AMICI

Generated by Doxygen 1.8.18

1 Main Page

1.0.1 Advanced Multilanguage Interface for CVODES and IDAS

1.0.2 About

AMICI provides a multi-language (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` or `PySB` and automatically compiles such models into `.mex` simulation files (Matlab), C++ executables or Python modules.

In contrast to the (no longer maintained) `sundialsTB` Matlab interface, all necessary functions are transformed into native C++ code, which allows for a significantly faster simulation.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 it is also applicable to a wider range of differential equation constrained optimization problems.

1.0.3 Current build status

1.0.4 Features

- SBML import (see details below)
- PySB import
- Generation of C++ code for model simulation and sensitivity computation
- Access to and high customizability of CVODES and IDAS solver
- Python, C++, Matlab interface
- Sensitivity analysis
  - forward
  - steady state
  - adjoint
  - first- and second-order
- Pre-equilibration and pre-simulation conditions
- Support for `discrete events and logical operations` (Matlab-only)

1.0.5 Interfaces & workflow

The AMICI workflow starts with importing a model from either `SBML` (Matlab, Python), `PySB` (Python), or a Matlab definition of the model (Matlab-only). From this input, all equations for model simulation are derived symbolically and C++ code is generated. This code is then compiled into a C++ library, a Python module, or a Matlab `.mex` file and is then used for model simulation.

1.0.6 Getting started

The AMICI source code is available at https://github.com/AMICI-dev/AMICI/. To install AMICI, first read the installation instructions.

To get you started with Python-AMICI, the best way might be checking out this Jupyter notebook.

To get started with Matlab-AMICI, various examples are available in matlab/examples/.

Comprehensive documentation on installation and usage of AMICI is available online for the python and MATLAB/C++ interfaces.

Any contributions to AMICI are welcome (code, bug reports, suggestions for improvements, ...).

1.0.6.1 Getting help

In case of questions or problems with using AMICI, feel free to post an issue on Github. We are trying to get back to you quickly.

1.0.7 Projects using AMICI

There are several tools for parameter estimation offering good integration with AMICI:

- pyPESTO: Python library for optimization, sampling and uncertainty analysis
- pyABC: Python library for parallel and scalable ABC-SMC (Approximate Bayesian Computation - Sequential Monte Carlo)
- parPE: C++ library for parameter estimation of ODE models offering distributed memory parallelism with focus on problems with many simulation conditions.

1.0.8 Publications

Citeable DOI for the latest AMICI release:

There is a list of publications using AMICI. If you used AMICI in your work, we are happy to include your project, please let us know via a Github issue.

When using AMICI in your project, please cite

- 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 and/or
- 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

When presenting work that employs AMICI, feel free to use one of the icons in documentation/gfx/, which are available under a CC0 license:

1.0.9 Status of SBML support in Python-AMICI

Python-AMICI currently passes 696 out of the 1780 (∼39%) test cases from the semantic `SBML Test Suite` (`current status`).

In addition, we currently plan to add support for the following features (see corresponding issues for details and progress):

- Events (currently Matlab-only)
- Algebraic rules
- Models without species

contributions are welcome.

However, the following features are unlikely to be supported:

- SBML extensions
- `factorial()`, `ceil()`, `floor()`, due to incompatibility with symbolic sensitivity computations
- initial assignments for parameters
- `delay()` due to missing SUNDIALS solver support

In addition to SBML, we also plan to implement support for the `Simulation Experiment Description Markup Language (SED-ML)`.

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, be it new functionality, documentation, bug reports, or anything else.

If you would to contribute to AMICI, a good start is looking for issues tagged `good first issue` or `help wanted`. For other ideas or questions, just post an issue.

For code contributions, please read our developer's guide first.

4 Installation

4.1 Table of Contents

1. Availability
2. Python
3. MATLAB
4. C++ only
5. Dependencies

4.2 Availability

The sources for AMICI are available as

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

A Python package is available on pypi, see below.

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.

4.2.1 Obtaining AMICI via the GIT version control 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/AMICI-dev/AMICI` and a direct clone is possible via

```
git clone https://github.com/AMICI-dev/AMICI.git AMICI
```

4.3 Python

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

```
pip3 install amici
```

You can now import it as python module:

```
import amici
```

For cases where this installation fails, check below for special setups and custom installations. For Python-AMICI usage see https://github.com/AMICI-dev/AMICI/blob/master/documentation/PYTHON.md.

4.3.1 Installation of development versions

To install development versions which have not been released to pypi yet, you can install AMICI with pip directly from GitHub using:

```
pip3 install -e git+https://github.com/AMICI-dev/amici.git@develop#egg=amici\&subdirectory=python/sdist
```

Replace `develop` by the branch or commit you want to install.

Note that this will probably not work on Windows which does not support symlinks by default (https://stackoverflow.com/questions/5917249/git-symlinks-in-windows/49913019#49913019).

4.3.2 Light installation

In case you only want to use the AMICI Python package for generating model code for use with Matlab or C++ and don't want to bothered with any unnecessary dependencies, you can run

```
pip3 install --install-option --no-clibs amici
```

Note, however, that you will not be able to compile any model into a Python extension with this installation.

NOTE: If you run into an error with above installation command, install all AMICI dependencies listed in `setup.py` manually, and try again. (This is because `pip --install-option`s are applied to *all* installed packages, including dependencies.)

4.3.3 Anaconda

To use an Anaconda installation of python (https://www.anaconda.com/distribution/, Python>=3.6), proceed as follows:

Since Anaconda provides own versions of some packages which might not work with amici (in particular the gcc compiler), create a minimal virtual environment via:

```
conda create --name ENV_NAME pip python
```

Here, replace ENV_NAME by some name for the environment. To activate the environment, do:

```
source activate ENV_NAME
```

(and `conda deactivate` later to deactivate it again).

SWIG must be installed and available in your `PATH`, and a CBLAS-compatible BLAS must be available. You can also use conda to install the latter locally, using:

```
conda install -c conda-forge openblas
```

To install AMICI, now do:

```
pip install amici
```

The option `--no-cache` may be helpful here to make sure the installation is done completely anew.

Now, you are ready to use AMICI in the virtual environment.

4.3.3.1 Anaconda on Mac

If the above installation does not work for you, try installing AMICI via:

```
CFLAGS="-stdlib=libc++" CC=clang CXX=clang pip3 install --verbose amici
```

This will use the `clang` compiler.

You will have to pass the same options when compiling any model later on. This can be done by inserting the following code before calling `sbml2amici`:

```
import os
os.environ['CC'] = 'clang'
os.environ['CXX'] = 'clang'
os.environ['CFLAGS'] = '-stdlib=libc++'
```

(For further discussion see https://github.com/AMICI-dev/AMICI/issues/357)

4.3.4 Windows using GCC (mingw)

To install AMICI on Windows using python, you can proceed as follows:

Some general remarks:

- Install all libraries in a path not containing white spaces, e.g. directly under C:.
- Replace the following paths according to your installation.
- Slashes can be preferable to backslashes for some environment variables.
- See also #425 for further discussion.

Then, follow these steps:

- A python environment for Windows is required. We recommend Anaconda with python >=3.7.
- Install MinGW-W64 (32bit will succeed to compile, but fail during linking). MinGW-W64 GCC-8.1.0 for `x86_64-posix-sjlj` (direct link has been shown to work on Windows 7 and 10 test systems.
- Add the following directory to `PATH`:
  - `C:\mingw-w64\x86_64-8.1.0-posix-sjlj-rt_v6-rev0\mingw64\bin`
- Make sure that this is the compiler that is found by the system (e.g. `where gcc` in a `cmd` should point to this installation).
- Download CBLAS headers and libraries, e.g. OpenBLAS, binary distribution 0.2.19. Set the following environment variables:
  - `BLAS_CFLAGS=-IC:/OpenBLAS-v0.2.19-Win64-int32/include`
  - `BLAS_LIBS=-Wl,-Bstatic -LC:/OpenBLAS-v0.2.19-Win64-int32/lib -lopenblas -Wl,-Bdynamic`
- Install SWIG (version swigwin-3.0.12 worked) and add the following directory to `PATH`:
  - `C:\swigwin-3.0.12`

- Install AMICI using:

  ```
  pip install --global-option="build_clib" --global-option="--compiler=mingw32"
  --global-option="build_ext" --global-option="--compiler=mingw32" amici
  --no-cache-dir --verbose
  ```

Possible sources of errors:

- On recent Windows versions, `anaconda3\Lib\distutils\cygwinccompiler.py` fails linking `msvcr140.dll` with `[...] x86_64-w64-mingw32/bin/ld.exe: cannot find -lmsvcr140`. This is not required for amici, so in `cygwinccompiler.py` 'return ['msvcr140']can be changed toreturn []`.
- If you use a python version where python/cpython#880 has not been fixed yet, you need to disable `define hypot _hypot in anaconda3\include/pyconfig.h` yourself.
- `import amici` in python resulting in the very informative

  ImportError: DLL load failed: The specified module could not be found.

means that some amici module dependencies were not found (not the AMICI module itself). `Dependency↩ Walker` will show you which ones.

Support for msvc is experimental. `installOpenBLAS.ps1` and `compileBLAS.cmd` may serve as guidance on how to install openBLAS using msvc.

4.3.5 Windows using MSVC (Visual Studio)

4.3.5.1 Visual Studio We assume that Visual Studio (not to be confused with Visual Studio Code) is already installed. Using Visual Studio Installer, the following components need to be included:

- Microsoft Visual C++ (MSVC). This is part of multiple packages, including Desktop Development with C++.
- Windows Universal C Runtime. This is an individual component and installs some DLLs that we need.

4.3.5.2 openBLAS To install open BLAS, download the following scripts from AMICI:

https://github.com/AMICI-dev/AMICI/blob/master/scripts/installOpenBLAS.ps1
https://github.com/AMICI-dev/AMICI/blob/master/scripts/compileBLAS.cmd

The first script needs to be called in Powershell, and it needs to call `compileBLAS.cmd`, so you will need to modify line 11:

```
C:\Users\travis\build\AMICI\scripts\compileBLAS.cmd
```

so that it matches your directory structure. This will download openBLAS and compile it, creating

```
C:\BLAS\lib\openblas.lib
C:\BLAS\bin\openblas.dll
```

You will also need to define two environment variables:

```
BLAS_LIBS="/LIBPATH:C:\BLAS\lib openblas.lib"
BLAS_CFLAGS="/IC:\BLAS\OpenBLAS-v0.3.10\OpenBLAS-0.3.10"
```

One way to do that is to run a PowerShell script with the following commands:

```
[System.Environment]::SetEnvironmentVariable("BLAS_LIBS", "/LIBPATH:C:\BLAS\lib openblas.lib", [System.Environ
[System.Environment]::SetEnvironmentVariable("BLAS_LIBS", "/LIBPATH:C:\BLAS\lib openblas.lib", [System.Environ
[System.Environment]::SetEnvironmentVariable("BLAS_CFLAGS", "-IC:\BLAS\OpenBLAS-v0.3.10\OpenBLAS-0.3.10", [Sys
[System.Environment]::SetEnvironmentVariable("BLAS_CFLAGS", "-IC:\BLAS\OpenBLAS-v0.3.10\OpenBLAS-0.3.10", [Sys
```

The call ending in `Process` sets the environment variable in the current process, and it is no longer in effect in the next process. The call ending in `User` is permanent, and takes effect the next time the user logs on.

4.3.5.3 PATH Now we need to make sure that all required DLLs are within the scope of the PATH variable. In particular, the following directories need to be included in PATH:

```
C:\BLAS\bin
C:\Program Files (x86)\Windows Kits\10\Redist\ucrt\DLLs\x64
```

The first one is needed for `openblas.dll` and the second is needed for the Windows Universal C Runtime. If any DLLs are missing in the PATH variable, Python will return the following error:

```
ImportError: DLL load failed: The specified module could not be found.
```

This can be tested using the "where" command. For example

```
where openblas.dll
```

should return

```
C:\BLAS\bin\openblas.dll
```

Almost all of the DLLs are standard Windows DLLs and should be included in either Windows or Visual Studio. But, in case it is necessary to test this, here is a list of some DLLs required by AMICI (when compiled with MSVC):

- `openblas.dll`
- `python37.dll`
- `MSVCP140.dll`
- `KERNEL32.dll`
- `VCRUNTIME140_1.dll`
- `VCRUNTIME140.dll`
- `api-ms-win-crt-convert-l1-1-0.dll`
- `api-ms-win-crt-heap-l1-1-0.dll`
- `api-ms-win-crt-stdio-l1-1-0.dll`
- `api-ms-win-crt-string-l1-1-0.dll`
- `api-ms-win-crt-runtime-l1-1-0.dll`
- `api-ms-win-crt-time-l1-1-0.dll`
- `api-ms-win-crt-math-l1-1-0.dll`

`MSVCP140.dll`, `VCRUNTIME140.dll`, and `VCRUNTIME140_1.dll` are needed by MSVC (see Visual Studio above). `KERNEL32.dll` is part of Windows and in `C:\Windows\System32`. The `api-ms-win-crt-XXX-l1-1-0.dll` are needed by `openblas.dll` and are part of the Windows Universal C Runtime (see Visual Studio above).

4.3.6 Custom installation

AMICI Python package installation can be customized using a number of environment variables:

Variable	Purpose	Example
CC	Setting the C(++) compiler	CC=/usr/bin/g++
CFLAGS	Extra compiler flags used in every compiler call	
BLAS_CFLAGS	Compiler flags for, e.g. BLAS include directories	
BLAS_LIBS	Flags for linking BLAS	
ENABLE_GCOV_COVERAGE	Set to build AMICI to provide code coverage information	ENABLE_GCOV_COVERA↩ GE=TRUE
ENABLE_AMICI_DEBUGGING	Set to build AMICI with debugging symbols	ENABLE_AMICI_DEBUGGI↩ NG=TRUE
AMICI_PARALLEL_COMPILE	Set to the number of parallel processes to be used for C(++) file compilation (defaults to 1)	AMICI_PARALLEL_COMPI↩ LE=4

4.4 MATLAB

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

```
installAMICI.m
```

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

```
savepath
```

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

```
mex -setup c++
```

For a list of supported compilers we refer to the mathworks documentation: mathworks.com Note that Microsoft Visual Studio compilers are currently not supported.

4.5 C++ only

To use AMICI from C++, run the

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

script to compile AMICI library.

NOTE: On some systems, the CMake executable may be named something other than `cmake`. In this case, set the `CMAKE` environment variable to the correct name (e.g. `export CMAKE=cmake3`, in case you have CMake available as `cmake3`).

The static library file can then be linked from

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

In CMake-based packages, amici can be linked via

```
find_package(Amici)
```

4.5.1 Optional SuperLU_MT support

To build AMICI with SuperLU_MT support, run

```
./scripts/buildSuperLUMT.sh
./scripts/buildSundials.sh
cd build/
cmake -DSUNDIALS_SUPERLUMT_ENABLE=ON ..
make
```

4.6 Dependencies

4.6.1 General

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

4.6.1.1 libsbml

To import Systems Biology Markup Language (SBML) models, AMICI relies on the Python or MATLAB SBML library.

4.6.1.2 Math Kernel Library (MKL)

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

On Ubuntu, this requirement can be satisfied with

```
apt install libatlas-base-dev
```

On Fedora (32):

```
sudo dnf install blas-devel
```

4.6.1.3 C++ compiler All AMICI installations require a C++11-compatible C++ compiler. AMICI has been tested with g++, mingw, clang and the Intel compiler. Visual C++ is not officially supported, but may work.

4.6.1.4 HDF5 The python and C++ interfaces provide routines to read and write options and results in `hdf5` format. For the python interface, the installation of hdf5 is optional, but for the C++ interace it is currently required.

HDF5 can be installed using package managers such as `brew` or `apt`:

```
brew install hdf5
```

or

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

4.6.1.5 SWIG The python interface requires `SWIG`, which has to be installed by the user. As root user, SWIG can be installed using package managers such as `brew` or `apt`:

```
brew install swig
```

or

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

Or by non-root users, using `scripts/downloadAndBuildSwig.sh` from the AMICI repository (not included in the PyPI package). The binary directory has to be added to the `PATH` environment variable, or `SWIG` has to be set as described in the following section.

4.6.1.5.1 Using a non-default SWIG executable We note here that some linux package managers may provide swig executables as `swig3.0`, but installation as `swig` is required. This can be fixed as root user using, e.g., symbolic links:

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

Non-root users can set the `SWIG` environment variable to the full path of the desired SWIG executable. This variable has be set during AMICI package installation as well as during model compilation.

4.6.2 Matlab

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/AMICI-dev/AMICI/issues/307).

The Symbolic Toolbox requirement can be circumvented by performing model import using the Python interface. The result code can then be used from Matlab.

4.6.3 Python

The python interface requires python 3.6 or newer and a cblas-compatible BLAS 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 python interface depends on some additional packages, e.g. `numpy`. They are automatically installed when installing the python package.

4.6.4 C++

The C++ interface requires `cmake` and a cblas-compatible BLAS to be installed.

4.6.5 Optional

SuperLU_MT, "a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations" (https://crd-legacy.lbl.gov/~xiaoye/SuperLU/#superlu_mt). SuperLU_MT is optional and is so far only available from the C++ interface.

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.

5.1 Model Definition

This guide will guide the user on how to specify models to import and simulate them using the Python interface. For example implementations see the examples in the `python/examples` directory.

5.1.1 SBML import

SBML is a commonly used format for specifying systems biology models. To import an SBML model into AMICI, first, load an SBML file using the `amici.sbml_import.SbmlImporter` class:

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

the SBML model as imported by libSBML is available as

```
sbml_model = sbml_importer.sbml
```

5.1.1.1 Constants Model parameters that should be considered constants can be specified in a list of strings specifying the SBML ID of the respective parameter, e.g.:

```
constant_parameters=['k4']
```

5.1.1.2 Observables Observables are specified as a dictionary with observable ID as key and observable formula as value.

A convenient way for specifying observables for an SBML model is storing them as `AssignmentRule`s. Assignment rules that should be considered as observables can then be extracted using the `amici.assignment↩Rules2observables` function, e.g.:

```
observables = amici.assignmentRules2observables(sbml, filter_function=lambda variable:
                                                variable.getId().startswith('observable_') and not variable.ge
```

5.1.1.3 Standard Deviations Standard deviations can be specified as dictionaries ...

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

5.1.1.4 Model Compilation To generate a Python module from the SBML model, call the method `amici.↩sbml_import.SbmlImporter.sbml2amici`, passing all the previously defined model specifications:

```
sbml_importer.sbml2amici('test', 'test',
                         observables=observables,
                         constant_parameters=constant_parameters,
                         sigmas=sigmas)
```

5.1.2 PySB import

`PySB` is a tool for specifying rule-based systems biology models as Python code. AMICI can import PySB models via `amici.pysb_import.pysb2amici`.

`BioNetGen` and `Kappa` models can be imported into AMICI using PySB.

5.1.3 PEtab import

`PEtab` is a format for specifying parameter estimation problems. It is based on an SBML model and tab-separated value files specifying the observation mdoel and experimental conditions.

AMICI can import PEtab-based model definitions and run simulations for the specified simulations conditions. For usage, see `python/examples/example_petab/petab.ipynb`.

5.1.4 Importing plain ODEs

The AMICI Python interface does not currently support direct import of ODEs. However, it is straightforward to encode them as RateRules in an SBML model. The `yaml2sbml` package may come in handy, as it facilitates generating SBML models from a YAML-based specification of an ODE model. Besides the SBML model it can also create `PEtab` files.

5.2 Model Simulation

AMICI model import creates a Python module for simulation of the respective model. To use the model module, the model directory has to be manually added to the python path:

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

the compiled model can then be imported as

```
import test as model_module
```

It is usually more convenient to use `amici.import_model_module()` for that purpose.

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

```
model = model_module.getModel()
```

Specify the simulation timepoints via `amici.Model.setTimepoints`:

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

For simulation, we need to generate a solver instance:

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

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

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

5.3 Building models with OpenMP support

AMICI can be built with OpenMP support, which allows to parallelize model simulations for multiple experimental conditions.

On Linux and OSX this is enabled by default. This can be verified using

```
import amici
amici.compiledWithOpenMP()
```

If not already enabled by default, you can enable OpenMP support by setting the environment variables `AMICI_CXXFLAGS` and `AMICI_LDFLAGS` to the correct OpenMP flags of your compiler and linker, respectively. This has to be done for both AMICI package installation *and* model compilation. When using `gcc` on Linux, this would be:

```
# on your shell:
AMICI_CXXFLAGS=-fopenmp AMICI_LDFLAGS=-fopenmp pip3 install amici

# in python, before model compilation:
import os
os.environ['AMICI_CXXFLAGS'] = '-fopenmp'
os.environ['AMICI_LDFLAGS'] = '-fopenmp'
```

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.

6.1 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.

6.1.1 Header

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

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

6.1.2 Options

Set the options by specifying the respective field of the modelstruct

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

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

field	description	default
.param	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.

6.1.3 States

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

```
syms state1 state2 state3
```

Create the state vector containing all states:

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

6.1.4 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 ];
```

6.1.5 Constants

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

```
syms const1 const2
```

Create the parameters vector

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

6.1.6 Differential Equation

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

```
syms t
```

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

```
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.

6.1.7 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 ];
```

6.1.8 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.

6.1.9 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

6.1.10 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.

6.1.11 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`.

6.1.12 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.

6.2 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.

6.2.1 Compiling a Python-generated model

Due to better performance or to avoid the Symbolic Toolbox requirement, it might be desirable to import a model in Python and compile the resulting code into a mex file. For Python model import, consult the respective section of the Python documentation. Once the imported succeeded, there will be a `compileMexFile.m` script inside the newly created model directory which can be invoked to compile the mex file. This mex file and `simulate_*.m` can be used as if fully created by matlab.

6.2.1.1 Using Python-AMICI model import from Matlab With recent matlab versions it is possible to use the AMICI python package from within Matlab. This not quite comfortable yet, but it is possible.

Here for proof of concept:

- Install the python package as described in the documentation
- Ensure `pyversion` shows the correct python version (3.6 or 3.7)
- Then, from within the AMICI `matlab/` directory:

  ``` sbml_importer = py.amici.SbmlImporter('../python/examples/example_steadystate/model_steadystate↩ _scaled.xml') sbml_importer.sbml2amici('steadystate', 'steadystate_example_from_python') model = py.↩ steadystate.getModel() solver = model.getSolver() model.setTimepoints(linspace(0, 50, 51)) rdata = py.↩ amici.runAmiciSimulation(model, solver) result = struct(py.dict(rdata.items())) t = double(py.array.array('d', result.ts)) x = double(py.array.array('d', result.x.flatten())) x = reshape(x, flip(double(py.array.array('d', result.↩ x.shape)))) plot(t, x) ```

6.3 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.

6.3.1 Integration

Define a time vector:

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

Generate a parameter vector:

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

Generate a constants vector:

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

Integrate:

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

The integration status will be indicated by the `sol.status` flag. Negative values indicated failed integration. The states will then be available as sol.x. The observables will then be available as `sol.y`. The 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.
```

6.3.2 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.

6.3.3 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.

6.3.4 Steady State Sensitivities

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

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

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. For more details see the function level documentation, e.g. in http://icb-dcm.github.io/AMICI/amici_8cpp.html.

7.1 Generation of C++ model files

`amiwrap.m` (Matlab) and `amici.SbmlImporter.sbml2amici` (Python) import a model and create C++ code for various model functions. The model source files are written 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
```

These files provide the implementation of a model-specific subclass of `amici::Model`. The `CMakeLists.txt` file can be used to build the library. `main.cpp` contains a simple scaffold for running a model simulation from C++. See next section for more details on these files.

7.2 Compiling and linking

In your application, you need to compile and link every model ${AMICI_ROOT_DIR}/models/${MODEL_N↩AME}/*.cpp, ${AMICI_ROOT_DIR}/src/*.cpp, the SUNDIALS and the SuiteSparse library. The simplest and recommended way is using the CMake package configuration from the build directory (AmiciConfig.↩cmake for use with find_package which tells CMake about all AMICI dependencies.

During model import a CMake file (CMakeLists.txt) will be generated automatically. The CMake file shows the above-mentioned library dependencies. CMakeLists.txt, together with main.cpp, provides a scaffold for a standalone simulation program. The required numerical libraries are shipped with AMICI. To compile them, run ${AMICI_ROOT_DIR}/scripts/buildAll.sh once. HDF5 libraries and header files need to be installed separately (see AMICI installation instructions).

More information on how to run the compiled sample program is provided in main.cpp.

The file wrapfunctions.h exports some functions with generic names e.g. to create a model instance. These functions are meant to be used by applications which may be linked to single, potentially changing model to avoid hard-coding the model name. Including multiple wrapfunctions.h files from different models at once is not possible.

7.3 Running a simulation

The complete AMICI API is available through amici/amici.h; this is the only header file that needs to be included for basic usage. Additionally, amici/hdf5.h may be helpful. This file provides some functions for reading and writing HDF5 files). All functions are declared within the amici namespace.

The entry function for running an AMICI simulation is amici::runAmiciSimulation(...), declared in amici/amici.h.

This function requires

- a amici::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.
- a amici::Solver instance. This solver instance needs to match the requirements of the model and can be generated using model->getSolver().
- optionally an amici::ExpData instance, which contains any experimental data (e.g. measurements, noise model parameters or model inputs) to compute residuals or an objective function.

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

For running simulations for multiple experimental conditions (multiple amici::ExpData instances), amici↩::runAmiciSimulations(...) provides an alternative entry point. If AMICI (and your application) have been compiled with OpenMP support (see installation guide), this allows for running those simulations in parallel.

7.4 Parameter estimation for AMICI models in high-performance computing environments

To perform parameter estimation for large or otherwise computationally demanding AMICI models from C++ in a high-performance computing environment, you may find the parPE library helpful. parPE allows for the private or shared memory parallel evaluation of a cost function requiring multiple simulations of the same model with different inputs. It provides interfaces to different optimizers, such as Ipopt.

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: Remove the directory AMICI/models/`mexext` and compile again.

Q: It still does not compile.

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

Q: My Python-generated model does not compile from MATLAB.

A: Try building any of the available examples before. If this succeeds, retry building the original model. Some dependencies might not be built correctly when using only the `compileMexFile.m` script.

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: How are events interpreted in a DAE context?

A: Currently we only support impulse free events. Also sensitivities have never been tested. Proceed with care and create an `issue` if any problems arise!

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!

Q: I am trying to install the AMICI Python package, but installation fails with something like

```
amici/src/cblas.cpp:16:13: fatal error: cblas.h: No such file or directory
#include <cblas.h>
         ^~~~~~~~~
compilation terminated.
error: command 'x86_64-linux-gnu-gcc' failed with exit status 1
```

A: You will have to install a CBLAS-compatible BLAS library and/or set `BLAS_CFLAGS` as described in the installation guide.

Q: Importing my model fails with something like `ImportError: _someModelName.cpython-37m-x86↩_64-linux-gnu.so: undefined symbol: omp_get_thread_num`.

A: You probably installed the AMICI package with OpenMP support, but did not have the relevant compiler/linker flags set when importing/building the model. See `Python-AMICI guide`.

9 AMICI developer's guide

This document contains information for AMICI developers, not too relevant to regular users.

9.1 Branches / releases

AMICI roughly follows the `GitFlow`. All new contributions are merged into `develop`. These changes are regularly merged into `master` as new releases. For release versioning we are trying to follow `semantic versioning`. New releases are created on Github and are automatically deployed to `Zenodo` for archiving and to obtain a digital object identifier (DOI) to make them citable. Furthermore, our CI pipeline will automatically create and deploy a new release on `PyPI`.
We try to keep a clean git history. Therefore, feature pull requests are squash-merged to `develop`. Merging of release branches to master is done via merge commits.

9.2 When starting to work on some issue

When starting to work on some Github issue, please assign yourself to let other developers know that you are working on it to avoid duplicate work. If the respective issue is not completely clear, it is generally a good idea to ask for clarification before starting to work on it.
If you want to work on something new, please create a Github issue first.

9.3 Code contributions

When making code contributions, please follow our style guide and the process described below:

- Check if you agree to release your contribution under the conditions provided in `LICENSE`. By opening a pull requests you confirm us that you do agree.

- Start a new branch from `develop` (on your fork, or at the main repository if you have access)
- Implement your changes
- Submit a pull request to the `develop` branch
- Make sure your code is documented appropriately
  - Run `scripts/run-doxygen.sh` to check completeness of your documentation
- Make sure your code is compatible with C++11, `gcc` and `clang` (our CI pipeline will do this for you)
- When adding new functionality, please also provide test cases (see `tests/cpputest/` and documentation/CI.md)
- Write meaningful commit messages
- Run all tests to ensure nothing was broken (more details)
  - Run `scripts/buildAll.sh && scripts/run-cpputest.sh`.
  - If you made changes to the Matlab or C++ code and have a Matlab license, please also run `tests/cpputest/wrapTestModels.m` and `tests/testModels.m`
  - If you made changes to the Python or C++ code, run `make python-tests` in `build`
- When all tests are passing and you think your code is ready to merge, request a code review (see also our code review guideline)
- Wait for feedback. If you do not receive feedback to your pull request within a week, please give us a friendly reminder.

9.3.1 Style guide

9.3.1.1 General

- All files and functions should come with file-level and function-level documentation.
- All new functionality should be covered by unit or integration tests. Runtime of those tests should be kept as short as possible.

9.3.1.2 Python

- We want to be compatible with Python 3.6
- For the Python code we want to follow `PEP8`. Although this is not the case for all existing code, any new contributions should do so.
- We use Python `type hints` for all functions (but not for class attributes, since they are not supported by the current Python doxygen filter). In Python code type hints should be used instead of doxygen `<br><b>Type</b>:`. (All legacy `<br><b>Type</b>:` attributes are to be removed.)

  For function docstrings, follow this format:

```
"""One-line description.
Possible a more detailed description
Arguments:
    Argument1: This needs to start on the same line, otherwise the current
        doxygen filter will fail.
    Returns:
        Return value
    Raises:
        SomeError in case of some error.
"""
```

9.3.1.3 C++

- We follow C++11
- We want to maintain compatibility with g++, clang and the Intel C++ compiler
- For code formatting, we use the settings from `.clang-format` in the root directory
- *Details to be defined*

9.3.1.4 Matlab *To be defined*

10 Continuous integration (CI) and tests

AMICI uses a continuous integration pipeline running on https://travis-ci.org/. This includes the following steps:

- Checking existence and format of documentation
- Static code analysis (http://cppcheck.sourceforge.net/)
- Unit and integration tests
- Memory leak detection

More details are provided in the sections below.
The CI scripts and tests can be found in `tests/` and `scripts/`. Some of the tests are integrated with CMake, see `make help` in the build directory.

10.1 C++ unit and integration tests

To run C++ tests, build AMICI with `make` or `scripts/buildAll.sh`, then run `scripts/run-cpputest.sh`.

10.2 Python unit and integration tests

To run Python tests, run `../scripts/run-python-tests.sh` from anywhere (assumes build directory is `build/`) or run `make python-tests` in your build directory.

10.2.1 SBML Test Suite

We test Python-AMICI SBML support using the test cases from the semantic SBML Test Suite. When making changes to the model import functions, make sure to run these tests.
To run the SBML Test Suite test cases, the easiest way is:

1. Running `scripts/installAmiciSource.sh` which creates a virtual Python environment and performs a development installation of AMICI from the current repository. (This needs to be run only once or after AMICI model generation or C++ changes).

2. Running `scripts/run-SBMLTestsuite.sh`. This will download the test cases if necessary and run them all. A subset of test cases can be selected with an optional argument (e.g. `scripts/run-SBMLTestsuite.sh 1,3-6,8`, to run cases 1, 3, 4, 5, 6 and 8).

   Once the test cases are available locally, for debugging it might be easier to directly use `pytest` with `tests/testSBMLSuite.py`.

10.3 Matlab tests (not included in CI pipeline)

To execute the Matlab test suite, run `tests/testModels.m`.

10.4 Model simulation integration tests

Many of our integration tests are model simulations. The simulation results obtained from the Python and C++ are compared to results saved in an HDF5 file (`tests/cpputest/expectedResults.h5`). Settings and data for the test simulations are also specified in this file.
Note: The C++ code for the models is included in the repository under `models/`. This code is to be updated whenever `amici::Model` changes.

10.4.1 Regenerating C++ code of the test models

Regeneration of the model code must done whenever `amici::Model` or the Matlab model import routines change.
This is done with

```
tests/cpputest/wrapTestModels.m
```

Note: This is currently only possible from Matlab < R2018a. This should change as soon as 1) all second-order sensitivity code is ported to C++/Python, 2) a non-SBML import exists for Python and 3) support for events has been added for Python.

10.4.2 Regenerating expected results

To update test results, run `make test` in the build directory, replace `tests/cpputest/expected↩Results.h5` by `tests/cpputest/writeResults.h5.bak` [ONLY DO THIS AFTER TRIPLE CHEC↩KING 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
```

10.5 Adding/Updating tests

To add new tests add a new corresponding python script (see, e.g., `./tests/generateTest↩Config/example_dirac.py`) and add it to and run `tests/generateTestConfigurationFor↩Examples.sh`. Then regenerate the expected test results (see above).

11 AMICI documentation

This file describes how the AMICI documentation is organized and compiled.
The AMICI documentation is created using `doxygen`. It combines Markdown files from the root directory, from `documentation/` and in-source documentation from the C++, Python and Matlab source files.
The documentation is generated by running

```
scripts/run-doxygen.sh
```

The resulting HTML and PDF documentation will be created in `doc/`. Upon merges to the `master` branch, the documentation will be deployed to github-pages at `http://icb-dcm.github.io/AMICI/`.
`scripts/run-doxygen.sh` also checks for any missing in-source documentation.

11.1 Doxygen configuration

The main doxygen configuration file is located in `matlab/mtoc/config/Doxyfile.template`. Edit this file for inclusion or exclusion of additional files.

11.2 Matlab documentation

Matlab documentation is processed by `mtoc++`. This is configured in `matlab/mtoc/config`.

11.3 Python documentation

Python documentation is processed by doxygen and doxypypy using the script and filters in `scripts/`.

11.4 Out-of-source documentation

Out-of-source documentation files should be written in Markdown and created within `documentation/`. Graphics for documentation are kept in `documentation/gfx/`.
Some guidelines:

- Note that there are some incompatibilities of Github Markdown and Doxygen Markdown. Ideally documentation should be written in a format compatible with both. This affects for example images links which currently cause trouble in Doxygen.

- Where possible, relative links are preferred over absolute links. However, they should work with both Github and Doxygen and ideally with local files for offline use.
- Please stick to the limit of 80 characters per line for readability of raw Markdown files where possible.

  However, note that some Markdown interpreters can handle line breaks within links and headings, whereas others cannot. Here, compatibility is preferred over linebreaks.
- Avoid trailing whitespace

11.5 Maintaining the list of publications

We want to maintain a list of publications / projects using AMICI. This is located at `documentation/references.md`. This file is created by `documentation/recreate_reference_list.py` based on the bibtex file `documentation/amici_refs.bib`.
After any changes to `documentation/amici_refs.bib`, please run

```
documentation/recreate_reference_list.py
```

12 Contributor Covenant Code of Conduct

12.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

12.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

12.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.
Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

12.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

12.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at `froehlichfab@gmail.com`. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.
Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

12.6 Attribution

This Code of Conduct is adapted from the `Contributor Covenant`, version 1.4, available at `https://www.contributor-covenant.org/version/1/4/code-of-conduct.html`
For answers to common questions about this code of conduct, see `https://www.contributor-covenant.org/faq`

13 AMICI with Docker

13.1 Create image

In the AMICI base directory run:

```
git archive -o docker/amici.tar.gz --format=tar.gz HEAD
cd docker && docker build -t $USER/amici:latest .
```

13.2 Published images

AMICI docker images are regularly published to `https://hub.docker.com/layers/dweindl/amici/`.

14 Code review

A guide for reviewing code and having your code reviewed by others.

14.1 Everyone

- Don't be too protective of your code
- Accept that, to a large extent, coding decisions are a matter of personal preference
- Don't get personal
- Ask for clarification
- Avoid strong language
- Try to understand your counterpart's perspective
- Clarify how strong you feel about each discussion point

14.2 Reviewing code

- If there are no objective advantages, don't force your style on others
- Ask questions instead of making demands
- Assume the author gave his best
- Mind the scope (many things are nice to have, but might be out of scope of the current change - open a new issue)
- The goal is "good enough", not "perfect"
- Be constructive
- You do not always have to request changes

14.3 Having your code reviewed

- Don't take it personal - the review is on the code, not on you
- Code reviews take time, appreciate the reviewer's comments
- Assume the reviewer did his best (but might still be wrong)
- Keep code changes small (e.g. separate wide reformatting from actual code changes to facility review)
- If the reviewer does not understand your code, probably many others won't either

14.4 Checklist

- [ ] Adherence to project-specific style guide
- [ ] The code is self-explanatory
- [ ] The code is concise / expressive
- [ ] Meaningful identifiers are used
- [ ] Corner-cases are covered, cases not covered fail loudly
- [ ] The code can be expected to scale well (enough)
- [ ] The code is well documented (e.g., input, operation, output), but without trivial comments
- [ ] The code is `SOLID`
- [ ] New code is added in the most meaningful place (i.e. matches the current architecture)
- [ ] No magic numbers
- [ ] No hard-coded values that should be user inputs
- [ ] No dead code left
- [ ] The changes make sense
- [ ] The changes are not obviously degrading performance
- [ ] There is no duplicated code
- [ ] The API is convenient
- [ ] Code block length and complexity is adequate
- [ ] Spelling okay
- [ ] The code is tested

15 LICENSE

Creative Commons Legal Code
CC0 1.0 Universal

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE

LEGAL SERVICES. DISTRIBUTION OF THIS DOCUMENT DOES NOT CREATE AN ATTORNEY-CLIENT REL↩ ATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE CO↩ MMONS MAKES NO WARRANTIES REGARDING THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER.

Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an "owner") of an original work of authorship and/or a database (each, a "Work").

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works ("Commons") that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the "Affirmer"), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

1. Copyright and Related Rights. A Work made available under CC0 may be protected by copyright and related or neighboring rights ("Copyright and Related Rights"). Copyright and Related Rights include, but are not limited to, the following:

   i. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work; ii. moral rights retained by the original author(s) and/or performer(s); iii. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work; iv. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below; v. rights protecting the extraction, dissemination, use and reuse of data in a Work; vi. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and vii. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

2. Waiver. To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

3. Public License Fallback. Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever,

including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.

4. Limitations and Disclaimers.

a. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document. b. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law. c. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work. d. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.

16 README

The graphics in this directory are under CC0 1.0 license. See LICENSE.md file.

17 References

List of publications using AMICI. Total number is 36.
If you applied AMICI in your work and your publication is missing, please let us know via a new Github issue.

2020

Alabert, Constance, Carolin Loos, Moritz Voelker-Albert, Simona Graziano, Ignasi Forné, Nazaret Reveron-Gomez, Lea Schuh, et al. 2020. "Domain Model Explains Propagation Dynamics and Stability of Histone H3k27 and H3k36 Methylation Landscapes." *Cell Reports* 30 (4): 1223–1234.e8. https://doi.org/10.1016/j.celrep.2019.12.060.

Schmiester, Leonard, Daniel Weindl, and Jan Hasenauer. 2020. "Parameterization of Mechanistic Models from Qualitative Data Using an Efficient Optimal Scaling Approach." *Journal of Mathematical Biology*, July. https://doi.org/10.1007/s00285-020-01522-w.

2019

Dharmarajan, Lekshmi, Hans-Michael Kaltenbach, Fabian Rudolf, and Joerg Stelling. 2019. "A Simple and Flexible Computational Framework for Inferring Sources of Heterogeneity from Single-Cell Dynamics." *Cell Systems* 8 (1): 15–26.e11. https://doi.org/10.1016/j.cels.2018.12.007.

Fischer, David S., Anna K. Fiedler, Eric Kernfeld, Ryan M. J. Genga, Aimée Bastidas-Ponce, Mostafa Bakhti, Heiko Lickert, Jan Hasenauer, Rene Maehr, and Fabian J. Theis. 2019. "Inferring Population Dynamics from Single-Cell Rna-Sequencing Time Series Data." *Nature Biotechnology* 37: 461–68. https://doi.org/10.1038/s41587-019-0088-0.

Gregg, Robert W, Saumendra N Sarkar, and Jason E Shoemaker. 2019. "Mathematical Modeling of the cGAS Pathway Reveals Robustness of Dna Sensing to Trex1 Feedback." *Journal of Theoretical Biology* 462 (February): 148–57. https://doi.org/10.1016/j.jtbi.2018.11.001.

Kapfer, Eva-Maria, Paul Stapor, and Jan Hasenauer. 2019. "Challenges in the Calibration of Large-Scale Ordinary Differential Equation Models." *IFAC-PapersOnLine* 52 (26): 58–64. https://doi.org/https://doi.org/10.1016/j.ifacol.2019.12.236.

Lines, Glenn Terje, Lukasz Paszkowski, Leonard Schmiester, Daniel Weindl, Paul Stapor, and Jan Hasenauer. 2019. "Efficient Computation of Steady States in Large-Scale Ode Models of Biochemical Reaction Networks." *Accepted for Publication in Proc. Of the Foundations of Syst. Biol. In Engin. (FOSBE)*.
Nousiainen, Kari, Jukka Intosalmi, and Harri Lähdesmäki. 2019. "A Mathematical Model for Enhancer Activation Kinetics During Cell Differentiation." In *Algorithms for Computational Biology*, edited by Ian Holmes, Carlos Martın-Vide, and Miguel A. Vega-Rodrıguez, 191–202. Cham: Springer International Publishing.
Pedretscher, B., B. Kaltenbacher, and O. Pfeiler. 2019. "Parameter Identification and Uncertainty Quantification in Stochastic State Space Models and Its Application to Texture Analysis." *Applied Numerical Mathematics* 146: 38–54. https://doi.org/https://doi.org/10.1016/j.apnum.2019.06.020.
Pitt, Jake Alan, and Julio R Banga. 2019. "Parameter Estimation in Models of Biological Oscillators: An Automated Regularised Estimation Approach." *BMC Bioinformatics* 20 (1): 82. https://doi.org/10.↩1186/s12859-019-2630-y.
Schmiester, Leonard, Yannik Schälte, Fabian Fröhlich, Jan Hasenauer, and Daniel Weindl. 2019. "Efficient parameterization of large-scale dynamic models based on relative measurements." *Bioinformatics*, July. https↩://doi.org/10.1093/bioinformatics/btz581.
Stapor, Paul, Leonard Schmiester, Christoph Wierling, Bodo Lange, Daniel Weindl, and Jan Hasenauer. 2019. "Mini-Batch Optimization Enables Training of Ode Models on Large-Scale Datasets." *bioRxiv*. https↩://doi.org/10.1101/859884.
Villaverde, Alejandro F., Elba Raimúndez, Jan Hasenauer, and Julio R. Banga. 2019. "A Comparison of Methods for Quantifying Prediction Uncertainty in Systems Biology." *Accepted for Publication in Proc. Of the Foundations of Syst. Biol. In Engin. (FOSBE)*.
Wang, Dantong, Paul Stapor, and Jan Hasenauer. 2019. "Dirac Mixture Distributions for the Approximation of Mixed Effects Models." *Accepted for Publication in Proc. Of the Foundations of Syst. Biol. In Engin. (FOSBE)*.

2018

Ballnus, Benjamin, Steffen Schaper, Fabian J Theis, and Jan Hasenauer. 2018. "Bayesian Parameter Estimation for Biochemical Reaction Networks Using Region-Based Adaptive Parallel Tempering." *Bioinformatics* 34 (13)↩: i494–i501. https://doi.org/10.1093/bioinformatics/bty229.
Bast, Lisa, Filippo Calzolari, Michael Strasser, Jan Hasenauer, Fabian J. Theis, Jovica Ninkovic, and Carsten Marr. 2018. "Subtle Changes in Clonal Dynamics Underlie the Age-Related Decline in Neurogenesis." *Cell Reports*.
Fröhlich, Fabian, Thomas Kessler, Daniel Weindl, Alexey Shadrin, Leonard Schmiester, Hendrik Hache, Artur Muradyan, et al. 2018. "Efficient Parameter Estimation Enables the Prediction of Drug Response Using a Mechanistic Pan-Cancer Pathway Model." *Cell Systems* 7 (6): 567–579.e6. https://doi.org/10.1016/j.cels.↩2018.10.013.
Fröhlich, Fabian, Anita Reiser, Laura Fink, Daniel Woschée, Thomas Ligon, Fabian Joachim Theis, Joachim Oskar Rädler, and Jan Hasenauer. 2018. "Multi-Experiment Nonlinear Mixed Effect Modeling of Single-Cell Translation Kinetics After Transfection." *Npj Systems Biology and Applications* 5 (1): 1. https://doi.org/10.↩1038/s41540-018-0079-7.
Kaltenbacher, Barbara, and Barbara Pedretscher. 2018. "Parameter Estimation in Sdes via the Fokker–Planck Equation: Likelihood Function and Adjoint Based Gradient Computation." *Journal of Mathematical Analysis and Applications* 465 (2): 872–84. https://doi.org/https://doi.org/10.1016/j.jmaa.2018.↩05.048.
Loos, Carolin, Sabrina Krause, and Jan Hasenauer. 2018. "Hierarchical Optimization for the Efficient Parametrization of ODE Models." *Bioinformatics* 34 (24): 4266–73. https://doi.org/10.↩1093/bioinformatics/bty514.
Loos, Carolin, Katharina Moeller, Fabian Fröhlich, Tim Hucho, and Jan Hasenauer. 2018. "A Hierarchical, Data-↩Driven Approach to Modeling Single-Cell Populations Predicts Latent Causes of Cell-to-Cell Variability." *Cell Systems* 6 (5): 593–603. https://doi.org/10.1016/j.cels.2018.04.008.
Schälte, Y., P. Stapor, and J. Hasenauer. 2018. "Evaluation of Derivative-Free Optimizers for Parameter Estimation in Systems Biology." *FAC-PapersOnLine* 51 (19): 98–101.
Stapor, Paul, Fabian Fröhlich, and Jan Hasenauer. 2018. "Optimization and Profile Calculation of ODE Models Using Second Order Adjoint Sensitivity Analysis." *Bioinformatics* 34 (13): i151–i159.
Villaverde, Alejandro F., Fabian Froehlich, Daniel Weindl, Jan Hasenauer, and Julio R Banga. 2018. "Benchmarking Optimization Methods for Parameter Estimation in Large Kinetic Models." *Bioinformatics*, bty736.

2017

Ballnus, B., S. Hug, K. Hatz, L. Görlitz, J. Hasenauer, and F. J. Theis. 2017. "Comprehensive Benchmarking of Markov Chain Monte Carlo Methods for Dynamical Systems." *BMC Syst. Biol.* 11 (63). https://doi.org/10.1186/s12918-017-0433-1.

Fröhlich, F., B. Kaltenbacher, F. J. Theis, and J. Hasenauer. 2017. "Scalable Parameter Estimation for Genome-Scale Biochemical Reaction Networks." *PLoS Comput. Biol.* 13 (1): e1005331. https://doi.org/10.1371/journal.pcbi.1005331.

Fröhlich, F., F. J. Theis, J. O. Rädler, and J. Hasenauer. 2017. "Parameter Estimation for Dynamical Systems with Discrete Events and Logical Operations." *Bioinformatics* 33 (7): 1049–56. https://doi.org/10.1093/bioinformatics/btw764.

Kazeroonian, A., F. J. Theis, and J. Hasenauer. 2017. "A Scalable Moment-Closure Approximation for Large-Scale Biochemical Reaction Networks." *Bioinformatics* 33 (14): i293–i300. https://doi.org/10.1093/bioinformatics/btx249.

Maier, C., C. Loos, and J. Hasenauer. 2017. "Robust Parameter Estimation for Dynamical Systems from Outlier-Corrupted Data." *Bioinformatics* 33 (5): 718–25. https://doi.org/10.1093/bioinformatics/btw703.

2016

Boiger, R., J. Hasenauer, S. Hross, and B. Kaltenbacher. 2016. "Integration Based Profile Likelihood Calculation for PDE Constrained Parameter Estimation Problems." *Inverse Prob.* 32 (12): 125009. https://doi.org/10.1088/0266-5611/32/12/125009.

Fiedler, A., S. Raeth, F. J. Theis, A. Hausser, and J. Hasenauer. 2016. "Tailored Parameter Optimization Methods for Ordinary Differential Equation Models with Steady-State Constraints." *BMC Syst. Biol.* 10 (80). https://doi.org/10.1186/s12918-016-0319-7.

Fröhlich, F., P. Thomas, A. Kazeroonian, F. J. Theis, R. Grima, and J. Hasenauer. 2016. "Inference for Stochastic Chemical Kinetics Using Moment Equations and System Size Expansion." *PLoS Comput. Biol.* 12 (7): e1005030. https://doi.org/10.1371/journal.pcbi.1005030.

Hross, S., A. Fiedler, F. J. Theis, and J. Hasenauer. 2016. "Quantitative Comparison of Competing PDE Models for Pom1p Dynamics in Fission Yeast." In *Proc. 6th IFAC Conf. Found. Syst. Biol. Eng.*, edited by R. Findeisen, E. Bullinger, E. Balsa-Canto, and K. Bernaerts, 49:264–69. 26. IFAC-PapersOnLine. https://doi.org/10.1016/j.ifacol.2016.12.136.

Kazeroonian, A., F. Fröhlich, A. Raue, F. J. Theis, and J. Hasenauer. 2016. "CERENA: Chemical REaction Network Analyzer – A Toolbox for the Simulation and Analysis of Stochastic Chemical Kinetics." *PLoS ONE* 11 (1): e0146732. https://doi.org/10.1371/journal.pone.0146732.

Loos, C., A. Fiedler, and J. Hasenauer. 2016. "Parameter Estimation for Reaction Rate Equation Constrained Mixture Models." In *Proc. 13th Int. Conf. Comp. Meth. Syst. Biol.*, edited by E. Bartocci, P. Lio, and N. Paoletti, 186–200. Lecture Notes in Bioinformatics. Springer International Publishing. https://doi.org/10.1007/978-3-319-45177-0.

2015

Loos, C., C. Marr, F. J. Theis, and J. Hasenauer. 2015. "Computational Methods in Systems Biology." In, edited by O. Roux and J. Bourdon, 9308:52–63. Lecture Notes in Computer Science. Springer International Publishing.

18 Class Documentation

18.1 AbstractModel Class Reference

Abstract base class of amici::Model defining functions that need to be implemented in an AMICI model.

```
#include <abstract_model.h>
```

Inheritance diagram for AbstractModel:

Public Member Functions

- virtual std::unique_ptr< Solver > getSolver ()=0

  Retrieves the solver object.
- virtual void froot (const realtype t, const AmiVector &x, const AmiVector &dx, gsl::span< realtype > root)=0

  Root function.
- virtual void fxdot (const realtype t, const AmiVector &x, const AmiVector &dx, AmiVector &xdot)=0

  Residual function.
- virtual void fsxdot (const realtype t, const AmiVector &x, const AmiVector &dx, int ip, const AmiVector &sx, const AmiVector &sdx, AmiVector &sxdot)=0

  Sensitivity Residual function.
- virtual void fxBdot_ss (const realtype t, const AmiVector &xB, const AmiVector &dxB, AmiVector &xBdot)=0

  Residual function backward when running in steady state mode.
- virtual void fJSparseB_ss (SUNMatrix JB)=0

  Sparse Jacobian function backward, steady state case.
- virtual void writeSteadystateJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot)=0

  Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.
- virtual void fJ (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J)=0

  Dense Jacobian function.
- virtual void fJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB)=0

  Dense Jacobian function.
- virtual void fJSparse (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J)=0

  Sparse Jacobian function.
- virtual void fJSparseB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB)=0

  Sparse Jacobian function.
- virtual void fJDiag (const realtype t, AmiVector &Jdiag, realtype cj, const AmiVector &x, const AmiVector &dx)=0

  Diagonal Jacobian function.

- virtual void fdxdotdp (const realtype t, const AmiVector &x, const AmiVector &dx)=0

  Model specific sparse implementation of explicit parameter derivative of right hand side.
- virtual void fJv (const realtype t, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, const AmiVector &v, AmiVector &nJv, realtype cj)=0

  Jacobian multiply function.
- virtual const std::string getAmiciVersion () const

  Returns the amici version that was used to generate the model.
- virtual const std::string getAmiciCommit () const

  Returns the amici commit that was used to generate the model.
- virtual void fx0 (realtype ∗x0, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fx0.
- virtual bool isFixedParameterStateReinitializationAllowed () const

  Function indicating whether reinitialization of states depending on fixed parameters is permissible.
- virtual void fx0_fixedParameters (realtype ∗x0, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fx0_fixedParameters.
- virtual void fsx0_fixedParameters (realtype ∗sx0, const realtype t, const realtype ∗x0, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsx0_fixedParameters.
- virtual void fsx0 (realtype ∗sx0, const realtype t, const realtype ∗x0, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsx0.
- virtual void fdx0 (AmiVector &x0, AmiVector &dx0)

  Initial value for time derivative of states (only necessary for DAEs)
- virtual void fstau (realtype ∗stau, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip, int ie)

  Model specific implementation of fstau.
- virtual void fy (realtype ∗y, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w)

  Model specific implementation of fy.
- virtual void fdydp (realtype ∗dydp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip, const realtype ∗w, const realtype ∗dwdp)

  Model specific implementation of fdydp.
- virtual void fdydx (realtype ∗dydx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗dwdx)

  Model specific implementation of fdydx.
- virtual void fz (realtype ∗z, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of fz.
- virtual void fsz (realtype ∗sz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip)

  Model specific implementation of fsz.
- virtual void frz (realtype ∗rz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of frz.
- virtual void fsrz (realtype ∗srz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip)

  Model specific implementation of fsrz.
- virtual void fdzdp (realtype ∗dzdp, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip)

  Model specific implementation of fdzdp.
- virtual void fdzdx (realtype ∗dzdx, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

Model specific implementation of fdzdx.

- virtual void fdrzdp (realtype ∗drzdp, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip)

  Model specific implementation of fdrzdp.
- virtual void fdrzdx (realtype ∗drzdx, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of fdrzdx.
- virtual void fdeltax (realtype ∗deltax, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ie, const realtype ∗xdot, const realtype ∗xdot_old)

  Model specific implementation of fdeltax.
- virtual void fdeltasx (realtype ∗deltasx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, int ip, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗sx, const realtype ∗stau)

  Model specific implementation of fdeltasx.
- virtual void fdeltaxB (realtype ∗deltaxB, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗xB)

  Model specific implementation of fdeltaxB.
- virtual void fdeltaqB (realtype ∗deltaqB, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗xB)

  Model specific implementation of fdeltaqB.
- virtual void fsigmay (realtype ∗sigmay, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fsigmay.
- virtual void fdsigmaydp (realtype ∗dsigmaydp, const realtype t, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsigmay.
- virtual void fsigmaz (realtype ∗sigmaz, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fsigmaz.
- virtual void fdsigmazdp (realtype ∗dsigmazdp, const realtype t, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsigmaz.
- virtual void fJy (realtype ∗nllh, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fJy.
- virtual void fJz (realtype ∗nllh, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fJz.
- virtual void fJrz (realtype ∗nllh, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz)

  Model specific implementation of fJrz.
- virtual void fdJydy (realtype ∗dJydy, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fdJydy.
- virtual void fdJydsigma (realtype ∗dJydsigma, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fdJydsigma.
- virtual void fdJzdz (realtype ∗dJzdz, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fdJzdz.
- virtual void fdJzdsigma (realtype ∗dJzdsigma, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fdJzdsigma.
- virtual void fdJrzdz (realtype ∗dJrzdz, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗rz, const realtype ∗sigmaz)

  Model specific implementation of fdJrzdz.

- virtual void fdJrzdsigma (realtype ∗dJrzdsigma, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗rz, const realtype ∗sigmaz)

  Model specific implementation of fdJrzdsigma.
- virtual void fw (realtype ∗w, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗tcl)

  Model specific implementation of fw.
- virtual void fdwdp (realtype ∗dwdp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl, const realtype ∗stcl)

  Model specific sparse implementation of dwdp.
- virtual void fdwdp_colptrs (sunindextype ∗indexptrs)

  Model specific implementation for dwdp, column pointers.
- virtual void fdwdp_rowvals (sunindextype ∗indexvals)

  Model specific implementation for dwdp, row values.
- virtual void fdwdp (realtype ∗dwdp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl, const realtype ∗stcl, int ip)

  Model specific sensitivity implementation of dwdp.
- virtual void fdwdx (realtype ∗dwdx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl)

  Model specific implementation of dwdx, data part.
- virtual void fdwdx_colptrs (sunindextype ∗indexptrs)

  Model specific implementation for dwdx, column pointers.
- virtual void fdwdx_rowvals (sunindextype ∗indexvals)

  Model specific implementation for dwdx, row values.

18.1.1 Detailed Description

Some functions have empty default implementations or throw. This class shall not have any data members.
Definition at line 25 of file abstract_model.h.

18.1.2 Member Function Documentation

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

Returns

The Solver instance

Implemented in Model_DAE, and Model_ODE.

18.1.2.2 froot()

```
virtual void froot (
    const realtype t,
    const AmiVector & x,
    const AmiVector & dx,
    gsl::span< realtype > root ) [pure virtual]
```

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_DAE, and Model_ODE.

Here is the caller graph for this function:

18.1.2.3 fxdot()

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

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_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.4 fsxdot()

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

Parameters

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

Implemented in Model_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.5 fxBdot_ss()

```
virtual void fxBdot_ss (
    const realtype t,
    const AmiVector & xB,
    const AmiVector & dxB,
    AmiVector & xBdot ) [pure virtual]
```

Parameters

t	time
xB	adjoint state
dxB	time derivative of state (DAE only)
xBdot	array to which values of the residual function will be written

Implemented in Model_DAE, and Model_ODE.

18.1.2.6 fJSparseB_ss()

```
virtual void fJSparseB_ss (
    SUNMatrix JB ) [pure virtual]
```

Parameters

JB	sparse matrix to which values of the Jacobian will be written

Implemented in Model_DAE, and Model_ODE.

18.1.2.7 writeSteadystateJB()

```
virtual void writeSteadystateJB (
    const realtype t,
    realtype cj,
```

```
const AmiVector & x,
const AmiVector & dx,
const AmiVector & xB,
const AmiVector & dxB,
const AmiVector & xBdot )  [pure virtual]
```

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
xBdot	Vector with the adjoint state right hand side

Implemented in Model_DAE, and Model_ODE.

18.1.2.8 fJ()

```
virtual void fJ (
const realtype t,
realtype cj,
const AmiVector & x,
const AmiVector & dx,
const AmiVector & xdot,
SUNMatrix J )  [pure virtual]
```

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_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.9 fJB()

```
virtual void fJB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB ) [pure virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implemented in Model_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.10 fJSparse()

```
virtual void fJSparse (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xdot,
    SUNMatrix J ) [pure virtual]
```

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)

Parameters

J	sparse matrix to which values of the Jacobian will be written

Implemented in Model_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.11 fJSparseB()

```
virtual void fJSparseB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB )  [pure virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implemented in Model_DAE, and Model_ODE.
Here is the caller graph for this function:

18.1.2.12 fJDiag()

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

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)

Implemented in Model_ODE, and Model_DAE.
Here is the caller graph for this function:

18.1.2.13 fdxdotdp()

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

Parameters

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

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

18.1.2.14 fJv()

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

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.

18.1.2.15 getAmiciVersion()

```
const std::string getAmiciVersion ( ) const [virtual]
```

Returns

ver amici version string

Definition at line 6 of file abstract_model.cpp.

18.1.2.16 getAmiciCommit()

```
const std::string getAmiciCommit ( ) const [virtual]
```

Returns

ver amici commit string

Definition at line 12 of file abstract_model.cpp.

18.1.2.17 fx0()

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

Parameters

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

Definition at line 18 of file abstract_model.cpp.

18.1.2.18 isFixedParameterStateReinitializationAllowed()

```
bool isFixedParameterStateReinitialization↩
Allowed ( ) const  [virtual]
```

Returns

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

Definition at line 29 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.19 fx0_fixedParameters()

```
void fx0_fixedParameters (
    realtype * x0,
    const realtype t,
    const realtype * p,
    const realtype * k )  [virtual]
```

Parameters

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

Definition at line 35 of file abstract_model.cpp.

18.1.2.20 fsx0_fixedParameters()

```
void fsx0_fixedParameters (
    realtype * sx0,
    const realtype t,
    const realtype * x0,
    const realtype * p,
    const realtype * k,
    int ip )  [virtual]
```

Parameters

sx0	initial state sensitivities
t	initial time

Parameters

x0	initial state
p	parameter vector
k	constant vector
ip	sensitivity index

Definition at line 44 of file abstract_model.cpp.

18.1.2.21 fsx0()

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

Parameters

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

Definition at line 55 of file abstract_model.cpp.

18.1.2.22 fdx0()

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

Parameters

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

Definition at line 68 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.23 fstau()

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

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 74 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.24 fy()

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

Parameters

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

Definition at line 90 of file abstract_model.cpp.

18.1.2.25 fdydp()

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

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
w	repeating elements vector
dwdp	Recurring terms in xdot, parameter derivative

Definition at line 104 of file abstract_model.cpp.

18.1.2.26 fdydx()

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

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
w	repeating elements vector
dwdx	Recurring terms in xdot, state derivative

Definition at line 120 of file abstract_model.cpp.

18.1.2.27 fz()

```
void fz (
    realtype * z,
```

```
int ie,
const realtype t,
const realtype * x,
const realtype * p,
const realtype * k,
const realtype * h )  [virtual]
```

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 135 of file abstract_model.cpp.

18.1.2.28 fsz()

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

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 149 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.29 frz()

```
void frz (
    realtype * rz,
    int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h )  [virtual]
```

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 165 of file abstract_model.cpp.

18.1.2.30 fsrz()

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

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 179 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.31 fdzdp()

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

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 195 of file abstract_model.cpp.

18.1.2.32 fdzdx()

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

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 210 of file abstract_model.cpp.

18.1.2.33 fdrzdp()

```
void fdrzdp (
    realtype * drzdp,
    int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
```

```
const realtype * h,
int ip )  [virtual]
```

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 224 of file abstract_model.cpp.

18.1.2.34 fdrzdx()

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

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 239 of file abstract_model.cpp.

18.1.2.35 fdeltax()

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

Parameters

deltax	state update
t	current time
x	current state
p	parameter vector

Parameters

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 253 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.36 fdeltasx()

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

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 269 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.37 fdeltaxB()

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

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 289 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.38 fdeltaqB()

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

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 306 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.39 fsigmay()

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

Parameters

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

Definition at line 324 of file abstract_model.cpp.

18.1.2.40 fdsigmaydp()

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

Parameters

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

Definition at line 335 of file abstract_model.cpp.

18.1.2.41 fsigmaz()

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

Parameters

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

Definition at line 347 of file abstract_model.cpp.

18.1.2.42 fdsigmazdp()

```
void fdsigmazdp (
    realtype * dsigmazdp,
    const realtype t,
    const realtype * p,
    const realtype * k,
    int ip ) [virtual]
```

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 358 of file abstract_model.cpp.

18.1.2.43 fJy()

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

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 370 of file abstract_model.cpp.

18.1.2.44 fJz()

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

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 384 of file abstract_model.cpp.

18.1.2.45 fJrz()

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

Parameters

nllh	regularization for event measurements z

Parameters

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 398 of file abstract_model.cpp.

18.1.2.46 fdJydy()

```
void fdJydy (
    realtype * dJydy,
    int iy,
    const realtype * p,
    const realtype * k,
    const realtype * y,
    const realtype * sigmay,
    const realtype * my )  [virtual]
```

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 411 of file abstract_model.cpp.

18.1.2.47 fdJydsigma()

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

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 425 of file abstract_model.cpp.

18.1.2.48 fdJzdz()

```
void fdJzdz (
    realtype * dJzdz,
    int iz,
    const realtype * p,
    const realtype * k,
    const realtype * z,
    const realtype * sigmaz,
    const realtype * mz )  [virtual]
```

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 439 of file abstract_model.cpp.

18.1.2.49 fdJzdsigma()

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

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 453 of file abstract_model.cpp.

18.1.2.50 fdJrzdz()

```
void fdJrzdz (
    realtype * dJrzdz,
    int iz,
    const realtype * p,
    const realtype * k,
    const realtype * rz,
    const realtype * sigmaz )  [virtual]
```

Parameters

dJrzdz	partial derivative of event penalization Jrz

Parameters

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 467 of file abstract_model.cpp.

18.1.2.51 fdJrzdsigma()

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

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 480 of file abstract_model.cpp.

18.1.2.52 fw()

```
void fw (
    realtype * w,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * tcl ) [virtual]
```

Parameters

w	Recurring terms in xdot
t	timepoint
x	vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
tcl	total abundances for conservations laws

Definition at line 493 of file abstract_model.cpp.

18.1.2.53 fdwdp() [1/2]

```
void fdwdp (
```

```
realtype * dwdp,
const realtype t,
const realtype * x,
const realtype * p,
const realtype * k,
const realtype * h,
const realtype * w,
const realtype * tcl,
const realtype * stcl )  [virtual]
```

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
tcl	total abundances for conservations laws
stcl	sensitivities of total abundances for conservations laws

Definition at line 505 of file abstract_model.cpp.

18.1.2.54 fdwdp_colptrs() `void fdwdp_colptrs ( sunindextype * indexptrs ) [virtual]`

Parameters

indexptrs	column pointers

Definition at line 519 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.55 fdwdp_rowvals() `void fdwdp_rowvals (`

```
sunindextype * indexvals )  [virtual]
```

Parameters

indexvals	row values

Definition at line 525 of file abstract_model.cpp.
Here is the caller graph for this function:

18.1.2.56 fdwdp() [2/2]

```
void fdwdp (
    realtype * dwdp,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w,
    const realtype * tcl,
    const realtype * stcl,
    int ip )  [virtual]
```

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
tcl	total abundances for conservations laws
stcl	sensitivities of total abundances for conservations laws
ip	sensitivity parameter index

Definition at line 531 of file abstract_model.cpp.

18.1.2.57 fdwdx()

```
void fdwdx (
    realtype * dwdx,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w,
    const realtype * tcl ) [virtual]
```

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
tcl	total abundances for conservations laws

Definition at line 546 of file abstract_model.cpp.

18.1.2.58 fdwdx_colptrs()

```
void fdwdx_colptrs (
    sunindextype * indexptrs ) [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 559 of file abstract_model.cpp.

Here is the caller graph for this function:

18.1.2.59 fdwdx_rowvals()

```
void fdwdx_rowvals (
    sunindextype * indexvals ) [virtual]
```

Parameters

indexvals	row values

Definition at line 565 of file abstract_model.cpp.

Here is the caller graph for this function:

18.2 AmiciApplication Class Reference

Main class for making calls to AMICI.

`#include <amici.h>`

Public Member Functions

- std::unique_ptr< ReturnData > runAmiciSimulation (Solver &solver, const ExpData *edata, Model &model, bool rethrow=false)

  Core integration routine. Initializes the solver and runs the forward and backward problem.
- std::vector< std::unique_ptr< ReturnData > > runAmiciSimulations (Solver const &solver, const std::vector< ExpData * > &edatas, Model const &model, bool failfast, int num_threads)

  Same as runAmiciSimulation, but for multiple ExpData instances.
- void warningF (const char *identifier, const char *format,...) const

  printf interface to warning()
- void errorF (const char *identifier, const char *format,...) const

  printf interface to error()
- int checkFinite (gsl::span< const realtype > array, const char *fun)

  Checks the values in an array for NaNs and Infs.

Public Attributes

- outputFunctionType warning = printWarnMsgIdAndTxt
- outputFunctionType error = printErrMsgIdAndTxt

18.2.1 Detailed Description

This class is used to provide separate AMICI contexts, for example, for use in multi-threaded applications where different threads want to use AMICI with different settings, such custom logging functions.
NOTE: For this moment, the context object needs to be set manually to any Model and Solver object. If not set, they will use the default output channel.
Definition at line 44 of file amici.h.

18.2.2 Member Function Documentation

18.2.2.1 runAmiciSimulation()

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

Parameters

solver	Solver instance
edata	pointer to experimental data object
model	model specification object
rethrow	rethrow integration exceptions?

Returns

rdata pointer to return data object

Definition at line 98 of file amici.cpp.
Here is the caller graph for this function:

18.2.2.2 runAmiciSimulations()

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

Parameters

solver	Solver instance
edatas	experimental data objects
model	model specification object
failfast	flag to allow early termination
num_threads	number of threads for parallel execution

Returns

vector of pointers to return data objects

Definition at line 200 of file amici.cpp.

Here is the call graph for this function:

18.2.2.3 warningF()

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

Parameters

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

Definition at line 241 of file amici.cpp.

Here is the caller graph for this function:

18.2.2.4 errorF()

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

Parameters

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

Definition at line 251 of file amici.cpp.

Here is the caller graph for this function:

18.2.2.5 checkFinite()

```
int checkFinite (
    gsl::span< const realtype > array,
    const char * fun )
```

Parameters

array	array
fun	name of calling function

Returns

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

Definition at line 261 of file amici.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.2.3 Member Data Documentation

18.2.3.1 warning `outputFunctionType warning = printWarnMsgIdAndTxt`

Function to process warnings

Definition at line 79 of file amici.h.

18.2.3.2 error `outputFunctionType error = printErrMsgIdAndTxt`

Function to process errors

Definition at line 82 of file amici.h.

18.3 AmiException Class Reference

AMICI exception class.

`#include <exception.h>`

Inheritance diagram for AmiException:

Public Member Functions

- AmiException (char const ∗fmt,...)

  Constructor with printf style interface.
- const char ∗ what () const noexcept override

  Override of default error message function.
- const char ∗ getBacktrace () const

  Returns the stored backtrace.
- void storeBacktrace (int nMaxFrames)

  Stores the current backtrace.

18.3.1 Detailed Description

Has a printf style interface to allow easy generation of error messages
Definition at line 16 of file exception.h.

18.3.2 Constructor & Destructor Documentation

18.3.2.1 AmiException()

```
AmiException (
    char const * fmt,
    ... ) [explicit]
```

Parameters

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

Definition at line 11 of file exception.cpp.

Here is the call graph for this function:

18.3.3 Member Function Documentation

18.3.3.1 what() `const char * what ( ) const [override], [noexcept]`

Returns

msg error message

Definition at line 19 of file exception.cpp.

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

Returns

trace backtrace

Definition at line 23 of file exception.cpp.

18.3.3.3 storeBacktrace() `void storeBacktrace ( int nMaxFrames )`

Parameters

nMaxFrames	number of frames to go back in stacktrace

Definition at line 27 of file exception.cpp.
Here is the caller graph for this function:

18.4 AmiVector Class Reference

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

Public Member Functions

- AmiVector ()=default

  Default constructor.
- AmiVector (const long int length)

  empty constructor
- AmiVector (std::vector< realtype > rvec)

  constructor from std::vector,
- AmiVector (gsl::span< realtype > rvec)

  constructor from gsl::span,
- AmiVector (const AmiVector &vold)

  copy constructor
- AmiVector (AmiVector &&other) noexcept

  move constructor
- ~AmiVector ()

  destructor
- AmiVector & operator= (AmiVector const &other)

  copy assignment operator
- realtype * data ()

  data accessor
- const realtype * data () const

  const data accessor
- N_Vector getNVector ()

  N_Vector accessor.
- const_N_Vector getNVector () const

  N_Vector accessor.
- std::vector< realtype > const & getVector () const

  Vector accessor.
- int getLength () const

  returns the length of the vector
- void reset ()

  resets the Vector by filling with zero values
- void minus ()

  changes the sign of data elements
- void set (realtype val)

  sets all data elements to a specific value
- realtype & operator[ ] (int pos)

  accessor to data elements of the vector
- realtype & at (int pos)

  accessor to data elements of the vector
- const realtype & at (int pos) const

  accessor to data elements of the vector
- void copy (const AmiVector &other)

  copies data from another AmiVector

18.4.1 Detailed Description

AmiVector class provides a generic interface to the NVector_Serial struct
Definition at line 20 of file vector.h.

18.4.2 Constructor & Destructor Documentation

18.4.2.1 AmiVector() [1/5]

```
AmiVector (
    const long int length ) [explicit]
```

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

Definition at line 35 of file vector.h.

18.4.2.2 AmiVector() [2/5]

```
AmiVector (
    std::vector< realtype > rvec ) [explicit]
```

Moves data from std::vector and constructs an nvec that points to the data

Parameters

rvec	vector from which the data will be moved

Definition at line 44 of file vector.h.

18.4.2.3 AmiVector() [3/5]

```
AmiVector (
    gsl::span< realtype > rvec ) [explicit]
```

Copy data from gsl::span and constructs a vector

Parameters

rvec	vector from which the data will be copied

Definition at line 52 of file vector.h.

18.4.2.4 AmiVector() [4/5]

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

Parameters

vold	vector from which the data will be copied

Definition at line 59 of file vector.h.

18.4.2.5 AmiVector() [5/5]

```
AmiVector (
    AmiVector && other ) [noexcept]
```

Parameters

other	vector from which the data will be moved

Definition at line 68 of file vector.h.

18.4.3 Member Function Documentation

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

Parameters

other	right hand side

Returns

left hand side

Definition at line 7 of file vector.cpp.

18.4.3.2 data() [1/2] `realtype * data ( )`

Returns

pointer to data array

Definition at line 13 of file vector.cpp.
Here is the caller graph for this function:

18.4.3.3 data() [2/2] `const realtype * data ( ) const`

Returns

const pointer to data array

Definition at line 15 of file vector.cpp.

18.4.3.4 getNVector() [1/2] `N_Vector getNVector ( )`

Returns

N_Vector

Definition at line 17 of file vector.cpp.

Here is the caller graph for this function:

18.4.3.5 getNVector() [2/2] `const_N_Vector getNVector ( ) const`

Returns

N_Vector

Definition at line 19 of file vector.cpp.

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

Returns

Vector

Definition at line 21 of file vector.cpp.
Here is the caller graph for this function:

18.4.3.7 getLength() `int getLength ( ) const`

Returns

length

Definition at line 23 of file vector.cpp.

Here is the caller graph for this function:

18.4.3.8 set()

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

Parameters

val	value for data elements

Definition at line 32 of file vector.cpp.
Here is the caller graph for this function:

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

Parameters

pos	index of element

Returns

element

Definition at line 34 of file vector.cpp.

18.4.3.10 at() [1/2] `realtype & at ( int pos )`

Parameters

pos	index of element

Returns

element

Definition at line 38 of file vector.cpp.
Here is the caller graph for this function:

18.4.3.11 at() [2/2]

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

Parameters

pos	index of element

Returns

element

Definition at line 42 of file vector.cpp.

18.4.3.12 copy()

```
void copy (
    const AmiVector & other )
```

Parameters

other	data source

Definition at line 46 of file vector.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.5 AmiVectorArray Class Reference

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

Public Member Functions

- AmiVectorArray ()=default

  Default constructor.
- AmiVectorArray (long int length_inner, long int length_outer)

  empty constructor
- AmiVectorArray (const AmiVectorArray &vaold)

  copy constructor
- AmiVectorArray & operator= (AmiVectorArray const &other)

  copy assignment operator
- realtype ∗ data (int pos)

  accessor to data of AmiVector elements
- const realtype ∗ data (int pos) const

  const accessor to data of AmiVector elements
- realtype & at (int ipos, int jpos)

  accessor to elements of AmiVector elements
- const realtype & at (int ipos, int jpos) const

  const accessor to elements of AmiVector elements
- N_Vector ∗ getNVectorArray ()

  accessor to NVectorArray
- N_Vector getNVector (int pos)

  accessor to NVector element
- const_N_Vector getNVector (int pos) const

const accessor to NVector element

- AmiVector & operator[ ] (int pos)

  accessor to AmiVector elements
- const AmiVector & operator[ ] (int pos) const

  const accessor to AmiVector elements
- int getLength () const

  length of AmiVectorArray
- void reset ()

  resets every AmiVector in AmiVectorArray
- void flatten_to_vector (std::vector< realtype > &vec) const

  flattens the AmiVectorArray to a vector in row-major format
- void copy (const AmiVectorArray &other)

  copies data from another AmiVectorArray

18.5.1 Detailed Description

Provides a generic interface to arrays of NVector_Serial structs
Definition at line 181 of file vector.h.

18.5.2 Constructor & Destructor Documentation

18.5.2.1 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

Definition at line 66 of file vector.cpp.

18.5.2.2 AmiVectorArray() [2/2]

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

Parameters

vaold	object to copy from

Definition at line 83 of file vector.cpp.

Here is the call graph for this function:

18.5.3 Member Function Documentation

18.5.3.1 operator=() `AmiVectorArray & operator= ( AmiVectorArray const & other )`

Parameters

other	right hand side

Returns

left hand side

Definition at line 74 of file vector.cpp.
Here is the call graph for this function:

18.5.3.2 data() [1/2] `realtype * data ( int pos )`

Parameters

pos	index of AmiVector

Returns

pointer to data array

Definition at line 91 of file vector.cpp.
Here is the caller graph for this function:

18.5.3.3 data() [2/2]

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

Parameters

pos	index of AmiVector

Returns

const pointer to data array

Definition at line 93 of file vector.cpp.

18.5.3.4 at() [1/2]

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

Parameters

ipos	inner index in AmiVector
jpos	outer index in AmiVectorArray

Returns

element

Definition at line 97 of file vector.cpp.
Here is the caller graph for this function:

18.5.3.5 at() [2/2] const realtype & at (

```
int ipos,
int jpos ) const
```

Parameters

ipos	inner index in AmiVector
jpos	outer index in AmiVectorArray

Returns

element

Definition at line 101 of file vector.cpp.

18.5.3.6 getNVectorArray() N_Vector * getNVectorArray ( )

Returns

N_VectorArray

Definition at line 105 of file vector.cpp.

18.5.3.7 getNVector() [1/2] N_Vector getNVector (

```
int pos )
```

Parameters

pos	index of corresponding AmiVector

Returns

N_Vector

Definition at line 107 of file vector.cpp.
Here is the caller graph for this function:

18.5.3.8 getNVector() [2/2] `const_N_Vector getNVector ( int pos ) const`

Parameters

pos	index of corresponding AmiVector

Returns

N_Vector

18.5.3.9 operator[]() [1/2] `AmiVector & operator[] ( int pos )`

Parameters

pos	index of AmiVector

Returns

AmiVector

Definition at line 109 of file vector.cpp.

18.5.3.10 operator[]() [2/2] `const AmiVector & operator[] ( int pos ) const`

Parameters

pos	index of AmiVector

Returns

const AmiVector

Definition at line 111 of file vector.cpp.

18.5.3.11 getLength() `int getLength ( ) const`

Returns

length

Definition at line 115 of file vector.cpp.
Here is the caller graph for this function:

18.5.3.12 flatten_to_vector() `void flatten_to_vector ( std::vector< realtype > & vec ) const`

Parameters

vec	vector into which the AmiVectorArray will be flattened. Must have length equal to number of elements.

Definition at line 124 of file vector.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.5.3.13 copy()

```
void copy (
    const AmiVectorArray & other )
```

Parameters

other	data source

Definition at line 142 of file vector.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.6 BackwardProblem Class Reference

class to solve backwards problems.
`#include <backwardproblem.h>`

Public Member Functions

- BackwardProblem (const ForwardProblem &fwd, const SteadystateProblem *posteq)

  Construct backward problem from forward problem.
- void workBackwardProblem ()

  Solve the backward problem.
- realtype gett () const

  Accessor for current time t.
- int getwhich () const

  Accessor for which.
- int * getwhichptr ()

  Accessor for pointer to which.
- std::vector< realtype > const & getdJydx () const

  Accessor for dJydx.
- AmiVector const & getAdjointState () const

  Accessor for xB.
- AmiVector const & getAdjointQuadrature () const

  Accessor for xQB.

18.6.1 Detailed Description

solves the backwards problem for adjoint sensitivity analysis and handles events and data-points
Definition at line 23 of file backwardproblem.h.

18.6.2 Constructor & Destructor Documentation

18.6.2.1 BackwardProblem()

```
BackwardProblem (
    const ForwardProblem & fwd,
    const SteadystateProblem * posteq ) [explicit]
```

Parameters

fwd	pointer to corresponding forward problem
posteq	pointer to postequilibration problem, can be nullptr

Definition at line 15 of file backwardproblem.cpp.

Here is the call graph for this function:

18.6.3 Member Function Documentation

18.6.3.1 workBackwardProblem() `void workBackwardProblem ( )`

If adjoint sensitivities are enabled this will also compute sensitivities. workForwardProblem must be called before this is function is called.
Definition at line 58 of file backwardproblem.cpp.
Here is the call graph for this function:

18.6.3.2 gett() `realtype gett ( ) const`

Returns

t

Definition at line 46 of file backwardproblem.h.

18.6.3.3 getwhich() `int getwhich ( ) const`

Returns

which

Definition at line 54 of file backwardproblem.h.

18.6.3.4 getwhichptr() `int* getwhichptr ( )`

Returns

which

Definition at line 62 of file backwardproblem.h.

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

Returns

dJydx

Definition at line 70 of file backwardproblem.h.

18.6.3.6 getAdjointState() `AmiVector const& getAdjointState ( ) const`

Returns

xB

Definition at line 78 of file backwardproblem.h.
Here is the caller graph for this function:

18.6.3.7 getAdjointQuadrature() `AmiVector const& getAdjointQuadrature ( ) const`

Returns

xQB

Definition at line 86 of file backwardproblem.h.
Here is the caller graph for this function:

18.7 ConditionContext Class Reference

The ConditionContext class applies condition-specific amici::Model settings and restores them when going out of scope.

`#include <edata.h>`

Inheritance diagram for ConditionContext:

Public Member Functions

- ConditionContext (Model ∗model, const ExpData ∗edata=nullptr, FixedParameterContext fpc=Fixed↩ ParameterContext::simulation)

  Apply condition-specific settings from edata to model while keeping a backup of the original values.
- ConditionContext & **operator=** (const ConditionContext &other)=delete
- void applyCondition (const ExpData ∗edata, FixedParameterContext fpc)

  Apply condition-specific settings from edata to the constructor-supplied model, not changing the settings which were backed-up in the constructor call.
- void restore ()

  Restore original settings on constructor-supplied amici::Model. Will be called during destruction. Explicit call is generally not necessary.

18.7.1 Detailed Description

Definition at line 524 of file edata.h.

18.7.2 Constructor & Destructor Documentation

18.7.2.1 ConditionContext()

```
ConditionContext (
    Model * model,
    const ExpData * edata = nullptr,
    FixedParameterContext fpc = FixedParameterContext::simulation ) [explicit]
```

Parameters

model	
edata	
fpc	flag indicating which fixedParmeter from edata to apply

Definition at line 326 of file edata.cpp.

Here is the call graph for this function:

18.7.3 Member Function Documentation

18.7.3.1 applyCondition()

```
void applyCondition (
    const ExpData * edata,
    FixedParameterContext fpc )
```

Parameters

edata	
fpc	flag indicating which fixedParmeter from edata to apply

Definition at line 351 of file edata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.8 ContextManager Class Reference

Generic implementation for a context manager, explicitely deletes copy and move operators for derived classes.
`#include <misc.h>`
Inheritance diagram for ContextManager:

Public Member Functions

- **ContextManager** (ContextManager &other)=delete
- **ContextManager** (ContextManager &&other)=delete

18.8.1 Detailed Description

Definition at line 183 of file misc.h.

18.9 CvodeException Class Reference

cvode exception handler class
`#include <exception.h>`
Inheritance diagram for CvodeException:

Public Member Functions

- CvodeException (int error_code, const char *function)

  Constructor.

18.9.1 Detailed Description

Definition at line 52 of file exception.h.

18.9.2 Constructor & Destructor Documentation

18.9.2.1 CvodeException()

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

Parameters

error_code	error code returned by cvode function
function	cvode function name

Definition at line 32 of file exception.cpp.

18.10 ExpData Class Reference

ExpData carries all information about experimental or condition-specific data.
`#include <edata.h>`

Public Member Functions

- ExpData ()=default

  default constructor
- ExpData (const ExpData &)=default

  Copy constructor, needs to be declared to be generated in swig.
- ExpData (int nytrue, int nztrue, int nmaxevent)

  constructor that only initializes dimensions
- ExpData (int nytrue, int nztrue, int nmaxevent, std::vector< realtype > ts)

  constructor that initializes timepoints from vectors
- ExpData (int nytrue, int nztrue, int nmaxevent, std::vector< realtype > ts, std::vector< realtype > fixedParameters)

  constructor that initializes timepoints and fixed parameters from vectors
- ExpData (int nytrue, int nztrue, int nmaxevent, std::vector< realtype > ts, std::vector< realtype > const &observedData, std::vector< realtype > const &observedDataStdDev, std::vector< realtype > const &observedEvents, std::vector< realtype > const &observedEventsStdDev)

  constructor that initializes timepoints and data from vectors
- ExpData (const Model &model)

  constructor that initializes with Model
- ExpData (const ReturnData &rdata, realtype sigma_y, realtype sigma_z)

  constructor that initializes with returnData, adds noise according to specified sigmas
- ExpData (const ReturnData &rdata, std::vector< realtype > sigma_y, std::vector< realtype > sigma_z)

  constructor that initializes with returnData, adds noise according to specified sigmas
- int nytrue () const

  number of observables of the non-augmented model
- int nztrue () const

  number of event observables of the non-augmented model
- int nmaxevent () const

  maximal number of events to track
- int nt () const

  number of timepoints
- void setTimepoints (const std::vector< realtype > &ts)

  set function that copies data from input to ExpData::ts
- std::vector< realtype > const & getTimepoints () const

  get function that copies data from ExpData::ts to output
- realtype getTimepoint (int it) const

  get function that returns timepoint at index
- void setObservedData (const std::vector< realtype > &observedData)

  set function that copies data from input to ExpData::my
- void setObservedData (const std::vector< realtype > &observedData, int iy)

  set function that copies observed data for specific observable
- bool isSetObservedData (int it, int iy) const

  get function that checks whether data at specified indices has been set
- std::vector< realtype > const & getObservedData () const

  get function that copies data from ExpData::observedData to output
- const realtype * getObservedDataPtr (int it) const

  get function that returns a pointer to observed data at index
- void setObservedDataStdDev (const std::vector< realtype > &observedDataStdDev)

  set function that copies data from input to ExpData::observedDataStdDev
- void setObservedDataStdDev (realtype stdDev)

  set function that sets all ExpData::observedDataStdDev to the input value
- void setObservedDataStdDev (const std::vector< realtype > &observedDataStdDev, int iy)

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

- void setObservedDataStdDev (realtype stdDev, int iy)

  set function that sets all standard deviation of a specific observable to the input value
- bool isSetObservedDataStdDev (int it, int iy) const

  get function that checks whether standard deviation of data at specified indices has been set
- std::vector< realtype > const & getObservedDataStdDev () const

  get function that copies data from ExpData::observedDataStdDev to output
- const realtype * getObservedDataStdDevPtr (int it) const

  get function that returns a pointer to standard deviation of observed data at index
- void setObservedEvents (const std::vector< realtype > &observedEvents)

  set function that copies observed event data from input to ExpData::observedEvents
- void setObservedEvents (const std::vector< realtype > &observedEvents, int iz)

  set function that copies observed event data for specific event observable
- bool isSetObservedEvents (int ie, int iz) const

  get function that checks whether event data at specified indices has been set
- std::vector< realtype > const & getObservedEvents () const

  get function that copies data from ExpData::mz to output
- const realtype * getObservedEventsPtr (int ie) const

  get function that returns a pointer to observed data at ieth occurence
- void setObservedEventsStdDev (const std::vector< realtype > &observedEventsStdDev)

  set function that copies data from input to ExpData::observedEventsStdDev
- void setObservedEventsStdDev (realtype stdDev)

  set function that sets all ExpData::observedDataStdDev to the input value
- void setObservedEventsStdDev (const std::vector< realtype > &observedEventsStdDev, int iz)

  set function that copies standard deviation of observed data for specific observable
- void setObservedEventsStdDev (realtype stdDev, int iz)

  set function that sets all standard deviation of a specific observable to the input value
- bool isSetObservedEventsStdDev (int ie, int iz) const

  get function that checks whether standard deviation of even data at specified indices has been set
- std::vector< realtype > const & getObservedEventsStdDev () const

  get function that copies data from ExpData::observedEventsStdDev to output
- const realtype * getObservedEventsStdDevPtr (int ie) const

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

Public Attributes

- std::vector< realtype > fixedParameters

  condition-specific fixed parameters of size Model::nk() or empty
- std::vector< realtype > fixedParametersPreequilibration

  condition-specific fixed parameters for pre-equilibration of size Model::nk() or empty. Overrides Solver::newton_preeq
- std::vector< realtype > fixedParametersPresimulation

  condition-specific fixed parameters for pre-simulation of size Model::nk() or empty.
- std::vector< realtype > parameters

  condition-specific parameters of size Model::np() or empty
- std::vector< realtype > x0

  condition-specific initial conditions of size Model::nx() or empty
- std::vector< realtype > sx0

  *condition-specific initial condition sensitivities of size Model::nx() * Model::nplist(), Model::nx() * ExpDataplist.size(), if ExpData::plist is not empty, or empty*
- std::vector< ParameterScaling > pscale

  condition-specific parameter scales of size Model::np()

- std::vector< int > plist

  condition-specific parameter list
- realtype t_presim = 0

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

Protected Member Functions

- void applyDimensions ()

  resizes observedData, observedDataStdDev, observedEvents and observedEventsStdDev
- void applyDataDimension ()

  resizes observedData and observedDataStdDev
- void applyEventDimension ()

  resizes observedEvents and observedEventsStdDev
- void checkDataDimension (std::vector< realtype > const &input, const char *fieldname) const

  checker for dimensions of input observedData or observedDataStdDev
- void checkEventsDimension (std::vector< realtype > const &input, const char *fieldname) const

  checker for dimensions of input observedEvents or observedEventsStdDev

Protected Attributes

- int nytrue_ {0}

  number of observables
- int nztrue_ {0}

  number of event observables
- int nmaxevent_ {0}

  maximal number of event occurences
- std::vector< realtype > ts_

  observation timepoints (dimension: nt)
- std::vector< realtype > observed_data_

  observed data (dimension: nt x nytrue, row-major)
- std::vector< realtype > observed_data_std_dev_

  standard deviation of observed data (dimension: nt x nytrue, row-major)
- std::vector< realtype > observed_events_

  observed events (dimension: nmaxevents x nztrue, row-major)
- std::vector< realtype > observed_events_std_dev_

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

18.10.1 Detailed Description

Definition at line 19 of file edata.h.

18.10.2 Constructor & Destructor Documentation

18.10.2.1 ExpData() [1/7]

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

Parameters

nytrue	
nztrue	
nmaxevent	

Definition at line 14 of file edata.cpp.
Here is the call graph for this function:

18.10.2.2 ExpData() [2/7]

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

Parameters

nytrue	(dimension: scalar)
nztrue	(dimension: scalar)
nmaxevent	(dimension: scalar)
ts	(dimension: nt)

Definition at line 20 of file edata.cpp.
Here is the call graph for this function:

18.10.2.3 ExpData() [3/7]

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

Parameters

nytrue	(dimension: scalar)
nztrue	(dimension: scalar)
nmaxevent	(dimension: scalar)
ts	(dimension: nt)

Parameters

fixedParameters	(dimension: nk)

Definition at line 27 of file edata.cpp.
Here is the call graph for this function:

18.10.2.4 ExpData() [4/7]

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

Parameters

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

Definition at line 36 of file edata.cpp.

Here is the call graph for this function:

18.10.2.5 ExpData() [5/7]

```
ExpData (
    const Model & model ) [explicit]
```

Parameters

model	pointer to model specification object

Definition at line 51 of file edata.cpp.
Here is the call graph for this function:

18.10.2.6 ExpData() [6/7]

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

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 58 of file edata.cpp.

18.10.2.7 ExpData() [7/7]

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

Parameters

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

Definition at line 61 of file edata.cpp.
Here is the call graph for this function:

18.10.3 Member Function Documentation

18.10.3.1 nytrue() `int nytrue ( ) const`

Returns

number of observables of the non-augmented model

Definition at line 311 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.2 nztrue() `int nztrue ( ) const`

Returns

number of event observables of the non-augmented model

Definition at line 316 of file edata.cpp.

18.10.3.3 nmaxevent() `int nmaxevent ( ) const`

Returns

maximal number of events to track

Definition at line 321 of file edata.cpp.

18.10.3.4 nt() `int nt ( ) const`

Returns

number of timepoints

Definition at line 109 of file edata.cpp.
Here is the caller graph for this function:

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

Parameters

ts	timepoints

Definition at line 98 of file edata.cpp.

Here is the call graph for this function:

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

Returns

ExpData::ts

Definition at line 105 of file edata.cpp.

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

Parameters

it	timepoint index

Returns

timepoint timepoint at index

Definition at line 113 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.8 setObservedData() [1/2] `void setObservedData ( const std::vector< realtype > & observedData )`

Parameters

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

Definition at line 117 of file edata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.10.3.9 setObservedData() [2/2]

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

Parameters

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

Definition at line 126 of file edata.cpp.
Here is the call graph for this function:

18.10.3.10 isSetObservedData()

```
bool isSetObservedData (
```

```
int it,
int iy ) const
```

Parameters

it	time index
iy	observable index

Returns

boolean specifying if data was set

Definition at line 134 of file edata.cpp.
Here is the caller graph for this function:

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

Returns

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

Definition at line 138 of file edata.cpp.

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

Parameters

it	timepoint index

Returns

pointer to observed data at index (dimension: nytrue)

Definition at line 142 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.13 setObservedDataStdDev() [1/4]

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

Parameters

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

Definition at line 149 of file edata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.10.3.14 setObservedDataStdDev() [2/4]

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

Parameters

stdDev	standard deviation (dimension: scalar)

Definition at line 159 of file edata.cpp.

18.10.3.15 setObservedDataStdDev() [3/4]

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

Parameters

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

Definition at line 164 of file edata.cpp.
Here is the call graph for this function:

18.10.3.16 setObservedDataStdDev() [4/4]

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

Parameters

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

Definition at line 173 of file edata.cpp.
Here is the call graph for this function:

18.10.3.17 isSetObservedDataStdDev()

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

Parameters

it	time index
iy	observable index

Returns

boolean specifying if standard deviation of data was set

Definition at line 179 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.18 getObservedDataStdDev()

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

Returns

standard deviation of observed data

Definition at line 183 of file edata.cpp.

18.10.3.19 getObservedDataStdDevPtr()

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

Parameters

it	timepoint index

Returns

pointer to standard deviation of observed data at index

Definition at line 187 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.20 setObservedEvents() [1/2] void setObservedEvents (const std::vector< realtype > & *observedEvents*)

Parameters

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

Definition at line 194 of file edata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.10.3.21 setObservedEvents() [2/2] void setObservedEvents (

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

Parameters

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

Definition at line 203 of file edata.cpp.

18.10.3.22 isSetObservedEvents()

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

Parameters

ie	event index
iz	event observable index

Returns

boolean specifying if data was set

Definition at line 212 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.23 getObservedEvents()

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

Returns

observed event data

Definition at line 216 of file edata.cpp.

18.10.3.24 getObservedEventsPtr()

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

Parameters

ie	event occurence

Returns

pointer to observed event data at ieth occurence

Definition at line 220 of file edata.cpp.
Here is the caller graph for this function:

18.10.3.25 setObservedEventsStdDev() [1/4]

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

Parameters

observedEventsStdDev	standard deviation of observed event data

Definition at line 227 of file edata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.10.3.26 setObservedEventsStdDev() [2/4]

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

Parameters

stdDev	standard deviation (dimension: scalar)

Definition at line 237 of file edata.cpp.

18.10.3.27 setObservedEventsStdDev() [3/4]

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

Parameters

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

Definition at line 242 of file edata.cpp.

18.10.3.28 setObservedEventsStdDev() [4/4]

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

Parameters

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

Definition at line 251 of file edata.cpp.

18.10.3.29 isSetObservedEventsStdDev()

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

Parameters

ie	event index
iz	event observable index

Returns

boolean specifying if standard deviation of event data was set

Definition at line 258 of file edata.cpp.
Here is the caller graph for this function:

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

Returns

standard deviation of observed event data

Definition at line 265 of file edata.cpp.

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

Parameters

ie	event occurence

Returns

pointer to standard deviation of observed event data at ieth occurence

Definition at line 269 of file edata.cpp.

Here is the caller graph for this function:

18.10.3.32 checkDataDimension()

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

Parameters

input	vector input to be checked
fieldname	name of the input

Definition at line 291 of file edata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.10.3.33 checkEventsDimension()

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

Parameters

input	vector input to be checkedjupyter_contrib_nbextensions
fieldname	name of the input

Definition at line 296 of file edata.cpp.
Here is the caller graph for this function:

18.10.4 Member Data Documentation

18.10.4.1 reinitializeFixedParameterInitialStates

```
bool reinitializeFixedParameterInitialStates = false
```

flag indicating whether reinitialization of states depending on fixed parameters is activated
Definition at line 434 of file edata.h.

18.11 FinalStateStorer Class Reference

stores the stimulation state when it goes out of scope
`#include <forwardproblem.h>`
Inheritance diagram for FinalStateStorer:

Public Member Functions

- FinalStateStorer (ForwardProblem *fwd)

  constructor, attaches problem pointer
- FinalStateStorer & **operator=** (const FinalStateStorer &other)=delete

- ~FinalStateStorer ()

  destructor, stores simulation state

18.11.1 Detailed Description

Definition at line 459 of file forwardproblem.h.

18.11.2 Constructor & Destructor Documentation

18.11.2.1 FinalStateStorer() `FinalStateStorer ( ForwardProblem * fwd ) [explicit]`

Parameters

fwd	problem from which the simulation state is to be stored

Definition at line 465 of file forwardproblem.h.

18.12 ForwardProblem Class Reference

The ForwardProblem class groups all functions for solving the forward problem.

`#include <forwardproblem.h>`

Public Member Functions

- ForwardProblem (const ExpData *edata, Model *model, Solver *solver, const SteadystateProblem *preeq)

  Constructor.
- void workForwardProblem ()

  Solve the forward problem.
- void getAdjointUpdates (Model &model, const ExpData &edata)

  computes adjoint updates dJydx according to provided model and expdata
- realtype getTime () const

  Accessor for t.
- AmiVector const & getState () const

  Accessor for x.
- AmiVector const & getStateDerivative () const

  Accessor for dx.
- AmiVectorArray const & getStateSensitivity () const

  Accessor for sx.
- std::vector< AmiVector > const & getStatesAtDiscontinuities () const

  Accessor for x_disc.
- std::vector< AmiVector > const & getRHSAtDiscontinuities () const

  Accessor for xdot_disc.
- std::vector< AmiVector > const & getRHSBeforeDiscontinuities () const

  Accessor for xdot_old_disc.
- std::vector< int > const & getNumberOfRoots () const

  Accessor for nroots.
- std::vector< realtype > const & getDiscontinuities () const

  Accessor for discs.
- std::vector< std::vector< int > > const & getRootIndexes () const

  Accessor for rootidx.
- std::vector< realtype > const & getDJydx () const

Accessor for dJydx.

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

  Accessor for dJzdx.
- AmiVector * getStatePointer ()

  Accessor for pointer to x.
- AmiVector * getStateDerivativePointer ()

  Accessor for pointer to dx.
- AmiVectorArray * getStateSensitivityPointer ()

  accessor for pointer to sx
- AmiVectorArray * getStateDerivativeSensitivityPointer ()

  Accessor for pointer to sdx.
- int getCurrentTimeIteration () const

  Accessor for it.
- realtype getFinalTime () const

  Returns final time point for which simulations are available.
- int getEventCounter () const

  Returns maximal event index for which simulations are available.
- int getRootCounter () const

  Returns maximal event index for which the timepoint is available.
- const SimulationState & getSimulationStateTimepoint (int it) const

  Retrieves the carbon copy of the simulation state variables at the specified timepoint index.
- const SimulationState & getSimulationStateEvent (int iroot) const

  Retrieves the carbon copy of the simulation state variables at the specified event index.
- const SimulationState & getInitialSimulationState () const

  Retrieves the carbon copy of the simulation state variables at the initial timepoint.
- const SimulationState & getFinalSimulationState () const

  Retrieves the carbon copy of the simulation state variables at the final timepoint (or when simulation failed)

Public Attributes

- friend FinalStateStorer
- Model * model
- Solver * solver
- const ExpData * edata

18.12.1 Detailed Description

Definition at line 43 of file forwardproblem.h.

18.12.2 Constructor & Destructor Documentation

18.12.2.1 ForwardProblem()

```
ForwardProblem (
    const ExpData * edata,
    Model * model,
    Solver * solver,
    const SteadystateProblem * preeq )
```

Parameters

edata	pointer to ExpData instance
model	pointer to Model instance
solver	pointer to Solver instance
preeq	preequilibration with which to initialize the forward problem, pass nullptr for no initialization

Definition at line 17 of file forwardproblem.cpp.
Here is the call graph for this function:

18.12.3 Member Function Documentation

18.12.3.1 workForwardProblem() `void workForwardProblem ( )`

If forward sensitivities are enabled this will also compute sensitivies.
Definition at line 46 of file forwardproblem.cpp.
Here is the call graph for this function:

18.12.3.2 getAdjointUpdates() `void getAdjointUpdates (`

```
Model & model,
const ExpData & edata )
```

Parameters

model	Model instance
edata	experimental data

Definition at line 313 of file forwardproblem.cpp.

Here is the call graph for this function:

18.12.3.3 getTime() `realtype getTime ( ) const`

Returns

t

Definition at line 79 of file forwardproblem.h.

18.12.3.4 getState() `AmiVector const& getState ( ) const`

Returns

x

Definition at line 87 of file forwardproblem.h.

18.12.3.5 getStateDerivative() `AmiVector const& getStateDerivative ( ) const`

Returns

dx

Definition at line 95 of file forwardproblem.h.

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

Returns

sx

Definition at line 103 of file forwardproblem.h.

18.12.3.7 getStatesAtDiscontinuities() `std::vector<AmiVector> const& getStatesAtDiscontinuities ( ) const`

Returns

x_disc

Definition at line 111 of file forwardproblem.h.

18.12.3.8 getRHSAtDiscontinuities() `std::vector<AmiVector> const& getRHSAtDiscontinuities ( ) const`

Returns

xdot_disc

Definition at line 119 of file forwardproblem.h.

18.12.3.9 getRHSBeforeDiscontinuities() `std::vector<AmiVector> const& getRHSBeforeDiscontinuities ( ) const`

Returns

xdot_old_disc

Definition at line 127 of file forwardproblem.h.

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

Returns

nroots

Definition at line 135 of file forwardproblem.h.

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

Returns

discs

Definition at line 143 of file forwardproblem.h.

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

Returns

rootidx

Definition at line 151 of file forwardproblem.h.
Here is the caller graph for this function:

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

Returns

dJydx

Definition at line 159 of file forwardproblem.h.

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

Returns

dJzdx

Definition at line 167 of file forwardproblem.h.

18.12.3.15 getStatePointer() `AmiVector* getStatePointer ( )`

Returns

&x

Definition at line 175 of file forwardproblem.h.

18.12.3.16 getStateDerivativePointer() `AmiVector* getStateDerivativePointer ( )`

Returns

&dx

Definition at line 183 of file forwardproblem.h.

18.12.3.17 getStateSensitivityPointer() `AmiVectorArray* getStateSensitivityPointer ( )`

Returns

&sx

Definition at line 191 of file forwardproblem.h.

18.12.3.18 getStateDerivativeSensitivityPointer() `AmiVectorArray* getStateDerivativeSensitivity↩ Pointer ( )`

Returns

&sdx

Definition at line 199 of file forwardproblem.h.

18.12.3.19 getCurrentTimeIteration() `int getCurrentTimeIteration ( ) const`

Returns

it

Definition at line 207 of file forwardproblem.h.

18.12.3.20 getFinalTime() `realtype getFinalTime ( ) const`

Returns

time point

Definition at line 215 of file forwardproblem.h.
Here is the caller graph for this function:

18.12.3.21 getEventCounter() `int getEventCounter ( ) const`

Returns

index

Definition at line 223 of file forwardproblem.h.
Here is the caller graph for this function:

18.12.3.22 getRootCounter() `int getRootCounter ( ) const`

Returns

index

Definition at line 231 of file forwardproblem.h.

18.12.3.23 getSimulationStateTimepoint() `const SimulationState& getSimulationStateTimepoint ( int it ) const`

Parameters

it	timpoint index

Returns

state

Definition at line 241 of file forwardproblem.h.
Here is the call graph for this function:

Here is the caller graph for this function:

18.12.3.24 getSimulationStateEvent() `const SimulationState& getSimulationStateEvent ( int iroot ) const`

Parameters

iroot	event index

Returns

SimulationState

Definition at line 253 of file forwardproblem.h.
Here is the caller graph for this function:

18.12.3.25 getInitialSimulationState() `const SimulationState& getInitialSimulationState ( ) const`

Returns

SimulationState

Definition at line 262 of file forwardproblem.h.
Here is the caller graph for this function:

18.12.3.26 getFinalSimulationState() `const SimulationState& getFinalSimulationState ( ) const`

Returns

SimulationState

Definition at line 271 of file forwardproblem.h.

18.12.4 Member Data Documentation

18.12.4.1 FinalStateStorer `friend FinalStateStorer`

allow FinalStateStorer to access private members and functions
Definition at line 59 of file forwardproblem.h.

18.12.4.2 model `Model* model`

pointer to model instance
Definition at line 276 of file forwardproblem.h.

18.12.4.3 solver `Solver* solver`

pointer to solver instance
Definition at line 279 of file forwardproblem.h.

18.12.4.4 edata `const ExpData* edata`

pointer to experimental data instance
Definition at line 282 of file forwardproblem.h.

18.13 IDAException Class Reference

ida exception handler class
`#include <exception.h>`

Inheritance diagram for IDAException:

Public Member Functions

- IDAException (int error_code, const char *function)

  Constructor.

18.13.1 Detailed Description

Definition at line 66 of file exception.h.

18.13.2 Constructor & Destructor Documentation

18.13.2.1 IDAException()

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

Parameters

error_code	error code returned by ida function
function	ida function name

Definition at line 35 of file exception.cpp.

18.14 IntegrationFailure Class Reference

Integration failure exception for the forward problem.

`#include <exception.h>`

Inheritance diagram for IntegrationFailure:

Public Member Functions

- IntegrationFailure (int code, realtype t)

  Constructor.

Public Attributes

- int error_code
- realtype time

18.14.1 Detailed Description

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
Definition at line 84 of file exception.h.

18.14.2 Constructor & Destructor Documentation

18.14.2.1 IntegrationFailure()

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

Parameters

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

Definition at line 38 of file exception.cpp.

18.14.3 Member Data Documentation

18.14.3.1 error_code

```
int error_code
```

error code returned by cvodes/idas
Definition at line 94 of file exception.h.

18.14.3.2 time `realtype time`

time of integration failure
Definition at line 97 of file exception.h.

18.15 IntegrationFailureB Class Reference

Integration failure exception for the backward problem.
`#include <exception.h>`
Inheritance diagram for IntegrationFailureB:

Public Member Functions

- IntegrationFailureB (int code, realtype t)

  Constructor.

Public Attributes

- int error_code
- realtype time

18.15.1 Detailed Description

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
Definition at line 108 of file exception.h.

18.15.2 Constructor & Destructor Documentation

18.15.2.1 IntegrationFailureB()

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

Parameters

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

Definition at line 42 of file exception.cpp.

18.15.3 Member Data Documentation

18.15.3.1 error_code `int error_code`

error code returned by cvode/ida
Definition at line 118 of file exception.h.

18.15.3.2 time `realtype time`

time of integration failure
Definition at line 121 of file exception.h.

18.16 Model Class Reference

The Model class represents an AMICI ODE/DAE 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 ()=default
- Model (int nx_rdata, int nxtrue_rdata, int nx_solver, int nxtrue_solver, int nx_solver_reinit, int ny, int nytrue, int nz, int nztrue, int ne, int nJ, int nw, int ndwdx, int ndwdp, int ndxdotdw, std::vector< int > ndJydy, int nnz, int ubw, int lbw, amici::SecondOrderMode o2mode, const std::vector< amici::realtype > &p, std::vector< amici::realtype > k, const std::vector< int > &plist, std::vector< amici::realtype > idlist, std::vector< int > z2event, bool pythonGenerated=false, int ndxdotdp_explicit=0, int ndxdotdp_implicit=0)

  Constructor with model dimensions.
- ~Model () override=default

- Model & operator= (Model const &other)=delete

  Copy assignment is disabled until const members are removed.
- virtual Model ∗ clone () const =0

  Clone this instance.
- void initialize (AmiVector &x, AmiVector &dx, AmiVectorArray &sx, AmiVectorArray &sdx, bool compute↩ Sensitivities)

  Initialization of model properties.
- void initializeB (AmiVector &xB, AmiVector &dxB, AmiVector &xQB, bool posteq) const

  Initialization of model properties.
- void initializeStates (AmiVector &x)

  Initialization of initial states.
- void initializeStateSensitivities (AmiVectorArray &sx, const AmiVector &x)

  Initialization of initial state sensitivities.
- void initHeaviside (const AmiVector &x, const AmiVector &dx)

  Initialises the heaviside variables h at the intial time t0 heaviside variables activate/deactivate on event occurences.
- int nplist () const

  Number of parameters wrt to which sensitivities are computed.
- int np () const

  Total number of model parameters.
- int nk () const

  Number of constants.
- int ncl () const

  Number of conservation laws.
- int nx_reinit () const

  Number of solver states subject to reinitialization.
- const double ∗ k () const

  Fixed parameters.
- int nMaxEvent () const

  Get nmaxevent.
- void setNMaxEvent (int nmaxevent)

  Set nmaxevent.
- int nt () const

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

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

  Set ParameterScale for each parameter, resets initial state sensitivities.
- void setParameterScale (const std::vector< ParameterScaling > &pscaleVec)

  Set ParameterScale for each parameter, resets initial state sensitivities.
- std::vector< realtype > const & getUnscaledParameters () const

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

  Get the parameter vector.
- realtype getParameterById (std::string const &par_id) const

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

  Get value of first model parameter with the specified name,.
- void setParameters (std::vector< realtype > const &p)

  Sets the parameter vector.
- void setParameterById (std::map< std::string, realtype > const &p, bool ignoreErrors=false)

  Sets model parameters according to the parameter IDs and mapped values.

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

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

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

  Set value of first model parameter with the specified name.
- void setParameterByName (std::map< std::string, realtype > const &p, bool ignoreErrors=false)

  Sets model parameters according to the parameter name and mapped values.
- int setParametersByNameRegex (std::string const &par_name_regex, realtype value)

  Set all values of all model parameters with names matching the specified regex.
- std::vector< realtype > const & getFixedParameters () const

  Gets the fixedParameter member.
- realtype getFixedParameterById (std::string const &par_id) const

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

  Get value of fixed parameter with the specified name, if multiple parameters have the same name, the first parameter with matching name is returned.
- void setFixedParameters (std::vector< realtype > const &k)

  Sets the fixedParameter member.
- void setFixedParameterById (std::string const &par_id, realtype value)

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

  Set values of all fixed parameters with the id matching the specified regex.
- void setFixedParameterByName (std::string const &par_name, realtype value)

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

  Set value of all fixed parameters with name matching the specified regex,.
- virtual std::string getName () const

  Returns the model name.
- virtual bool hasParameterNames () const

  Reports whether the model has parameter names set. Also returns true if the number of corresponding variables is just zero.
- 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. Also returns true if the number of corresponding variables is just zero.
- 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. Also returns true if the number of corresponding variables is just zero.
- 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. Also returns true if the number of corresponding variables is just zero.
- virtual std::vector< std::string > getObservableNames () const

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

  Reports whether the model has parameter ids set. Also returns true if the number of corresponding variables is just zero.

- 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. Also returns true if the number of corresponding variables is just zero.
- 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. Also returns true if the number of corresponding variables is just zero.
- 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. Also returns true if the number of corresponding variables is just zero.
- virtual std::vector< std::string > getObservableIds () const

  Get ids of the observables.
- std::vector< realtype > const & getTimepoints () const

  Get the timepoint vector.
- realtype getTimepoint (int it) const

  get simulation timepoint for time index it
- void setTimepoints (std::vector< realtype > const &ts)

  Set the timepoint vector.
- double t0 () const

  get simulation start time
- void setT0 (double t0)

  set simulation start time
- std::vector< bool > const & getStateIsNonNegative () const

  gets flags indicating whether states should be treated as non-negative
- void setStateIsNonNegative (std::vector< bool > const &stateIsNonNegative)

  sets flags indicating whether states should be treated as non-negative
- void setAllStatesNonNegative ()

  sets flags indicating that all states should be treated as non-negative
- ModelState const & getModelState () const

  retrieves the current model state
- void setModelState (ModelState const &state)

  sets the current model state
- std::vector< int > const & getParameterList () const

  Get the list of parameters for which sensitivities are computed.
- int plist (int pos) const

  entry in parameter list
- void setParameterList (std::vector< int > const &plist)

  Set the list of parameters for which sensitivities are computed, resets initial state sensitivities.
- std::vector< realtype > getInitialStates ()

  Get the initial states.
- void setInitialStates (std::vector< realtype > const &x0)

  Set the initial states.
- bool hasCustomInitialStates () const

  Returns whether custom initial states have been set.
- std::vector< realtype > getInitialStateSensitivities ()

  Get the initial states sensitivities.
- void setInitialStateSensitivities (std::vector< realtype > const &sx0)

Set the initial state sensitivities.

- bool hasCustomInitialStateSensitivities () const

  Returns whether custom initial state sensitivitiess have been set.
- void setUnscaledInitialStateSensitivities (std::vector< realtype > const &sx0)

  Set the initial state sensitivities.
- void setSteadyStateSensitivityMode (SteadyStateSensitivityMode mode)

  Sets the mode how sensitivities are computed in the steadystate simulation.
- SteadyStateSensitivityMode getSteadyStateSensitivityMode () const

  Gets the mode how sensitivities are computed in the steadystate simulation.
- void setReinitializeFixedParameterInitialStates (bool flag)

  Set whether initial states depending on fixedParmeters are to be reinitialized after preequilibration and presimulation.
- bool getReinitializeFixedParameterInitialStates () const

  Get whether initial states depending on fixedParmeters are to be reinitialized after preequilibration and presimulation.
- void requireSensitivitiesForAllParameters ()

  Require computation of sensitivities for all parameters p [0..np[in natural order, resets initial state sensitivities.
- void getExpression (gsl::span< realtype > w, const realtype t, const AmiVector &x)

  Time-resolved w,.
- void getObservable (gsl::span< realtype > y, const realtype t, const AmiVector &x)

  Time-resolved observables,.
- void getObservableSensitivity (gsl::span< realtype > sy, const realtype t, const AmiVector &x, const AmiVectorArray &sx)

  Sensitivity of time-resolved observables, total derivative sy = dydx ∗ sx + dydp (only for forward sensitivities)
- void getObservableSigma (gsl::span< realtype > sigmay, const int it, const ExpData ∗edata)

  Time-resolved observable standard deviations.
- void getObservableSigmaSensitivity (gsl::span< realtype > ssigmay, const int it, const ExpData ∗edata)

  Sensitivity of time-resolved observable standard deviation, total derivative (can be used with both adjoint and forward sensitivity)
- void addObservableObjective (realtype &Jy, const int it, const AmiVector &x, const ExpData &edata)

  Time-resolved measurement negative log-likelihood Jy.
- void addObservableObjectiveSensitivity (std::vector< realtype > &sllh, std::vector< realtype > &s2llh, const int it, const AmiVector &x, const AmiVectorArray &sx, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy, total derivative (to be used with forward senstivities)
- void addPartialObservableObjectiveSensitivity (std::vector< realtype > &sllh, std::vector< realtype > &s2llh, const int it, const AmiVector &x, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy, partial derivative (to be used with adjoint senstivities)
- void getAdjointStateObservableUpdate (gsl::span< realtype > dJydx, const int it, const AmiVector &x, const ExpData &edata)

  State sensitivity of the negative loglikelihood Jy, partial derivative (to be used with adjoint senstivities)
- void getEvent (gsl::span< realtype > z, const int ie, const realtype t, const AmiVector &x)

  Event-resolved observables.
- void getEventSensitivity (gsl::span< realtype > sz, const int ie, const realtype t, const AmiVector &x, const AmiVectorArray &sx)

  Sensitivities of event-resolved observables, total derivative, total derivative (only forward sensitivities)
- void getUnobservedEventSensitivity (gsl::span< realtype > sz, const int ie)

  Sensitivity of z at final timepoint (ignores sensitivity of timepoint), total derivative.
- void getEventRegularization (gsl::span< realtype > rz, const int ie, const realtype t, const AmiVector &x)

  Regularization for event-resolved observables.
- void getEventRegularizationSensitivity (gsl::span< realtype > srz, const int ie, const realtype t, const AmiVector &x, const AmiVectorArray &sx)

  Sensitivities of regularization for event-resolved observables, total derivative (only forward sensitivities)

- void getEventSigma (gsl::span< realtype > sigmaz, const int ie, const int nroots, const realtype t, const ExpData *edata)

  Event-resolved observable standard deviations.
- void getEventSigmaSensitivity (gsl::span< realtype > ssigmaz, const int ie, const int nroots, const realtype t, const ExpData *edata)

  Sensitivities of event-resolved observable standard deviations, total derivative (only forward sensitivities)
- void addEventObjective (realtype &Jz, const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Event-resolved observable negative log-likelihood,.
- void addEventObjectiveRegularization (realtype &Jrz, const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Event-resolved observable negative log-likelihood,.
- void addEventObjectiveSensitivity (std::vector< realtype > &sllh, std::vector< realtype > &s2llh, const int ie, const int nroots, const realtype t, const AmiVector &x, const AmiVectorArray &sx, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy, total derivative (to be used with forward senstivities)
- void addPartialEventObjectiveSensitivity (std::vector< realtype > &sllh, std::vector< realtype > &s2llh, const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy, partial derivative (to be used with adjoint senstivities)
- void getAdjointStateEventUpdate (gsl::span< realtype > dJzdx, const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  State sensitivity of the negative loglikelihood Jz, partial derivative (to be used with adjoint senstivities)
- void getEventTimeSensitivity (std::vector< realtype > &stau, const realtype t, const int ie, const AmiVector &x, const AmiVectorArray &sx)

  Sensitivity of event timepoint, total derivative (only forward sensi)
- void addStateEventUpdate (AmiVector &x, const int ie, const realtype t, const AmiVector &xdot, const AmiVector &xdot_old)

  Update state variables after event.
- void addStateSensitivityEventUpdate (AmiVectorArray &sx, const int ie, const realtype t, const AmiVector &x_old, const AmiVector &xdot, const AmiVector &xdot_old, const std::vector< realtype > &stau)

  Update state sensitivity after event.
- void addAdjointStateEventUpdate (AmiVector &xB, const int ie, const realtype t, const AmiVector &x, const AmiVector &xdot, const AmiVector &xdot_old)

  Update adjoint state after event.
- void addAdjointQuadratureEventUpdate (AmiVector xQB, const int ie, const realtype t, const AmiVector &x, const AmiVector &xB, const AmiVector &xdot, const AmiVector &xdot_old)

  Update adjoint quadratures after event.
- void updateHeaviside (const std::vector< int > &rootsfound)

  Update the heaviside variables h on event occurences.
- void updateHeavisideB (const int *rootsfound)

  Updates the heaviside variables h on event occurences in the backward problem.
- int checkFinite (gsl::span< 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.
- void setAlwaysCheckFinite (bool alwaysCheck)

  Set if the result of every call to Model::f should be checked for finiteness.*
- bool getAlwaysCheckFinite () const

  Get setting of whether the result of every call to Model::f should be checked for finiteness.*
- void fx0 (AmiVector &x)

  Initial states.
- void fx0_fixedParameters (AmiVector &x)

  Sets only those initial states that are specified via fixedParmeters.
- void fsx0 (AmiVectorArray &sx, const AmiVector &x)

Initial value for initial state sensitivities.

- void fsx0_fixedParameters (AmiVectorArray &sx, const AmiVector &x)

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

- virtual void fsdx0 ()

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

- void fx_rdata (AmiVector &x_rdata, const AmiVector &x_solver)

  Expands conservation law for states.

- void fsx_rdata (AmiVectorArray &sx_rdata, const AmiVectorArray &sx_solver)

  Expands conservation law for state sensitivities.

- virtual void fdeltaqB (realtype ∗deltaqB, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗xB)

  Model specific implementation of fdeltaqB.

- virtual void fdeltasx (realtype ∗deltasx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, int ip, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗sx, const realtype ∗stau)

  Model specific implementation of fdeltasx.

- virtual void fdeltax (realtype ∗deltax, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ie, const realtype ∗xdot, const realtype ∗xdot_old)

  Model specific implementation of fdeltax.

- virtual void fdeltaxB (realtype ∗deltaxB, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ie, const realtype ∗xdot, const realtype ∗xdot_old, const realtype ∗xB)

  Model specific implementation of fdeltaxB.

- virtual void fdJrzdsigma (realtype ∗dJrzdsigma, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗rz, const realtype ∗sigmaz)

  Model specific implementation of fdJrzdsigma.

- virtual void fdJrzdz (realtype ∗dJrzdz, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗rz, const realtype ∗sigmaz)

  Model specific implementation of fdJrzdz.

- virtual void fdJydsigma (realtype ∗dJydsigma, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fdJydsigma.

- virtual void fdJydy (realtype ∗dJydy, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fdJydy.

- virtual void fdJzdsigma (realtype ∗dJzdsigma, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fdJzdsigma.

- virtual void fdJzdz (realtype ∗dJzdz, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fdJzdz.

- virtual void fdrzdp (realtype ∗drzdp, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip)

  Model specific implementation of fdrzdp.

- virtual void fdrzdx (realtype ∗drzdx, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of fdrzdx.

- virtual void fdsigmaydp (realtype ∗dsigmaydp, const realtype t, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsigmay.

- virtual void fdsigmazdp (realtype ∗dsigmazdp, const realtype t, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsigmaz.

- virtual void fdwdp (realtype ∗dwdp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl, const realtype ∗stcl)

Model specific sparse implementation of dwdp.

- virtual void fdwdp (realtype ∗dwdp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl, const realtype ∗stcl, int ip)

  Model specific sensitivity implementation of dwdp.
- virtual void fdwdp_colptrs (sunindextype ∗indexptrs)

  Model specific implementation for dwdp, column pointers.
- virtual void fdwdp_rowvals (sunindextype ∗indexvals)

  Model specific implementation for dwdp, row values.
- virtual void fdwdx (realtype ∗dwdx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗tcl)

  Model specific implementation of dwdx, data part.
- virtual void fdwdx_colptrs (sunindextype ∗indexptrs)

  Model specific implementation for dwdx, column pointers.
- virtual void fdwdx_rowvals (sunindextype ∗indexvals)

  Model specific implementation for dwdx, row values.
- virtual void fdydp (realtype ∗dydp, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip, const realtype ∗w, const realtype ∗dwdp)

  Model specific implementation of fdydp.
- virtual void fdydx (realtype ∗dydx, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗dwdx)

  Model specific implementation of fdydx.
- virtual void fdzdp (realtype ∗dzdp, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip)

  Model specific implementation of fdzdp.
- virtual void fdzdx (realtype ∗dzdx, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of fdzdx.
- virtual void fJrz (realtype ∗nllh, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz)

  Model specific implementation of fJrz.
- virtual void fJy (realtype ∗nllh, int iy, const realtype ∗p, const realtype ∗k, const realtype ∗y, const realtype ∗sigmay, const realtype ∗my)

  Model specific implementation of fJy.
- virtual void fJz (realtype ∗nllh, int iz, const realtype ∗p, const realtype ∗k, const realtype ∗z, const realtype ∗sigmaz, const realtype ∗mz)

  Model specific implementation of fJz.
- virtual void frz (realtype ∗rz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of frz.
- virtual void fsigmay (realtype ∗sigmay, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fsigmay.
- virtual void fsigmaz (realtype ∗sigmaz, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fsigmaz.
- virtual void fsrz (realtype ∗srz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip)

  Model specific implementation of fsrz.
- virtual void fstau (realtype ∗stau, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip, int ie)

  Model specific implementation of fstau.
- virtual void fsx0 (realtype ∗sx0, const realtype t, const realtype ∗x0, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsx0.

- virtual void fsx0_fixedParameters (realtype ∗sx0, const realtype t, const realtype ∗x0, const realtype ∗p, const realtype ∗k, int ip)

  Model specific implementation of fsx0_fixedParameters.
- virtual void fsz (realtype ∗sz, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗sx, int ip)

  Model specific implementation of fsz.
- virtual void fw (realtype ∗w, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗tcl)

  Model specific implementation of fw.
- virtual void fx0 (realtype ∗x0, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fx0.
- virtual void fx0_fixedParameters (realtype ∗x0, const realtype t, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fx0_fixedParameters.
- virtual void fy (realtype ∗y, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w)

  Model specific implementation of fy.
- virtual void fz (realtype ∗z, int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h)

  Model specific implementation of fz.

Public Attributes

- int nx_rdata {0}
- int nxtrue_rdata {0}
- int nx_solver {0}
- int nxtrue_solver {0}
- int nx_solver_reinit {0}
- int ny {0}
- int nytrue {0}
- int nz {0}
- int nztrue {0}
- int ne {0}
- int nw {0}
- int ndwdx {0}
- int ndwdp {0}
- int ndxdotdw {0}
- std::vector< int > ndJydy
- int nnz {0}
- int nJ {0}
- int ubw {0}
- int lbw {0}
- bool pythonGenerated
- int ndxdotdp_explicit {0}
- int ndxdotdp_implicit {0}
- SecondOrderMode o2mode {SecondOrderMode::none}
- std::vector< realtype > idlist
- SUNMatrixWrapper dxdotdp_explicit
- SUNMatrixWrapper dxdotdp_implicit
- AmiVectorArray dxdotdp {0, 0}
- AmiciApplication ∗ app = &defaultContext

Protected Member Functions

- void writeSliceEvent (gsl::span< const realtype > slice, gsl::span< realtype > buffer, const int ie)

  Writes part of a slice to a buffer according to indices specified in z2event.
- void writeSensitivitySliceEvent (gsl::span< const realtype > slice, gsl::span< realtype > buffer, const int ie)

  Writes part of a sensitivity slice to a buffer according to indices specified in z2event.
- void writeLLHSensitivitySlice (const std::vector< realtype > &dLLhdp, std::vector< realtype > &sllh, std↩ ::vector< realtype > &s2llh)

  Seperates first and second order objective sensitivity information and writes them into the respective buffers.
- void checkLLHBufferSize (const std::vector< realtype > &sllh, const std::vector< realtype > &s2llh) const

  Verifies that the provided buffers have the expected size.
- void initializeVectors ()

  Set the nplist-dependent vectors to their proper sizes.
- void fy (realtype t, const AmiVector &x)

  Observables / measurements.
- void fdydp (realtype t, const AmiVector &x)

  Partial derivative of observables y w.r.t. model parameters p.
- void fdydx (realtype t, const AmiVector &x)

  Partial derivative of observables y w.r.t. state variables x.
- void fsigmay (int it, const ExpData *edata)

  Standard deviation of measurements.
- void fdsigmaydp (int it, const ExpData *edata)

  Partial derivative of standard deviation of measurements w.r.t. model.
- void fJy (realtype &Jy, int it, const AmiVector &y, const ExpData &edata)

  Negative log-likelihood of measurements y.
- virtual void fdJydy_colptrs (sunindextype *indexptrs, int index)

  Model specific implementation of fdJydy colptrs.
- virtual void fdJydy_rowvals (sunindextype *indexptrs, int index)

  Model specific implementation of fdJydy row vals.
- void fdJydy (int it, const AmiVector &x, const ExpData &edata)

  Partial derivative of time-resolved measurement negative log-likelihood Jy.
- void fdJydsigma (int it, const AmiVector &x, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigma.
- void fdJydp (const int it, const AmiVector &x, const ExpData &edata)

  Compute sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. parameters for the given timepoint. Add result to respective fields in rdata.
- void fdJydx (const int it, const AmiVector &x, const ExpData &edata)

  Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. state variables.
- void fz (int ie, realtype t, const AmiVector &x)

  Event-resolved output.
- void fdzdp (int ie, realtype t, const AmiVector &x)

  Partial derivative of event-resolved output z w.r.t. to model parameters p.
- void fdzdx (int ie, realtype t, const AmiVector &x)

  Partial derivative of event-resolved output z w.r.t. to model states x.
- void frz (int ie, realtype t, const AmiVector &x)

  Event root function of events (equal to froot but does not include non-output events)
- void fdrzdp (int ie, realtype t, const AmiVector &x)

  Sensitivity of event-resolved root output w.r.t. to model parameters p.
- void fdrzdx (int ie, realtype t, const AmiVector &x)

  Sensitivity of event-resolved measurements rz w.r.t. to model states x.
- void fsigmaz (const int ie, const int nroots, const realtype t, const ExpData *edata)

Standard deviation of events.

- void fdsigmazdp (int ie, int nroots, realtype t, const ExpData ∗edata)

  Sensitivity of standard deviation of events measurements w.r.t. model parameters p.
- void fJz (realtype &Jz, int nroots, const AmiVector &z, const ExpData &edata)

  Negative log-likelihood of event-resolved measurements z.
- void fdJzdz (const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Partial derivative of event measurement negative log-likelihood Jz.
- void fdJzdsigma (const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz.
- void fdJzdp (const int ie, const int nroots, realtype t, const AmiVector &x, const ExpData &edata)

  Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. parameters.
- void fdJzdx (const int ie, const int nroots, realtype t, const AmiVector &x, const ExpData &edata)

  Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. state variables.
- void fJrz (realtype &Jrz, int nroots, const AmiVector &rz, const ExpData &edata)

  Regularization of negative log-likelihood with roots of event-resolved measurements rz.
- void fdJrzdz (const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Partial derivative of event measurement negative log-likelihood Jz.
- void fdJrzdsigma (const int ie, const int nroots, const realtype t, const AmiVector &x, const ExpData &edata)

  Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz.
- void fw (realtype t, const realtype ∗x)

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

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

  Recurring terms in xdot, state derivative.
- virtual void fx_rdata (realtype ∗x_rdata, const realtype ∗x_solver, const realtype ∗tcl)

  Model specific implementation of fx_rdata.
- virtual void fsx_rdata (realtype ∗sx_rdata, const realtype ∗sx_solver, const realtype ∗stcl, int ip)

  Model specific implementation of fsx_solver.
- virtual void fx_solver (realtype ∗x_solver, const realtype ∗x_rdata)

  Model specific implementation of fx_solver.
- virtual void fsx_solver (realtype ∗sx_solver, const realtype ∗sx_rdata)

  Model specific implementation of fsx_solver.
- virtual void ftotal_cl (realtype ∗total_cl, const realtype ∗x_rdata)

  Model specific implementation of ftotal_cl.
- virtual void fstotal_cl (realtype ∗stotal_cl, const realtype ∗sx_rdata, int ip)

  Model specific implementation of fstotal_cl.
- N_Vector computeX_pos (const_N_Vector x)

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

Protected Attributes

- ModelState state_
- SUNMatrixWrapper J_
- SUNMatrixWrapper dxdotdw_
- SUNMatrixWrapper dwdp_
- SUNMatrixWrapper dwdx_
- SUNMatrixWrapper M_
- std::vector< realtype > my_
- std::vector< realtype > mz_

- std::vector< SUNMatrixWrapper > dJydy_
- std::vector< realtype > dJydy_matlab_
- std::vector< realtype > dJydsigma_
- std::vector< realtype > dJydx_
- std::vector< realtype > dJydp_
- std::vector< realtype > dJzdz_
- std::vector< realtype > dJzdsigma_
- std::vector< realtype > dJrzdz_
- std::vector< realtype > dJrzdsigma_
- std::vector< realtype > dJzdx_
- std::vector< realtype > dJzdp_
- 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 > sx_
- std::vector< realtype > x_rdata_
- std::vector< realtype > sx_rdata_
- std::vector< realtype > y_
- std::vector< realtype > sigmay_
- std::vector< realtype > dsigmaydp_
- std::vector< realtype > z_
- std::vector< realtype > rz_
- std::vector< realtype > sigmaz_
- std::vector< realtype > dsigmazdp_
- std::vector< realtype > deltax_
- std::vector< realtype > deltasx_
- std::vector< realtype > deltaxB_
- std::vector< realtype > deltaqB_
- AmiVector x_pos_tmp_ {0}
- std::vector< realtype > original_parameters_
- std::vector< int > z2event_
- std::vector< realtype > x0data_
- std::vector< realtype > sx0data_
- std::vector< realtype > ts_
- std::vector< bool > state_is_non_negative_
- bool any_state_non_negative_ {false}
- int nmaxevent_ {10}
- std::vector< ParameterScaling > pscale_
- realtype tstart_ {0.0}
- SteadyStateSensitivityMode steadystate_sensitivity_mode_ {SteadyStateSensitivityMode::newtonOnly}
- bool reinitialize_fixed_parameter_initial_states_ {false}
- bool always_check_finite_ {false}

18.16.1 Detailed Description

Definition at line 70 of file model.h.

18.16.2 Constructor & Destructor Documentation

18.16.2.1 Model() [1/2] Model ( ) [default]

default constructor

18.16.2.2 Model() [2/2] Model (

```
int nx_rdata,
int nxtrue_rdata,
int nx_solver,
int nxtrue_solver,
int nx_solver_reinit,
int ny,
int nytrue,
int nz,
int nztrue,
int ne,
int nJ,
int nw,
int ndwdx,
int ndwdp,
int ndxdotdw,
std::vector< int > ndJydy,
int nnz,
int ubw,
int lbw,
amici::SecondOrderMode o2mode,
const std::vector< amici::realtype > & p,
std::vector< amici::realtype > k,
const std::vector< int > & plist,
std::vector< amici::realtype > idlist,
std::vector< int > z2event,
bool pythonGenerated = false,
int ndxdotdp_explicit = 0,
int ndxdotdp_implicit = 0 )
```

Parameters

nx_rdata	number of state variables
nxtrue_rdata	number of state variables of the non-augmented model
nx_solver	number of state variables with conservation laws applied
nxtrue_solver	number of state variables of the non-augmented model with conservation laws applied
nx_solver_reinit	number of state variables with conservation laws subject to reinitialization
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
ndxdotdw	number of nonzero elements in the w derivative of xdot
ndJydy	number of nonzero elements in the y derivative of dJy (dimension nytrue)
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

Parameters

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
pythonGenerated	flag indicating matlab or python wrapping
ndxdotdp_explicit	number of nonzero elements dxdotdp_explicit
ndxdotdp_implicit	number of nonzero elements dxdotdp_implicit

Definition at line 97 of file model.cpp.
Here is the call graph for this function:

18.16.2.3 ~Model() `~Model ( ) [override], [default]`

destructor

18.16.3 Member Function Documentation

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

Parameters

other	object to copy from

Returns

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

Returns

The clone

Here is the caller graph for this function:

18.16.3.3 initialize()

```
void initialize (
    AmiVector & x,
    AmiVector & dx,
    AmiVectorArray & sx,
    AmiVectorArray & sdx,
    bool computeSensitivities )
```

Parameters

x	pointer to state variables
dx	pointer to time derivative of states (DAE only)
sx	pointer to state variable sensititivies
sdx	pointer to time derivative of state sensitivities (DAE only)
computeSensitivities	flag indicating whether sensitivities are to be computed

Definition at line 190 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.4 initializeB()

```
void initializeB (
    AmiVector & xB,
    AmiVector & dxB,
    AmiVector & xQB,
    bool posteq ) const
```

Parameters

xB	adjoint state variables
dxB	time derivative of adjoint states (DAE only)
xQB	adjoint quadratures
posteq	flag indicating whether postequilibration was performed

Definition at line 204 of file model.cpp.
Here is the call graph for this function:

18.16.3.5 initializeStates()

```
void initializeStates (
    AmiVector & x )
```

Parameters

x	pointer to state variables

Definition at line 212 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.6 initializeStateSensitivities()

```
void initializeStateSensitivities (
    AmiVectorArray & sx,
    const AmiVector & x )
```

Parameters

sx	pointer to state variable sensititivies
x	pointer to state variables

Definition at line 223 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.7 initHeaviside()

```
void initHeaviside (
    const AmiVector & x,
    const AmiVector & dx )
```

Parameters

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

Definition at line 242 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.8 nplist() `int nplist ( ) const`

Returns

length of sensitivity index vector

Definition at line 260 of file model.cpp.

Here is the caller graph for this function:

18.16.3.9 np() `int np ( ) const`

Returns

length of parameter vector

Definition at line 262 of file model.cpp.

Here is the caller graph for this function:

18.16.3.10 nk() `int nk ( ) const`

Returns

length of constant vector

Definition at line 264 of file model.cpp.

Here is the caller graph for this function:

18.16.3.11 ncl() `int ncl ( ) const`

Returns

difference between nx_rdata and nx_solver

Definition at line 266 of file model.cpp.
Here is the caller graph for this function:

18.16.3.12 nx_reinit() `int nx_reinit ( ) const`

Returns

model member nx_solver_reinit

Definition at line 268 of file model.cpp.
Here is the caller graph for this function:

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

Returns

pointer to constants array

Definition at line 270 of file model.cpp.
Here is the caller graph for this function:

18.16.3.14 nMaxEvent() `int nMaxEvent ( ) const`

Returns

maximum number of events that may occur for each type

Definition at line 272 of file model.cpp.

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

Parameters

nmaxevent	maximum number of events that may occur for each type

Definition at line 274 of file model.cpp.

18.16.3.16 nt() `int nt ( ) const`

Returns

number of timepoints

Definition at line 276 of file model.cpp.
Here is the caller graph for this function:

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

Returns

vector of parameter scale

Definition at line 278 of file model.cpp.
Here is the caller graph for this function:

18.16.3.18 setParameterScale() [1/2] `void setParameterScale ( ParameterScaling pscale )`

Parameters

pscale	scalar parameter scale for all parameters

Definition at line 282 of file model.cpp.

Here is the caller graph for this function:

18.16.3.19 setParameterScale() [2/2] `void setParameterScale ( const std::vector< ParameterScaling > & pscaleVec )`

Parameters

pscaleVec	vector of parameter scales

Definition at line 288 of file model.cpp.

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

Returns

unscaled parameters

Definition at line 297 of file model.cpp.

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

Returns

The user-set parameters (see also getUnscaledParameters)

Definition at line 301 of file model.cpp.
Here is the caller graph for this function:

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

Parameters

par↩ _id	parameter id

Returns

parameter value

Definition at line 305 of file model.cpp.
Here is the call graph for this function:

18.16.3.23 getParameterByName()

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

Parameters

par_name	parameter name

Returns

parameter value

Definition at line 313 of file model.cpp.
Here is the call graph for this function:

18.16.3.24 setParameters()

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

Parameters

p	vector of parameters

Definition at line 321 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.25 setParameterById() [1/2]

```
void setParameterById (
    std::map< std::string, realtype > const & p,
    bool ignoreErrors = false )
```

Parameters

p	map of parameters IDs and values
ignoreErrors	Ignore errors such as parameter IDs in p which are not model parameters

Definition at line 330 of file model.cpp.

18.16.3.26 setParameterById() [2/2]

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

Parameters

par↩ _id	parameter id
value	parameter value

Definition at line 343 of file model.cpp.

Here is the call graph for this function:

18.16.3.27 setParametersByIdRegex()

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

Parameters

par_id_regex	parameter id regex
value	parameter value

Returns

number of parameter ids that matched the regex

Definition at line 353 of file model.cpp.
Here is the call graph for this function:

18.16.3.28 setParameterByName() [1/2]

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

Parameters

par_name	parameter name
value	parameter value

Definition at line 364 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.29 setParameterByName() [2/2]

```
void setParameterByName (
    std::map< std::string, realtype > const & p,
    bool ignoreErrors = false )
```

Parameters

p	map of parameters names and values
ignoreErrors	Ignore errors such as parameter names in p which are not model parameters

Definition at line 374 of file model.cpp.
Here is the call graph for this function:

18.16.3.30 setParametersByNameRegex()

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

Parameters

par_name_regex	parameter name regex
value	parameter value

Returns

number of fixed parameter names that matched the regex

Definition at line 387 of file model.cpp.
Here is the call graph for this function:

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

Returns

vector of fixed parameters

Definition at line 401 of file model.cpp.

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

Parameters

par_id	parameter id

Returns

parameter value

Definition at line 405 of file model.cpp.
Here is the call graph for this function:

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

Parameters

par_name	parameter name

Returns

parameter value

Definition at line 414 of file model.cpp.
Here is the call graph for this function:

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

Parameters

k	vector of fixed parameters

Definition at line 423 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.35 setFixedParameterById()

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

Parameters

par_id	fixed parameter id
value	fixed parameter value

Definition at line 430 of file model.cpp.
Here is the call graph for this function:

18.16.3.36 setFixedParametersByIdRegex()

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

Parameters

par_id_regex	fixed parameter name regex
value	fixed parameter value

Returns

number of fixed parameter ids that matched the regex

Definition at line 439 of file model.cpp.
Here is the call graph for this function:

18.16.3.37 setFixedParameterByName()

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

Parameters

par_name	fixed parameter id
value	fixed parameter value

Definition at line 449 of file model.cpp.
Here is the call graph for this function:

18.16.3.38 setFixedParametersByNameRegex()

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

Parameters

par_name_regex	fixed parameter name regex
value	fixed parameter value

Returns

number of fixed parameter names that matched the regex

Definition at line 459 of file model.cpp.
Here is the call graph for this function:

18.16.3.39 getName()

```
std::string getName ( ) const [virtual]
```

Returns

model name

Definition at line 469 of file model.cpp.

18.16.3.40 hasParameterNames()

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

Returns

boolean indicating whether parameter names were set

Definition at line 473 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

the names

Definition at line 477 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether state names were set

Definition at line 481 of file model.cpp.
Here is the call graph for this function:

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

Returns

the names

Definition at line 485 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether fixed parameter names were set

Definition at line 489 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

the names

Definition at line 493 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether observabke names were set

Definition at line 497 of file model.cpp.
Here is the call graph for this function:

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

Returns

the names

Definition at line 501 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether parameter ids were set

Definition at line 505 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

the ids

Definition at line 509 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether state ids were set

Definition at line 513 of file model.cpp.
Here is the call graph for this function:

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

Returns

the ids

Definition at line 517 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether fixed parameter ids were set

Definition at line 521 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

the ids

Definition at line 525 of file model.cpp.
Here is the caller graph for this function:

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

Returns

boolean indicating whether observale ids were set

Definition at line 529 of file model.cpp.
Here is the call graph for this function:

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

Returns

the ids

Definition at line 533 of file model.cpp.
Here is the caller graph for this function:

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

Returns

timepoint vector

Definition at line 537 of file model.cpp.

18.16.3.57 getTimepoint() `double getTimepoint ( int it ) const`

Parameters

it	time index

Returns

t timepoint

Definition at line 539 of file model.cpp.
Here is the caller graph for this function:

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

Parameters

ts	timepoint vector

Definition at line 541 of file model.cpp.
Here is the caller graph for this function:

18.16.3.59 t0() `double t0 ( ) const`

Returns

simulation start time

Definition at line 549 of file model.cpp.
Here is the caller graph for this function:

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

Parameters

t0	simulation start time

Definition at line 551 of file model.cpp.
Here is the call graph for this function:

18.16.3.61 getStateIsNonNegative() `std::vector< bool > const & getStateIsNonNegative ( ) const`

Returns

vector of flags

Definition at line 553 of file model.cpp.

18.16.3.62 setStateIsNonNegative() `void setStateIsNonNegative ( std::vector< bool > const & stateIsNonNegative )`

Parameters

stateIsNonNegative	vector of flags

Definition at line 557 of file model.cpp.
Here is the caller graph for this function:

18.16.3.63 getModelState() `ModelState const& getModelState ( ) const`

Returns

current model state

Definition at line 627 of file model.h.
Here is the caller graph for this function:

18.16.3.64 setModelState() `void setModelState ( ModelState const & state )`

Parameters

state	model state

Definition at line 635 of file model.h.

Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

list of parameter indices

Definition at line 577 of file model.cpp.
Here is the caller graph for this function:

18.16.3.66 plist()

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

Parameters

pos	index

Returns

entry

Definition at line 579 of file model.cpp.
Here is the caller graph for this function:

18.16.3.67 setParameterList()

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

Parameters

plist	list of parameter indices

Definition at line 581 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

initial state vector

Definition at line 592 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Parameters

x0	initial state vector

Definition at line 607 of file model.cpp.
Here is the caller graph for this function:

18.16.3.70 hasCustomInitialStates() `bool hasCustomInitialStates ( ) const`

Returns

True if has custom initial states, otherwise false

Definition at line 620 of file model.cpp.
Here is the caller graph for this function:

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

Returns

vector of initial state sensitivities

Definition at line 625 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Parameters

sx0	vector of initial state sensitivities with chainrule applied. This could be a slice of ReturnData::sx or ReturnData::sx0

Definition at line 644 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.73 hasCustomInitialStateSensitivities() `bool hasCustomInitialStateSensitivities ( ) const`

Returns

True if has custom initial state sensitivitiess, otherwise false

Definition at line 680 of file model.cpp.
Here is the caller graph for this function:

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

Parameters

sx0	vector of initial state sensitivities without chainrule applied. This could be the readin from a model.sx0data saved to hdf5.

Definition at line 685 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.75 setSteadyStateSensitivityMode()

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

Parameters

mode	steadyStateSensitivityMode

Definition at line 700 of file model.cpp.

18.16.3.76 getSteadyStateSensitivityMode()

```
SteadyStateSensitivityMode getSteadyStateSensitivity↩
Mode ( ) const
```

Returns

flag value

Definition at line 705 of file model.cpp.

18.16.3.77 setReinitializeFixedParameterInitialStates()

```
void setReinitializeFixedParameterInitial↩
States (
    bool flag )
```

Parameters

flag	true/false

Definition at line 709 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.78 getReinitializeFixedParameterInitialStates() `bool getReinitializeFixedParameterInitial↩ States ( ) const`

Returns

flag true/false

Definition at line 719 of file model.cpp.
Here is the caller graph for this function:

18.16.3.79 getExpression() `void getExpression ( gsl::span< realtype > w, const realtype t, const AmiVector & x )`

Parameters

w	buffer (dimension: nw)
t	current timepoint
x	current state

Definition at line 729 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.80 getObservable()

```
void getObservable (
    gsl::span< realtype > y,
    const realtype t,
    const AmiVector & x )
```

Parameters

y	buffer (dimension: ny)
t	current timepoint
x	current state

Definition at line 735 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.81 getObservableSensitivity()

```
void getObservableSensitivity (
    gsl::span< realtype > sy,
    const realtype t,
    const AmiVector & x,
    const AmiVectorArray & sx )
```

Parameters

sy	buffer (dimension: ny x nplist, row-major)
t	timpoint
x	state variables
sx	state sensitivities

Definition at line 741 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.82 getObservableSigma()

```
void getObservableSigma (
    gsl::span< realtype > sigmay,
    const int it,
    const ExpData * edata )
```

Parameters

sigmay	buffer (dimension: ny)
it	timepoint index
edata	pointer to experimental data instance (optional, pass nullptr to ignore)

Definition at line 768 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.83 getObservableSigmaSensitivity()

```
void getObservableSigmaSensitivity (
    gsl::span< realtype > ssigmay,
    const int it,
    const ExpData * edata )
```

Parameters

ssigmay	buffer (dimension: ny x nplist, row-major)
it	timepoint index
edata	pointer to experimental data instance (optional, pass nullptr to ignore)

Definition at line 774 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.84 addObservableObjective()

```
void addObservableObjective (
    realtype & Jy,
    const int it,
    const AmiVector & x,
    const ExpData & edata )
```

Parameters

Jy	buffer (dimension: 1)
it	timepoint index
x	state variables
edata	experimental data instance

Definition at line 780 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.85 addObservableObjectiveSensitivity()

```
void addObservableObjectiveSensitivity (
    std::vector< realtype > & sllh,
    std::vector< realtype > & s2llh,
    const int it,
    const AmiVector & x,
    const AmiVectorArray & sx,
    const ExpData & edata )
```

Parameters

sllh	first order buffer (dimension: nplist)
s2llh	second order buffer (dimension: nJ-1 x nplist, row-major)
it	timepoint index
x	state variables

Parameters

sx	state sensitivities
edata	experimental data instance

Definition at line 797 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.86 addPartialObservableObjectiveSensitivity()

```
void addPartialObservableObjectiveSensitivity
(
        std::vector< realtype > & sllh,
        std::vector< realtype > & s2llh,
        const int it,
```

```
const AmiVector & x,
const ExpData & edata )
```

Parameters

sllh	first order buffer (dimension: nplist)
s2llh	second order buffer (dimension: nJ-1 x nplist, row-major)
it	timepoint index
x	state variables
edata	experimental data instance

Definition at line 823 of file model.cpp.
Here is the call graph for this function:

18.16.3.87 getAdjointStateObservableUpdate()

```
void getAdjointStateObservableUpdate (
gsl::span< realtype > dJydx,
const int it,
const AmiVector & x,
const ExpData & edata )
```

Parameters

dJydx	buffer (dimension: nJ x nx_solver, row-major)
it	timepoint index
x	state variables
edata	experimental data instance

Definition at line 834 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.88 getEvent()

```
void getEvent (
    gsl::span< realtype > z,
    const int ie,
    const realtype t,
    const AmiVector & x )
```

Parameters

z	buffer (dimension: nz)
ie	event index
t	timepoint
x	state variables

Definition at line 841 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.89 getEventSensitivity()

```
void getEventSensitivity (
    gsl::span< realtype > sz,
    const int ie,
    const realtype t,
    const AmiVector & x,
    const AmiVectorArray & sx )
```

Parameters

sz	buffer (dimension: nz x nplist, row-major)
ie	event index
t	timepoint
x	state variables
sx	state sensitivities

Definition at line 847 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.90 getUnobservedEventSensitivity()

```
void getUnobservedEventSensitivity (
    gsl::span< realtype > sz,
    const int ie )
```

Parameters

sz	output buffer (dimension: nz x nplist, row-major)
ie	event index

Definition at line 857 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.91 getEventRegularization()

```
void getEventRegularization (
    gsl::span< realtype > rz,
    const int ie,
    const realtype t,
    const AmiVector & x )
```

Parameters

rz	buffer (dimension: nz)
ie	event index
t	timepoint
x	state variables

Definition at line 867 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.92 getEventRegularizationSensitivity()

```
void getEventRegularizationSensitivity (
    gsl::span< realtype > srz,
    const int ie,
    const realtype t,
    const AmiVector & x,
    const AmiVectorArray & sx )
```

Parameters

srz	buffer (dimension: nz x nplist, row-major)
ie	event index
t	timepoint
x	state variables
sx	state sensitivities

Definition at line 873 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.93 getEventSigma()

```
void getEventSigma (
    gsl::span< realtype > sigmaz,
    const int ie,
    const int nroots,
    const realtype t,
    const ExpData * edata )
```

Parameters

sigmaz	buffer (dimension: nz)
ie	event index
nroots	event occurence
t	timepoint
edata	pointer to experimental data instance (optional, pass nullptr to ignore)

Definition at line 884 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.94 getEventSigmaSensitivity()

```
void getEventSigmaSensitivity (
    gsl::span< realtype > ssigmaz,
    const int ie,
    const int nroots,
    const realtype t,
    const ExpData * edata )
```

Parameters

ssigmaz	buffer (dimension: nz x nplist, row-major)
ie	event index
nroots	event occurence
t	timepoint
edata	pointer to experimental data instance (optional, pass nullptr to ignore)

Definition at line 891 of file model.cpp.
Here is the call graph for this function:

18.16.3.95 addEventObjective()

```
void addEventObjective (
    realtype & Jz,
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )
```

Parameters

Jz	buffer (dimension: 1)
ie	event index
nroots	event occurence
t	timepoint
x	state variables
edata	experimental data instance

Definition at line 898 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.96 addEventObjectiveRegularization()

```
void addEventObjectiveRegularization (
    realtype & Jrz,
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )
```

Parameters

Jrz	buffer (dimension: 1)
ie	event index
nroots	event occurence
t	timepoint
x	state variables
edata	experimental data instance

Definition at line 916 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.97 addEventObjectiveSensitivity()

```
void addEventObjectiveSensitivity (
    std::vector< realtype > & sllh,
    std::vector< realtype > & s2llh,
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const AmiVectorArray & sx,
    const ExpData & edata )
```

Parameters

sllh	first order buffer (dimension: nplist)
s2llh	second order buffer (dimension: nJ-1 x nplist, row-major)
ie	event index
nroots	event occurence
t	timepoint
x	state variables
sx	state sensitivities
edata	experimental data instance

Definition at line 934 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.98 addPartialEventObjectiveSensitivity()

```
void addPartialEventObjectiveSensitivity (
    std::vector< realtype > & sllh,
    std::vector< realtype > & s2llh,
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )
```

Parameters

sllh	first order buffer (dimension: nplist)
s2llh	second order buffer (dimension: nJ-1 x nplist, row-major)
ie	event index
nroots	event occurence
t	timepoint
x	state variables
edata	experimental data instance

Definition at line 975 of file model.cpp.
Here is the call graph for this function:

18.16.3.99 getAdjointStateEventUpdate()

```
void getAdjointStateEventUpdate (
    gsl::span< realtype > dJzdx,
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )
```

Parameters

dJzdx	buffer (dimension: nJ x nx_solver, row-major)
ie	event index
nroots	event occurence
t	timepoint
x	state variables
edata	experimental data instance

Definition at line 967 of file model.cpp.
Here is the call graph for this function:

18.16.3.100 getEventTimeSensitivity()

```
void getEventTimeSensitivity (
    std::vector< realtype > & stau,
    const realtype t,
    const int ie,
    const AmiVector & x,
    const AmiVectorArray & sx )
```

Parameters

stau	current timepoint (dimension: nplist)
t	timepoint
ie	event index
x	state variables
sx	state sensitivities

Definition at line 989 of file model.cpp.
Here is the call graph for this function:

18.16.3.101 addStateEventUpdate()

```
void addStateEventUpdate (
    AmiVector & x,
    const int ie,
    const realtype t,
    const AmiVector & xdot,
    const AmiVector & xdot_old )
```

Parameters

x	current state (will be overwritten)
ie	event index
t	current timepoint
xdot	current residual function values

Parameters

xdot_old	value of residual function before event

Definition at line 1003 of file model.cpp.
Here is the call graph for this function:

18.16.3.102 addStateSensitivityEventUpdate()

```
void addStateSensitivityEventUpdate (
    AmiVectorArray & sx,
    const int ie,
    const realtype t,
    const AmiVector & x_old,
    const AmiVector & xdot,
    const AmiVector & xdot_old,
    const std::vector< realtype > & stau )
```

Parameters

sx	current state sensitivity (will be overwritten)
ie	event index
t	current timepoint
x_old	current state
xdot	current residual function values
xdot_old	value of residual function before event
stau	timepoint sensitivity, to be computed with Model::getEventTimeSensitivity

Definition at line 1022 of file model.cpp.

Here is the call graph for this function:

18.16.3.103 addAdjointStateEventUpdate()

```
void addAdjointStateEventUpdate (
    AmiVector & xB,
    const int ie,
    const realtype t,
    const AmiVector & x,
    const AmiVector & xdot,
    const AmiVector & xdot_old )
```

Parameters

xB	current adjoint state (will be overwritten)
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 1048 of file model.cpp.
Here is the call graph for this function:

18.16.3.104 addAdjointQuadratureEventUpdate()

```
void addAdjointQuadratureEventUpdate (
    AmiVector xQB,
    const int ie,
    const realtype t,
    const AmiVector & x,
    const AmiVector & xB,
    const AmiVector & xdot,
    const AmiVector & xdot_old )
```

Parameters

xQB	current quadrature state (will be overwritten)
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 1071 of file model.cpp.
Here is the call graph for this function:

18.16.3.105 updateHeaviside()

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

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 1090 of file model.cpp.

18.16.3.106 updateHeavisideB()

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

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 1096 of file model.cpp.

18.16.3.107 checkFinite()

```
int checkFinite (
    gsl::span< const realtype > array,
    const char * fun ) const
```

Parameters

array	arrays of values
fun	name of the fucntion that generated the values

Returns

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

Definition at line 1103 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.108 setAlwaysCheckFinite()

```
void setAlwaysCheckFinite (
    bool alwaysCheck )
```

Parameters

alwaysCheck	

Definition at line 1115 of file model.cpp.

18.16.3.109 getAlwaysCheckFinite() `bool getAlwaysCheckFinite ( ) const`

Returns

that

Definition at line 1119 of file model.cpp.

18.16.3.110 fx0() [1/2] `void fx0 ( AmiVector & x )`

Parameters

x	pointer to state variables

Definition at line 1121 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.111 fx0_fixedParameters() [1/2]

```
void fx0_fixedParameters (
    AmiVector & x )
```

Parameters

x	pointer to state variables

Definition at line 1135 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.112 fsx0() [1/2]

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

Parameters

sx	pointer to state sensitivity variables
x	pointer to state variables

Definition at line 1150 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.113 fsx0_fixedParameters() [1/2]

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

Parameters

sx	pointer to state sensitivity variables
x	pointer to state variables

Definition at line 1164 of file model.cpp.

Here is the call graph for this function:

18.16.3.114 fx_rdata() [1/2]

```
void fx_rdata (
    AmiVector & x_rdata,
    const AmiVector & x_solver )
```

Parameters

x_rdata	pointer to state variables with conservation laws expanded (stored in rdata)
x_solver	pointer to state variables with conservation laws applied (solver returns this)

Definition at line 1183 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.115 fsx_rdata() [1/2]

```
void fsx_rdata (
    AmiVectorArray & sx_rdata,
    const AmiVectorArray & sx_solver )
```

Parameters

sx_rdata	pointer to state variable sensitivities with conservation laws expanded (stored in rdata)
sx_solver	pointer to state variable sensitivities with conservation laws applied (solver returns this)

Definition at line 1189 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.116 writeSliceEvent()

```
void writeSliceEvent (
    gsl::span< const realtype > slice,
    gsl::span< realtype > buffer,
    const int ie ) [protected]
```

Parameters

slice	source data slice
buffer	output data slice
ie	event index

Definition at line 1198 of file model.cpp.
Here is the caller graph for this function:

18.16.3.117 writeSensitivitySliceEvent()

```
void writeSensitivitySliceEvent (
    gsl::span< const realtype > slice,
    gsl::span< realtype > buffer,
    const int ie ) [protected]
```

Parameters

slice	source data slice
buffer	output data slice
ie	event index

Definition at line 1207 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.118 writeLLHSensitivitySlice()

```
void writeLLHSensitivitySlice (
    const std::vector< realtype > & dLLhdp,
    std::vector< realtype > & sllh,
    std::vector< realtype > & s2llh ) [protected]
```

Parameters

dLLhdp	data with mangled first and second order information
sllh	first order buffer
s2llh	second order buffer

Definition at line 1218 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.119 checkLLHBufferSize()

```
void checkLLHBufferSize (
    const std::vector< realtype > & sllh,
    const std::vector< realtype > & s2llh ) const [protected]
```

Parameters

sllh	first order buffer
s2llh	second order buffer

Definition at line 1229 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.120 fy() [1/2]

```
void fy (
    realtype t,
    const AmiVector & x ) [protected]
```

Parameters

t	current timepoint
x	current state

Definition at line 1246 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.121 fdydp() [1/2]

```
void fdydp (
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

t	current timepoint
x	current state

Definition at line 1262 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.122 fdydx() [1/2]

```
void fdydx (
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

t	current timepoint
x	current state

Definition at line 1281 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.123 fsigmay() [1/2]

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

Parameters

it	timepoint index
edata	pointer to experimental data instance

Definition at line 1297 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.124 fdsigmaydp() [1/2]

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

Parameters

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

Definition at line 1326 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.125 fJy() [1/2]

```
void fJy (
    realtype & Jy,
    int it,
    const AmiVector & y,
    const ExpData & edata )  [protected]
```

Parameters

Jy	variable to which llh will be added
it	timepoint index
y	simulated observable
edata	pointer to experimental data instance

Here is the caller graph for this function:

18.16.3.126 fdJydy_colptrs()

```
void fdJydy_colptrs (
    sunindextype * indexptrs,
    int index ) [protected], [virtual]
```

Parameters

indexptrs	column pointers
index	ytrue index

Definition at line 1356 of file model.cpp.
Here is the caller graph for this function:

18.16.3.127 fdJydy_rowvals()

```
void fdJydy_rowvals (
    sunindextype * indexptrs,
    int index ) [protected], [virtual]
```

Parameters

indexptrs	row val pointers
index	ytrue index

Definition at line 1362 of file model.cpp.

Here is the caller graph for this function:

18.16.3.128 fdJydy() [1/2]

```
void fdJydy (
    int it,
    const AmiVector & x,
    const ExpData & edata ) [protected]
```

Parameters

it	timepoint index
x	state variables
edata	pointer to experimental data instance

Definition at line 1368 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.129 fdJydsigma() [1/2]

```
void fdJydsigma (
    int it,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

it	timepoint index
x	state variables
edata	pointer to experimental data instance

Definition at line 1408 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.130 fdJydp()

```
void fdJydp (
    const int it,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

it	timepoint index
x	state variables
edata	pointer to experimental data instance

Definition at line 1429 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.131 fdJydx()

```
void fdJydx (
    const int it,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

it	timepoint index
x	state variables
edata	pointer to experimental data instance

Definition at line 1468 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.132 fz() [1/2]

```
void fz (
    int ie,
```

```
realtype t,
const AmiVector & x )  [protected]
```

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 1505 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.133 fdzdp() [1/2]

```
void fdzdp (
int ie,
realtype t,
const AmiVector & x )  [protected]
```

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 1513 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.134 fdzdx() [1/2]

```
void fdzdx (
    int ie,
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 1527 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.135 frz() [1/2]

```
void frz (
    int ie,
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 1539 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.136 fdrzdp() [1/2]

```
void fdrzdp (
    int ie,
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 1547 of file model.cpp.
Here is the call graph for this function:

18.16.3.137 fdrzdx() [1/2]

```
void fdrzdx (
    int ie,
    realtype t,
    const AmiVector & x )  [protected]
```

Parameters

ie	event index

Parameters

t	current timepoint
x	current state

Definition at line 1561 of file model.cpp.
Here is the call graph for this function:

18.16.3.138 fsigmaz() [1/2]

```
void fsigmaz (
    const int ie,
    const int nroots,
    const realtype t,
    const ExpData * edata )  [protected]
```

Parameters

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

Definition at line 1573 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.139 fdsigmazdp() [1/2]

```
void fdsigmazdp (
    int ie,
    int nroots,
    realtype t,
    const ExpData * edata )  [protected]
```

Parameters

ie	event index
nroots	event occurence
t	current timepoint
edata	pointer to experimental data instance

Definition at line 1604 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.140 fJz() [1/2]

```
void fJz (
    realtype & Jz,
    int nroots,
    const AmiVector & z,
    const ExpData & edata )  [protected]
```

Parameters

Jz	variable to which llh will be added
nroots	event index
z	simulated event
edata	pointer to experimental data instance

Here is the caller graph for this function:

18.16.3.141 fdJzdz() [1/2]

```
void fdJzdz (
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1632 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.142 fdJzdsigma() [1/2]

```
void fdJzdsigma (
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1654 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.143 fdJzdp()

```
void fdJzdp (
    const int ie,
    const int nroots,
    realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1676 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.144 fdJzdx()

```
void fdJzdx (
    const int ie,
    const int nroots,
    realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1725 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.145 fJrz() [1/2]

```
void fJrz (
    realtype & Jrz,
    int nroots,
    const AmiVector & rz,
    const ExpData & edata )  [protected]
```

Parameters

Jrz	variable to which regularization will be added
nroots	event index
rz	regularization variable
edata	pointer to experimental data instance

Here is the caller graph for this function:

18.16.3.146 fdJrzdz() [1/2]

```
void fdJrzdz (
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1758 of file model.cpp.
Here is the call graph for this function:

18.16.3.147 fdJrzdsigma() [1/2]

```
void fdJrzdsigma (
    const int ie,
    const int nroots,
    const realtype t,
    const AmiVector & x,
    const ExpData & edata )  [protected]
```

Parameters

ie	event index
nroots	event index
t	current timepoint
x	state variables
edata	pointer to experimental data instance

Definition at line 1779 of file model.cpp.
Here is the call graph for this function:

18.16.3.148 fw() [1/2]

```
void fw (
    realtype t,
    const realtype * x ) [protected]
```

Parameters

t	timepoint
x	array with the states

Definition at line 1800 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.149 fdwdp() [1/3]

```
void fdwdp (
    realtype t,
    const realtype * x )  [protected]
```

Parameters

t	timepoint
x	array with the states

Definition at line 1810 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.150 fdwdx() [1/2]

```
void fdwdx (
    realtype t,
    const realtype * x ) [protected]
```

Parameters

t	timepoint
x	array with the states

Definition at line 1837 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.151 fx_rdata() [2/2]

```
void fx_rdata (
    realtype * x_rdata,
    const realtype * x_solver,
    const realtype * tcl ) [protected], [virtual]
```

Parameters

x_rdata	state variables with conservation laws expanded
x_solver	state variables with conservation laws applied
tcl	total abundances for conservation laws

Definition at line 1852 of file model.cpp.

18.16.3.152 fsx_rdata() [2/2]

```
void fsx_rdata (
    realtype * sx_rdata,
    const realtype * sx_solver,
    const realtype * stcl,
    int ip ) [protected], [virtual]
```

Parameters

sx_rdata	state sensitivity variables with conservation laws expanded
sx_solver	state sensitivity variables with conservation laws applied
stcl	sensitivities of total abundances for conservation laws
ip	sensitivity index

Definition at line 1861 of file model.cpp.
Here is the call graph for this function:

18.16.3.153 fx_solver()

```
void fx_solver (
    realtype * x_solver,
    const realtype * x_rdata ) [protected], [virtual]
```

Parameters

x_solver	state variables with conservation laws applied
x_rdata	state variables with conservation laws expanded

Definition at line 1866 of file model.cpp.

Here is the caller graph for this function:

18.16.3.154 fsx_solver()

```
void fsx_solver (
    realtype * sx_solver,
    const realtype * sx_rdata ) [protected], [virtual]
```

Parameters

sx_rdata	state sensitivity variables with conservation laws expanded
sx_solver	state sensitivity variables with conservation laws applied

Definition at line 1874 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.155 ftotal_cl()

```
void ftotal_cl (
    realtype * total_cl,
    const realtype * x_rdata ) [protected], [virtual]
```

Parameters

total↩ _cl	total abundances of conservation laws
x_rdata	state variables with conservation laws expanded

Definition at line 1881 of file model.cpp.
Here is the caller graph for this function:

18.16.3.156 fstotal_cl()

```
void fstotal_cl (
    realtype * stotal_cl,
    const realtype * sx_rdata,
    int ip ) [protected], [virtual]
```

Parameters

stotal↩ _cl	sensitivites for the total abundances of conservation laws
sx_rdata	state sensitivity variables with conservation laws expanded
ip	sensitivity index

Definition at line 1888 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.157 computeX_pos() `N_Vector computeX_pos ( const_N_Vector x ) [protected]`

Parameters

x	state vector possibly containing negative values

Returns

state vector with negative values replaced by 0 according to stateIsNonNegative

Definition at line 1896 of file model.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.16.3.158 fdeltaqB() `void 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 525 of file abstract_model.cpp.

18.16.3.159 fdeltasx() `void 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 485 of file abstract_model.cpp.

18.16.3.160 fdeltax() `void 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 464 of file abstract_model.cpp.

18.16.3.161 fdeltaxB() `void 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 505 of file abstract_model.cpp.

18.16.3.162 fdJrzdsigma() [2/2] void 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 698 of file abstract_model.cpp.

18.16.3.163 fdJrzdz() [2/2] void 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 684 of file abstract_model.cpp.

18.16.3.164 fdJydsigma() [2/2] void 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 641 of file abstract_model.cpp.

18.16.3.165 fdJydy() [2/2] void fdJydy

Parameters

dJydy	partial derivative of time-resolved measurement negative log-likelihood Jy
iy	output index
p	parameter vector

Parameters

k	constant vector
y	model output at timepoint
sigmay	measurement standard deviation at timepoint
my	measurement at timepoint

Definition at line 626 of file abstract_model.cpp.

18.16.3.166 fdJzdsigma() [2/2] `void 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 671 of file abstract_model.cpp.

18.16.3.167 fdJzdz() [2/2] `void 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 656 of file abstract_model.cpp.

18.16.3.168 fdrzdp() [2/2] `void 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 434 of file abstract_model.cpp.

18.16.3.169 fdrzdx() [2/2] void 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 448 of file abstract_model.cpp.

18.16.3.170 fdsigmaydp() [2/2] void fdsigmaydp

Parameters

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

Definition at line 549 of file abstract_model.cpp.

18.16.3.171 fdsigmazdp() [2/2] void fdsigmazdp

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 571 of file abstract_model.cpp.

18.16.3.172 fdwdp() [2/3] void fdwdp

Parameters

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

Parameters

w	vector with helper variables
tcl	total abundances for conservations laws
stcl	sensitivities of total abundances for conservations laws

Definition at line 728 of file abstract_model.cpp.

18.16.3.173 fdwdp() [3/3] void fdwdp

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
tcl	total abundances for conservations laws
stcl	sensitivities of total abundances for conservations laws
ip	sensitivity parameter index

Definition at line 758 of file abstract_model.cpp.

18.16.3.174 fdwdp_colptrs() void fdwdp_colptrs

Parameters

indexptrs	column pointers

Definition at line 737 of file abstract_model.cpp.

18.16.3.175 fdwdp_rowvals() void fdwdp_rowvals

Parameters

indexvals	row values

Definition at line 743 of file abstract_model.cpp.

18.16.3.176 fdwdx() [2/2] void fdwdx

Parameters

dwdx	Recurring terms in xdot, state derivative
t	timepoint
x	vector with the states
p	parameter vector
k	constants vector

Parameters

h	heavyside vector
w	vector with helper variables
tcl	total abundances for conservations laws

Definition at line 774 of file abstract_model.cpp.

18.16.3.177 fdwdx_colptrs() `void fdwdx_colptrs`

Parameters

indexptrs	column pointers

Definition at line 782 of file abstract_model.cpp.

18.16.3.178 fdwdx_rowvals() `void fdwdx_rowvals`

Parameters

indexvals	row values

Definition at line 788 of file abstract_model.cpp.

18.16.3.179 fdydp() [2/2] `void 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
w	repeating elements vector
dwdp	Recurring terms in xdot, parameter derivative

Definition at line 313 of file abstract_model.cpp.

18.16.3.180 fdydx() [2/2] `void 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

Parameters

w	repeating elements vector
dwdx	Recurring terms in xdot, state derivative

Definition at line 328 of file abstract_model.cpp.

18.16.3.181 fdzdp() [2/2] void 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 403 of file abstract_model.cpp.

18.16.3.182 fdzdx() [2/2] void 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 418 of file abstract_model.cpp.

18.16.3.183 fJrz() [2/2] void 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 611 of file abstract_model.cpp.

18.16.3.184 fJy() [2/2] void 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 584 of file abstract_model.cpp.

18.16.3.185 fJz() [2/2] void 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 598 of file abstract_model.cpp.

18.16.3.186 frz() [2/2] void 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 372 of file abstract_model.cpp.

18.16.3.187 fsigmay() [2/2] void fsigmay

Parameters

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

Definition at line 538 of file abstract_model.cpp.

18.16.3.188 fsigmaz() [2/2] void fsigmaz

Parameters

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

Definition at line 559 of file abstract_model.cpp.

18.16.3.189 fsrz() void 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 387 of file abstract_model.cpp.

18.16.3.190 fstau() void 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 283 of file abstract_model.cpp.

18.16.3.191 fsx0() [2/2] void fsx0

Parameters

sx0	initial state sensitivities

Parameters

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

Definition at line 259 of file abstract_model.cpp.

18.16.3.192 fsx0_fixedParameters() [2/2] `void fsx0_fixedParameters`

Parameters

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

Definition at line 246 of file abstract_model.cpp.

18.16.3.193 fsz() `void 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 357 of file abstract_model.cpp.

18.16.3.194 fw() [2/2] `void fw`

Parameters

w	Recurring terms in xdot
t	timepoint
x	vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
tcl	total abundances for conservations laws

Definition at line 712 of file abstract_model.cpp.

18.16.3.195 fx0() [2/2] void fx0

Parameters

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

Definition at line 216 of file abstract_model.cpp.

18.16.3.196 fx0_fixedParameters() [2/2] void fx0_fixedParameters

Parameters

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

Definition at line 234 of file abstract_model.cpp.

18.16.3.197 fy() [2/2] void fy

Parameters

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

Definition at line 297 of file abstract_model.cpp.

18.16.3.198 fz() [2/2] void 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 342 of file abstract_model.cpp.

18.16.4 Member Data Documentation

18.16.4.1 nx_rdata `int nx_rdata {0}`
number of states
Definition at line 1175 of file model.h.

18.16.4.2 nxtrue_rdata `int nxtrue_rdata {0}`
number of states in the unaugmented system
Definition at line 1178 of file model.h.

18.16.4.3 nx_solver `int nx_solver {0}`
number of states with conservation laws applied
Definition at line 1181 of file model.h.

18.16.4.4 nxtrue_solver `int nxtrue_solver {0}`
number of states in the unaugmented system with conservation laws applied
Definition at line 1187 of file model.h.

18.16.4.5 nx_solver_reinit `int nx_solver_reinit {0}`
number of solver states subject to reinitialization
Definition at line 1190 of file model.h.

18.16.4.6 ny `int ny {0}`
number of observables
Definition at line 1193 of file model.h.

18.16.4.7 nytrue `int nytrue {0}`
number of observables in the unaugmented system
Definition at line 1196 of file model.h.

18.16.4.8 nz `int nz {0}`
number of event outputs
Definition at line 1199 of file model.h.

18.16.4.9 nztrue `int nztrue {0}`
number of event outputs in the unaugmented system
Definition at line 1202 of file model.h.

18.16.4.10 ne `int ne {0}`
number of events
Definition at line 1205 of file model.h.

18.16.4.11 nw `int nw {0}`
number of common expressions
Definition at line 1208 of file model.h.

18.16.4.12 ndwdx `int ndwdx {0}`
number of derivatives of common expressions wrt x
Definition at line 1211 of file model.h.

18.16.4.13 ndwdp `int ndwdp {0}`
number of derivatives of common expressions wrt p
Definition at line 1214 of file model.h.

18.16.4.14 ndxdotdw `int ndxdotdw {0}`
number of nonzero entries in dxdotdw
Definition at line 1217 of file model.h.

18.16.4.15 ndJydy `std::vector<int> ndJydy`
number of nonzero entries in dJydy
Definition at line 1219 of file model.h.

18.16.4.16 nnz `int nnz {0}`
number of nonzero entries in jacobian
Definition at line 1222 of file model.h.

18.16.4.17 nJ `int nJ {0}`
dimension of the augmented objective function for 2nd order ASA
Definition at line 1225 of file model.h.

18.16.4.18 ubw `int ubw {0}`
upper bandwith of the jacobian
Definition at line 1228 of file model.h.

18.16.4.19 lbw `int lbw {0}`
lower bandwith of the jacobian
Definition at line 1231 of file model.h.

18.16.4.20 pythonGenerated `bool pythonGenerated`
flag indicating Matlab or python based model generation
Definition at line 1234 of file model.h.

18.16.4.21 ndxdotdp_explicit `int ndxdotdp_explicit {0}`
number of nonzero entries in ndxdotdp_explicit
Definition at line 1237 of file model.h.

18.16.4.22 ndxdotdp_implicit `int ndxdotdp_implicit {0}`

number of nonzero entries in ndxdotdp_implicit
Definition at line 1240 of file model.h.

18.16.4.23 o2mode `SecondOrderMode o2mode {SecondOrderMode::none}`

flag indicating whether for sensi == AMICI_SENSI_ORDER_SECOND directional or full second order derivative will be computed
Definition at line 1246 of file model.h.

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

flag array for DAE equations
Definition at line 1249 of file model.h.

18.16.4.25 dxdotdp_explicit `SUNMatrixWrapper dxdotdp_explicit [mutable]`

temporary storage of dxdotdp data across functions, Python only (dimension: nplist x nx_solver, nnz: ndxdotdp_↩explicit, type CSC_MAT)
Definition at line 1255 of file model.h.

18.16.4.26 dxdotdp_implicit `SUNMatrixWrapper dxdotdp_implicit [mutable]`

temporary storage of dxdotdp_implicit data across functions, Python only (dimension: nplist x ∗ nx_solver, nnz: ndxdotdp_implicit, type CSC_MAT)
Definition at line 1261 of file model.h.

18.16.4.27 dxdotdp `AmiVectorArray dxdotdp {0, 0}`

temporary storage of dxdotdp data across functions, Matlab only (dimension: nplist x nx_solver, row-major)
Definition at line 1267 of file model.h.

18.16.4.28 app `AmiciApplication* app = &defaultContext`

AMICI context
Definition at line 1269 of file model.h.

18.16.4.29 state_ `ModelState state_ [protected]`

all variables necessary for function evaluation
Definition at line 1666 of file model.h.

18.16.4.30 J_ `SUNMatrixWrapper J_ [mutable], [protected]`

Sparse Jacobian (dimension: nnz)
Definition at line 1669 of file model.h.

18.16.4.31 dxdotdw_ `SUNMatrixWrapper dxdotdw_ [mutable], [protected]`

Sparse dxdotdw temporary storage (dimension: ndxdotdw)
Definition at line 1672 of file model.h.

18.16.4.32 dwdp_ `SUNMatrixWrapper dwdp_ [mutable], [protected]`
Sparse dwdp temporary storage (dimension: ndwdp)
Definition at line 1675 of file model.h.

18.16.4.33 dwdx_ `SUNMatrixWrapper dwdx_ [mutable], [protected]`
Sparse dwdx temporary storage (dimension: ndwdx)
Definition at line 1678 of file model.h.

18.16.4.34 M_ `SUNMatrixWrapper M_ [mutable], [protected]`
Dense Mass matrix (dimension: nx_solver x nx_solver)
Definition at line 1681 of file model.h.

18.16.4.35 my_ `std::vector<realtype> my_ [mutable], [protected]`
current observable (dimension: nytrue)
Definition at line 1684 of file model.h.

18.16.4.36 mz_ `std::vector<realtype> mz_ [mutable], [protected]`
current event measurement (dimension: nztrue)
Definition at line 1687 of file model.h.

18.16.4.37 dJydy_ `std::vector<SUNMatrixWrapper> dJydy_ [mutable], [protected]`
Sparse observable derivative of data likelihood, only used if pythonGenerated==true (dimension nytrue, nJ x ny, row-major)
Definition at line 1691 of file model.h.

18.16.4.38 dJydy_matlab_ `std::vector<realtype> dJydy_matlab_ [mutable], [protected]`
observable derivative of data likelihood, only used if pythonGenerated==false (dimension nJ x ny x nytrue, row-major)
Definition at line 1696 of file model.h.

18.16.4.39 dJydsigma_ `std::vector<realtype> dJydsigma_ [mutable], [protected]`
observable sigma derivative of data likelihood (dimension nJ x ny x nytrue, row-major)
Definition at line 1701 of file model.h.

18.16.4.40 dJydx_ `std::vector<realtype> dJydx_ [mutable], [protected]`
state derivative of data likelihood (dimension nJ x nx_solver, row-major)
Definition at line 1706 of file model.h.

18.16.4.41 dJydp_ `std::vector<realtype> dJydp_ [mutable], [protected]`
parameter derivative of data likelihood for current timepoint (dimension: nJ x nplist, row-major)
Definition at line 1711 of file model.h.

18.16.4.42 dJzdz_ `std::vector<realtype> dJzdz_ [mutable], [protected]`
event output derivative of event likelihood (dimension nJ x nz x nztrue, row-major)
Definition at line 1716 of file model.h.

18.16.4.43 dJzdsigma_ `std::vector<realtype> dJzdsigma_ [mutable], [protected]`

event sigma derivative of event likelihood (dimension nJ x nz x nztrue, row-major)
Definition at line 1721 of file model.h.

18.16.4.44 dJrzdz_ `std::vector<realtype> dJrzdz_ [mutable], [protected]`

event output derivative of event likelihood at final timepoint (dimension nJ x nz x nztrue, row-major)
Definition at line 1726 of file model.h.

18.16.4.45 dJrzdsigma_ `std::vector<realtype> dJrzdsigma_ [mutable], [protected]`

event sigma derivative of event likelihood at final timepoint (dimension nJ x nz x nztrue, row-major)
Definition at line 1731 of file model.h.

18.16.4.46 dJzdx_ `std::vector<realtype> dJzdx_ [mutable], [protected]`

state derivative of event likelihood (dimension nJ x nx_solver, row-major)
Definition at line 1736 of file model.h.

18.16.4.47 dJzdp_ `std::vector<realtype> dJzdp_ [mutable], [protected]`

parameter derivative of event likelihood for current timepoint (dimension: nJ x nplist x, row-major)
Definition at line 1741 of file model.h.

18.16.4.48 dzdx_ `std::vector<realtype> dzdx_ [mutable], [protected]`

state derivative of event output (dimension: nz x nx_solver, row-major)
Definition at line 1746 of file model.h.

18.16.4.49 dzdp_ `std::vector<realtype> dzdp_ [mutable], [protected]`

parameter derivative of event output (dimension: nz x nplist, row-major)
Definition at line 1751 of file model.h.

18.16.4.50 drzdx_ `std::vector<realtype> drzdx_ [mutable], [protected]`

state derivative of event regularization variable (dimension: nz x nx_solver, row-major)
Definition at line 1756 of file model.h.

18.16.4.51 drzdp_ `std::vector<realtype> drzdp_ [mutable], [protected]`

parameter derivative of event regularization variable (dimension: nz x nplist, row-major)
Definition at line 1761 of file model.h.

18.16.4.52 dydp_ `std::vector<realtype> dydp_ [mutable], [protected]`

parameter derivative of observable (dimension: ny x nplist, row-major)
Definition at line 1766 of file model.h.

18.16.4.53 dydx_ `std::vector<realtype> dydx_ [mutable], [protected]`

state derivative of time-resolved observable (dimension: nx_solver x ny, row-major)
Definition at line 1771 of file model.h.

18.16.4.54 w_ `std::vector<realtype> w_ [mutable], [protected]`
tempory storage of w data across functions (dimension: nw)
Definition at line 1774 of file model.h.

18.16.4.55 sx_ `std::vector<realtype> sx_ [mutable], [protected]`
tempory storage for flattened sx, (dimension: nx_solver x nplist, row-major)
Definition at line 1779 of file model.h.

18.16.4.56 x_rdata_ `std::vector<realtype> x_rdata_ [mutable], [protected]`
tempory storage for x_rdata (dimension: nx_rdata)
Definition at line 1782 of file model.h.

18.16.4.57 sx_rdata_ `std::vector<realtype> sx_rdata_ [mutable], [protected]`
tempory storage for sx_rdata slice (dimension: nx_rdata)
Definition at line 1785 of file model.h.

18.16.4.58 y_ `std::vector<realtype> y_ [mutable], [protected]`
temporary storage for time-resolved observable (dimension: ny)
Definition at line 1788 of file model.h.

18.16.4.59 sigmay_ `std::vector<realtype> sigmay_ [mutable], [protected]`
data standard deviation for current timepoint (dimension: ny)
Definition at line 1791 of file model.h.

18.16.4.60 dsigmaydp_ `std::vector<realtype> dsigmaydp_ [mutable], [protected]`
temporary storage for parameter derivative of data standard deviation, (dimension: ny x nplist, row-major)
Definition at line 1796 of file model.h.

18.16.4.61 z_ `std::vector<realtype> z_ [mutable], [protected]`
temporary storage for event-resolved observable (dimension: nz)
Definition at line 1799 of file model.h.

18.16.4.62 rz_ `std::vector<realtype> rz_ [mutable], [protected]`
temporary storage for event regularization (dimension: nz)
Definition at line 1802 of file model.h.

18.16.4.63 sigmaz_ `std::vector<realtype> sigmaz_ [mutable], [protected]`
temporary storage for event standard deviation (dimension: nz)
Definition at line 1805 of file model.h.

18.16.4.64 dsigmazdp_ `std::vector<realtype> dsigmazdp_ [mutable], [protected]`
temporary storage for parameter derivative of event standard deviation, (dimension: nz x nplist, row-major)
Definition at line 1810 of file model.h.

18.16.4.65 deltax_ `std::vector<realtype> deltax_ [mutable], [protected]`

temporary storage for change in x after event (dimension: nx_solver)
Definition at line 1813 of file model.h.

18.16.4.66 deltasx_ `std::vector<realtype> deltasx_ [mutable], [protected]`

temporary storage for change in sx after event (dimension: nx_solver x nplist, row-major)
Definition at line 1818 of file model.h.

18.16.4.67 deltaxB_ `std::vector<realtype> deltaxB_ [mutable], [protected]`

temporary storage for change in xB after event (dimension: nx_solver)
Definition at line 1821 of file model.h.

18.16.4.68 deltaqB_ `std::vector<realtype> deltaqB_ [mutable], [protected]`

temporary storage for change in qB after event (dimension: nJ x nplist, row-major)
Definition at line 1826 of file model.h.

18.16.4.69 x_pos_tmp_ `AmiVector x_pos_tmp_ {0} [mutable], [protected]`

temporary storage of positified state variables according to stateIsNonNegative (dimension: nx_solver)
Definition at line 1830 of file model.h.

18.16.4.70 original_parameters_ `std::vector<realtype> original_parameters_ [protected]`

orignal user-provided, possibly scaled parameter array (dimension: np)
Definition at line 1834 of file model.h.

18.16.4.71 z2event_ `std::vector<int> z2event_ [protected]`

index indicating to which event an event output belongs
Definition at line 1837 of file model.h.

18.16.4.72 x0data_ `std::vector<realtype> x0data_ [protected]`

state initialisation (size nx_solver)
Definition at line 1840 of file model.h.

18.16.4.73 sx0data_ `std::vector<realtype> sx0data_ [protected]`

sensitivity initialisation (size nx_rdata x nplist, row-major)
Definition at line 1843 of file model.h.

18.16.4.74 ts_ `std::vector<realtype> ts_ [protected]`

timepoints (size nt)
Definition at line 1846 of file model.h.

18.16.4.75 state_is_non_negative_ `std::vector<bool> state_is_non_negative_ [protected]`

vector of bools indicating whether state variables are to be assumed to be positive
Definition at line 1850 of file model.h.

18.16.4.76 any_state_non_negative_ `bool any_state_non_negative_ {false} [protected]`

boolean indicating whether any entry in stateIsNonNegative is `true`
Definition at line 1853 of file model.h.

18.16.4.77 nmaxevent_ `int nmaxevent_ {10} [protected]`

maximal number of events to track
Definition at line 1856 of file model.h.

18.16.4.78 pscale_ `std::vector<ParameterScaling> pscale_ [protected]`

parameter transformation of `originalParameters` (dimension np)
Definition at line 1859 of file model.h.

18.16.4.79 tstart_ `realtype tstart_ {0.0} [protected]`

starting time
Definition at line 1862 of file model.h.

18.16.4.80 steadystate_sensitivity_mode_ `SteadyStateSensitivityMode steadystate_sensitivity_mode_ {SteadyStateSensitivityMode::newtonOnly} [protected]`

flag indicating whether steadystate sensivities are to be computed via FSA when steadyStateSimulation is used
Definition at line 1867 of file model.h.

18.16.4.81 reinitialize_fixed_parameter_initial_states_ `bool reinitialize_fixed_parameter_initial_states_ {false} [protected]`

flag indicating whether reinitialization of states depending on fixed parameters is activated
Definition at line 1872 of file model.h.

18.16.4.82 always_check_finite_ `bool always_check_finite_ {false} [protected]`

Indicates whether the result of every call to Model::f∗ should be checked for finiteness
Definition at line 1876 of file model.h.

18.17 Model_DAE Class Reference

The Model class represents an AMICI DAE model.
`#include <model_dae.h>`

Inheritance diagram for Model_DAE:

Public Member Functions

- Model_DAE ()=default
- Model_DAE (const int nx_rdata, const int nxtrue_rdata, const int nx_solver, const int nxtrue_solver, const int nx_solver_reinit, 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 ndxdotdw, std::vector< int > ndJydy, const int nnz, const int ubw, const int lbw, const SecondOrderMode o2mode, std::vector< realtype > const &p, std::vector< realtype > const &k, std::vector< int > const &plist, std::vector< realtype > const &idlist, std::vector< int > const &z2event, const bool pythonGenerated=false, const int ndxdotdp_explicit=0, const int ndxdotdp_implicit=0)

  Constructor with model dimensions.
- void fJ (realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J) override

  Dense Jacobian function.
- void fJ (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xdot, SUNMatrix J)

  Jacobian of xdot with respect to states x.
- void fJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB) override

  Dense Jacobian function.
- void fJB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, SUNMatrix JB)

  Jacobian of xBdot with respect to adjoint state xB.
- void fJSparse (realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J) override

  Sparse Jacobian function.
- void fJSparse (realtype t, realtype cj, N_Vector x, N_Vector dx, SUNMatrix J)

  J in sparse form (for sparse solvers from the SuiteSparse Package)
- void fJSparseB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB) override

  Sparse Jacobian function.
- void fJSparseB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, SUNMatrix JB)

  JB in sparse form (for sparse solvers from the SuiteSparse Package)
- void fJDiag (realtype t, AmiVector &JDiag, realtype cj, const AmiVector &x, const AmiVector &dx) override

  Diagonal of the Jacobian (for preconditioning)
- void fJv (realtype t, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, const AmiVector &v, AmiVector &nJv, realtype cj) override

Jacobian multiply function.

- 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)
- 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)
- void froot (realtype t, const AmiVector &x, const AmiVector &dx, gsl::span< realtype > root) override

  Root function.
- void froot (realtype t, N_Vector x, N_Vector dx, gsl::span< realtype > root)

  Event trigger function for events.
- void fxdot (realtype t, const AmiVector &x, const AmiVector &dx, AmiVector &xdot) override

  Residual function.
- void fxdot (realtype t, N_Vector x, N_Vector dx, N_Vector xdot)

  Residual function of the DAE.
- 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.
- 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.
- void fxBdot_ss (const realtype t, const AmiVector &xB, const AmiVector &dxB, AmiVector &xBdot) override

  Residual function backward when running in steady state mode.
- void fxBdot_ss (realtype t, N_Vector xB, N_Vector dxB, N_Vector xBdot) const

  Implementation of fxBdot for steady state case at the N_Vector level.
- void fqBdot_ss (realtype t, N_Vector xB, N_Vector dxB, N_Vector qBdot) const

  Implementation of fqBdot for steady state at the N_Vector level.
- void fJSparseB_ss (SUNMatrix JB) override

  Sparse Jacobian function backward, steady state case.
- void writeSteadystateJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot) override

  Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.
- void fdxdotdp (realtype t, const N_Vector x, const N_Vector dx)

  Sensitivity of dx/dt wrt model parameters p.
- void fdxdotdp (const realtype t, const AmiVector &x, const AmiVector &dx) override

  Model specific sparse implementation of explicit parameter derivative of right hand side.
- void fsxdot (realtype t, const AmiVector &x, const AmiVector &dx, int ip, const AmiVector &sx, const AmiVector &sdx, AmiVector &sxdot) override

  Sensitivity Residual function.
- 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.
- void fM (realtype t, const N_Vector x)

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

  Retrieves the solver object.

Protected Member Functions

- virtual void fJ (realtype *J, realtype t, const realtype *x, const double *p, const double *k, const realtype *h, realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)=0

  Model specific implementation for fJ.
- virtual void fJB (realtype *JB, realtype t, const realtype *x, const double *p, const double *k, const realtype *h, realtype cj, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJB.

- virtual void fJSparse (SUNMatrixContent_Sparse JSparse, realtype t, const realtype ∗x, const double ∗p, const double ∗k, const realtype ∗h, realtype cj, const realtype ∗dx, const realtype ∗w, const realtype ∗dwdx)=0

  Model specific implementation for fJSparse.
- virtual void fJSparseB (SUNMatrixContent_Sparse 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)

  Model specific implementation for fJSparseB.
- virtual void fJDiag (realtype ∗JDiag, realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, realtype cj, const realtype ∗dx, const realtype ∗w, const realtype ∗dwdx)

  Model specific implementation for fJDiag.
- virtual void fJvB (realtype ∗JvB, realtype t, const realtype ∗x, const double ∗p, const double ∗k, const realtype ∗h, realtype cj, const realtype ∗xB, const realtype ∗dx, const realtype ∗dxB, const realtype ∗vB, const realtype ∗w, const realtype ∗dwdx)

  Model specific implementation for fJvB.
- virtual void froot (realtype ∗root, realtype t, const realtype ∗x, const double ∗p, const double ∗k, const realtype ∗h, const realtype ∗dx)

  Model specific implementation for froot.
- virtual void fxdot (realtype ∗xdot, realtype t, const realtype ∗x, const double ∗p, const double ∗k, const realtype ∗h, const realtype ∗dx, const realtype ∗w)=0

  Model specific implementation for fxdot.
- virtual void fdxdotdp (realtype ∗dxdotdp, realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, int ip, const realtype ∗dx, const realtype ∗w, const realtype ∗dwdp)

  Model specific implementation of fdxdotdp.
- virtual void fM (realtype ∗M, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k)

  Model specific implementation of fM.

Additional Inherited Members

18.17.1 Detailed Description

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.
Definition at line 27 of file model_dae.h.

18.17.2 Constructor & Destructor Documentation

18.17.2.1 Model_DAE() [1/2] `Model_DAE ( ) [default]`

default constructor

18.17.2.2 Model_DAE() [2/2] `Model_DAE (`

```
const int nx_rdata,
const int nxtrue_rdata,
const int nx_solver,
const int nxtrue_solver,
const int nx_solver_reinit,
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 ndxdotdw,
std::vector< int > ndJydy,
const int nnz,
const int ubw,
const int lbw,
const SecondOrderMode o2mode,
std::vector< realtype > const & p,
std::vector< realtype > const & k,
std::vector< int > const & plist,
std::vector< realtype > const & idlist,
std::vector< int > const & z2event,
const bool pythonGenerated = false,
const int ndxdotdp_explicit = 0,
const int ndxdotdp_implicit = 0 )
```

Parameters

nx_rdata	number of state variables
nxtrue_rdata	number of state variables of the non-augmented model
nx_solver	number of state variables with conservation laws applied
nxtrue_solver	number of state variables of the non-augmented model with conservation laws applied
nx_solver_reinit	number of state variables with conservation laws subject to reinitialization
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
ndxdotdw	number of nonzero elements dxdotdw
ndJydy	number of nonzero elements dJydy
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
pythonGenerated	flag indicating matlab or python wrapping
ndxdotdp_explicit	number of nonzero elements dxdotdp_explicit
ndxdotdp_implicit	number of nonzero elements dxdotdp_implicit

Definition at line 67 of file model_dae.h.

18.17.3 Member Function Documentation

18.17.3.1 fJ() [1/3]

```
void fJ (
    realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xdot,
    SUNMatrix J ) [override], [virtual]
```

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 AbstractModel.
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:

18.17.3.2 fJ() [2/3]

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

Parameters

t	timepoint
cj	scaling factor, inverse of the step size

Parameters

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 11 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.3 fJB() [1/3]

```
void fJB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB ) [override], [virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implements AbstractModel.
Definition at line 190 of file model_dae.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.4 fJB() [2/3]

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

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 198 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.5 fJSparse() [1/3]

```
void fJSparse (
    realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xdot,
    SUNMatrix J ) [override], [virtual]
```

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 AbstractModel.
Definition at line 21 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.6 fJSparse() [2/3]

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

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 27 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.7 fJSparseB() [1/3]

```
void fJSparseB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB ) [override], [virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implements AbstractModel.
Definition at line 209 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.8 fJSparseB() [2/3]

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

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 216 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.9 fJDiag() [1/2]

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

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

Implements AbstractModel.
Definition at line 92 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.10 fJv() [1/2]

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

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 AbstractModel.
Definition at line 50 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.11 fJv() [2/2]

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

Parameters

t	timepoint
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 57 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.12 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 )
```

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 228 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.13 froot() [1/3]

```
void froot (
realtype t,
const AmiVector & x,
const AmiVector & dx,
gsl::span< realtype > root ) [override], [virtual]
```

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 AbstractModel.
Definition at line 64 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.14 froot() [2/3]

```
void froot (
    realtype t,
    N_Vector x,
    N_Vector dx,
    gsl::span< realtype > root )
```

Parameters

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

Definition at line 69 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.15 fxdot() [1/3]

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

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 AbstractModel.
Definition at line 78 of file model_dae.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.16 fxdot() [2/3]

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

Parameters

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

Definition at line 83 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.17 fxBdot()

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

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 235 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.18 fqBdot()

```
void fqBdot (
    realtype t,
    N_Vector x,
    N_Vector dx,
```

```
N_Vector xB,
N_Vector dxB,
N_Vector qBdot )
```

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 243 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.19 fxBdot_ss() [1/2]

```
void fxBdot_ss (
    const realtype t,
    const AmiVector & xB,
    const AmiVector & dxB,
    AmiVector & xBdot ) [override], [virtual]
```

Parameters

t	time
xB	adjoint state
dxB	time derivative of state (DAE only)
xBdot	array to which values of the residual function will be written

Implements AbstractModel.
Definition at line 259 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.20 fxBdot_ss() [2/2]

```
void fxBdot_ss (
    realtype t,
    N_Vector xB,
    N_Vector dxB,
    N_Vector xBdot ) const
```

Parameters

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

Definition at line 264 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.21 fqBdot_ss()

```
void fqBdot_ss (
    realtype t,
    N_Vector xB,
    N_Vector dxB,
    N_Vector qBdot ) const
```

Parameters

t	timepoint
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 274 of file model_dae.cpp.

18.17.3.22 fJSparseB_ss()

```
void fJSparseB_ss (
    SUNMatrix JB ) [override], [virtual]
```

Parameters

JB	sparse matrix to which values of the Jacobian will be written

Implements AbstractModel.
Definition at line 281 of file model_dae.cpp.
Here is the call graph for this function:

18.17.3.23 writeSteadystateJB()

```
void writeSteadystateJB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot ) [override], [virtual]
```

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
xBdot	Vector with the adjoint state right hand side

Implements AbstractModel.
Definition at line 286 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.24 fdxdotdp() [1/3]

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

Parameters

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

Definition at line 105 of file model_dae.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.17.3.25 fdxdotdp() [2/3]

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

Parameters

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

Implements AbstractModel.
Definition at line 292 of file model_dae.h.

Here is the call graph for this function:

18.17.3.26 fsxdot() [1/2]

```
void fsxdot (
    realtype t,
    const AmiVector & x,
    const AmiVector & dx,
    int ip,
    const AmiVector & sx,
    const AmiVector & sdx,
    AmiVector & sxdot ) [override], [virtual]
```

Parameters

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

Implements AbstractModel.
Definition at line 297 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.27 fsxdot() [2/2]

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

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 304 of file model_dae.cpp.

Here is the call graph for this function:

18.17.3.28 fM() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 126 of file model_dae.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Returns

The Solver instance

Implements AbstractModel.
Definition at line 133 of file model_dae.cpp.

18.17.3.30 fJ() [3/3]

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

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

18.17.3.31 fJB() [3/3]

```
void fJB (
    realtype * JB,
    realtype t,
```

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

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
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.cpp.

18.17.3.32 fJSparse() [3/3]

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

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

18.17.3.33 fJSparseB() [3/3]

```
void fJSparseB (
    SUNMatrixContent_Sparse 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]
```

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 38 of file model_dae.cpp.

18.17.3.34 fJDiag() [2/2]

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

Parameters

JDiag	array to which the Jacobian diagonal 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

Parameters

dwdx	derivative of w wrt x

Definition at line 148 of file model_dae.cpp.

18.17.3.35 fJvB() [2/2]

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

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 158 of file model_dae.cpp.

18.17.3.36 froot() [3/3]

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

Parameters

root	values of the trigger function
t	timepoint
x	Vector with the states

Parameters

p	parameter vector
k	constants vector
h	heavyside vector
dx	Vector with the derivative states

Definition at line 169 of file model_dae.cpp.

18.17.3.37 fxdot() [3/3]

```
virtual void fxdot (
    realtype * xdot,
    realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype * dx,
    const realtype * w )  [protected], [pure virtual]
```

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

18.17.3.38 fdxdotdp() [3/3]

```
void fdxdotdp (
    realtype * dxdotdp,
    realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    int ip,
    const realtype * dx,
    const realtype * w,
    const realtype * dwdp )  [protected], [virtual]
```

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

Parameters

w	vector with helper variables
dwdp	derivative of w wrt p

Definition at line 177 of file model_dae.cpp.

18.17.3.39 fM() [2/2]

```
void fM (
    realtype * M,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k )  [protected], [virtual]
```

Parameters

M	mass matrix
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector

Definition at line 187 of file model_dae.cpp.

18.18 Model_ODE Class Reference

The Model class represents an AMICI ODE model.

`#include <model_ode.h>`

Inheritance diagram for Model_ODE:

Public Member Functions

- Model_ODE ()=default
- Model_ODE (const int nx_rdata, const int nxtrue_rdata, const int nx_solver, const int nxtrue_solver, const int nx_solver_reinit, 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 ndxdotdw, std::vector< int > ndJydy, const int nnz, const int

ubw, const int lbw, const SecondOrderMode o2mode, std::vector< realtype > const &p, std::vector< realtype > const &k, std::vector< int > const &plist, std::vector< realtype > const &idlist, std::vector< int > const &z2event, const bool pythonGenerated=false, const int ndxdotdp_explicit=0, const int ndxdotdp_implicit=0)

Constructor with model dimensions.

- void fJ (realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J) override

  Dense Jacobian function.
- void fJ (realtype t, N_Vector x, N_Vector xdot, SUNMatrix J)

  Implementation of fJ at the N_Vector level.
- void fJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB) override

  Dense Jacobian function.
- void fJB (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, SUNMatrix JB)

  Implementation of fJB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation.
- void fJSparse (realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, SUNMatrix J) override

  Sparse Jacobian function.
- void fJSparse (realtype t, N_Vector x, SUNMatrix 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.
- void fJSparseB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot, SUNMatrix JB) override

  Sparse Jacobian function.
- void fJSparseB (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, SUNMatrix JB)

  Implementation of fJSparseB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation.
- 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.
- void fJDiag (realtype t, AmiVector &JDiag, realtype cj, const AmiVector &x, const AmiVector &dx) override

  Diagonal of the Jacobian (for preconditioning)
- void fJv (realtype t, const AmiVector &x, const AmiVector &dx, const AmiVector &xdot, const AmiVector &v, AmiVector &nJv, realtype cj) override

  Jacobian multiply function.
- void fJv (N_Vector v, N_Vector Jv, realtype t, N_Vector x)

  Implementation of fJv at the N_Vector level.
- void fJvB (N_Vector vB, N_Vector JvB, realtype t, N_Vector x, N_Vector xB)

  Implementation of fJvB at the N_Vector level.
- void froot (realtype t, const AmiVector &x, const AmiVector &dx, gsl::span< realtype > root) override

  Root function.
- void froot (realtype t, N_Vector x, gsl::span< 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.
- void fxdot (realtype t, const AmiVector &x, const AmiVector &dx, AmiVector &xdot) override

  Residual function.
- 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.
- void fxBdot (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot)

  Implementation of fxBdot at the N_Vector level.
- void fqBdot (realtype t, N_Vector x, N_Vector xB, N_Vector qBdot)

  Implementation of fqBdot at the N_Vector level.

- void fxBdot_ss (const realtype t, const AmiVector &xB, const AmiVector &, AmiVector &xBdot) override

  Residual function backward when running in steady state mode.
- void fxBdot_ss (realtype t, N_Vector xB, N_Vector xBdot) const

  Implementation of fxBdot for steady state at the N_Vector level.
- void fqBdot_ss (realtype t, N_Vector xB, N_Vector qBdot) const

  Implementation of fqBdot for steady state case at the N_Vector level.
- void fJSparseB_ss (SUNMatrix JB) override

  Sparse Jacobian function backward, steady state case.
- void writeSteadystateJB (const realtype t, realtype cj, const AmiVector &x, const AmiVector &dx, const AmiVector &xB, const AmiVector &dxB, const AmiVector &xBdot) override

  Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.
- void fdxdotdw (realtype t, const N_Vector x)

  Sensitivity of dx/dt wrt model parameters w.
- void fdxdotdp (realtype t, const N_Vector x)
- void fdxdotdp (realtype t, const AmiVector &x, const AmiVector &dx) override

  Model specific sparse implementation of explicit parameter derivative of right hand side.
- void fsxdot (realtype t, const AmiVector &x, const AmiVector &dx, int ip, const AmiVector &sx, const AmiVector &sdx, AmiVector &sxdot) override

  Sensitivity Residual function.
- void fsxdot (realtype t, N_Vector x, int ip, N_Vector sx, N_Vector sxdot)

  Implementation of fsxdot at the N_Vector level.
- std::unique_ptr< Solver > getSolver () override

  Retrieves the solver object.

Protected Member Functions

- virtual void fJ (realtype *J, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)=0

  Model specific implementation for fJ.
- virtual void fJB (realtype *JB, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJB.
- virtual void fJSparse (SUNMatrixContent_Sparse JSparse, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJSparse.
- virtual void fJSparse (realtype *JSparse, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJSparse, data only.
- virtual void fJSparse_colptrs (sunindextype *indexptrs)

  Model specific implementation for fJSparse, column pointers.
- virtual void fJSparse_rowvals (sunindextype *indexvals)

  Model specific implementation for fJSparse, row values.
- virtual void fJSparseB (SUNMatrixContent_Sparse JSparseB, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJSparseB.
- virtual void fJSparseB (realtype *JSparseB, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJSparseB.
- virtual void fJSparseB_colptrs (sunindextype *indexptrs)

  Model specific implementation for fJSparse, column pointers.
- virtual void fJSparseB_rowvals (sunindextype *indexvals)

  Model specific implementation for fJSparse, row values.

- virtual void fJDiag (realtype *JDiag, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)

  Model specific implementation for fJDiag.
- virtual void froot (realtype *root, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)

  Model specific implementation for froot.
- virtual void fxdot (realtype *xdot, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)=0

  Model specific implementation for fxdot.
- virtual void fdxdotdp (realtype *dxdotdp, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, int ip, const realtype *w, const realtype *dwdp)

  Model specific implementation of fdxdotdp, with w chainrule (Matlab)
- virtual void fdxdotdp_explicit (realtype *dxdotdp_explicit, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)

  Model specific implementation of fdxdotdp_explicit, no w chainrule (Py)
- virtual void fdxdotdp_explicit_colptrs (sunindextype *indexptrs)

  Model specific implementation of fdxdotdp_explicit, colptrs part.
- virtual void fdxdotdp_explicit_rowvals (sunindextype *indexvals)

  Model specific implementation of fdxdotdp_explicit, rowvals part.
- virtual void fdxdotdp_implicit_colptrs (sunindextype *indexptrs)

  Model specific implementation of fdxdotdp_implicit, colptrs part.
- virtual void fdxdotdp_implicit_rowvals (sunindextype *indexvals)

  Model specific implementation of fdxdotdp_implicit, rowvals part.
- virtual void fdxdotdw (realtype *dxdotdw, realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)

  Model specific implementation of fdxdotdw, data part.
- virtual void fdxdotdw_colptrs (sunindextype *indexptrs)

  Model specific implementation of fdxdotdw, colptrs part.
- virtual void fdxdotdw_rowvals (sunindextype *indexvals)

  Model specific implementation of fdxdotdw, rowvals part.

Additional Inherited Members

18.18.1 Detailed Description

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.
Definition at line 27 of file model_ode.h.

18.18.2 Constructor & Destructor Documentation

18.18.2.1 Model_ODE() [1/2] Model_ODE ( ) [default]

default constructor

18.18.2.2 Model_ODE() [2/2] Model_ODE (

```
const int nx_rdata,
const int nxtrue_rdata,
const int nx_solver,
const int nxtrue_solver,
const int nx_solver_reinit,
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 ndxdotdw,
std::vector< int > ndJydy,
const int nnz,
const int ubw,
const int lbw,
const SecondOrderMode o2mode,
std::vector< realtype > const & p,
std::vector< realtype > const & k,
std::vector< int > const & plist,
std::vector< realtype > const & idlist,
std::vector< int > const & z2event,
const bool pythonGenerated = false,
const int ndxdotdp_explicit = 0,
const int ndxdotdp_implicit = 0 )
```

Parameters

nx_rdata	number of state variables
nxtrue_rdata	number of state variables of the non-augmented model
nx_solver	number of state variables with conservation laws applied
nxtrue_solver	number of state variables of the non-augmented model with conservation laws applied
nx_solver_reinit	number of state variables with conservation laws subject to reinitialization
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
ndxdotdw	number of nonzero elements dxdotdw
ndJydy	number of nonzero elements dJydy
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
pythonGenerated	flag indicating matlab or python wrapping
ndxdotdp_explicit	number of nonzero elements dxdotdp_explicit
ndxdotdp_implicit	number of nonzero elements dxdotdp_implicit

Definition at line 67 of file model_ode.h.

18.18.3 Member Function Documentation

18.18.3.1 fJ() [1/3]

```
void fJ (
    realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xdot,
    SUNMatrix J ) [override], [virtual]
```

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 AbstractModel.
Definition at line 7 of file model_ode.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.2 fJ() [2/3]

```
void fJ (
    realtype t,
    N_Vector x,
    N_Vector xdot,
    SUNMatrix J )
```

This function provides an interface to the model specific routines for the solver implementation as well 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 13 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.3 fJB() [1/3]

```
void fJB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB ) [override], [virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implements AbstractModel.
Definition at line 304 of file model_ode.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.4 fJB() [2/3]

```
void fJB (
    realtype t,
    N_Vector x,
    N_Vector xB,
    N_Vector xBdot,
    SUNMatrix JB )
```

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 311 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.5 fJSparse() [1/4]

```
void fJSparse (
    realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xdot,
    SUNMatrix J ) [override], [virtual]
```

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 AbstractModel.
Definition at line 22 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.6 fJSparse() [2/4]

```
void fJSparse (
    realtype t,
    N_Vector x,
    SUNMatrix J )
```

Parameters

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

Definition at line 28 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.7 fJSparseB() [1/4]

```
void fJSparseB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot,
    SUNMatrix JB ) [override], [virtual]
```

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side (unused)
JB	dense matrix to which values of the jacobian will be written

Implements AbstractModel.
Definition at line 321 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.8 fJSparseB() [2/4]

```
void fJSparseB (
    realtype t,
    N_Vector x,
    N_Vector xB,
    N_Vector xBdot,
    SUNMatrix JB )
```

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 328 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.9 fJDiag() [1/3]

```
void fJDiag (
    realtype t,
    N_Vector JDiag,
    N_Vector x )
```

Parameters

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

Definition at line 348 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.10 fJDiag() [2/3]

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

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

Implements AbstractModel.
Definition at line 85 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.11 fJv() [1/2]

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

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 AbstractModel.
Definition at line 46 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.12 fJv() [2/2]

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

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 52 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.13 fJvB()

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

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 357 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.14 froot() [1/3]

```
void froot (
    realtype t,
    const AmiVector & x,
    const AmiVector & dx,
    gsl::span< realtype > root ) [override], [virtual]
```

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 AbstractModel.
Definition at line 58 of file model_ode.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.15 froot() [2/3]

```
void froot (
    realtype t,
    N_Vector x,
    gsl::span< realtype > root )
```

Parameters

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

Definition at line 63 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.16 fxdot() [1/3]

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

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 AbstractModel.
Definition at line 71 of file model_ode.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.17 fxdot() [2/3]

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

Parameters

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

Definition at line 76 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.18 fxBdot()

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

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 364 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.19 fqBdot()

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

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 370 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.20 fxBdot_ss() [1/2]

```
void fxBdot_ss (
    const realtype t,
    const AmiVector & xB,
    const AmiVector & dxB,
    AmiVector & xBdot ) [override], [virtual]
```

Parameters

t	time
xB	adjoint state
dxB	time derivative of state (DAE only)
xBdot	array to which values of the residual function will be written

Implements AbstractModel.
Definition at line 397 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.21 fxBdot_ss() [2/2]

```
void fxBdot_ss (
    realtype t,
    N_Vector xB,
    N_Vector xBdot ) const
```

Parameters

t	timepoint
xB	Vector with the states
xBdot	Vector with the adjoint right hand side

Definition at line 402 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.22 fqBdot_ss()

```
void fqBdot_ss (
    realtype t,
    N_Vector xB,
    N_Vector qBdot ) const
```

Parameters

t	timepoint
xB	Vector with the adjoint states
qBdot	Vector with the adjoint quadrature right hand side

Definition at line 409 of file model_ode.cpp.

18.18.3.23 fJSparseB_ss()

```
void fJSparseB_ss (
```

```
SUNMatrix JB )  [override], [virtual]
```

Parameters

JB	sparse matrix to which values of the Jacobian will be written

Implements AbstractModel.
Definition at line 415 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.24 writeSteadystateJB()

```
void writeSteadystateJB (
    const realtype t,
    realtype cj,
    const AmiVector & x,
    const AmiVector & dx,
    const AmiVector & xB,
    const AmiVector & dxB,
    const AmiVector & xBdot )  [override], [virtual]
```

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
xBdot	Vector with the adjoint state right hand side

Implements AbstractModel.
Definition at line 420 of file model_ode.cpp.

Here is the call graph for this function:

18.18.3.25 fdxdotdw() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 93 of file model_ode.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.26 fdxdotdp() [1/3]

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

Explicit sensitivity of dx/dt wrt model parameters p

Parameters

t	timepoint
x	Vector with the states

Definition at line 105 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.18.3.27 fdxdotdp() [2/3]

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

Parameters

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

Implements AbstractModel.
Definition at line 141 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.28 fsxdot() [1/2]

```
void fsxdot (
    realtype t,
    const AmiVector & x,
```

```
const AmiVector & dx,
int ip,
const AmiVector & sx,
const AmiVector & sdx,
AmiVector & sxdot )  [override], [virtual]
```

Parameters

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

Implements AbstractModel.
Definition at line 430 of file model_ode.cpp.
Here is the call graph for this function:

18.18.3.29 fsxdot() [2/2]

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

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 437 of file model_ode.cpp.

Here is the call graph for this function:

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

Returns

The Solver instance

Implements AbstractModel.
Definition at line 146 of file model_ode.cpp.

18.18.3.31 fJ() [3/3]

```
virtual void fJ (
    realtype * J,
    realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
```

```
const realtype * h,
const realtype * w,
const realtype * dwdx )  [protected], [pure virtual]
```

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

18.18.3.32 fJB() [3/3]

```
void fJB (
realtype * JB,
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]
```

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 150 of file model_ode.cpp.

18.18.3.33 fJSparse() [3/4]

```
void fJSparse (
SUNMatrixContent_Sparse JSparse,
realtype t,
const realtype * x,
const realtype * p,
const realtype * k,
const realtype * h,
const realtype * w,
const realtype * dwdx )  [protected], [virtual]
```

Parameters

JSparse	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 160 of file model_ode.cpp.

18.18.3.34 fJSparse() [4/4]

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

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 170 of file model_ode.cpp.

18.18.3.35 fJSparse_colptrs()

```
void fJSparse_colptrs (
    sunindextype * indexptrs ) [protected], [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 179 of file model_ode.cpp.

Here is the caller graph for this function:

18.18.3.36 fJSparse_rowvals()

```
void fJSparse_rowvals (
    sunindextype * indexvals ) [protected], [virtual]
```

Parameters

indexvals	row values

Definition at line 185 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.37 fJSparseB() [3/4]

```
void fJSparseB (
    SUNMatrixContent_Sparse JSparseB,
    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]
```

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

Parameters

w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 191 of file model_ode.cpp.

18.18.3.38 fJSparseB() [4/4]

```
void fJSparseB (
    realtype * JSparseB,
    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]
```

Parameters

JSparseB	data array
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 201 of file model_ode.cpp.

18.18.3.39 fJSparseB_colptrs()

```
void fJSparseB_colptrs (
    sunindextype * indexptrs ) [protected], [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 211 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.40 fJSparseB_rowvals()

```
void fJSparseB_rowvals (
    sunindextype * indexvals ) [protected], [virtual]
```

Parameters

indexvals	row values

Definition at line 217 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.41 fJDiag() [3/3]

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

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 223 of file model_ode.cpp.

18.18.3.42 froot() [3/3]

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

Parameters

root	values of the trigger function

Parameters

t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector

Definition at line 232 of file model_ode.cpp.

18.18.3.43 fxdot() [3/3]

```
virtual void fxdot (
    realtype * xdot,
    realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w )  [protected], [pure virtual]
```

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

18.18.3.44 fdxdotdp() [3/3]

```
void fdxdotdp (
    realtype * dxdotdp,
    realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    int ip,
    const realtype * w,
    const realtype * dwdp )  [protected], [virtual]
```

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 240 of file model_ode.cpp.

18.18.3.45 fdxdotdp_explicit()

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

Parameters

dxdotdp_explicit	partial derivative xdot wrt p
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

Definition at line 250 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.46 fdxdotdp_explicit_colptrs()

```
void fdxdotdp_explicit_colptrs (
    sunindextype * indexptrs ) [protected], [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 259 of file model_ode.cpp.

Here is the caller graph for this function:

18.18.3.47 fdxdotdp_explicit_rowvals()

```
void fdxdotdp_explicit_rowvals ( sunindextype * indexvals ) [protected], [virtual]
```

Parameters

indexvals	row values

Definition at line 265 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.48 fdxdotdp_implicit_colptrs()

```
void fdxdotdp_implicit_colptrs ( sunindextype * indexptrs ) [protected], [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 271 of file model_ode.cpp.

Here is the caller graph for this function:

18.18.3.49 fdxdotdp_implicit_rowvals()

```
void fdxdotdp_implicit_rowvals (
    sunindextype * indexvals )  [protected], [virtual]
```

Parameters

indexvals	row values

Definition at line 277 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.50 fdxdotdw() [2/2]

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

Parameters

dxdotdw	partial derivative xdot wrt w

Parameters

t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

Definition at line 283 of file model_ode.cpp.

18.18.3.51 fdxdotdw_colptrs()

```
void fdxdotdw_colptrs ( sunindextype * indexptrs ) [protected], [virtual]
```

Parameters

indexptrs	column pointers

Definition at line 292 of file model_ode.cpp.
Here is the caller graph for this function:

18.18.3.52 fdxdotdw_rowvals()

```
void fdxdotdw_rowvals ( sunindextype * indexvals ) [protected], [virtual]
```

Parameters

indexvals	row values

Definition at line 298 of file model_ode.cpp.

Here is the caller graph for this function:

18.19 ModelContext Class Reference

The ModelContext temporarily stores amici::Model::state and restores it when going out of scope.
`#include <rdata.h>`
Inheritance diagram for ModelContext:

Public Member Functions

- ModelContext (Model *model)

  initialize backup of the original values.
- ModelContext & **operator=** (const ModelContext &other)=delete
- void restore ()

  Restore original state on constructor-supplied amici::Model. Will be called during destruction. Explicit call is generally not necessary.

18.19.1 Detailed Description

Definition at line 687 of file rdata.h.

18.19.2 Constructor & Destructor Documentation

18.19.2.1 ModelContext()

```
ModelContext (
    Model * model ) [explicit]
```

Parameters

model	

Definition at line 827 of file rdata.cpp.

18.20 ModelState Struct Reference

implements an exchange format to store and transfer the state of the model at a specific timepoint. This is designed to only encompass the minimal number of attributes that need to be transferred.

`#include <model.h>`

Public Attributes

- std::vector< realtype > h
- std::vector< realtype > total_cl
- std::vector< realtype > stotal_cl
- std::vector< realtype > unscaledParameters
- std::vector< realtype > fixedParameters
- std::vector< int > plist

18.20.1 Detailed Description

Definition at line 39 of file model.h.

18.20.2 Member Data Documentation

18.20.2.1 h `std::vector<realtype> h`

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

Definition at line 42 of file model.h.

18.20.2.2 total_cl `std::vector<realtype> total_cl`

total abundances for conservation laws (dimension: nx_rdata-nx_solver)

Definition at line 46 of file model.h.

18.20.2.3 stotal_cl `std::vector<realtype> stotal_cl`

sensitivities of total abundances for conservation laws (dimension: (nx_rdata-nx_solver) x np, row-major)

Definition at line 50 of file model.h.

18.20.2.4 unscaledParameters `std::vector<realtype> unscaledParameters`

unscaled parameters (dimension: np)

Definition at line 53 of file model.h.

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

constants (dimension: nk)

Definition at line 56 of file model.h.

18.20.2.6 plist `std::vector<int> plist`

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

Definition at line 61 of file model.h.

18.21 NewtonFailure Class Reference

Newton failure exception.
`#include <exception.h>`
Inheritance diagram for NewtonFailure:

Public Member Functions

- NewtonFailure (int code, const char *function)

  Constructor, simply calls AmiException constructor.

Public Attributes

- int error_code

18.21.1 Detailed Description

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
Definition at line 149 of file exception.h.

18.21.2 Constructor & Destructor Documentation

18.21.2.1 NewtonFailure()

```
NewtonFailure (
    int code,
    const char * function )
```

Parameters

function	name of the function in which the error occured
code	error code

Definition at line 46 of file exception.cpp.

18.21.3 Member Data Documentation

18.21.3.1 error_code `int error_code`

error code returned by solver
Definition at line 159 of file exception.h.

18.22 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)

  Initializes all members with the provided objects.
- void getStep (int ntry, int nnewt, AmiVector &delta)

  Computes the solution of one Newton iteration.
- void computeNewtonSensis (AmiVectorArray &sx)

  Computes steady state sensitivities.
- const std::vector< int > & getNumLinSteps () const

  Accessor for numlinsteps.
- virtual void prepareLinearSystem (int ntry, int nnewt)=0

  Writes the Jacobian for the Newton iteration and passes it to the linear solver.
- virtual void prepareLinearSystemB (int ntry, int nnewt)=0
- virtual void solveLinearSystem (AmiVector &rhs)=0

  Solves the linear system for the Newton step.

Static Public Member Functions

- static std::unique_ptr< NewtonSolver > getSolver (realtype ∗t, AmiVector ∗x, Solver &simulationSolver, Model ∗model)

  Factory method to create a NewtonSolver based on linsolType.

Public Attributes

- int max_lin_steps_ {0}
- int max_steps {0}
- realtype atol_ {1e-16}
- realtype rtol_ {1e-8}
- NewtonDampingFactorMode damping_factor_mode_ {NewtonDampingFactorMode::on}
- realtype damping_factor_lower_bound {1e-8}

Protected Attributes

- realtype * t_
- Model * model_
- AmiVector xdot_
- AmiVector * x_
- AmiVector dx_
- std::vector< int > num_lin_steps_
- AmiVector xB_
- AmiVector dxB_

18.22.1 Detailed Description

Definition at line 22 of file newton_solver.h.

18.22.2 Constructor & Destructor Documentation

18.22.2.1 NewtonSolver()

```
NewtonSolver (
    realtype * t,
    AmiVector * x,
    Model * model )
```

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the model object

Definition at line 18 of file newton_solver.cpp.

18.22.3 Member Function Documentation

18.22.3.1 getSolver()

```
std::unique_ptr< NewtonSolver > getSolver (
    realtype * t,
    AmiVector * x,
    Solver & simulationSolver,
    Model * model )  [static]
```

Parameters

t	pointer to time variable
x	pointer to state variables
simulationSolver	solver with settings
model	pointer to the model object

Returns

solver NewtonSolver according to the specified linsolType

Definition at line 25 of file newton_solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.22.3.2 getStep()

```
void getStep (
    int ntry,
    int nnewt,
    AmiVector & delta )
```

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 85 of file newton_solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.22.3.3 computeNewtonSensis()

```
void computeNewtonSensis (
    AmiVectorArray & sx )
```

Parameters

sx	pointer to state variable sensitivities

Definition at line 94 of file newton_solver.cpp.

Here is the call graph for this function:

18.22.3.4 getNumLinSteps() `const std::vector<int>& getNumLinSteps ( ) const`

Returns

numlinsteps

Definition at line 69 of file newton_solver.h.

18.22.3.5 prepareLinearSystem() `virtual void prepareLinearSystem ( int ntry, int nnewt ) [pure virtual]`

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:

18.22.3.6 prepareLinearSystemB()

```
virtual void prepareLinearSystemB (
    int ntry,
    int nnewt ) [pure virtual]
```

Writes the Jacobian (JB) 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:

18.22.3.7 solveLinearSystem()

```
virtual void solveLinearSystem (
    AmiVector & rhs ) [pure virtual]
```

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:

18.22.4 Member Data Documentation

18.22.4.1 max_lin_steps_ `int max_lin_steps_ {0}`

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

Definition at line 105 of file newton_solver.h.

18.22.4.2 max_steps `int max_steps {0}`

maximum number of allowed Newton steps for steady state computation

Definition at line 107 of file newton_solver.h.

18.22.4.3 atol_ `realtype atol_ {1e-16}`

absolute tolerance

Definition at line 109 of file newton_solver.h.

18.22.4.4 rtol_ `realtype rtol_ {1e-8}`

relative tolerance

Definition at line 111 of file newton_solver.h.

18.22.4.5 damping_factor_mode_ `NewtonDampingFactorMode damping_factor_mode_ {NewtonDamping↩ FactorMode::on}`

damping factor flag

Definition at line 113 of file newton_solver.h.

18.22.4.6 damping_factor_lower_bound `realtype damping_factor_lower_bound {1e-8}`

damping factor lower bound

Definition at line 115 of file newton_solver.h.

18.22.4.7 t_ `realtype* t_ [protected]`

time variable

Definition at line 119 of file newton_solver.h.

18.22.4.8 model_ `Model* model_ [protected]`

pointer to the model object

Definition at line 121 of file newton_solver.h.

18.22.4.9 xdot_ `AmiVector xdot_ [protected]`

right hand side AmiVector

Definition at line 123 of file newton_solver.h.

18.22.4.10 x_ `AmiVector* x_ [protected]`

current state

Definition at line 125 of file newton_solver.h.

18.22.4.11 dx_ `AmiVector dx_ [protected]`

current state time derivative (DAE)

Definition at line 127 of file newton_solver.h.

18.22.4.12 num_lin_steps_ `std::vector<int> num_lin_steps_ [protected]`

history of number of linear steps

Definition at line 129 of file newton_solver.h.

18.22.4.13 xB_ `AmiVector xB_ [protected]`

current adjoint state

Definition at line 131 of file newton_solver.h.

18.22.4.14 dxB_ `AmiVector dxB_ [protected]`

current adjoint state time derivative (DAE)

Definition at line 133 of file newton_solver.h.

18.23 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)

  Constructor, initializes all members with the provided objects and initializes temporary storage objects.
- **NewtonSolverDense** (const NewtonSolverDense &)=delete
- void solveLinearSystem (AmiVector &rhs) override

  Solves the linear system for the Newton step.

- void prepareLinearSystem (int ntry, int nnewt) override

  Writes the Jacobian for the Newton iteration and passes it to the linear solver.
- void prepareLinearSystemB (int ntry, int nnewt) override

Additional Inherited Members

18.23.1 Detailed Description

Definition at line 141 of file newton_solver.h.

18.23.2 Constructor & Destructor Documentation

18.23.2.1 NewtonSolverDense()

```
NewtonSolverDense (
    realtype * t,
    AmiVector * x,
    Model * model )
```

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the model object

Definition at line 139 of file newton_solver.cpp.

18.23.3 Member Function Documentation

18.23.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector & rhs ) [override], [virtual]
```

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implements NewtonSolver.
Definition at line 168 of file newton_solver.cpp.
Here is the call graph for this function:

18.23.3.2 prepareLinearSystem()

```
void prepareLinearSystem (
```

```
int ntry,
int nnewt )  [override], [virtual]
```

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 149 of file newton_solver.cpp.
Here is the call graph for this function:

18.23.3.3 prepareLinearSystemB()

```
void prepareLinearSystemB (
    int ntry,
    int nnewt )  [override], [virtual]
```

Writes the Jacobian (JB) 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 158 of file newton_solver.cpp.
Here is the call graph for this function:

18.24 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)

  Constructor, initializes all members with the provided objects.
- void solveLinearSystem (AmiVector &rhs) override

  Solves the linear system for the Newton step by passing it to linsolveSPBCG.
- void prepareLinearSystem (int ntry, int nnewt) override
- void prepareLinearSystemB (int ntry, int nnewt) override
- void linsolveSPBCG (int ntry, int nnewt, AmiVector &ns_delta)

Additional Inherited Members

18.24.1 Detailed Description

Definition at line 258 of file newton_solver.h.

18.24.2 Constructor & Destructor Documentation

18.24.2.1 NewtonSolverIterative()

```
NewtonSolverIterative (
    realtype * t,
    AmiVector * x,
    Model * model )
```

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the model object

Definition at line 242 of file newton_solver.cpp.

18.24.3 Member Function Documentation

18.24.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector & rhs ) [override], [virtual]
```

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implements NewtonSolver.
Definition at line 303 of file newton_solver.cpp.
Here is the call graph for this function:

18.24.3.2 prepareLinearSystem()

```
void prepareLinearSystem (
    int ntry,
    int nnewt ) [override], [virtual]
```

Writes the Jacobian (J) 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 254 of file newton_solver.cpp.

Here is the call graph for this function:

18.24.3.3 prepareLinearSystemB()

```
void prepareLinearSystemB (
    int ntry,
    int nnewt ) [override], [virtual]
```

Writes the Jacobian (JB) 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 277 of file newton_solver.cpp.

Here is the call graph for this function:

18.24.3.4 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 310 of file newton_solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.25 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)

  Constructor, initializes all members with the provided objects, initializes temporary storage objects and the klu solver.
- **NewtonSolverSparse** (const NewtonSolverSparse &)=delete
- void solveLinearSystem (AmiVector &rhs) override

  Solves the linear system for the Newton step.

- void prepareLinearSystem (int ntry, int nnewt) override

  Writes the Jacobian for the Newton iteration and passes it to the linear solver.
- void prepareLinearSystemB (int ntry, int nnewt) override

Additional Inherited Members

18.25.1 Detailed Description

Definition at line 200 of file newton_solver.h.

18.25.2 Constructor & Destructor Documentation

18.25.2.1 NewtonSolverSparse()

```
NewtonSolverSparse (
    realtype * t,
    AmiVector * x,
    Model * model )
```

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the model object

Definition at line 190 of file newton_solver.cpp.

18.25.3 Member Function Documentation

18.25.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector & rhs ) [override], [virtual]
```

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implements NewtonSolver.
Definition at line 221 of file newton_solver.cpp.
Here is the call graph for this function:

18.25.3.2 prepareLinearSystem()

```
void prepareLinearSystem (
```

```
int ntry,
int nnewt )  [override], [virtual]
```

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 201 of file newton_solver.cpp.
Here is the call graph for this function:

18.25.3.3 prepareLinearSystemB()

```
void prepareLinearSystemB (
int ntry,
int nnewt )  [override], [virtual]
```

Writes the Jacobian (JB) 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 211 of file newton_solver.cpp.
Here is the call graph for this function:

18.26 ReturnData Class Reference

Stores all data to be returned by amici::runAmiciSimulation.

`#include <rdata.h>`

Public Member Functions

- ReturnData ()=default

  default constructor
- ReturnData (std::vector< realtype > ts, int np, int nk, int nx, int nx_solver, int nxtrue, int nx_solver_reinit, int ny, int nytrue, int nz, int nztrue, int ne, int nJ, int nplist, int nmaxevent, int nt, int newton_maxsteps, int nw, std::vector< ParameterScaling > pscale, SecondOrderMode o2mode, SensitivityOrder sensi, Sensitivity↩ Method sensi_meth, RDataReporting rdrm)

  ReturnData.
- ReturnData (Solver const &solver, const Model &model)

  constructor that uses information from model and solver to appropriately initialize fields
- void processSimulationObjects (SteadystateProblem const ∗preeq, ForwardProblem const ∗fwd, BackwardProblem const ∗bwd, SteadystateProblem const ∗posteq, Model &model, Solver const &solver, ExpData const ∗edata)

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

Public Attributes

- std::vector< realtype > ts
- std::vector< realtype > xdot
- std::vector< realtype > J
- std::vector< realtype > w
- 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
- double cpu_time = 0.0
- double cpu_timeB = 0.0
- std::vector< SteadyStateStatus > preeq_status
- double preeq_cpu_time = 0.0

- double preeq_cpu_timeB = 0.0
- std::vector< SteadyStateStatus > posteq_status
- double posteq_cpu_time = 0.0
- double posteq_cpu_timeB = 0.0
- std::vector< int > preeq_numsteps
- std::vector< int > preeq_numlinsteps
- int preeq_numstepsB = 0
- std::vector< int > posteq_numsteps
- std::vector< int > posteq_numlinsteps
- int posteq_numstepsB = 0
- realtype preeq_t = NAN
- realtype preeq_wrms = NAN
- realtype posteq_t = NAN
- realtype posteq_wrms = NAN
- std::vector< realtype > x0
- std::vector< realtype > x_ss
- std::vector< realtype > sx0
- std::vector< realtype > sx_ss
- realtype llh = 0.0
- realtype chi2 = 0.0
- std::vector< realtype > sllh
- std::vector< realtype > s2llh
- int status = 0
- int np {0}
- int nk {0}
- int nx {0}
- int nx_solver {0}
- int nxtrue {0}
- int nx_solver_reinit {0}
- int ny {0}
- int nytrue {0}
- int nz {0}
- int nztrue {0}
- int ne {0}
- int nJ {0}
- int nplist {0}
- int nmaxevent {0}
- int nt {0}
- int nw {0}
- int newton_maxsteps {0}
- std::vector< ParameterScaling > pscale
- SecondOrderMode o2mode {SecondOrderMode::none}
- SensitivityOrder sensi {SensitivityOrder::none}
- SensitivityMethod sensi_meth {SensitivityMethod::none}
- RDataReporting rdata_reporting {RDataReporting::full}

Protected Member Functions

- void initializeLikelihoodReporting ()

  initializes storage for likelihood reporting mode
- void initializeResidualReporting ()

  initializes storage for residual reporting mode
- void initializeFullReporting ()

  initializes storage for full reporting mode

- void initializeObjectiveFunction ()

  initialize values for chi2 and llh and derivatives
- void processPreEquilibration (SteadystateProblem const &preeq, Model &model)

  extracts data from a preequilibration steadystateproblem
- void processPostEquilibration (SteadystateProblem const &posteq, Model &model, ExpData const ∗edata)

  extracts data from a preequilibration steadystateproblem
- void processForwardProblem (ForwardProblem const &fwd, Model &model, ExpData const ∗edata)

  extracts results from forward problem
- void processBackwardProblem (ForwardProblem const &fwd, BackwardProblem const &bwd, SteadystateProblem const ∗preeq, Model &model)

  extracts results from backward problem
- void processSolver (Solver const &solver)

  extracts results from solver
- template<class T >
  void storeJacobianAndDerivativeInReturnData (T const &problem, Model &model)

  Evaluates and stores the Jacobian and right hand side at final timepoint.
- void readSimulationState (SimulationState const &state, Model &model)

  sets member variables and model state according to provided simulation state
- void fres (int it, Model &model, const ExpData &edata)

  Residual function.
- void fchi2 (int it)

  Chi-squared function.
- void fsres (int it, Model &model, const ExpData &edata)

  Residual sensitivity function.
- void fFIM (int it, Model &model, const ExpData &edata)

  Fisher information matrix function.
- void invalidate (int it_start)

  Set likelihood, state variables, outputs and respective sensitivities to NaN (typically after integration failure)
- void invalidateLLH ()

  Set likelihood and chi2 to NaN (typically after integration failure)
- void invalidateSLLH ()

  Set likelihood sensitivities to NaN (typically after integration failure)
- void applyChainRuleFactorToSimulationResults (const Model &model)

  applies the chain rule to account for parameter transformation in the sensitivities of simulation results
- bool computingFSA () const

  Checks whether forward sensitivity analysis is performed.
- void getDataOutput (int it, Model &model, ExpData const ∗edata)

  Extracts output information for data-points, expects that x_solver and sx_solver were were set appropriately.
- void getDataSensisFSA (int it, Model &model, ExpData const ∗edata)

  Extracts data information for forward sensitivity analysis, expects that x_solver and sx_solver were were set appropriately.
- void getEventOutput (int iroot, realtype t, const std::vector< int > rootidx, Model &model, ExpData const ∗edata)

  Extracts output information for events, expects that x_solver and sx_solver were were set appropriately.
- void getEventSensisFSA (int iroot, int ie, realtype t, Model &model, ExpData const ∗edata)

  Extracts event information for forward sensitivity analysis, expects that x_solver and sx_solver were set appropriately.
- void handleSx0Backward (const Model &model, SteadystateProblem const &preeq, AmiVector &xQB) const

  Updates contribution to likelihood from quadratures (xQB), if preequilibration was run in adjoint mode.
- void handleSx0Forward (const Model &model, std::vector< realtype > &llhS0, AmiVector &xB) const

  Updates contribution to likelihood for inital state sensitivities (llhS0), if no preequilibration was run or if forward sensitivities were used.

Protected Attributes

- realtype t_
- AmiVector x_solver_
- AmiVector dx_solver_
- AmiVectorArray sx_solver_
- AmiVector x_rdata_
- AmiVectorArray sx_rdata_
- std::vector< int > nroots_

18.26.1 Detailed Description

NOTE: multidimensional arrays are stored in row-major order (FORTRAN-style)
Definition at line 36 of file rdata.h.

18.26.2 Constructor & Destructor Documentation

18.26.2.1 ReturnData() [1/2]

```
ReturnData (
    std::vector< realtype > ts,
    int np,
    int nk,
    int nx,
    int nx_solver,
    int nxtrue,
    int nx_solver_reinit,
    int ny,
    int nytrue,
    int nz,
    int nztrue,
    int ne,
    int nJ,
    int nplist,
    int nmaxevent,
    int nt,
    int newton_maxsteps,
    int nw,
    std::vector< ParameterScaling > pscale,
    SecondOrderMode o2mode,
    SensitivityOrder sensi,
    SensitivityMethod sensi_meth,
    RDataReporting rdrm )
```

Parameters

ts	see amici::Model::ts
np	see amici::Model::np
nk	see amici::Model::nk
nx	see amici::Model::nx_rdata
nx_solver	see amici::Model::nx_solver
nxtrue	see amici::Model::nxtrue_rdata
nx_solver_reinit	see amici::Model::nx_solver_reinit
ny	see amici::Model::ny
nytrue	see amici::Model::nytrue
nz	see amici::Model::nz
nztrue	see amici::Model::nztrue

Parameters

ne	see amici::Model::ne
nJ	see amici::Model::nJ
nplist	see amici::Model::nplist
nmaxevent	see amici::Model::nmaxevent
nt	see amici::Model::nt
newton_maxsteps	see amici::Solver::newton_maxsteps
nw	see amici::Model::nw
pscale	see amici::Model::pscale
o2mode	see amici::Model::o2mode
sensi	see amici::Solver::sensi
sensi_meth	see amici::Solver::sensi_meth
rdrm	see amici::Solver::rdata_reporting

Definition at line 27 of file rdata.cpp.

18.26.2.2 ReturnData() [2/2]

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

Parameters

solver	solver instance
model	model instance

Definition at line 17 of file rdata.cpp.

18.26.3 Member Function Documentation

18.26.3.1 processSimulationObjects()

```
void processSimulationObjects (
    SteadystateProblem const * preeq,
    ForwardProblem const * fwd,
    BackwardProblem const * bwd,
    SteadystateProblem const * posteq,
    Model & model,
    Solver const & solver,
    ExpData const * edata )
```

Parameters

preeq	simulated preequilibration problem, pass nullptr to ignore
fwd	simulated forward problem, pass nullptr to ignore
bwd	simulat ed backward problem, pass nullptr to ignore
posteq	simulated postequilibration problem, pass nullptr to ignore
model	matching model instance
solver	matching solver instance
edata	matching experimental data

Definition at line 146 of file rdata.cpp.

Here is the call graph for this function:

18.26.3.2 processPreEquilibration()

```
void processPreEquilibration (
    SteadystateProblem const & preeq,
    Model & model ) [protected]
```

Parameters

preeq	Steadystateproblem for preequilibration

Parameters

model	Model instance to compute return values

Definition at line 180 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.3 processPostEquilibration()

```
void processPostEquilibration (
    SteadystateProblem const & posteq,
    Model & model,
    ExpData const * edata ) [protected]
```

Parameters

posteq	Steadystateproblem for postequilibration
model	Model instance to compute return values
edata	ExpData instance containing observable data

Definition at line 209 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.4 processForwardProblem()

```
void processForwardProblem (
    ForwardProblem const & fwd,
    Model & model,
    ExpData const * edata ) [protected]
```

Parameters

fwd	forward problem
model	model that was used for forward simulation
edata	ExpData instance containing observable data

Definition at line 234 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.5 processBackwardProblem()

```
void processBackwardProblem (
    ForwardProblem const & fwd,
    BackwardProblem const & bwd,
    SteadystateProblem const * preeq,
    Model & model ) [protected]
```

Parameters

fwd	forward problem
bwd	backward problem
preeq	Steadystateproblem for preequilibration
model	model that was used for forward/backward simulation

Definition at line 402 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.6 processSolver()

```
void processSolver (
    Solver const & solver ) [protected]
```

Parameters

solver	solver that was used for forward/backward simulation

Definition at line 475 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.7 storeJacobianAndDerivativeInReturnData()

```
void storeJacobianAndDerivativeInReturnData (
    T const & problem,
    Model & model ) [protected]
```

Parameters

problem	forward problem or steadystate problem
model	model that was used for forward/backward simulation

Definition at line 524 of file rdata.h.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.8 readSimulationState()

```
void readSimulationState (
    SimulationState const & state,
    Model & model ) [protected]
```

Parameters

state	simulation state provided by Problem
model	model that was used for forward/backward simulation

Definition at line 500 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.9 fres()

```
void fres (
    int it,
    Model & model,
    const ExpData & edata )  [protected]
```

Parameters

it	time index
model	model that was used for forward/backward simulation
edata	ExpData instance containing observable data

Definition at line 737 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.10 fchi2()

```
void fchi2 (
    int it ) [protected]
```

Parameters

it	time index

Definition at line 757 of file rdata.cpp.
Here is the caller graph for this function:

18.26.3.11 fsres()

```
void fsres (
    int it,
    Model & model,
    const ExpData & edata ) [protected]
```

Parameters

it	time index
model	model that was used for forward/backward simulation
edata	ExpData instance containing observable data

Definition at line 767 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.12 fFIM()

```
void fFIM (
    int it,
    Model & model,
    const ExpData & edata ) [protected]
```

Parameters

it	time index
model	model that was used for forward/backward simulation
edata	ExpData instance containing observable data

Definition at line 794 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.13 invalidate() `void invalidate ( int it_start ) [protected]`

Parameters

it_start	time index at which to start invalidating

Definition at line 510 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.14 applyChainRuleFactorToSimulationResults()

```
void applyChainRuleFactorToSimulation↩
Results (
            const Model & model )  [protected]
```

Parameters

model	Model from which the ReturnData was obtained

Definition at line 542 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.15 computingFSA() `bool computingFSA ( ) const [protected]`

Returns

boolean indicator

Definition at line 613 of file rdata.h.
Here is the caller graph for this function:

18.26.3.16 getDataOutput() `void getDataOutput ( int it, Model & model, ExpData const * edata ) [protected]`

Parameters

it	timepoint index
model	model that was used in forward solve
edata	ExpData instance carrying experimental data

Definition at line 278 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.17 getDataSensisFSA()

```
void getDataSensisFSA (
    int it,
    Model & model,
    ExpData const * edata ) [protected]
```

Parameters

it	index of current timepoint
model	model that was used in forward solve
edata	ExpData instance carrying experimental data

Definition at line 312 of file rdata.cpp.

Here is the call graph for this function:

18.26.3.18 getEventOutput()

```
void getEventOutput (
    int iroot,
    realtype t,
    const std::vector< int > rootidx,
    Model & model,
    ExpData const * edata )  [protected]
```

Parameters

iroot	event index
t	event timepoint
rootidx	information about which roots fired (1 indicating fired, 0/-1 for not)
model	model that was used in forward solve
edata	ExpData instance carrying experimental data

Definition at line 335 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.19 getEventSensisFSA()

```
void getEventSensisFSA (
    int iroot,
    int ie,
    realtype t,
    Model & model,
    ExpData const * edata )  [protected]
```

Parameters

iroot	event index
ie	index of event type
t	event timepoint
model	model that was used in forward solve
edata	ExpData instance carrying experimental data

Definition at line 380 of file rdata.cpp.

Here is the call graph for this function:

18.26.3.20 handleSx0Backward()

```
void handleSx0Backward (
    const Model & model,
    SteadystateProblem const & preeq,
    AmiVector & xQB ) const [protected]
```

Parameters

model	model that was used for forward/backward simulation
preeq	Steadystateproblem for preequilibration
xQB	vector with quadratures from adjoint computation

Definition at line 432 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.3.21 handleSx0Forward()

```
void handleSx0Forward (
    const Model & model,
    std::vector< realtype > & llhS0,
    AmiVector & xB ) const [protected]
```

Parameters

model	model that was used for forward/backward simulation
llhS0	contribution to likelihood for initial state sensitivities
xB	vector with final adjoint state (exluding conservation laws)

Definition at line 447 of file rdata.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.26.4 Member Data Documentation

18.26.4.1 ts

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

timepoints (dimension: nt)
Definition at line 107 of file rdata.h.

18.26.4.2 xdot `std::vector<realtype> xdot`
time derivative (dimension: nx)
Definition at line 110 of file rdata.h.

18.26.4.3 J `std::vector<realtype> J`
Jacobian of differential equation right hand side (dimension: nx x nx, row-major)
Definition at line 116 of file rdata.h.

18.26.4.4 w `std::vector<realtype> w`
w data from the model (recurring terms in xdot, for imported SBML models from python, this contains the flux vector) (dimensions: nt x nw, row major)
Definition at line 123 of file rdata.h.

18.26.4.5 z `std::vector<realtype> z`
event output (dimension: nmaxevent x nz, row-major)
Definition at line 126 of file rdata.h.

18.26.4.6 sigmaz `std::vector<realtype> sigmaz`
event output sigma standard deviation (dimension: nmaxevent x nz, row-major)
Definition at line 132 of file rdata.h.

18.26.4.7 sz `std::vector<realtype> sz`
parameter derivative of event output (dimension: nmaxevent x nz, row-major)
Definition at line 138 of file rdata.h.

18.26.4.8 ssigmaz `std::vector<realtype> ssigmaz`
parameter derivative of event output standard deviation (dimension: nmaxevent x nz, row-major)
Definition at line 144 of file rdata.h.

18.26.4.9 rz `std::vector<realtype> rz`
event trigger output (dimension: nmaxevent x nz, row-major)
Definition at line 147 of file rdata.h.

18.26.4.10 srz `std::vector<realtype> srz`
parameter derivative of event trigger output (dimension: nmaxevent x nz x nplist, row-major)
Definition at line 153 of file rdata.h.

18.26.4.11 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 159 of file rdata.h.

18.26.4.12 x `std::vector<realtype> x`
state (dimension: nt x nx, row-major)
Definition at line 162 of file rdata.h.

18.26.4.13 sx `std::vector<realtype> sx`
parameter derivative of state (dimension: nt x nplist x nx, row-major)
Definition at line 167 of file rdata.h.

18.26.4.14 y `std::vector<realtype> y`
observable (dimension: nt x ny, row-major)
Definition at line 170 of file rdata.h.

18.26.4.15 sigmay `std::vector<realtype> sigmay`
observable standard deviation (dimension: nt x ny, row-major)
Definition at line 173 of file rdata.h.

18.26.4.16 sy `std::vector<realtype> sy`
parameter derivative of observable (dimension: nt x nplist x ny, row-major)
Definition at line 179 of file rdata.h.

18.26.4.17 ssigmay `std::vector<realtype> ssigmay`
parameter derivative of observable standard deviation (dimension: nt x nplist x ny, row-major)
Definition at line 185 of file rdata.h.

18.26.4.18 res `std::vector<realtype> res`
observable (dimension: nt∗ny, row-major)
Definition at line 188 of file rdata.h.

18.26.4.19 sres `std::vector<realtype> sres`
parameter derivative of residual (dimension: nt∗ny x nplist, row-major)
Definition at line 193 of file rdata.h.

18.26.4.20 FIM `std::vector<realtype> FIM`
fisher information matrix (dimension: nplist x nplist, row-major)
Definition at line 196 of file rdata.h.

18.26.4.21 numsteps `std::vector<int> numsteps`
number of integration steps forward problem (dimension: nt)
Definition at line 199 of file rdata.h.

18.26.4.22 numstepsB `std::vector<int> numstepsB`
number of integration steps backward problem (dimension: nt)
Definition at line 202 of file rdata.h.

18.26.4.23 numrhsevals `std::vector<int> numrhsevals`
number of right hand side evaluations forward problem (dimension: nt)
Definition at line 205 of file rdata.h.

18.26.4.24 numrhsevalsB `std::vector<int> numrhsevalsB`
number of right hand side evaluations backwad problem (dimension: nt)
Definition at line 208 of file rdata.h.

18.26.4.25 numerrtestfails `std::vector<int> numerrtestfails`
number of error test failures forward problem (dimension: nt)
Definition at line 211 of file rdata.h.

18.26.4.26 numerrtestfailsB `std::vector<int> numerrtestfailsB`
number of error test failures backwad problem (dimension: nt)
Definition at line 214 of file rdata.h.

18.26.4.27 numnonlinsolvconvfails `std::vector<int> numnonlinsolvconvfails`
number of linear solver convergence failures forward problem (dimension: nt)
Definition at line 219 of file rdata.h.

18.26.4.28 numnonlinsolvconvfailsB `std::vector<int> numnonlinsolvconvfailsB`
number of linear solver convergence failures backwad problem (dimension: nt)
Definition at line 224 of file rdata.h.

18.26.4.29 order `std::vector<int> order`
employed order forward problem (dimension: nt)
Definition at line 227 of file rdata.h.

18.26.4.30 cpu_time `double cpu_time = 0.0`
computation time of forward solve [ms]
Definition at line 230 of file rdata.h.

18.26.4.31 cpu_timeB `double cpu_timeB = 0.0`
computation time of backward solve [ms]
Definition at line 233 of file rdata.h.

18.26.4.32 preeq_status `std::vector<SteadyStateStatus> preeq_status`
flags indicating success of steady state solver (preequilibration)
Definition at line 236 of file rdata.h.

18.26.4.33 preeq_cpu_time `double preeq_cpu_time = 0.0`
computation time of the steady state solver [ms] (preequilibration)
Definition at line 239 of file rdata.h.

18.26.4.34 preeq_cpu_timeB `double preeq_cpu_timeB = 0.0`
computation time of the steady state solver of the backward problem [ms] (preequilibration)
Definition at line 243 of file rdata.h.

18.26.4.35 posteq_status `std::vector<SteadyStateStatus> posteq_status`

flags indicating success of steady state solver (postequilibration)
Definition at line 246 of file rdata.h.

18.26.4.36 posteq_cpu_time `double posteq_cpu_time = 0.0`

computation time of the steady state solver [ms] (postequilibration)
Definition at line 249 of file rdata.h.

18.26.4.37 posteq_cpu_timeB `double posteq_cpu_timeB = 0.0`

computation time of the steady state solver of the backward problem [ms] (postequilibration)
Definition at line 253 of file rdata.h.

18.26.4.38 preeq_numsteps `std::vector<int> preeq_numsteps`

number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3)
Definition at line 259 of file rdata.h.

18.26.4.39 preeq_numlinsteps `std::vector<int> preeq_numlinsteps`

number of linear steps by Newton step for steady state problem. this will only be filled for iterative solvers (preequilibration) (length = newton_maxsteps $*$ 2)
Definition at line 266 of file rdata.h.

18.26.4.40 preeq_numstepsB `int preeq_numstepsB = 0`

number of simulation steps for adjoint steady state problem (preequilibration) [== 0 if analytical solution worked, $>$ 0 otherwise]
Definition at line 272 of file rdata.h.

18.26.4.41 posteq_numsteps `std::vector<int> posteq_numsteps`

number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3) (postequilibration)
Definition at line 278 of file rdata.h.

18.26.4.42 posteq_numlinsteps `std::vector<int> posteq_numlinsteps`

number of linear steps by Newton step for steady state problem. this will only be filled for iterative solvers (postequilibration) (length = newton_maxsteps $*$ 2)
Definition at line 285 of file rdata.h.

18.26.4.43 posteq_numstepsB `int posteq_numstepsB = 0`

number of simulation steps for adjoint steady state problem (postequilibration) [== 0 if analytical solution worked, $>$ 0 otherwise]
Definition at line 291 of file rdata.h.

18.26.4.44 preeq_t `realtype preeq_t = NAN`

time when steadystate was reached via simulation (preequilibration)
Definition at line 296 of file rdata.h.

18.26.4.45 preeq_wrms `realtype preeq_wrms = NAN`
weighted root-mean-square of the rhs when steadystate was reached (preequilibration)
Definition at line 302 of file rdata.h.

18.26.4.46 posteq_t `realtype posteq_t = NAN`
time when steadystate was reached via simulation (postequilibration)
Definition at line 307 of file rdata.h.

18.26.4.47 posteq_wrms `realtype posteq_wrms = NAN`
weighted root-mean-square of the rhs when steadystate was reached (postequilibration)
Definition at line 313 of file rdata.h.

18.26.4.48 x0 `std::vector<realtype> x0`
initial state (dimension: nx)
Definition at line 316 of file rdata.h.

18.26.4.49 x_ss `std::vector<realtype> x_ss`
preequilibration steady state found by Newton solver (dimension: nx)
Definition at line 319 of file rdata.h.

18.26.4.50 sx0 `std::vector<realtype> sx0`
initial sensitivities (dimension: nplist x nx, row-major)
Definition at line 322 of file rdata.h.

18.26.4.51 sx_ss `std::vector<realtype> sx_ss`
preequilibration sensitivities found by Newton solver (dimension: nplist x nx, row-major)
Definition at line 328 of file rdata.h.

18.26.4.52 llh `realtype llh = 0.0`
loglikelihood value
Definition at line 331 of file rdata.h.

18.26.4.53 chi2 `realtype chi2 = 0.0`
chi2 value
Definition at line 334 of file rdata.h.

18.26.4.54 sllh `std::vector<realtype> sllh`
parameter derivative of loglikelihood (dimension: nplist)
Definition at line 337 of file rdata.h.

18.26.4.55 s2llh `std::vector<realtype> s2llh`
second order parameter derivative of loglikelihood (dimension: (nJ-1) x nplist, row-major)
Definition at line 343 of file rdata.h.

18.26.4.56 status `int status = 0`
status code
Definition at line 346 of file rdata.h.

18.26.4.57 np `int np {0}`
total number of model parameters
Definition at line 349 of file rdata.h.

18.26.4.58 nk `int nk {0}`
number of fixed parameters
Definition at line 352 of file rdata.h.

18.26.4.59 nx `int nx {0}`
number of states
Definition at line 355 of file rdata.h.

18.26.4.60 nx_solver `int nx_solver {0}`
number of states with conservation laws applied
Definition at line 358 of file rdata.h.

18.26.4.61 nxtrue `int nxtrue {0}`
number of states in the unaugmented system
Definition at line 361 of file rdata.h.

18.26.4.62 nx_solver_reinit `int nx_solver_reinit {0}`
number of solver states to be reinitilized after preequilibration
Definition at line 364 of file rdata.h.

18.26.4.63 ny `int ny {0}`
number of observables
Definition at line 367 of file rdata.h.

18.26.4.64 nytrue `int nytrue {0}`
number of observables in the unaugmented system
Definition at line 370 of file rdata.h.

18.26.4.65 nz `int nz {0}`
number of event outputs
Definition at line 373 of file rdata.h.

18.26.4.66 nztrue `int nztrue {0}`
number of event outputs in the unaugmented system
Definition at line 376 of file rdata.h.

18.26.4.67 ne `int ne {0}`
number of events
Definition at line 379 of file rdata.h.

18.26.4.68 nJ `int nJ {0}`
dimension of the augmented objective function for 2nd order ASA
Definition at line 382 of file rdata.h.

18.26.4.69 nplist `int nplist {0}`
number of parameter for which sensitivities were requested
Definition at line 385 of file rdata.h.

18.26.4.70 nmaxevent `int nmaxevent {0}`
maximal number of occuring events (for every event type)
Definition at line 388 of file rdata.h.

18.26.4.71 nt `int nt {0}`
number of considered timepoints
Definition at line 391 of file rdata.h.

18.26.4.72 nw `int nw {0}`
number of columns in w
Definition at line 394 of file rdata.h.

18.26.4.73 newton_maxsteps `int newton_maxsteps {0}`
maximal number of newton iterations for steady state calculation
Definition at line 397 of file rdata.h.

18.26.4.74 pscale `std::vector<ParameterScaling> pscale`
scaling of parameterization (lin,log,log10)
Definition at line 400 of file rdata.h.

18.26.4.75 o2mode `SecondOrderMode o2mode {SecondOrderMode::none}`
flag indicating whether second order sensitivities were requested
Definition at line 403 of file rdata.h.

18.26.4.76 sensi `SensitivityOrder sensi {SensitivityOrder::none}`
sensitivity order
Definition at line 406 of file rdata.h.

18.26.4.77 sensi_meth `SensitivityMethod sensi_meth {SensitivityMethod::none}`
sensitivity method
Definition at line 409 of file rdata.h.

18.26.4.78 rdata_reporting `RDataReporting rdata_reporting {RDataReporting::full}`

reporting mode

Definition at line 412 of file rdata.h.

18.26.4.79 t_ `realtype t_ [protected]`

timepoint for model evaluation

Definition at line 427 of file rdata.h.

18.26.4.80 x_solver_ `AmiVector x_solver_ [protected]`

partial state vector, excluding states eliminated from conservation laws

Definition at line 430 of file rdata.h.

18.26.4.81 dx_solver_ `AmiVector dx_solver_ [protected]`

partial time derivative of state vector, excluding states eliminated from conservation laws

Definition at line 433 of file rdata.h.

18.26.4.82 sx_solver_ `AmiVectorArray sx_solver_ [protected]`

partial sensitivity state vector array, excluding states eliminated from conservation laws

Definition at line 437 of file rdata.h.

18.26.4.83 x_rdata_ `AmiVector x_rdata_ [protected]`

full state vector, including states eliminated from conservation laws

Definition at line 440 of file rdata.h.

18.26.4.84 sx_rdata_ `AmiVectorArray sx_rdata_ [protected]`

full sensitivity state vector array, including states eliminated from conservation laws

Definition at line 444 of file rdata.h.

18.26.4.85 nroots_ `std::vector<int> nroots_ [protected]`

array of number of found roots for a certain event type (dimension: ne)

Definition at line 448 of file rdata.h.

18.27 SetupFailure Class Reference

Setup failure exception.

`#include <exception.h>`

Inheritance diagram for SetupFailure:

Public Member Functions

- SetupFailure (const char *msg)

  Constructor, simply calls AmiException constructor.

18.27.1 Detailed Description

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
Definition at line 132 of file exception.h.

18.27.2 Constructor & Destructor Documentation

18.27.2.1 SetupFailure() `SetupFailure ( const char * msg ) [explicit]`

Parameters

msg	

Definition at line 138 of file exception.h.

18.28 SimulationState Struct Reference

implements an exchange format to store and transfer the state of a simulation at a specific timepoint.
`#include <forwardproblem.h>`

Public Attributes

- realtype t
- AmiVector x
- AmiVector dx
- AmiVectorArray sx
- ModelState state

18.28.1 Detailed Description

Definition at line 25 of file forwardproblem.h.

18.28.2 Member Data Documentation

18.28.2.1 t `realtype t`

timepoint

Definition at line 27 of file forwardproblem.h.

18.28.2.2 x `AmiVector x`

state variables

Definition at line 29 of file forwardproblem.h.

18.28.2.3 dx `AmiVector dx`

state variables

Definition at line 31 of file forwardproblem.h.

18.28.2.4 sx `AmiVectorArray sx`

state variable sensitivity

Definition at line 33 of file forwardproblem.h.

18.28.2.5 state `ModelState state`

state of the model that was used for simulation

Definition at line 35 of file forwardproblem.h.

18.29 Solver Class Reference

`#include <solver.h>`

Public Member Functions

- Solver (AmiciApplication ∗app)

  Constructor.
- Solver (const Solver &other)

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

  Clone this instance.
- int run (realtype tout) const

  runs a forward simulation until the specified timepoint
- int step (realtype tout) const

  makes a single step in the simulation
- void runB (realtype tout) const

  runs a backward simulation until the specified timepoint
- void setup (realtype t0, Model ∗model, const AmiVector &x0, const AmiVector &dx0, const AmiVectorArray &sx0, const AmiVectorArray &sdx0) const

  Initialises the ami memory object and applies specified options.
- void setupB (int ∗which, realtype tf, Model ∗model, const AmiVector &xB0, const AmiVector &dxB0, const AmiVector &xQB0) const

Initialises the AMI memory object for the backwards problem.

- void setupSteadystate (const realtype t0, Model ∗model, const AmiVector &x0, const AmiVector &dx0, const AmiVector &xB0, const AmiVector &dxB0, const AmiVector &xQ0) const

  Initialises the ami memory for quadrature computation.
- void updateAndReinitStatesAndSensitivities (Model ∗model)

  Reinitializes state and respective sensitivities (if necessary) according to changes in fixedParameters.
- virtual void getRootInfo (int ∗rootsfound) const =0
- virtual void calcIC (realtype tout1) const =0

  Calculates consistent initial conditions, assumes initial states to be correct (DAE only)
- virtual void calcICB (int which, realtype tout1) const =0

  Calculates consistent initial conditions for the backwards problem, assumes initial states to be correct (DAE only)
- virtual void solveB (realtype tBout, int itaskB) const =0

  Solves the backward problem until a predefined timepoint (adjoint only)
- virtual void turnOffRootFinding () const =0

  Disable rootfinding.
- SensitivityMethod getSensitivityMethod () const

  Return current sensitivity method.
- void setSensitivityMethod (SensitivityMethod sensi_meth)

  Set sensitivity method.
- SensitivityMethod getSensitivityMethodPreequilibration () const

  Return current sensitivity method during preequilibration.
- void setSensitivityMethodPreequilibration (SensitivityMethod sensi_meth_preeq)

  Set sensitivity method for preequilibration.
- void switchForwardSensisOff () const

  Disable forward sensitivity integration (used in steady state sim)
- int getNewtonMaxSteps () const

  Get maximum number of allowed Newton steps for steady state computation.
- void setNewtonMaxSteps (int newton_maxsteps)

  Set maximum number of allowed Newton steps for steady state computation.
- bool getPreequilibration () const

  Get if model preequilibration is enabled.
- void setPreequilibration (bool require_preequilibration)

  Enable/disable model preequilibration.
- int getNewtonMaxLinearSteps () const

  Get maximum number of allowed linear steps per Newton step for steady state computation.
- void setNewtonMaxLinearSteps (int newton_maxlinsteps)

  Set maximum number of allowed linear steps per Newton step for steady state computation.
- NewtonDampingFactorMode getNewtonDampingFactorMode () const

  Get a state of the damping factor used in the Newton solver.
- void setNewtonDampingFactorMode (NewtonDampingFactorMode dampingFactorMode)

  Turn on/off a damping factor in the Newton method.
- double getNewtonDampingFactorLowerBound () const

  Get a lower bound of the damping factor used in the Newton solver.
- void setNewtonDampingFactorLowerBound (double dampingFactorLowerBound)

  Set a lower bound of the damping factor in the Newton solver.
- SensitivityOrder getSensitivityOrder () const

  Get sensitvity order.
- void setSensitivityOrder (SensitivityOrder sensi)

  Set the sensitvity order.
- double getRelativeTolerance () const

  Get the relative tolerances for the forward problem.

- void setRelativeTolerance (double rtol)

  Sets the relative tolerances for the forward problem.
- double getAbsoluteTolerance () const

  Get the absolute tolerances for the forward problem.
- void setAbsoluteTolerance (double atol)

  Sets the absolute tolerances for the forward problem.
- double getRelativeToleranceFSA () const

  Returns the relative tolerances for the forward sensitivity problem.
- void setRelativeToleranceFSA (double rtol)

  Sets the relative tolerances for the forward sensitivity problem.
- double getAbsoluteToleranceFSA () const

  Returns the absolute tolerances for the forward sensitivity problem.
- void setAbsoluteToleranceFSA (double atol)

  Sets the absolute tolerances for the forward sensitivity problem.
- double getRelativeToleranceB () const

  Returns the relative tolerances for the adjoint sensitivity problem.
- void setRelativeToleranceB (double rtol)

  Sets the relative tolerances for the adjoint sensitivity problem.
- double getAbsoluteToleranceB () const

  Returns the absolute tolerances for the backward problem for adjoint sensitivity analysis.
- void setAbsoluteToleranceB (double atol)

  Sets the absolute tolerances for the backward problem for adjoint sensitivity analysis.
- double getRelativeToleranceQuadratures () const

  Returns the relative tolerance for the quadrature problem.
- void setRelativeToleranceQuadratures (double rtol)

  sets the relative tolerance for the quadrature problem
- double getAbsoluteToleranceQuadratures () const

  returns the absolute tolerance for the quadrature problem
- void setAbsoluteToleranceQuadratures (double atol)

  sets the absolute tolerance for the quadrature problem
- double getRelativeToleranceSteadyState () const

  returns the relative tolerance for the steady state problem
- void setRelativeToleranceSteadyState (double rtol)

  sets the relative tolerance for the steady state problem
- double getAbsoluteToleranceSteadyState () const

  returns the absolute tolerance for the steady state problem
- void setAbsoluteToleranceSteadyState (double atol)

  sets the absolute tolerance for the steady state problem
- double getRelativeToleranceSteadyStateSensi () const

  returns the relative tolerance for the sensitivities of the steady state problem
- void setRelativeToleranceSteadyStateSensi (double rtol)

  sets the relative tolerance for the sensitivities of the steady state problem
- double getAbsoluteToleranceSteadyStateSensi () const

  returns the absolute tolerance for the sensitivities of the steady state problem
- void setAbsoluteToleranceSteadyStateSensi (double atol)

  sets the absolute tolerance for the sensitivities of the steady state problem
- long int getMaxSteps () const

  returns the maximum number of solver steps for the forward problem
- void setMaxSteps (long int maxsteps)

  sets the maximum number of solver steps for the forward problem
- long int getMaxStepsBackwardProblem () const

returns the maximum number of solver steps for the backward problem

- void setMaxStepsBackwardProblem (long int maxsteps)

  sets the maximum number of solver steps for the backward problem
- LinearMultistepMethod getLinearMultistepMethod () const

  returns the linear system multistep method
- void setLinearMultistepMethod (LinearMultistepMethod lmm)

  sets the linear system multistep method
- NonlinearSolverIteration getNonlinearSolverIteration () const

  returns the nonlinear system solution method
- void setNonlinearSolverIteration (NonlinearSolverIteration iter)

  sets the nonlinear system solution method
- InterpolationType getInterpolationType () const

  getInterpolationType
- void setInterpolationType (InterpolationType interpType)

  sets the interpolation of the forward solution that is used for the backwards problem
- int getStateOrdering () const

  Gets KLU / SuperLUMT state ordering mode.
- void setStateOrdering (int ordering)

  Sets KLU / SuperLUMT state ordering mode.
- bool getStabilityLimitFlag () const

  returns stability limit detection mode
- void setStabilityLimitFlag (bool stldet)

  set stability limit detection mode
- LinearSolver getLinearSolver () const

  getLinearSolver
- void setLinearSolver (LinearSolver linsol)

  setLinearSolver
- InternalSensitivityMethod getInternalSensitivityMethod () const

  returns the internal sensitivity method
- void setInternalSensitivityMethod (InternalSensitivityMethod ism)

  sets the internal sensitivity method
- RDataReporting getReturnDataReportingMode () const

  returns the ReturnData reporting mode
- void setReturnDataReportingMode (RDataReporting rdrm)

  sets the ReturnData reporting mode
- void writeSolution (realtype ∗t, AmiVector &x, AmiVector &dx, AmiVectorArray &sx, AmiVector &xQ) const

  write solution from forward simulation
- void writeSolutionB (realtype ∗t, AmiVector &xB, AmiVector &dxB, AmiVector &xQB, int which) const

  write solution from forward simulation
- const AmiVector & getState (realtype t) const

  Access state solution at time t.
- const AmiVector & getDerivativeState (realtype t) const

  Access derivative state solution at time t.
- const AmiVectorArray & getStateSensitivity (realtype t) const

  Access state sensitivity solution at time t.
- const AmiVector & getAdjointState (int which, realtype t) const

  Access adjoint solution at time t.
- const AmiVector & getAdjointDerivativeState (int which, realtype t) const

  Access adjoint derivative solution at time t.
- const AmiVector & getAdjointQuadrature (int which, realtype t) const

  Access adjoint quadrature solution at time t.

- const AmiVector & getQuadrature (realtype t) const

  Access quadrature solution at time t.
- virtual void reInit (realtype t0, const AmiVector &yy0, const AmiVector &yp0) const =0

  Reinitializes the states in the solver after an event occurence.
- virtual void sensReInit (const AmiVectorArray &yyS0, const AmiVectorArray &ypS0) const =0

  Reinitializes the state sensitivites in the solver after an event occurence.
- virtual void sensToggleOff () const =0

  Switches off computation of state sensitivites without deallocating the memory for sensitivities.
- virtual void reInitB (int which, realtype tB0, const AmiVector &yyB0, const AmiVector &ypB0) const =0

  Reinitializes the adjoint states after an event occurence.
- virtual void quadReInitB (int which, const AmiVector &yQB0) const =0

  Reinitialize the adjoint states after an event occurence.
- realtype gett () const

  current solver timepoint
- realtype getCpuTime () const

  Reads out the cpu time needed for forward solve.
- realtype getCpuTimeB () const

  Reads out the cpu time needed for bavkward solve.
- int nx () const

  number of states with which the solver was initialized
- int nplist () const

  number of parameters with which the solver was initialized
- int nquad () const

  number of quadratures with which the solver was initialized
- bool computingFSA () const

  check if FSA is being computed
- bool computingASA () const

  check if ASA is being computed
- void resetDiagnosis () const

  Resets vectors containing diagnosis information.
- void storeDiagnosis () const

  Stores diagnosis information from solver memory block for forward problem.
- void storeDiagnosisB (int which) const

  Stores diagnosis information from solver memory block for backward problem.
- std::vector< int > const & getNumSteps () const

  Accessor ns.
- std::vector< int > const & getNumStepsB () const

  Accessor nsB.
- std::vector< int > const & getNumRhsEvals () const

  Accessor nrhs.
- std::vector< int > const & getNumRhsEvalsB () const

  Accessor nrhsB.
- std::vector< int > const & getNumErrTestFails () const

  Accessor netf.
- std::vector< int > const & getNumErrTestFailsB () const

  Accessor netfB.
- std::vector< int > const & getNumNonlinSolvConvFails () const

  Accessor nnlscf.
- std::vector< int > const & getNumNonlinSolvConvFailsB () const

  Accessor nnlscfB.
- std::vector< int > const & getLastOrder () const

  Accessor order.

Public Attributes

- AmiciApplication ∗ app = &defaultContext

Protected Member Functions

- virtual void setStopTime (realtype tstop) const =0

  Sets a timepoint at which the simulation will be stopped.
- virtual int solve (realtype tout, int itask) const =0

  Solves the forward problem until a predefined timepoint.
- virtual int solveF (realtype tout, int itask, int ∗ncheckPtr) const =0

  Solves the forward problem until a predefined timepoint (adjoint only)
- virtual void reInitPostProcessF (realtype tnext) const =0

  reInitPostProcessF postprocessing of the solver memory after a discontinuity in the forward problem
- virtual void reInitPostProcessB (realtype tnext) const =0

  reInitPostProcessB postprocessing of the solver memory after a discontinuity in the backward problem
- virtual void getSens () const =0

  extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable
- virtual void getB (int which) const =0

  extracts the adjoint state at the current timepoint from solver memory and writes it to the xB member variable
- virtual void getQuadB (int which) const =0

  extracts the adjoint quadrature state at the current timepoint from solver memory and writes it to the xQB member variable
- virtual void getQuad (realtype &t) const =0

  extracts the quadrature at the current timepoint from solver memory and writes it to the xQ member variable
- virtual void init (realtype t0, const AmiVector &x0, const AmiVector &dx0) const =0

  Initialises the states at the specified initial timepoint.
- virtual void initSteadystate (realtype t0, const AmiVector &x0, const AmiVector &dx0) const =0

  Initialises the states at the specified initial timepoint.
- virtual void sensInit1 (const AmiVectorArray &sx0, const AmiVectorArray &sdx0) const =0

  initialises the forward sensitivities
- virtual void binit (int which, realtype tf, const AmiVector &xB0, const AmiVector &dxB0) const =0

  Initialise the adjoint states at the specified final timepoint.
- virtual void qbinit (int which, const AmiVector &xQB0) const =0

  Initialise the quadrature states at the specified final timepoint.
- virtual void rootInit (int ne) const =0

  Initialises the rootfinding for events.
- void initializeNonLinearSolverSens (const Model ∗model) const

  Initalize non-linear solver for sensitivities.
- virtual void setDenseJacFn () const =0

  Set the dense Jacobian function.
- virtual void setSparseJacFn () const =0

  sets the sparse Jacobian function
- virtual void setBandJacFn () const =0

  sets the banded Jacobian function
- virtual void setJacTimesVecFn () const =0

  sets the Jacobian vector multiplication function
- virtual void setDenseJacFnB (int which) const =0

  sets the dense Jacobian function
- virtual void setSparseJacFnB (int which) const =0

  sets the sparse Jacobian function
- virtual void setBandJacFnB (int which) const =0

sets the banded Jacobian function

- virtual void setJacTimesVecFnB (int which) const =0

  sets the Jacobian vector multiplication function

- virtual void setSparseJacFn_ss () const =0

  sets the sparse Jacobian function for backward steady state case

- virtual void allocateSolver () const =0

  Create specifies solver method and initializes solver memory for the forward problem.

- virtual void setSStolerances (double rtol, double atol) const =0

  sets scalar relative and absolute tolerances for the forward problem

- virtual void setSensSStolerances (double rtol, const double ∗atol) const =0

  activates sets scalar relative and absolute tolerances for the sensitivity variables

- virtual void setSensErrCon (bool error_corr) const =0
- virtual void setQuadErrConB (int which, bool flag) const =0

  Specifies whether error control is also enforced for the backward quadrature problem.

- virtual void setQuadErrCon (bool flag) const =0

  Specifies whether error control is also enforced for the forward quadrature problem.

- virtual void setErrHandlerFn () const =0

  Attaches the error handler function (errMsgIdAndTxt) to the solver.

- virtual void setUserData (Model ∗model) const =0

  Attaches the user data instance (here this is a Model) to the forward problem.

- virtual void setUserDataB (int which, Model ∗model) const =0

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

- virtual void setMaxNumSteps (long int mxsteps) const =0

  specifies the maximum number of steps for the forward problem

- virtual void setMaxNumStepsB (int which, long int mxstepsB) const =0

  specifies the maximum number of steps for the forward problem

- virtual void setStabLimDet (int stldet) const =0

  activates stability limit detection for the forward problem

- virtual void setStabLimDetB (int which, int stldet) const =0

  activates stability limit detection for the backward problem

- virtual void setId (const Model ∗model) const =0

  specify algebraic/differential components (DAE only)

- virtual void setSuppressAlg (bool flag) const =0

  deactivates error control for algebraic components (DAE only)

- virtual void setSensParams (const realtype ∗p, const realtype ∗pbar, const int ∗plist) const =0

  specifies the scaling and indexes for sensitivity computation

- virtual void getDky (realtype t, int k) const =0

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

- virtual void getDkyB (realtype t, int k, int which) const =0

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

- virtual void getSensDky (realtype t, int k) const =0

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

- virtual void getQuadDkyB (realtype t, int k, int which) const =0

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

- virtual void getQuadDky (realtype t, int k) const =0

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

- virtual void adjInit () const =0

  initializes the adjoint problem

- virtual void quadInit (const AmiVector &xQ0) const =0

  initializes the quadratures

- virtual void allocateSolverB (int ∗which) const =0

Specifies solver method and initializes solver memory for the backward problem.

- virtual void setSStolerancesB (int which, realtype relTolB, realtype absTolB) const =0

  sets relative and absolute tolerances for the backward problem
- virtual void quadSStolerancesB (int which, realtype reltolQB, realtype abstolQB) const =0

  sets relative and absolute tolerances for the quadrature backward problem
- virtual void quadSStolerances (realtype reltolQB, realtype abstolQB) const =0

  sets relative and absolute tolerances for the quadrature problem
- virtual void getNumSteps (const void ∗ami_mem, long int ∗numsteps) const =0

  reports the number of solver steps
- virtual void getNumRhsEvals (const void ∗ami_mem, long int ∗numrhsevals) const =0

  reports the number of right hand evaluations
- virtual void getNumErrTestFails (const void ∗ami_mem, long int ∗numerrtestfails) const =0

  reports the number of local error test failures
- virtual void getNumNonlinSolvConvFails (const void ∗ami_mem, long int ∗numnonlinsolvconvfails) const =0

  reports the number of nonlinear convergence failures
- virtual void getLastOrder (const void ∗ami_mem, int ∗order) const =0

  Reports the order of the integration method during the last internal step.
- void initializeLinearSolver (const Model ∗model) const

  Initializes and sets the linear solver for the forward problem.
- void initializeNonLinearSolver () const

  Sets the non-linear solver.
- virtual void setLinearSolver () const =0

  Sets the linear solver for the forward problem.
- virtual void setLinearSolverB (int which) const =0

  Sets the linear solver for the backward problem.
- virtual void setNonLinearSolver () const =0

  Set the non-linear solver for the forward problem.
- virtual void setNonLinearSolverB (int which) const =0

  Set the non-linear solver for the backward problem.
- virtual void setNonLinearSolverSens () const =0

  Set the non-linear solver for sensitivities.
- void initializeLinearSolverB (const Model ∗model, int which) const

  Initializes the linear solver for the backward problem.
- void initializeNonLinearSolverB (int which) const

  Initializes the non-linear solver for the backward problem.
- virtual const Model ∗ getModel () const =0
- bool getInitDone () const

  checks whether memory for the forward problem has been allocated
- bool getSensInitDone () const

  checks whether memory for forward sensitivities has been allocated
- bool getAdjInitDone () const

  checks whether memory for forward interpolation has been allocated
- bool getInitDoneB (int which) const

  checks whether memory for the backward problem has been allocated
- bool getQuadInitDoneB (int which) const

  checks whether memory for backward quadratures has been allocated
- bool getQuadInitDone () const

  checks whether memory for quadratures has been allocated
- virtual void diag () const =0

  attaches a diagonal linear solver to the forward problem
- virtual void diagB (int which) const =0

attaches a diagonal linear solver to the backward problem

- void resetMutableMemory (int nx, int nplist, int nquad) const

  resets solverMemory and solverMemoryB
- virtual void ∗ getAdjBmem (void ∗ami_mem, int which) const =0

  Retrieves the solver memory instance for the backward problem.
- void applyTolerances () const

  updates solver tolerances according to the currently specified member variables
- void applyTolerancesFSA () const

  updates FSA solver tolerances according to the currently specified member variables
- void applyTolerancesASA (int which) const

  updates ASA solver tolerances according to the currently specified member variables
- void applyQuadTolerancesASA (int which) const

  updates ASA quadrature solver tolerances according to the currently specified member variables
- void applyQuadTolerances () const

  updates quadrature solver tolerances according to the currently specified member variables
- void applySensitivityTolerances () const

  updates all senstivivity solver tolerances according to the currently specified member variables
- void setInitDone () const

  sets that memory for the forward problem has been allocated
- void setSensInitDone () const

  sets that memory for forward sensitivities has been allocated
- void setSensInitOff () const

  sets that memory for forward sensitivities has not been allocated
- void setAdjInitDone () const

  sets that memory for forward interpolation has been allocated
- void setInitDoneB (int which) const

  sets that memory for the backward problem has been allocated
- void setQuadInitDoneB (int which) const

  sets that memory for backward quadratures has been allocated
- void setQuadInitDone () const

  sets that memory for quadratures has been allocated
- void checkSensitivityMethod (const SensitivityMethod sensi_meth, bool preequilibration) const

  Sets sensitivity method (for simulation or preequilibration)

Protected Attributes

- std::unique_ptr< void, std::function< void(void ∗)> > solver_memory_
- std::vector< std::unique_ptr< void, std::function< void(void ∗)> > > solver_memory_B_
- InternalSensitivityMethod ism_ {InternalSensitivityMethod::simultaneous}
- LinearMultistepMethod lmm_ {LinearMultistepMethod::BDF}
- NonlinearSolverIteration iter_ {NonlinearSolverIteration::newton}
- InterpolationType interp_type_ {InterpolationType::hermite}
- long int maxsteps_ {10000}
- std::unique_ptr< SUNLinSolWrapper > linear_solver_
- std::unique_ptr< SUNLinSolWrapper > linear_solver_B_
- std::unique_ptr< SUNNonLinSolWrapper > non_linear_solver_
- std::unique_ptr< SUNNonLinSolWrapper > non_linear_solver_B_
- std::unique_ptr< SUNNonLinSolWrapper > non_linear_solver_sens_
- bool solver_was_called_F_ {false}
- bool solver_was_called_B_ {false}
- AmiVector x_ {0}
- AmiVector dky_ {0}

- AmiVector dx_ {0}
- AmiVectorArray sx_ {0, 0}
- AmiVectorArray sdx_ {0, 0}
- AmiVector xB_ {0}
- AmiVector dxB_ {0}
- AmiVector xQB_ {0}
- AmiVector xQ_ {0}
- realtype t_ {std::nan("")}
- bool force_reinit_postprocess_F_ {false}
- bool force_reinit_postprocess_B_ {false}

18.29.1 Detailed Description

The Solver class provides a generic interface to CVODES and IDAS solvers, individual realizations are realized in the CVodeSolver and the IDASolver class. All transient private/protected members (CVODES/IDAS memory, interface variables and status flags) are specified as mutable and not included in serialization or equality checks. No solver setting parameter should be marked mutable.
NOTE: Any changes in data members here must be propagated to copy ctor, equality operator, serialization functions in serialization.h, and amici::hdf5::readSolverSettingsFromHDF5 in hdf5.cpp.
Definition at line 48 of file solver.h.

18.29.2 Constructor & Destructor Documentation

18.29.2.1 Solver() [1/2]

```
Solver (
    AmiciApplication * app )
```

Parameters

app	AMICI application context

Definition at line 15 of file solver.cpp.

18.29.2.2 Solver() [2/2]

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

Parameters

other	

Definition at line 20 of file solver.cpp.

18.29.3 Member Function Documentation

18.29.3.1 clone()

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

Returns

The clone

Here is the caller graph for this function:

18.29.3.2 run()

```
int run (
    realtype tout ) const
```

Parameters

tout	next timepooint

Returns

status flag

Definition at line 39 of file solver.cpp.
Here is the call graph for this function:

18.29.3.3 step()

```
int step (
    realtype tout ) const
```

Parameters

tout	next timepooint

Returns

status flag

Definition at line 52 of file solver.cpp.
Here is the call graph for this function:

18.29.3.4 runB()

```
void runB (
    realtype tout ) const
```

Parameters

tout	next timepooint

Definition at line 62 of file solver.cpp.
Here is the call graph for this function:

18.29.3.5 setup()

```
void setup (
    realtype t0,
    Model * model,
    const AmiVector & x0,
    const AmiVector & dx0,
    const AmiVectorArray & sx0,
    const AmiVectorArray & sdx0 ) const
```

Parameters

t0	initial timepoint

Parameters

model	pointer to the model instance
x0	initial states
dx0	initial derivative states
sx0	initial state sensitivities
sdx0	initial derivative state sensitivities

Definition at line 69 of file solver.cpp.
Here is the call graph for this function:

18.29.3.6 setupB()

```
void setupB (
    int * which,
    realtype tf,
    Model * model,
    const AmiVector & xB0,
    const AmiVector & dxB0,
    const AmiVector & xQB0 ) const
```

Parameters

which	index of the backward problem, will be set by this routine
tf	final timepoint (initial timepoint for the bwd problem)
model	pointer to the model instance
xB0	initial adjoint states
dxB0	initial adjoint derivative states
xQB0	initial adjoint quadratures

Definition at line 130 of file solver.cpp.
Here is the call graph for this function:

18.29.3.7 setupSteadystate()

```
void setupSteadystate (
    const realtype t0,
    Model * model,
    const AmiVector & x0,
    const AmiVector & dx0,
    const AmiVector & xB0,
    const AmiVector & dxB0,
    const AmiVector & xQ0 ) const
```

Parameters

t0	initial timepoint
model	pointer to the model instance
x0	initial states
dx0	initial derivative states
xB0	initial adjoint states
dxB0	initial derivative adjoint states
xQ0	initial quadrature vector

Definition at line 161 of file solver.cpp.
Here is the call graph for this function:

18.29.3.8 updateAndReinitStatesAndSensitivities()

```
void updateAndReinitStatesAndSensitivities ( Model * model )
```

Parameters

model	pointer to the model instance

Definition at line 182 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.9 getRootInfo()

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

getRootInfo extracts information which event occured

Parameters

rootsfound	array with flags indicating whether the respective event occured

18.29.3.10 calcIC()

```
virtual void calcIC (
    realtype tout1 ) const  [pure virtual]
```

Parameters

tout1	next timepoint to be computed (sets timescale)

18.29.3.11 calcICB()

```
virtual void calcICB (
    int which,
    realtype tout1 ) const  [pure virtual]
```

Parameters

which	identifier of the backwards problem
tout1	next timepoint to be computed (sets timescale)

18.29.3.12 solveB()

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

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:

18.29.3.13 getSensitivityMethod() `SensitivityMethod getSensitivityMethod ( ) const`

Returns

method enum

Definition at line 554 of file solver.cpp.

18.29.3.14 setSensitivityMethod() `void setSensitivityMethod ( SensitivityMethod sensi_meth )`

Parameters

sensi_meth	

Definition at line 558 of file solver.cpp.
Here is the call graph for this function:

18.29.3.15 getSensitivityMethodPreequilibration() `SensitivityMethod getSensitivityMethodPreequilibration ( ) const`

Returns

method enum

Definition at line 556 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.16 setSensitivityMethodPreequilibration() `void setSensitivityMethodPreequilibration ( SensitivityMethod sensi_meth_preeq )`

Parameters

sensi_meth_preeq	

Definition at line 563 of file solver.cpp.
Here is the call graph for this function:

18.29.3.17 getNewtonMaxSteps() `int getNewtonMaxSteps ( ) const`

Returns

Definition at line 578 of file solver.cpp.

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

Parameters

newton_maxsteps	

Definition at line 580 of file solver.cpp.

18.29.3.19 getPreequilibration() `bool getPreequilibration ( ) const`

Returns

Definition at line 586 of file solver.cpp.

18.29.3.20 setPreequilibration() `void setPreequilibration ( bool require_preequilibration )`

Parameters

require_preequilibration	

Definition at line 588 of file solver.cpp.

18.29.3.21 getNewtonMaxLinearSteps() `int getNewtonMaxLinearSteps ( ) const`

Returns

Definition at line 592 of file solver.cpp.

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

Parameters

newton_maxlinsteps	

Definition at line 594 of file solver.cpp.

18.29.3.23 getNewtonDampingFactorMode() `NewtonDampingFactorMode getNewtonDampingFactorMode ( ) const`

Returns

Definition at line 600 of file solver.cpp.

18.29.3.24 setNewtonDampingFactorMode() `void setNewtonDampingFactorMode ( NewtonDampingFactorMode dampingFactorMode )`

Parameters

dampingFactorMode	

Definition at line 602 of file solver.cpp.

18.29.3.25 getNewtonDampingFactorLowerBound() `double getNewtonDampingFactorLowerBound ( ) const`

Returns

Definition at line 606 of file solver.cpp.

18.29.3.26 setNewtonDampingFactorLowerBound() `void setNewtonDampingFactorLowerBound ( double dampingFactorLowerBound )`

Parameters

dampingFactorLowerBound	

Definition at line 608 of file solver.cpp.

18.29.3.27 getSensitivityOrder() `SensitivityOrder getSensitivityOrder ( ) const`

Returns

sensitivity order

Definition at line 612 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.28 setSensitivityOrder() `void setSensitivityOrder ( SensitivityOrder sensi )`

Parameters

sensi	sensitivity order

Definition at line 614 of file solver.cpp.

Here is the call graph for this function:

18.29.3.29 getRelativeTolerance() `double getRelativeTolerance ( ) const`

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Returns

relative tolerances

Definition at line 623 of file solver.cpp.

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

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 627 of file solver.cpp.
Here is the call graph for this function:

18.29.3.31 getAbsoluteTolerance() `double getAbsoluteTolerance ( ) const`

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Returns

absolute tolerances

Definition at line 639 of file solver.cpp.

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

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 643 of file solver.cpp.
Here is the call graph for this function:

18.29.3.33 getRelativeToleranceFSA() `double getRelativeToleranceFSA ( ) const`

Returns

relative tolerances

Definition at line 655 of file solver.cpp.

18.29.3.34 setRelativeToleranceFSA() `void setRelativeToleranceFSA ( double rtol )`

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 659 of file solver.cpp.

Here is the call graph for this function:

18.29.3.35 getAbsoluteToleranceFSA() `double getAbsoluteToleranceFSA ( ) const`

Returns

absolute tolerances

Definition at line 670 of file solver.cpp.

18.29.3.36 setAbsoluteToleranceFSA() `void setAbsoluteToleranceFSA ( double atol )`

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 674 of file solver.cpp.
Here is the call graph for this function:

18.29.3.37 getRelativeToleranceB() `double getRelativeToleranceB ( ) const`

Returns

relative tolerances

Definition at line 685 of file solver.cpp.

18.29.3.38 setRelativeToleranceB() `void setRelativeToleranceB ( double rtol )`

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 689 of file solver.cpp.
Here is the call graph for this function:

18.29.3.39 getAbsoluteToleranceB() `double getAbsoluteToleranceB ( ) const`

Returns

absolute tolerances

Definition at line 700 of file solver.cpp.

18.29.3.40 setAbsoluteToleranceB() `void setAbsoluteToleranceB ( double atol )`

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 704 of file solver.cpp.
Here is the call graph for this function:

18.29.3.41 getRelativeToleranceQuadratures() `double getRelativeToleranceQuadratures ( ) const`

Returns

relative tolerance

Definition at line 715 of file solver.cpp.
Here is the caller graph for this function:

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

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 719 of file solver.cpp.
Here is the call graph for this function:

18.29.3.43 getAbsoluteToleranceQuadratures() `double getAbsoluteToleranceQuadratures ( ) const`

Returns

absolute tolerance

Definition at line 733 of file solver.cpp.
Here is the caller graph for this function:

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

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 737 of file solver.cpp.
Here is the call graph for this function:

18.29.3.45 getRelativeToleranceSteadyState() `double getRelativeToleranceSteadyState ( ) const`

Returns

relative tolerance

Definition at line 751 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.46 setRelativeToleranceSteadyState() `void setRelativeToleranceSteadyState ( double rtol )`

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 755 of file solver.cpp.

18.29.3.47 getAbsoluteToleranceSteadyState() `double getAbsoluteToleranceSteadyState ( ) const`

Returns

absolute tolerance

Definition at line 762 of file solver.cpp.

Here is the caller graph for this function:

18.29.3.48 setAbsoluteToleranceSteadyState() `void setAbsoluteToleranceSteadyState ( double atol )`

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 766 of file solver.cpp.

18.29.3.49 getRelativeToleranceSteadyStateSensi() `double getRelativeToleranceSteadyStateSensi ( ) const`

Returns

relative tolerance

Definition at line 773 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.50 setRelativeToleranceSteadyStateSensi() `void setRelativeToleranceSteadyStateSensi ( double rtol )`

Parameters

rtol	relative tolerance (non-negative number)

Definition at line 777 of file solver.cpp.

18.29.3.51 getAbsoluteToleranceSteadyStateSensi() `double getAbsoluteToleranceSteadyStateSensi (`

```
) const
```

Returns

absolute tolerance

Definition at line 784 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.52 setAbsoluteToleranceSteadyStateSensi()

```
void setAbsoluteToleranceSteadyStateSensi (
double atol )
```

Parameters

atol	absolute tolerance (non-negative number)

Definition at line 788 of file solver.cpp.

18.29.3.53 getMaxSteps()

```
long int getMaxSteps ( ) const
```

Returns

maximum number of solver steps

Definition at line 795 of file solver.cpp.

18.29.3.54 setMaxSteps()

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

Parameters

maxsteps	maximum number of solver steps (non-negative number)

Definition at line 797 of file solver.cpp.

Here is the call graph for this function:

18.29.3.55 getMaxStepsBackwardProblem() `long int getMaxStepsBackwardProblem ( ) const`

Returns

maximum number of solver steps

Definition at line 806 of file solver.cpp.

18.29.3.56 setMaxStepsBackwardProblem() `void setMaxStepsBackwardProblem ( long int maxsteps )`

Parameters

maxsteps	maximum number of solver steps (non-negative number)

Definition at line 808 of file solver.cpp.
Here is the call graph for this function:

18.29.3.57 getLinearMultistepMethod() `LinearMultistepMethod getLinearMultistepMethod ( ) const`

Returns

linear system multistep method

Definition at line 818 of file solver.cpp.

18.29.3.58 setLinearMultistepMethod()

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

Parameters

lmm	linear system multistep method

Definition at line 820 of file solver.cpp.
Here is the call graph for this function:

18.29.3.59 getNonlinearSolverIteration() `NonlinearSolverIteration getNonlinearSolverIteration ( ) const`

Returns

Definition at line 826 of file solver.cpp.

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

Parameters

iter	nonlinear system solution method

Definition at line 830 of file solver.cpp.
Here is the call graph for this function:

18.29.3.61 getInterpolationType() `InterpolationType getInterpolationType ( ) const`

Returns

Definition at line 836 of file solver.cpp.

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

Parameters

interpType	interpolation type

Definition at line 838 of file solver.cpp.
Here is the call graph for this function:

18.29.3.63 getStateOrdering() `int getStateOrdering ( ) const`

Returns

State-ordering as integer according to SUNLinSolKLU::StateOrdering or SUNLinSolSuperLUMT::State↩ Ordering (which differ).

Definition at line 844 of file solver.cpp.

18.29.3.64 setStateOrdering() `void setStateOrdering ( int ordering )`

This only applies when linsol is set to LinearSolver::KLU or LinearSolver::SuperLUMT. Mind the difference between SUNLinSolKLU::StateOrdering and SUNLinSolSuperLUMT::StateOrdering.

Parameters

ordering	state ordering

Definition at line 846 of file solver.cpp.

18.29.3.65 getStabilityLimitFlag() `bool getStabilityLimitFlag ( ) const`

Returns

stldet can be false (deactivated) or true (activated)

Definition at line 866 of file solver.cpp.

18.29.3.66 setStabilityLimitFlag() `void setStabilityLimitFlag ( bool stldet )`

Parameters

stldet	can be false (deactivated) or true (activated)

Definition at line 868 of file solver.cpp.
Here is the call graph for this function:

18.29.3.67 getLinearSolver() `LinearSolver getLinearSolver ( ) const`

Returns

Definition at line 878 of file solver.cpp.
Here is the caller graph for this function:

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

Parameters

linsol	

Definition at line 880 of file solver.cpp.

Here is the call graph for this function:

18.29.3.69 getInternalSensitivityMethod() `InternalSensitivityMethod getInternalSensitivityMethod ( ) const`

Returns

internal sensitivity method

Definition at line 886 of file solver.cpp.

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

Parameters

ism	internal sensitivity method

Definition at line 890 of file solver.cpp.
Here is the call graph for this function:

18.29.3.71 getReturnDataReportingMode() `RDataReporting getReturnDataReportingMode ( ) const`

Returns

ReturnData reporting mode

Definition at line 896 of file solver.cpp.

18.29.3.72 setReturnDataReportingMode() `void setReturnDataReportingMode ( RDataReporting rdrm )`

Parameters

rdrm	ReturnData reporting mode

Definition at line 900 of file solver.cpp.

18.29.3.73 writeSolution()

```
void writeSolution (
    realtype * t,
    AmiVector & x,
    AmiVector & dx,
    AmiVectorArray & sx,
    AmiVector & xQ ) const
```

Parameters

t	time
x	state
dx	derivative state
sx	state sensitivity
xQ	quadrature

Definition at line 1034 of file solver.cpp.
Here is the call graph for this function:

18.29.3.74 writeSolutionB()

```
void writeSolutionB (
    realtype * t,
    AmiVector & xB,
    AmiVector & dxB,
    AmiVector & xQB,
    int which ) const
```

Parameters

t	time
xB	adjoint state

Parameters

dxB	adjoint derivative state
xQB	adjoint quadrature
which	index of adjoint problem

Definition at line 1045 of file solver.cpp.
Here is the call graph for this function:

18.29.3.75 getState() `const AmiVector & getState ( realtype t ) const`

Parameters

t	time

Returns

x or interpolated solution dky

Definition at line 1053 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.76 getDerivativeState() `const AmiVector & getDerivativeState ( realtype t ) const`

Parameters

t	time

Returns

dx or interpolated solution dky

Definition at line 1063 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.77 getStateSensitivity() `const AmiVectorArray & getStateSensitivity ( realtype t ) const`

Parameters

t	time

Returns

(interpolated) solution sx

Definition at line 1073 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.78 getAdjointState()

```
const AmiVector & getAdjointState (
    int which,
    realtype t ) const
```

Parameters

which	adjoint problem index
t	time

Returns

(interpolated) solution xB

Definition at line 1084 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.79 getAdjointDerivativeState()

```
const AmiVector & getAdjointDerivativeState (
    int which,
    realtype t ) const
```

Parameters

which	adjoint problem index
t	time

Returns

(interpolated) solution dxB

Definition at line 1100 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.80 getAdjointQuadrature()

```
const AmiVector & getAdjointQuadrature (
    int which,
    realtype t ) const
```

Parameters

which	adjoint problem index
t	time

Returns

(interpolated) solution xQB

Definition at line 1116 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.81 getQuadrature() `const AmiVector & getQuadrature ( realtype t ) const`

Parameters

t	time

Returns

(interpolated) solution xQ

Definition at line 1132 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.82 reInit()

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

Parameters

t0	reinitialization timepoint
yy0	inital state variables
yp0	initial derivative state variables (DAE only)

Here is the caller graph for this function:

18.29.3.83 sensReInit()

```
virtual void sensReInit (
    const AmiVectorArray & yyS0,
    const AmiVectorArray & ypS0 ) const  [pure virtual]
```

Parameters

yyS0	new state sensitivity
ypS0	new derivative state sensitivities (DAE only)

18.29.3.84 reInitB()

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

Parameters

which	identifier of the backwards problem

Parameters

tB0	reinitialization timepoint
yyB0	new adjoint state
ypB0	new adjoint derivative state

18.29.3.85 quadReInitB()

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

Parameters

which	identifier of the backwards problem
yQB0	new adjoint quadrature state

18.29.3.86 gett()

```
realtype gett ( ) const
```

Returns

t

Definition at line 1148 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.87 getCpuTime()

```
realtype getCpuTime ( ) const
```

Returns

cpu_time

Definition at line 1001 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.88 getCpuTimeB() `realtype getCpuTimeB ( ) const`

Returns

cpu_timeB

Definition at line 1005 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.89 nx() `int nx ( ) const`

Returns

x.getLength()

Definition at line 952 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.90 nplist() `int nplist ( ) const`

Returns

sx.getLength()

Definition at line 950 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.91 nquad() `int nquad ( ) const`

Returns

xQB.getLength()

Definition at line 954 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.92 computingFSA() `bool computingFSA ( ) const`

Returns

flag

Definition at line 747 of file solver.h.
Here is the call graph for this function:

18.29.3.93 computingASA() `bool computingASA ( ) const`

Returns

flag

Definition at line 756 of file solver.h.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.94 storeDiagnosisB() `void storeDiagnosisB ( int which ) const`

Parameters

which	identifier of the backwards problem

Definition at line 234 of file solver.cpp.

Here is the call graph for this function:

18.29.3.95 getNumSteps() [1/2] `std::vector<int> const& getNumSteps ( ) const`

Returns

ns

Definition at line 782 of file solver.h.
Here is the caller graph for this function:

18.29.3.96 getNumStepsB() `std::vector<int> const& getNumStepsB ( ) const`

Returns

nsB

Definition at line 790 of file solver.h.
Here is the caller graph for this function:

18.29.3.97 getNumRhsEvals() [1/2] `std::vector<int> const& getNumRhsEvals ( ) const`

Returns

nrhs

Definition at line 798 of file solver.h.
Here is the caller graph for this function:

18.29.3.98 getNumRhsEvalsB() `std::vector<int> const& getNumRhsEvalsB ( ) const`

Returns

nrhsB

Definition at line 806 of file solver.h.
Here is the caller graph for this function:

18.29.3.99 getNumErrTestFails() [1/2] std::vector<int> const& getNumErrTestFails ( ) const

Returns

netf

Definition at line 814 of file solver.h.
Here is the caller graph for this function:

18.29.3.100 getNumErrTestFailsB() std::vector<int> const& getNumErrTestFailsB ( ) const

Returns

netfB

Definition at line 822 of file solver.h.
Here is the caller graph for this function:

18.29.3.101 getNumNonlinSolvConvFails() [1/2] std::vector<int> const& getNumNonlinSolvConv↩ Fails ( ) const

Returns

nnlscf

Definition at line 830 of file solver.h.
Here is the caller graph for this function:

18.29.3.102 getNumNonlinSolvConvFailsB() `std::vector<int> const& getNumNonlinSolvConvFailsB ( ) const`

Returns

nnlscfB

Definition at line 838 of file solver.h.
Here is the caller graph for this function:

18.29.3.103 getLastOrder() [1/2] `std::vector<int> const& getLastOrder ( ) const`

Returns

order

Definition at line 846 of file solver.h.
Here is the caller graph for this function:

18.29.3.104 setStopTime()

```
virtual void setStopTime (
    realtype tstop ) const [protected], [pure virtual]
```

Parameters

tstop	timepoint until which simulation should be performed

Here is the caller graph for this function:

18.29.3.105 solve()

```
virtual int solve (
    realtype tout,
    int itask ) const [protected], [pure virtual]
```

Parameters

tout	timepoint until which simulation should be performed
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:

18.29.3.106 solveF()

```
virtual int solveF (
    realtype tout,
    int itask,
    int * ncheckPtr ) const [protected], [pure virtual]
```

Parameters

tout	timepoint until which simulation should be performed
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:

18.29.3.107 reInitPostProcessF()

`virtual void reInitPostProcessF ( realtype tnext ) const [protected], [pure virtual]`

Parameters

tnext	next timepoint (defines integration direction)

18.29.3.108 reInitPostProcessB()

`virtual void reInitPostProcessB ( realtype tnext ) const [protected], [pure virtual]`

Parameters

tnext	next timepoint (defines integration direction)

18.29.3.109 getB()

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

Parameters

which	index of the backwards problem

Here is the caller graph for this function:

18.29.3.110 getQuadB() `virtual void getQuadB ( int which ) const [protected], [pure virtual]`

Parameters

which	index of the backwards problem

Here is the caller graph for this function:

18.29.3.111 getQuad() `virtual void getQuad ( realtype & t ) const [protected], [pure virtual]`

Parameters

t	timepoint for quadrature extraction

Here is the caller graph for this function:

18.29.3.112 init()

```
virtual void init (
    realtype t0,
    const AmiVector & x0,
    const AmiVector & dx0 ) const  [protected], [pure virtual]
```

Parameters

t0	initial timepoint
x0	initial states
dx0	initial derivative states

Here is the caller graph for this function:

18.29.3.113 initSteadystate()

```
virtual void initSteadystate (
    realtype t0,
    const AmiVector & x0,
    const AmiVector & dx0 ) const  [protected], [pure virtual]
```

Parameters

t0	initial timepoint
x0	initial states
dx0	initial derivative states

Here is the caller graph for this function:

18.29.3.114 sensInit1()

```
virtual void sensInit1 (
    const AmiVectorArray & sx0,
    const AmiVectorArray & sdx0 ) const  [protected], [pure virtual]
```

Parameters

sx0	initial states semsitivities

Parameters

sdx0	initial derivative states sensitivities

18.29.3.115 binit()

```
virtual void binit (
    int which,
    realtype tf,
    const AmiVector & xB0,
    const AmiVector & dxB0 ) const [protected], [pure virtual]
```

Parameters

which	identifier of the backwards problem
tf	final timepoint
xB0	initial adjoint state
dxB0	initial adjoint derivative state

Here is the caller graph for this function:

18.29.3.116 qbinit()

```
virtual void qbinit (
    int which,
    const AmiVector & xQB0 ) const [protected], [pure virtual]
```

Parameters

which	identifier of the backwards problem
xQB0	intial adjoint quadrature state

Here is the caller graph for this function:

18.29.3.117 rootInit() `virtual void rootInit ( int ne ) const [protected], [pure virtual]`

Parameters

ne	number of different events

Here is the caller graph for this function:

18.29.3.118 initializeNonLinearSolverSens() `void initializeNonLinearSolverSens ( const Model * model ) const [protected]`

Parameters

model	Model instance

Definition at line 908 of file solver.cpp.

18.29.3.119 setDenseJacFnB() `virtual void setDenseJacFnB ( int which ) const [protected], [pure virtual]`

Parameters

which	identifier of the backwards problem

18.29.3.120 setSparseJacFnB() `virtual void setSparseJacFnB ( int which ) const [protected], [pure virtual]`

Parameters

which	identifier of the backwards problem

18.29.3.121 setBandJacFnB() `virtual void setBandJacFnB ( int which ) const [protected], [pure virtual]`

Parameters

which	identifier of the backwards problem

18.29.3.122 setJacTimesVecFnB()

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

Parameters

which	identifier of the backwards problem

18.29.3.123 setSStolerances()

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

Parameters

rtol	relative tolerances
atol	absolute tolerances

Here is the caller graph for this function:

18.29.3.124 setSensSStolerances()

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

Parameters

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

18.29.3.125 setSensErrCon()

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

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

Parameters

error_corr	activation flag

18.29.3.126 setQuadErrConB()

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

Parameters

which	identifier of the backwards problem
flag	activation flag

18.29.3.127 setQuadErrCon()

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

Parameters

flag	activation flag

18.29.3.128 setErrHandlerFn()

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

Here is the caller graph for this function:

18.29.3.129 setUserData()

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

Parameters

model	Model instance

Here is the caller graph for this function:

18.29.3.130 setUserDataB()

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

Parameters

which	identifier of the backwards problem
model	Model instance

Here is the caller graph for this function:

18.29.3.131 setMaxNumSteps()

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

Parameters

mxsteps	number of steps

Here is the caller graph for this function:

18.29.3.132 setMaxNumStepsB()

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

Parameters

which	identifier of the backwards problem
mxstepsB	number of steps

Here is the caller graph for this function:

18.29.3.133 setStabLimDet()

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

Parameters

stldet	flag for stability limit detection (TRUE or FALSE)

Here is the caller graph for this function:

18.29.3.134 setStabLimDetB()

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

Parameters

which	identifier of the backwards problem
stldet	flag for stability limit detection (TRUE or FALSE)

Here is the caller graph for this function:

18.29.3.135 setId()

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

Parameters

model	model specification

18.29.3.136 setSuppressAlg()

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

Parameters

flag	deactivation flag

18.29.3.137 setSensParams()

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

Parameters

p	paramaters
pbar	parameter scaling constants
plist	parameter index list

18.29.3.138 getDky()

```
virtual void getDky (
    realtype t,
    int k ) const [protected], [pure virtual]
```

Parameters

t	timepoint
k	derivative order

Here is the caller graph for this function:

18.29.3.139 getDkyB()

```
virtual void getDkyB (
    realtype t,
    int k,
    int which ) const [protected], [pure virtual]
```

Parameters

t	timepoint
k	derivative order
which	index of backward problem

Here is the caller graph for this function:

18.29.3.140 getSensDky()

```
virtual void getSensDky (
    realtype t,
    int k ) const [protected], [pure virtual]
```

Parameters

t	timepoint
k	derivative order

Here is the caller graph for this function:

18.29.3.141 getQuadDkyB()

```
virtual void getQuadDkyB (
    realtype t,
    int k,
    int which ) const [protected], [pure virtual]
```

Parameters

t	timepoint
k	derivative order
which	index of backward problem

Here is the caller graph for this function:

18.29.3.142 getQuadDky()

```
virtual void getQuadDky (
    realtype t,
    int k ) const [protected], [pure virtual]
```

Parameters

t	timepoint
k	derivative order

Here is the caller graph for this function:

18.29.3.143 quadInit()

```
virtual void quadInit (
    const AmiVector & xQ0 ) const [protected], [pure virtual]
```

Parameters

xQ0	vector with initial values for xQ

Here is the caller graph for this function:

18.29.3.144 allocateSolverB()

```
virtual void allocateSolverB (
    int * which ) const [protected], [pure virtual]
```

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

18.29.3.145 setSStolerancesB()

```
virtual void setSStolerancesB (
```

```
int which,
realtype relTolB,
realtype absTolB ) const [protected], [pure virtual]
```

Parameters

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

18.29.3.146 quadSStolerancesB()

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

Parameters

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

18.29.3.147 quadSStolerances()

```
virtual void quadSStolerances (
realtype reltolQB,
realtype abstolQB ) const [protected], [pure virtual]
```

Parameters

reltolQB	relative tolerances
abstolQB	absolute tolerances

18.29.3.148 getNumSteps() [2/2]

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numsteps	output array

18.29.3.149 getNumRhsEvals() [2/2]

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numrhsevals	output array

18.29.3.150 getNumErrTestFails() [2/2]

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numerrtestfails	output array

18.29.3.151 getNumNonlinSolvConvFails() [2/2]

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numnonlinsolvconvfails	output array

18.29.3.152 getLastOrder() [2/2]

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
order	output array

18.29.3.153 initializeLinearSolver()

```
void initializeLinearSolver (
    const Model * model ) const [protected]
```

Parameters

model	pointer to the model object

Definition at line 257 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.154 setLinearSolverB()

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

Parameters

which	index of the backward problem

18.29.3.155 setNonLinearSolverB()

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

Parameters

which	index of the backward problem

18.29.3.156 initializeLinearSolverB()

```
void initializeLinearSolverB (
    const Model * model,
    int which ) const [protected]
```

Parameters

model	pointer to the model object
which	index of the backward problem

Definition at line 350 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.157 initializeNonLinearSolverB()

```
void initializeNonLinearSolverB (
    int which ) const [protected]
```

Parameters

which	index of the backward problem

Definition at line 422 of file solver.cpp.

Here is the caller graph for this function:

18.29.3.158 getModel() `virtual const Model* getModel ( ) const [protected], [pure virtual]`

Accessor function to the model stored in the user data

Returns

user data model

18.29.3.159 getInitDone() `bool getInitDone ( ) const [protected]`

Returns

proxy for solverMemory->(cv|ida)_MallocDone

Definition at line 956 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.160 getSensInitDone() `bool getSensInitDone ( ) const [protected]`

Returns

proxy for solverMemory->(cv|ida)_SensMallocDone

Definition at line 958 of file solver.cpp.

18.29.3.161 getAdjInitDone() `bool getAdjInitDone ( ) const [protected]`

Returns

proxy for solverMemory->(cv|ida)_adjMallocDone

Definition at line 960 of file solver.cpp.
Here is the caller graph for this function:

18.29.3.162 getInitDoneB() `bool getInitDoneB ( int which ) const [protected]`

Parameters

which	adjoint problem index

Returns

proxy for solverMemoryB->(cv|ida)_MallocDone

Definition at line 962 of file solver.cpp.

18.29.3.163 getQuadInitDoneB() `bool getQuadInitDoneB ( int which ) const [protected]`

Parameters

which	adjoint problem index

Returns

proxy for solverMemoryB->(cv|ida)_QuadMallocDone

Definition at line 967 of file solver.cpp.

18.29.3.164 getQuadInitDone() `bool getQuadInitDone ( ) const [protected]`

Returns

proxy for solverMemory->(cv|ida)_QuadMallocDone

Definition at line 972 of file solver.cpp.
Here is the caller graph for this function:

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

Parameters

which	identifier of the backwards problem

18.29.3.166 resetMutableMemory() `void resetMutableMemory ( int nx, int nplist, int nquad ) const [protected]`

Parameters

nx	new number of state variables
nplist	new number of sensitivity parameters
nquad	new number of quadratures (only differs from nplist for higher order senisitivity computation)

Definition at line 1009 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.167 getAdjBmem()

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

Parameters

which	identifier of the backwards problem
ami_mem	pointer to the forward solver memory instance

Returns

A (void ∗) pointer to the CVODES memory allocated for the backward problem.

18.29.3.168 applyTolerancesASA() `void applyTolerancesASA ( int which ) const [protected]`

Parameters

which	identifier of the backwards problem

Definition at line 496 of file solver.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.169 applyQuadTolerancesASA() `void applyQuadTolerancesASA ( int which ) const [protected]`

Parameters

which	identifier of the backwards problem

Definition at line 508 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.29.3.170 setInitDoneB() `void setInitDoneB ( int which ) const [protected]`

Parameters

which	adjoint problem index

Definition at line 982 of file solver.cpp.

18.29.3.171 setQuadInitDoneB() `void setQuadInitDoneB ( int which ) const [protected]`

Parameters

which	adjoint problem index

Definition at line 988 of file solver.cpp.

18.29.3.172 checkSensitivityMethod() `void checkSensitivityMethod ( const SensitivityMethod sensi_meth, bool preequilibration ) const [protected]`

Parameters

sensi_meth	new value for sensi_meth[_preeq]
preequilibration	flag indicating preequilibration or simulation

Definition at line 568 of file solver.cpp.
Here is the caller graph for this function:

18.29.4 Member Data Documentation

18.29.4.1 app `AmiciApplication* app = &defaultContext`

AMICI context
Definition at line 869 of file solver.h.

18.29.4.2 solver_memory_ `std::unique_ptr<void, std::function<void(void *)> > solver_memory_ [mutable], [protected]`

pointer to solver memory block
Definition at line 1510 of file solver.h.

18.29.4.3 solver_memory_B_ `std::vector<std::unique_ptr<void, std::function<void(void *)> > > solver_memory_B_ [mutable], [protected]`

pointer to solver memory block
Definition at line 1514 of file solver.h.

18.29.4.4 ism_ `InternalSensitivityMethod ism_ {InternalSensitivityMethod::simultaneous} [protected]`

internal sensitivity method flag used to select the sensitivity solution method. Only applies for Forward Sensitivities.
Definition at line 1518 of file solver.h.

18.29.4.5 lmm_ `LinearMultistepMethod lmm_ {LinearMultistepMethod::BDF} [protected]`

specifies the linear multistep method.
Definition at line 1522 of file solver.h.

18.29.4.6 iter_ `NonlinearSolverIteration iter_ {NonlinearSolverIteration::newton} [protected]`

specifies the type of nonlinear solver iteration
Definition at line 1527 of file solver.h.

18.29.4.7 interp_type_ `InterpolationType interp_type_ {InterpolationType::hermite} [protected]`

interpolation type for the forward problem solution which is then used for the backwards problem.
Definition at line 1532 of file solver.h.

18.29.4.8 maxsteps_ `long int maxsteps_ {10000} [protected]`

maximum number of allowed integration steps

Definition at line 1535 of file solver.h.

18.29.4.9 linear_solver_ `std::unique_ptr<SUNLinSolWrapper> linear_solver_ [mutable], [protected]`

linear solver for the forward problem

Definition at line 1538 of file solver.h.

18.29.4.10 linear_solver_B_ `std::unique_ptr<SUNLinSolWrapper> linear_solver_B_ [mutable], [protected]`

linear solver for the backward problem

Definition at line 1541 of file solver.h.

18.29.4.11 non_linear_solver_ `std::unique_ptr<SUNNonLinSolWrapper> non_linear_solver_ [mutable], [protected]`

non-linear solver for the forward problem

Definition at line 1544 of file solver.h.

18.29.4.12 non_linear_solver_B_ `std::unique_ptr<SUNNonLinSolWrapper> non_linear_solver_B_↩ [mutable], [protected]`

non-linear solver for the backward problem

Definition at line 1547 of file solver.h.

18.29.4.13 non_linear_solver_sens_ `std::unique_ptr<SUNNonLinSolWrapper> non_linear_solver_↩ sens_ [mutable], [protected]`

non-linear solver for the sensitivities

Definition at line 1550 of file solver.h.

18.29.4.14 solver_was_called_F_ `bool solver_was_called_F_ {false} [mutable], [protected]`

flag indicating whether the forward solver has been called

Definition at line 1553 of file solver.h.

18.29.4.15 solver_was_called_B_ `bool solver_was_called_B_ {false} [mutable], [protected]`

flag indicating whether the backward solver has been called

Definition at line 1556 of file solver.h.

18.29.4.16 x_ `AmiVector x_ {0} [mutable], [protected]`

state (dimension: nx_solver)

Definition at line 1604 of file solver.h.

18.29.4.17 dky_ `AmiVector dky_ {0} [mutable], [protected]`

state interface variable (dimension: nx_solver)

Definition at line 1607 of file solver.h.

18.29.4.18 dx_ `AmiVector dx_ {0} [mutable], [protected]`
state derivative dummy (dimension: nx_solver)
Definition at line 1610 of file solver.h.

18.29.4.19 sx_ `AmiVectorArray sx_ {0, 0} [mutable], [protected]`
state sensititivities interface variable (dimension: nx_solver x nplist)
Definition at line 1614 of file solver.h.

18.29.4.20 sdx_ `AmiVectorArray sdx_ {0, 0} [mutable], [protected]`
state derivative sensititivities dummy (dimension: nx_solver x nplist)
Definition at line 1617 of file solver.h.

18.29.4.21 xB_ `AmiVector xB_ {0} [mutable], [protected]`
adjoint state interface variable (dimension: nx_solver)
Definition at line 1620 of file solver.h.

18.29.4.22 dxB_ `AmiVector dxB_ {0} [mutable], [protected]`
adjoint derivative dummy variable (dimension: nx_solver)
Definition at line 1623 of file solver.h.

18.29.4.23 xQB_ `AmiVector xQB_ {0} [mutable], [protected]`
adjoint quadrature interface variable (dimension: nJ x nplist)
Definition at line 1626 of file solver.h.

18.29.4.24 xQ_ `AmiVector xQ_ {0} [mutable], [protected]`
forward quadrature interface variable (dimension: nx_solver)
Definition at line 1629 of file solver.h.

18.29.4.25 t_ `realtype t_ {std::nan("")} [mutable], [protected]`
integration time of the forward problem
Definition at line 1632 of file solver.h.

18.29.4.26 force_reinit_postprocess_F_ `bool force_reinit_postprocess_F_ {false} [mutable], [protected]`
flag to force reInitPostProcessF before next call to solve
Definition at line 1635 of file solver.h.

18.29.4.27 force_reinit_postprocess_B_ `bool force_reinit_postprocess_B_ {false} [mutable], [protected]`
flag to force reInitPostProcessB before next call to solveB
Definition at line 1638 of file solver.h.

18.30 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

- SteadystateProblem (const Solver &solver, const Model &model)

  constructor
- void workSteadyStateProblem (Solver ∗solver, Model ∗model, int it)

  Handles steady state computation in the forward case: tries to determine the steady state of the ODE system and computes steady state sensitivities if requested.
- void workSteadyStateBackwardProblem (Solver ∗solver, Model ∗model, const BackwardProblem ∗bwd)
- void findSteadyState (Solver ∗solver, NewtonSolver ∗newtonSolver, Model ∗model, int it)

  Handles the computation of the steady state, throws an AmiException, if no steady state was found.
- void findSteadyStateByNewtonsMethod (NewtonSolver ∗newtonSolver, Model ∗model, bool newton_retry)

  Tries to determine the steady state by using Newton's method.
- void findSteadyStateBySimulation (const Solver ∗solver, Model ∗model, int it)

  Tries to determine the steady state by using forward simulation.
- void computeSteadyStateQuadrature (NewtonSolver ∗newtonSolver, const Solver ∗solver, Model ∗model)

  Handles the computation of quadratures in adjoint mode.
- void getQuadratureByLinSolve (NewtonSolver ∗newtonSolver, Model ∗model)

  Computes the quadrature in steady state backward mode by solving the linear system defined by the backward Jacobian.
- void getQuadratureBySimulation (const Solver ∗solver, Model ∗model)

  Computes the quadrature in steady state backward mode by numerical integration of xB forward in time.
- void handleSteadyStateFailure (const Solver ∗solver, Model ∗model)

  Stores state and throws an exception if equilibration failed.
- void writeErrorString (std::string ∗errorString, SteadyStateStatus status) const

  Assembles the error message to be thrown.
- bool getSensitivityFlag (const Model ∗model, const Solver ∗solver, int it, SteadyStateContext context)

  Checks depending on the status of the Newton solver, solver settings, and the model, whether state sensitivities still need to be computed via a linear system solve or stored.
- realtype getWrmsNorm (AmiVector const &x, AmiVector const &xdot, realtype atol, realtype rtol, AmiVector &ewt) const

  Computes the weighted root mean square of xdot the weights are computed according to x: w_i = 1 / (rtol ∗ x_i + atol)
- bool checkConvergence (const Solver ∗solver, Model ∗model, SensitivityMethod checkSensitivities)

  Checks convergence for state and respective sensitivities.
- void applyNewtonsMethod (Model ∗model, NewtonSolver ∗newtonSolver, bool newton_retry)

  Runs the Newton solver iterations and checks for convergence to steady state.
- void runSteadystateSimulation (const Solver ∗solver, Model ∗model, bool backward)

  Simulation is launched, if Newton solver or linear system solve fails.
- std::unique_ptr< Solver > createSteadystateSimSolver (const Solver ∗solver, Model ∗model, bool forward↩ Sensis, bool backward) const

  Initialize CVodeSolver instance for preequilibration simulation.
- bool initializeBackwardProblem (Solver ∗solver, Model ∗model, const BackwardProblem ∗bwd)

  Initialize backward computation by setting state, time, adjoint state and checking for preequilibration mode.
- void computeQBfromQ (Model ∗model, const AmiVector &yQ, AmiVector &yQB) const

  Compute the backward quadratures, which contribute to the gradient (xQB) from the quadrature over the backward state itself (xQ)
- void storeSimulationState (Model ∗model, bool storesensi)

  Store carbon copy of current simulation state variables as SimulationState.
- const SimulationState & getFinalSimulationState () const

  Returns the stored SimulationState.
- const AmiVector & getEquilibrationQuadratures () const

  Returns the quadratures from pre- or postequilibration.
- const AmiVector & getState () const

Returns state at steadystate.

- const AmiVectorArray & getStateSensitivity () const

  Returns state sensitivity at steadystate.
- std::vector< realtype > const & getDJydx () const

  Accessor for dJydx.
- double getCPUTime () const

  Accessor for run_time of the forward problem.
- double getCPUTimeB () const

  Accessor for run_time of the backward problem.
- std::vector< SteadyStateStatus > const & getSteadyStateStatus () const

  Accessor for steady_state_status.
- realtype getSteadyStateTime () const

  Accessor for t.
- realtype getResidualNorm () const

  Accessor for wrms.
- const std::vector< int > & getNumSteps () const

  Accessor for numsteps.
- int getNumStepsB () const

  Accessor for numstepsB.
- const std::vector< int > & getNumLinSteps () const

  Accessor for numlinsteps.
- void getAdjointUpdates (Model &model, const ExpData &edata)

  computes adjoint updates dJydx according to provided model and expdata
- AmiVector const & getAdjointQuadrature () const

  Accessor for xQB.
- bool hasQuadrature () const

  Accessor for hasQuadrature_.
- bool checkSteadyStateSuccess () const

  computes adjoint updates dJydx according to provided model and expdata

18.30.1 Detailed Description

Definition at line 25 of file steadystateproblem.h.

18.30.2 Constructor & Destructor Documentation

18.30.2.1 SteadystateProblem()

```
SteadystateProblem (
    const Solver & solver,
    const Model & model ) [explicit]
```

Parameters

solver	Solver instance
model	Model instance

Definition at line 21 of file steadystateproblem.cpp.

Here is the call graph for this function:

18.30.3 Member Function Documentation

18.30.3.1 workSteadyStateProblem()

```
void workSteadyStateProblem (
    Solver * solver,
    Model * model,
    int it )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object
it	integer with the index of the current time step

Definition at line 44 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.2 workSteadyStateBackwardProblem()

```
void workSteadyStateBackwardProblem (
    Solver * solver,
    Model * model,
    const BackwardProblem * bwd )
```

Integrates over the adjoint state backward in time by solving a linear system of equations, which gives the analytical solution. Computes the gradient via adjoint steady state sensitivities

Parameters

solver	pointer to the solver object
model	pointer to the model object
bwd	backward problem

Definition at line 90 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.3 findSteadyState()

```
void findSteadyState (
    Solver * solver,
    NewtonSolver * newtonSolver,
    Model * model,
    int it )
```

Parameters

solver	pointer to the solver object
newtonSolver	pointer to the newtonSolver solver object
model	pointer to the model object
it	integer with the index of the current time step

Definition at line 109 of file steadystateproblem.cpp.

18.30.3.4 findSteadyStateByNewtonsMethod()

```
void findSteadyStateByNewtonsMethod (
    NewtonSolver * newtonSolver,
    Model * model,
    bool newton_retry )
```

Parameters

newtonSolver	pointer to the newtonSolver solver object
model	pointer to the model object
newton_retry	bool flag indicating whether being relaunched

Definition at line 129 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.5 findSteadyStateBySimulation()

```
void findSteadyStateBySimulation (
    const Solver * solver,
    Model * model,
    int it )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object
it	integer with the index of the current time step

Definition at line 172 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.6 computeSteadyStateQuadrature()

```
void computeSteadyStateQuadrature (
    NewtonSolver * newtonSolver,
```

```
const Solver * solver,
Model * model )
```

Parameters

newtonSolver	pointer to the newtonSolver solver object
solver	pointer to the solver object
model	pointer to the model object

Definition at line 241 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.7 getQuadratureByLinSolve()

```
void getQuadratureByLinSolve (
NewtonSolver * newtonSolver,
Model * model )
```

Parameters

newtonSolver	pointer to the newtonSolver solver object
model	pointer to the model object

Definition at line 265 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.8 getQuadratureBySimulation()

```
void getQuadratureBySimulation (
    const Solver * solver,
    Model * model )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object

Definition at line 293 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.9 handleSteadyStateFailure()

```
void handleSteadyStateFailure (
    const Solver * solver,
    Model * model )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object

Definition at line 316 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.10 writeErrorString()

```
void writeErrorString (
    std::string * errorString,
    SteadyStateStatus status ) const
```

Parameters

errorString	const pointer to string with error message
status	Entry of steady_state_staus to be processed

Definition at line 334 of file steadystateproblem.cpp.

18.30.3.11 getSensitivityFlag()

```
bool getSensitivityFlag (
    const Model * model,
    const Solver * solver,
    int it,
    SteadyStateContext context )
```

Parameters

model	pointer to the model object
solver	pointer to the solver object
it	integer with the index of the current time step
context	SteadyStateContext giving the situation for the flag

Returns

flag telling how to process state sensis

Definition at line 359 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.12 getWrmsNorm()

```
realtype getWrmsNorm (
    AmiVector const & x,
    AmiVector const & xdot,
```

```
realtype atol,
realtype rtol,
AmiVector & ewt ) const
```

Parameters

x	current state
xdot	current rhs
atol	absolute tolerance
rtol	relative tolerance
ewt	error weight vector

Returns

root-mean-square norm

Definition at line 414 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.13 checkConvergence()

```
bool checkConvergence (
const Solver * solver,
Model * model,
SensitivityMethod checkSensitivities )
```

Parameters

solver	Solver instance
model	instance
checkSensitivities	flag whether sensitivities should be checked

Returns

boolean indicating convergence

Definition at line 428 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.14 applyNewtonsMethod()

```
void applyNewtonsMethod (
    Model * model,
    NewtonSolver * newtonSolver,
    bool newton_retry )
```

Parameters

model	pointer to the model object
newtonSolver	pointer to the NewtonSolver object **Type**: NewtonSolver
newton_retry	flag indicating if Newton solver is rerun

Definition at line 476 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.15 runSteadystateSimulation()

```
void runSteadystateSimulation (
    const Solver * solver,
    Model * model,
    bool backward )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object
backward	flag indicating adjoint mode (including quadrature)

Definition at line 571 of file steadystateproblem.cpp.
Here is the caller graph for this function:

18.30.3.16 createSteadystateSimSolver()

```
std::unique_ptr< Solver > createSteadystateSimSolver (
    const Solver * solver,
    Model * model,
    bool forwardSensis,
    bool backward ) const
```

Parameters

solver	pointer to the solver object
model	pointer to the model object
forwardSensis	flag switching on integration with FSA
backward	flag switching on quadratures computation

Returns

solver instance

Definition at line 642 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.17 initializeBackwardProblem()

```
bool initializeBackwardProblem (
    Solver * solver,
    Model * model,
    const BackwardProblem * bwd )
```

Parameters

solver	pointer to the solver object
model	pointer to the model object
bwd	pointer to backward problem

Returns

flag indicating whether backward computation to be carried out

Definition at line 211 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.18 computeQBfromQ()

```
void computeQBfromQ (
    Model * model,
    const AmiVector & yQ,
    AmiVector & yQB ) const
```

Parameters

model	pointer to the model object
yQ	vector to be multiplied with dxdotdp
yQB	resulting vector after multiplication

Definition at line 678 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.19 storeSimulationState()

```
void storeSimulationState (
    Model * model,
    bool storesensi )
```

Parameters

model	model carrying the ModelState to be used
storesensi	flag to enable storage of sensitivities

Definition at line 716 of file steadystateproblem.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.30.3.20 getFinalSimulationState() `const SimulationState& getFinalSimulationState ( ) const`

Returns

stored SimulationState

Definition at line 233 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.21 getEquilibrationQuadratures() `const AmiVector& getEquilibrationQuadratures ( ) const`

Returns

xQB Vector with quadratures

Definition at line 241 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.22 getState() `const AmiVector& getState ( ) const`

Returns

x

Definition at line 248 of file steadystateproblem.h.
Here is the caller graph for this function:

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

Returns

sx

Definition at line 257 of file steadystateproblem.h.
Here is the caller graph for this function:

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

Returns

dJydx

Definition at line 265 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.25 getCPUTime() `double getCPUTime ( ) const`

Returns

run_time

Definition at line 273 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.26 getCPUTimeB() `double getCPUTimeB ( ) const`

Returns

run_time

Definition at line 279 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.27 getSteadyStateStatus() `std::vector<SteadyStateStatus> const& getSteadyStateStatus ( ) const`

Returns

steady_state_status

Definition at line 285 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.28 getSteadyStateTime() `realtype getSteadyStateTime ( ) const`

Returns

t

Definition at line 292 of file steadystateproblem.h.

18.30.3.29 getResidualNorm() `realtype getResidualNorm ( ) const`

Returns

wrms

Definition at line 298 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.30 getNumSteps() `const std::vector<int>& getNumSteps ( ) const`

Returns

numsteps

Definition at line 304 of file steadystateproblem.h.

18.30.3.31 getNumStepsB() `int getNumStepsB ( ) const`

Returns

numstepsB

Definition at line 310 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.32 getNumLinSteps() `const std::vector<int>& getNumLinSteps ( ) const`

Returns

numlinsteps

Definition at line 316 of file steadystateproblem.h.

18.30.3.33 getAdjointUpdates()

```
void getAdjointUpdates (
    Model & model,
    const ExpData & edata )
```

Parameters

model	Model instance
edata	experimental data

Definition at line 703 of file steadystateproblem.cpp.
Here is the call graph for this function:

18.30.3.34 getAdjointQuadrature()

```
AmiVector const& getAdjointQuadrature ( ) const
```

Returns

xQB

Definition at line 329 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.35 hasQuadrature()

```
bool hasQuadrature ( ) const
```

Returns

hasQuadrature_

Definition at line 335 of file steadystateproblem.h.
Here is the caller graph for this function:

18.30.3.36 checkSteadyStateSuccess() `bool checkSteadyStateSuccess ( ) const`

Returns

covergence of steady state solver

Definition at line 465 of file steadystateproblem.cpp.

18.31 SUNLinSolBand Class Reference

SUNDIALS band direct solver.
`#include <sundials_linsol_wrapper.h>`
Inheritance diagram for SUNLinSolBand:

Public Member Functions

- SUNLinSolBand (N_Vector x, SUNMatrix A)

  Create solver using existing matrix A without taking ownership of A.
- SUNLinSolBand (AmiVector const &x, int ubw, int lbw)

  Create new band solver and matrix A.
- SUNMatrix getMatrix () const override

  Get the matrix A (matrix solvers only).

Additional Inherited Members

18.31.1 Detailed Description

Definition at line 141 of file sundials_linsol_wrapper.h.

18.31.2 Constructor & Destructor Documentation

18.31.2.1 SUNLinSolBand() [1/2]

```
SUNLinSolBand (
    N_Vector x,
    SUNMatrix A )
```

Parameters

x	A template for cloning vectors needed within the solver.
A	square matrix

Definition at line 148 of file sundials_linsol_wrapper.cpp.

18.31.2.2 SUNLinSolBand() [2/2]

```
SUNLinSolBand (
    AmiVector const & x,
    int ubw,
    int lbw )
```

Parameters

x	A template for cloning vectors needed within the solver.
ubw	upper bandwidth of band matrix A
lbw	lower bandwidth of band matrix A

Definition at line 154 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.31.3 Member Function Documentation

18.31.3.1 getMatrix()

```
SUNMatrix getMatrix ( ) const [override], [virtual]
```

Returns

A

Reimplemented from SUNLinSolWrapper.
Definition at line 162 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.32 SUNLinSolDense Class Reference

SUNDIALS dense direct solver.

```
#include <sundials_linsol_wrapper.h>
```

Inheritance diagram for SUNLinSolDense:

Public Member Functions

- SUNLinSolDense (AmiVector const &x)

  Create dense solver.
- SUNMatrix getMatrix () const override

  Get the matrix A (matrix solvers only).

Additional Inherited Members

18.32.1 Detailed Description

Definition at line 170 of file sundials_linsol_wrapper.h.

18.32.2 Constructor & Destructor Documentation

18.32.2.1 SUNLinSolDense() `SUNLinSolDense ( AmiVector const & x ) [explicit]`

Parameters

x	A template for cloning vectors needed within the solver.

Definition at line 164 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.32.3 Member Function Documentation

18.32.3.1 getMatrix() `SUNMatrix getMatrix ( ) const [override], [virtual]`

Returns

A

Reimplemented from SUNLinSolWrapper.
Definition at line 171 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.33 SUNLinSolKLU Class Reference

SUNDIALS KLU sparse direct solver.
`#include <sundials_linsol_wrapper.h>`
Inheritance diagram for SUNLinSolKLU:

Public Types

- enum StateOrdering { **AMD**, **COLAMD**, **natural** }

Public Member Functions

- SUNLinSolKLU (N_Vector x, SUNMatrix A)

  Create KLU solver with given matrix.
- SUNLinSolKLU (AmiVector const &x, int nnz, int sparsetype, StateOrdering ordering)

  Create KLU solver and matrix to operate on.
- SUNMatrix getMatrix () const override

  Get the matrix A (matrix solvers only).
- void reInit (int nnz, int reinit_type)

Reinitializes memory and flags for a new factorization (symbolic and numeric) to be conducted at the next solver setup call.

- void setOrdering (StateOrdering ordering)

  Sets the ordering used by KLU for reducing fill in the linear solve.

Additional Inherited Members

18.33.1 Detailed Description

Definition at line 189 of file sundials_linsol_wrapper.h.

18.33.2 Member Enumeration Documentation

18.33.2.1 StateOrdering `enum StateOrdering [strong]`

KLU state reordering (different from SuperLUMT ordering!)
Definition at line 192 of file sundials_linsol_wrapper.h.

18.33.3 Constructor & Destructor Documentation

18.33.3.1 SUNLinSolKLU() [1/2]

```
SUNLinSolKLU (
    N_Vector x,
    SUNMatrix A )
```

Parameters

x	A template for cloning vectors needed within the solver.
A	sparse matrix

Definition at line 173 of file sundials_linsol_wrapper.cpp.

18.33.3.2 SUNLinSolKLU() [2/2]

```
SUNLinSolKLU (
    AmiVector const & x,
    int nnz,
    int sparsetype,
    StateOrdering ordering )
```

Parameters

x	A template for cloning vectors needed within the solver.
nnz	Number of non-zeros in matrix A
sparsetype	Sparse matrix type (CSC_MAT, CSR_MAT)
ordering	

Definition at line 179 of file sundials_linsol_wrapper.cpp.

Here is the call graph for this function:

18.33.4 Member Function Documentation

18.33.4.1 getMatrix() `SUNMatrix getMatrix ( ) const [override], [virtual]`

Returns

A

Reimplemented from SUNLinSolWrapper.
Definition at line 189 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.33.4.2 reInit()

```
void reInit (
    int nnz,
    int reinit_type )
```

For more details see sunlinsol/sunlinsol_klu.h

Parameters

nnz	Number of non-zeros
reinit_type	SUNKLU_REINIT_FULL or SUNKLU_REINIT_PARTIAL

Definition at line 191 of file sundials_linsol_wrapper.cpp.

Here is the call graph for this function:

18.33.4.3 setOrdering()

```
void setOrdering (
    StateOrdering ordering )
```

Parameters

ordering	

Definition at line 197 of file sundials_linsol_wrapper.cpp.
Here is the caller graph for this function:

18.34 SUNLinSolPCG Class Reference

SUNDIALS scaled preconditioned CG (Conjugate Gradient method) (PCG) solver.
`#include <sundials_linsol_wrapper.h>`
Inheritance diagram for SUNLinSolPCG:

Public Member Functions

- SUNLinSolPCG (N_Vector y, int pretype, int maxl)

Create PCG solver.

- int setATimes (void ∗A_data, ATimesFn ATimes)

  Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).
- int setPreconditioner (void ∗P_data, PSetupFn Pset, PSolveFn Psol)

  Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials↩_linearsolver.h).
- int setScalingVectors (N_Vector s, N_Vector nul)

  Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).
- int getNumIters () const

  Returns the number of linear iterations performed in the last 'Solve' call.
- realtype getResNorm () const

  Returns the final residual norm from the last 'Solve' call.
- N_Vector getResid () const

  Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Additional Inherited Members

18.34.1 Detailed Description

Definition at line 306 of file sundials_linsol_wrapper.h.

18.34.2 Constructor & Destructor Documentation

18.34.2.1 SUNLinSolPCG()

```
SUNLinSolPCG (
    N_Vector y,
    int pretype,
    int maxl )
```

Parameters

y	
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 203 of file sundials_linsol_wrapper.cpp.

18.34.3 Member Function Documentation

18.34.3.1 setATimes()

```
int setATimes (
    void * A_data,
    ATimesFn ATimes )
```

Parameters

A_data	
ATimes	

Returns

Definition at line 209 of file sundials_linsol_wrapper.cpp.

18.34.3.2 setPreconditioner()

```
int setPreconditioner (
    void * P_data,
    PSetupFn Pset,
    PSolveFn Psol )
```

Parameters

P_data	
Pset	
Psol	

Returns

Definition at line 213 of file sundials_linsol_wrapper.cpp.

18.34.3.3 setScalingVectors()

```
int setScalingVectors (
    N_Vector s,
    N_Vector nul )
```

Parameters

s	
nul	

Returns

Definition at line 218 of file sundials_linsol_wrapper.cpp.

18.34.3.4 getNumIters()

```
int getNumIters ( ) const
```

Returns

Number of iterations

Definition at line 222 of file sundials_linsol_wrapper.cpp.

18.34.3.5 getResNorm()

```
realtype getResNorm ( ) const
```

Returns

residual norm

Definition at line 224 of file sundials_linsol_wrapper.cpp.

18.34.3.6 getResid()

```
N_Vector getResid ( ) const
```

Returns

Definition at line 226 of file sundials_linsol_wrapper.cpp.

18.35 SUNLinSolSPBCGS Class Reference

SUNDIALS scaled preconditioned Bi-CGStab (Bi-Conjugate Gradient Stable method) (SPBCGS) solver.

`#include <sundials_linsol_wrapper.h>`

Inheritance diagram for SUNLinSolSPBCGS:

Public Member Functions

- SUNLinSolSPBCGS (N_Vector x, int pretype=PREC_NONE, int maxl=SUNSPBCGS_MAXL_DEFAULT)

  SUNLinSolSPBCGS.
- SUNLinSolSPBCGS (AmiVector const &x, int pretype=PREC_NONE, int maxl=SUNSPBCGS_MAXL_DE↩ FAULT)

  SUNLinSolSPBCGS.
- int setATimes (void ∗A_data, ATimesFn ATimes)

  Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).
- int setPreconditioner (void ∗P_data, PSetupFn Pset, PSolveFn Psol)

  Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials↩ _linearsolver.h).
- int setScalingVectors (N_Vector s, N_Vector nul)

  Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).
- int getNumIters () const

  Returns the number of linear iterations performed in the last 'Solve' call.
- realtype getResNorm () const

  Returns the final residual norm from the last 'Solve' call.
- N_Vector getResid () const

  Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Additional Inherited Members

18.35.1 Detailed Description

Definition at line 372 of file sundials_linsol_wrapper.h.

18.35.2 Constructor & Destructor Documentation

18.35.2.1 SUNLinSolSPBCGS() [1/2]

```
SUNLinSolSPBCGS (
    N_Vector x,
    int pretype = PREC_NONE,
    int maxl = SUNSPBCGS_MAXL_DEFAULT ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 228 of file sundials_linsol_wrapper.cpp.

18.35.2.2 SUNLinSolSPBCGS() [2/2]

```
SUNLinSolSPBCGS (
    AmiVector const & x,
    int pretype = PREC_NONE,
    int maxl = SUNSPBCGS_MAXL_DEFAULT ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 234 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.35.3 Member Function Documentation

18.35.3.1 setATimes()

```
int setATimes (
    void * A_data,
    ATimesFn ATimes )
```

Parameters

A_data	
ATimes	

Returns

Definition at line 240 of file sundials_linsol_wrapper.cpp.

18.35.3.2 setPreconditioner()

```
int setPreconditioner (
    void * P_data,
    PSetupFn Pset,
    PSolveFn Psol )
```

Parameters

P_data	
Pset	
Psol	

Returns

Definition at line 244 of file sundials_linsol_wrapper.cpp.

18.35.3.3 setScalingVectors() `int setScalingVectors ( N_Vector s, N_Vector nul )`

Parameters

s	
nul	

Returns

Definition at line 249 of file sundials_linsol_wrapper.cpp.

18.35.3.4 getNumIters() `int getNumIters ( ) const`

Returns

Number of iterations

Definition at line 253 of file sundials_linsol_wrapper.cpp.

18.35.3.5 getResNorm() `realtype getResNorm ( ) const`

Returns

residual norm

Definition at line 255 of file sundials_linsol_wrapper.cpp.

18.35.3.6 getResid() `N_Vector getResid ( ) const`

Returns

Definition at line 259 of file sundials_linsol_wrapper.cpp.

18.36 SUNLinSolSPFGMR Class Reference

SUNDIALS scaled preconditioned FGMRES (Flexible Generalized Minimal Residual method) (SPFGMR) solver.

`#include <sundials_linsol_wrapper.h>`

Inheritance diagram for SUNLinSolSPFGMR:

Public Member Functions

- SUNLinSolSPFGMR (AmiVector const &x, int pretype, int maxl)

  SUNLinSolSPFGMR.
- int setATimes (void ∗A_data, ATimesFn ATimes)

  Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).
- int setPreconditioner (void ∗P_data, PSetupFn Pset, PSolveFn Psol)

  Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials↩ _linearsolver.h).
- int setScalingVectors (N_Vector s, N_Vector nul)

  Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).
- int getNumIters () const

  Returns the number of linear iterations performed in the last 'Solve' call.
- realtype getResNorm () const

  Returns the final residual norm from the last 'Solve' call.
- N_Vector getResid () const

  Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Additional Inherited Members

18.36.1 Detailed Description

Definition at line 449 of file sundials_linsol_wrapper.h.

18.36.2 Constructor & Destructor Documentation

18.36.2.1 SUNLinSolSPFGMR()

```
SUNLinSolSPFGMR (
    AmiVector const & x,
    int pretype,
    int maxl )
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 261 of file sundials_linsol_wrapper.cpp.

18.36.3 Member Function Documentation

18.36.3.1 setATimes()

```
int setATimes (
    void * A_data,
    ATimesFn ATimes )
```

Parameters

A_data	
ATimes	

Returns

Definition at line 267 of file sundials_linsol_wrapper.cpp.

18.36.3.2 setPreconditioner()

```
int setPreconditioner (
    void * P_data,
    PSetupFn Pset,
    PSolveFn Psol )
```

Parameters

P_data	
Pset	
Psol	

Returns

Definition at line 271 of file sundials_linsol_wrapper.cpp.

18.36.3.3 setScalingVectors()

```
int setScalingVectors (
    N_Vector s,
    N_Vector nul )
```

Parameters

s	
nul	

Returns

Definition at line 276 of file sundials_linsol_wrapper.cpp.

18.36.3.4 getNumIters()

```
int getNumIters ( ) const
```

Returns

Number of iterations

Definition at line 280 of file sundials_linsol_wrapper.cpp.

18.36.3.5 getResNorm() `realtype getResNorm ( ) const`

Returns

residual norm

Definition at line 282 of file sundials_linsol_wrapper.cpp.

18.36.3.6 getResid() `N_Vector getResid ( ) const`

Returns

Definition at line 286 of file sundials_linsol_wrapper.cpp.

18.37 SUNLinSolSPGMR Class Reference

SUNDIALS scaled preconditioned GMRES (Generalized Minimal Residual method) solver (SPGMR).
`#include <sundials_linsol_wrapper.h>`
Inheritance diagram for SUNLinSolSPGMR:

Public Member Functions

- SUNLinSolSPGMR (AmiVector const &x, int pretype=PREC_NONE, int maxl=SUNSPGMR_MAXL_DEFA↩ ULT)

  Create SPGMR solver.
- int setATimes (void ∗A_data, ATimesFn ATimes)

  Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).
- int setPreconditioner (void ∗P_data, PSetupFn Pset, PSolveFn Psol)

  Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials↩ _linearsolver.h).
- int setScalingVectors (N_Vector s, N_Vector nul)

  Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).
- int getNumIters () const

  Returns the number of linear iterations performed in the last 'Solve' call.
- realtype getResNorm () const

Returns the final residual norm from the last 'Solve' call.

- N_Vector getResid () const

  Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Additional Inherited Members

18.37.1 Detailed Description

Definition at line 515 of file sundials_linsol_wrapper.h.

18.37.2 Constructor & Destructor Documentation

18.37.2.1 SUNLinSolSPGMR()

```
SUNLinSolSPGMR (
    AmiVector const & x,
    int pretype = PREC_NONE,
    int maxl = SUNSPGMR_MAXL_DEFAULT ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 288 of file sundials_linsol_wrapper.cpp.

18.37.3 Member Function Documentation

18.37.3.1 setATimes()

```
int setATimes (
    void * A_data,
    ATimesFn ATimes )
```

Parameters

A_data	
ATimes	

Returns

Definition at line 294 of file sundials_linsol_wrapper.cpp.

18.37.3.2 setPreconditioner()

```
int setPreconditioner (
    void * P_data,
    PSetupFn Pset,
    PSolveFn Psol )
```

Parameters

P_data	
Pset	
Psol	

Returns

Definition at line 298 of file sundials_linsol_wrapper.cpp.

18.37.3.3 setScalingVectors()

```
int setScalingVectors (
    N_Vector s,
    N_Vector nul )
```

Parameters

s	
nul	

Returns

Definition at line 303 of file sundials_linsol_wrapper.cpp.

18.37.3.4 getNumIters()

```
int getNumIters ( ) const
```

Returns

Number of iterations

Definition at line 307 of file sundials_linsol_wrapper.cpp.

18.37.3.5 getResNorm()

```
realtype getResNorm ( ) const
```

Returns

residual norm

Definition at line 309 of file sundials_linsol_wrapper.cpp.

18.37.3.6 getResid()

```
N_Vector getResid ( ) const
```

Returns

Definition at line 311 of file sundials_linsol_wrapper.cpp.

18.38 SUNLinSolSPTFQMR Class Reference

SUNDIALS scaled preconditioned TFQMR (Transpose-Free Quasi-Minimal Residual method) (SPTFQMR) solver.

```
#include <sundials_linsol_wrapper.h>
```

Inheritance diagram for SUNLinSolSPTFQMR:

Public Member Functions

- SUNLinSolSPTFQMR (N_Vector x, int pretype=PREC_NONE, int maxl=SUNSPTFQMR_MAXL_DEFAULT)

  Create SPTFQMR solver.
- SUNLinSolSPTFQMR (AmiVector const &x, int pretype=PREC_NONE, int maxl=SUNSPTFQMR_MAXL_↩ DEFAULT)

  Create SPTFQMR solver.
- int setATimes (void ∗A_data, ATimesFn ATimes)

  Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).
- int setPreconditioner (void ∗P_data, PSetupFn Pset, PSolveFn Psol)

  Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials↩ _linearsolver.h).
- int setScalingVectors (N_Vector s, N_Vector nul)

  Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).
- int getNumIters () const

  Returns the number of linear iterations performed in the last 'Solve' call.
- realtype getResNorm () const

  Returns the final residual norm from the last 'Solve' call.
- N_Vector getResid () const

  Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Additional Inherited Members

18.38.1 Detailed Description

Definition at line 582 of file sundials_linsol_wrapper.h.

18.38.2 Constructor & Destructor Documentation

18.38.2.1 SUNLinSolSPTFQMR() [1/2]

```
SUNLinSolSPTFQMR (
    N_Vector x,
    int pretype = PREC_NONE,
    int maxl = SUNSPTFQMR_MAXL_DEFAULT ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 313 of file sundials_linsol_wrapper.cpp.

18.38.2.2 SUNLinSolSPTFQMR() [2/2]

```
SUNLinSolSPTFQMR (
    AmiVector const & x,
    int pretype = PREC_NONE,
    int maxl = SUNSPTFQMR_MAXL_DEFAULT ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.
pretype	Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
maxl	Maximum number of solver iterations

Definition at line 319 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.38.3 Member Function Documentation

18.38.3.1 setATimes()

```
int setATimes (
    void * A_data,
    ATimesFn ATimes )
```

Parameters

A_data	
ATimes	

Returns

Definition at line 325 of file sundials_linsol_wrapper.cpp.

18.38.3.2 setPreconditioner()

```
int setPreconditioner (
    void * P_data,
    PSetupFn Pset,
    PSolveFn Psol )
```

Parameters

P_data	
Pset	
Psol	

Returns

Definition at line 329 of file sundials_linsol_wrapper.cpp.

18.38.3.3 setScalingVectors()

```
int setScalingVectors (
    N_Vector s,
    N_Vector nul )
```

Parameters

s	
nul	

Returns

Definition at line 334 of file sundials_linsol_wrapper.cpp.

18.38.3.4 getNumIters()

```
int getNumIters ( ) const
```

Returns

Number of iterations

Definition at line 338 of file sundials_linsol_wrapper.cpp.

18.38.3.5 getResNorm()

```
realtype getResNorm ( ) const
```

Returns

residual norm

Definition at line 342 of file sundials_linsol_wrapper.cpp.

18.38.3.6 getResid()

```
N_Vector getResid ( ) const
```

Returns

Definition at line 346 of file sundials_linsol_wrapper.cpp.

18.39 SUNLinSolWrapper Class Reference

A RAII wrapper for SUNLinearSolver structs.

```
#include <sundials_linsol_wrapper.h>
```

Inheritance diagram for SUNLinSolWrapper:

Public Member Functions

- SUNLinSolWrapper (SUNLinearSolver linsol)

  Wrap existing SUNLinearSolver.
- SUNLinSolWrapper (const SUNLinSolWrapper &other)=delete

  Copy constructor.
- SUNLinSolWrapper (SUNLinSolWrapper &&other) noexcept

  Move constructor.
- SUNLinSolWrapper & operator= (const SUNLinSolWrapper &other)=delete

  Copy assignment.
- SUNLinSolWrapper & operator= (SUNLinSolWrapper &&other) noexcept

  Move assignment.
- SUNLinearSolver get () const

  Returns the wrapped SUNLinSol.
- SUNLinearSolver_Type getType () const

  Returns an identifier for the linear solver type.
- void setup (SUNMatrix A) const

  Performs any linear solver setup needed, based on an updated system matrix A.
- void setup (const SUNMatrixWrapper &A) const

  Performs any linear solver setup needed, based on an updated system matrix A.
- int Solve (SUNMatrix A, N_Vector x, N_Vector b, realtype tol) const

Solves a linear system A∗x = b.

- long int getLastFlag () const

  Returns the last error flag encountered within the linear solver.

- int space (long int ∗lenrwLS, long int ∗leniwLS) const

  Returns the integer and real workspace sizes for the linear solver.

- virtual SUNMatrix getMatrix () const

  Get the matrix A (matrix solvers only).

Protected Member Functions

- int initialize ()

  Performs linear solver initialization (assumes that all solver-specific options have been set).

Protected Attributes

- SUNLinearSolver solver_ {nullptr}

18.39.1 Detailed Description

For details on member functions see documentation in sunlinsol/sundials_linearsolver.h.
Definition at line 31 of file sundials_linsol_wrapper.h.

18.39.2 Constructor & Destructor Documentation

18.39.2.1 SUNLinSolWrapper() [1/3] SUNLinSolWrapper (SUNLinearSolver *linsol*) [explicit]

Parameters

linsol	

Definition at line 10 of file sundials_linsol_wrapper.cpp.

18.39.2.2 SUNLinSolWrapper() [2/3] SUNLinSolWrapper (const SUNLinSolWrapper & *other*) [delete]

Parameters

other	

18.39.2.3 SUNLinSolWrapper() [3/3] SUNLinSolWrapper (SUNLinSolWrapper && *other*) [noexcept]

Parameters

other	

Definition at line 17 of file sundials_linsol_wrapper.cpp.

18.39.3 Member Function Documentation

18.39.3.1 operator=() [1/2] SUNLinSolWrapper& operator= (const SUNLinSolWrapper & *other*) [delete]

Parameters

other	

Returns

18.39.3.2 operator=() [2/2] SUNLinSolWrapper& operator= (SUNLinSolWrapper && *other*) [noexcept]

Parameters

other	

Returns

18.39.3.3 get() SUNLinearSolver get ( ) const

Returns

SUNLinearSolver

Definition at line 21 of file sundials_linsol_wrapper.cpp.

18.39.3.4 getType() SUNLinearSolver_Type getType ( ) const

Returns

Definition at line 23 of file sundials_linsol_wrapper.cpp.

18.39.3.5 setup() [1/2] void setup (SUNMatrix *A*) const

Parameters

A	

Definition at line 34 of file sundials_linsol_wrapper.cpp.

Here is the caller graph for this function:

18.39.3.6 setup() [2/2]

```
void setup (
    const SUNMatrixWrapper & A ) const
```

Parameters

A	

Definition at line 40 of file sundials_linsol_wrapper.cpp.
Here is the call graph for this function:

18.39.3.7 Solve()

```
int Solve (
    SUNMatrix A,
    N_Vector x,
    N_Vector b,
    realtype tol ) const
```

Parameters

A	
x	A template for cloning vectors needed within the solver.
b	
tol	Tolerance (weighted 2-norm), iterative solvers only

Returns

error flag

Definition at line 42 of file sundials_linsol_wrapper.cpp.

18.39.3.8 getLastFlag() `long getLastFlag ( ) const`

Returns

error flag

Definition at line 46 of file sundials_linsol_wrapper.cpp.

18.39.3.9 space()

```
int space (
    long int * lenrwLS,
    long int * leniwLS ) const
```

Parameters

lenrwLS	output argument for size of real workspace
leniwLS	output argument for size of interger workspace

Returns

workspace size

Definition at line 50 of file sundials_linsol_wrapper.cpp.

18.39.3.10 getMatrix() `SUNMatrix getMatrix ( ) const [virtual]`

Returns

A

Reimplemented in SUNLinSolKLU, SUNLinSolDense, and SUNLinSolBand.
Definition at line 54 of file sundials_linsol_wrapper.cpp.

18.39.3.11 initialize() `int initialize ( ) [protected]`

Returns

error code

Definition at line 27 of file sundials_linsol_wrapper.cpp.

18.39.4 Member Data Documentation

18.39.4.1 solver_ `SUNLinearSolver solver_ {nullptr} [protected]`

Wrapped solver
Definition at line 134 of file sundials_linsol_wrapper.h.

18.40 SUNMatrixWrapper Class Reference

A RAII wrapper for SUNMatrix structs.
`#include <sundials_matrix_wrapper.h>`

Public Member Functions

- SUNMatrixWrapper (int M, int N, int NNZ, int sparsetype)

  Create sparse matrix. See SUNSparseMatrix in sunmatrix_sparse.h.
- SUNMatrixWrapper (int M, int N)

  Create dense matrix. See SUNDenseMatrix in sunmatrix_dense.h.
- SUNMatrixWrapper (int M, int ubw, int lbw)

  Create banded matrix. See SUNBandMatrix in sunmatrix_band.h.
- SUNMatrixWrapper (const SUNMatrixWrapper &A, realtype droptol, int sparsetype)

  Create sparse matrix from dense or banded matrix. See SUNSparseFromDenseMatrix and SUNSparseFromBand↩ Matrix in sunmatrix_sparse.h.
- SUNMatrixWrapper (SUNMatrix mat)

  Wrap existing SUNMatrix.
- SUNMatrixWrapper (const SUNMatrixWrapper &other)

  Copy constructor.
- SUNMatrixWrapper (SUNMatrixWrapper &&other)

  Move constructor.
- SUNMatrixWrapper & operator= (const SUNMatrixWrapper &other)

  Copy assignment.
- SUNMatrixWrapper & operator= (SUNMatrixWrapper &&other)

  Move assignment.
- realtype ∗ data () const

  Access raw data.
- SUNMatrix get () const

  Get the wrapped SUNMatrix.
- sunindextype rows () const

  Get the number of rows.
- sunindextype columns () const

  Get the number of columns.
- sunindextype nonzeros () const

  Get the number of non-zero elements (sparse matrices only)
- sunindextype ∗ indexvals () const

  Get the index values of a sparse matrix.
- sunindextype ∗ indexptrs () const

  Get the index pointers of a sparse matrix.
- int sparsetype () const

  Get the type of sparse matrix.
- void reset ()

  reset data to zeroes
- void scale (realtype a)

  multiply with a scalar (in-place)
- void multiply (N_Vector c, const_N_Vector b) const

  N_Vector interface for multiply.
- void multiply (gsl::span< realtype > c, gsl::span< const realtype > b) const

  Perform matrix vector multiplication c += A∗b.
- void multiply (N_Vector c, const N_Vector b, gsl::span< const int > cols, bool transpose) const

  Perform reordered matrix vector multiplication c += A[:,cols]∗b.
- void multiply (gsl::span< realtype > c, gsl::span< const realtype > b, gsl::span< const int > cols, bool transpose) const

  Perform reordered matrix vector multiplication c += A[:,cols]∗b.
- void sparse_multiply (SUNMatrixWrapper ∗C, SUNMatrixWrapper ∗B) const

  Perform matrix matrix multiplication C[:, :] += A ∗ B for sparse A, B, C.
- void zero ()

  Set to 0.0.

18.40.1 Detailed Description

This can create dense, sparse, or banded matrices using the respective constructor.
Definition at line 22 of file sundials_matrix_wrapper.h.

18.40.2 Constructor & Destructor Documentation

18.40.2.1 SUNMatrixWrapper() [1/7]

```
SUNMatrixWrapper (
    int M,
    int N,
    int NNZ,
    int sparsetype )
```

Parameters

M	Number of rows
N	Number of columns
NNZ	Number of nonzeros
sparsetype	Sparse type

Definition at line 11 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

18.40.2.2 SUNMatrixWrapper() [2/7]

```
SUNMatrixWrapper (
    int M,
    int N )
```

Parameters

M	Number of rows
N	Number of columns

Definition at line 24 of file sundials_matrix_wrapper.cpp.

18.40.2.3 SUNMatrixWrapper() [3/7]

```
SUNMatrixWrapper (
    int M,
    int ubw,
    int lbw )
```

Parameters

M	Number of rows and columns

Parameters

ubw	Upper bandwidth
lbw	Lower bandwidth

Definition at line 32 of file sundials_matrix_wrapper.cpp.

18.40.2.4 SUNMatrixWrapper() [4/7]

```
SUNMatrixWrapper (
    const SUNMatrixWrapper & A,
    realtype droptol,
    int sparsetype )
```

Parameters

A	Wrapper for dense matrix
droptol	tolerance for dropping entries
sparsetype	Sparse type

Definition at line 40 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

18.40.2.5 SUNMatrixWrapper() [5/7]

```
SUNMatrixWrapper (
    SUNMatrix mat ) [explicit]
```

Parameters

mat	

Definition at line 64 of file sundials_matrix_wrapper.cpp.

18.40.2.6 SUNMatrixWrapper() [6/7]

```
SUNMatrixWrapper (
    const SUNMatrixWrapper & other )
```

Parameters

other	

Definition at line 73 of file sundials_matrix_wrapper.cpp.

18.40.2.7 SUNMatrixWrapper() [7/7] `SUNMatrixWrapper ( SUNMatrixWrapper && other )`

Parameters

other	

Definition at line 85 of file sundials_matrix_wrapper.cpp.

18.40.3 Member Function Documentation

18.40.3.1 operator=() [1/2] `SUNMatrixWrapper & operator= ( const SUNMatrixWrapper & other )`

Parameters

other	

Returns

Definition at line 90 of file sundials_matrix_wrapper.cpp.

18.40.3.2 operator=() [2/2] `SUNMatrixWrapper & operator= ( SUNMatrixWrapper && other )`

Parameters

other	

Returns

Definition at line 96 of file sundials_matrix_wrapper.cpp.

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

Returns

raw data pointer

Definition at line 103 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.4 get() `SUNMatrix get ( ) const`

Returns

SlsMat

Definition at line 417 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.5 rows() `sunindextype rows ( ) const`

Returns

number

Definition at line 107 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.6 columns() `sunindextype columns ( ) const`

Returns

number

Definition at line 126 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.7 nonzeros() `sunindextype nonzeros ( ) const`

Returns

number

Definition at line 145 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.8 indexvals() `sunindextype * indexvals ( ) const`

Returns

index array

Definition at line 158 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.9 indexptrs() `sunindextype * indexptrs ( ) const`

Returns

index array

Definition at line 162 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.10 sparsetype() `int sparsetype ( ) const`

Returns

index array

Definition at line 166 of file sundials_matrix_wrapper.cpp.
Here is the caller graph for this function:

18.40.3.11 scale()

```
void scale (
    realtype a )
```

Parameters

a	scalar value to multiply matrix

Definition at line 177 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.40.3.12 multiply() [1/4]

```
void multiply (
    N_Vector c,
    const_N_Vector b ) const
```

Parameters

c	output vector, may already contain values
b	multiplication vector

Definition at line 185 of file sundials_matrix_wrapper.cpp.

Here is the caller graph for this function:

18.40.3.13 multiply() [2/4]

```
void multiply (
    gsl::span< realtype > c,
    gsl::span< const realtype > b ) const
```

Parameters

c	output vector, may already contain values
b	multiplication vector

Definition at line 190 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

18.40.3.14 multiply() [3/4]

```
void multiply (
    N_Vector c,
    const N_Vector b,
    gsl::span< const int > cols,
    bool transpose ) const
```

Parameters

c	output vector, may already contain values
b	multiplication vector
cols	int vector for column reordering
transpose	bool transpose A before multiplication

Definition at line 251 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

18.40.3.15 multiply() [4/4]

```
void multiply (
    gsl::span< realtype > c,
    gsl::span< const realtype > b,
    gsl::span< const int > cols,
    bool transpose ) const
```

Parameters

c	output vector, may already contain values
b	multiplication vector
cols	int vector for column reordering
transpose	bool transpose A before multiplication

Definition at line 260 of file sundials_matrix_wrapper.cpp.

Here is the call graph for this function:

18.40.3.16 sparse_multiply()

```
void sparse_multiply (
    SUNMatrixWrapper * C,
    SUNMatrixWrapper * B ) const
```

Parameters

C	output matrix, may already contain values
B	multiplication matrix

Definition at line 319 of file sundials_matrix_wrapper.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

18.41 SUNNonLinSolFixedPoint Class Reference

SUNDIALS Fixed point non-linear solver to solve G(y) = y.

`#include <sundials_linsol_wrapper.h>`

Inheritance diagram for SUNNonLinSolFixedPoint:

Public Member Functions

- SUNNonLinSolFixedPoint (const_N_Vector x, int m=0)

  Create fixed-point solver.
- SUNNonLinSolFixedPoint (int count, const_N_Vector x, int m=0)

  Create fixed-point solver for use with sensitivity analysis.
- int getSysFn (SUNNonlinSolSysFn ∗SysFn) const

  Get function to evaluate the fixed point function G(y) = y.

Additional Inherited Members

18.41.1 Detailed Description

Definition at line 831 of file sundials_linsol_wrapper.h.

18.41.2 Constructor & Destructor Documentation

18.41.2.1 SUNNonLinSolFixedPoint() [1/2]

```
SUNNonLinSolFixedPoint (
    const_N_Vector x,
    int m = 0 ) [explicit]
```

Parameters

x	template for cloning vectors needed within the solver.
m	number of acceleration vectors to use

Definition at line 362 of file sundials_linsol_wrapper.cpp.

18.41.2.2 SUNNonLinSolFixedPoint() [2/2]

```
SUNNonLinSolFixedPoint (
    int count,
    const_N_Vector x,
    int m = 0 )
```

Parameters

count	Number of vectors in the nonlinear solve. When integrating a system containing Ns sensitivities the value of count is: • Ns+1 if using a simultaneous corrector approach. • Ns if using a staggered corrector approach.
x	template for cloning vectors needed within the solver.
m	number of acceleration vectors to use

Definition at line 366 of file sundials_linsol_wrapper.cpp.

18.41.3 Member Function Documentation

18.41.3.1 getSysFn()

```
int getSysFn (
    SUNNonlinSolSysFn * SysFn ) const
```

Parameters

SysFn	

Returns

Definition at line 370 of file sundials_linsol_wrapper.cpp.

18.42 SUNNonLinSolNewton Class Reference

SUNDIALS Newton non-linear solver to solve F (y) = 0.

```
#include <sundials_linsol_wrapper.h>
```

Inheritance diagram for SUNNonLinSolNewton:

Public Member Functions

- SUNNonLinSolNewton (N_Vector x)

  Create Newton solver.
- SUNNonLinSolNewton (int count, N_Vector x)

  Create Newton solver for enabled sensitivity analysis.
- int getSysFn (SUNNonlinSolSysFn *SysFn) const

  Get function to evaluate the nonlinear residual function F(y) = 0.

Additional Inherited Members

18.42.1 Detailed Description

Definition at line 801 of file sundials_linsol_wrapper.h.

18.42.2 Constructor & Destructor Documentation

18.42.2.1 SUNNonLinSolNewton() [1/2]

```
SUNNonLinSolNewton (
    N_Vector x ) [explicit]
```

Parameters

x	A template for cloning vectors needed within the solver.

Definition at line 348 of file sundials_linsol_wrapper.cpp.

18.42.2.2 SUNNonLinSolNewton() [2/2]

```
SUNNonLinSolNewton (
    int count,
    N_Vector x )
```

Parameters

count	Number of vectors in the nonlinear solve. When integrating a system containing Ns sensitivities the value of count is: • Ns+1 if using a simultaneous corrector approach. • Ns if using a staggered corrector approach.

Parameters

x	A template for cloning vectors needed within the solver.

Definition at line 352 of file sundials_linsol_wrapper.cpp.

18.42.3 Member Function Documentation

18.42.3.1 getSysFn() `int getSysFn ( SUNNonlinSolSysFn * SysFn ) const`

Parameters

SysFn	

Returns

Definition at line 358 of file sundials_linsol_wrapper.cpp.

18.43 SUNNonLinSolWrapper Class Reference

A RAII wrapper for SUNNonLinearSolver structs which solve the nonlinear system F (y) = 0 or G(y) = y.

`#include <sundials_linsol_wrapper.h>`

Inheritance diagram for SUNNonLinSolWrapper:

Public Member Functions

- SUNNonLinSolWrapper (SUNNonlinearSolver sol)

  SUNNonLinSolWrapper from existing SUNNonlinearSolver.
- SUNNonLinSolWrapper (const SUNNonLinSolWrapper &other)=delete

  Copy constructor.
- SUNNonLinSolWrapper (SUNNonLinSolWrapper &&other) noexcept

  Move constructor.
- SUNNonLinSolWrapper & operator= (const SUNNonLinSolWrapper &other)=delete

  Copy assignment.
- SUNNonLinSolWrapper & operator= (SUNNonLinSolWrapper &&other) noexcept

  Move assignment.

- SUNNonlinearSolver get () const

  Get the wrapped SUNNonlinearSolver.
- SUNNonlinearSolver_Type getType () const

  Get type ID of the solver.
- int setup (N_Vector y, void ∗mem)

  Setup solver.
- int Solve (N_Vector y0, N_Vector y, N_Vector w, realtype tol, bool callLSetup, void ∗mem)

  Solve the nonlinear system F (y) = 0 or G(y) = y.
- int setSysFn (SUNNonlinSolSysFn SysFn)

  Set function to evaluate the nonlinear residual function F(y) = 0 or the fixed point function G(y) = y.
- int setLSetupFn (SUNNonlinSolLSetupFn SetupFn)

  Set linear solver setup function.
- int setLSolveFn (SUNNonlinSolLSolveFn SolveFn)

  Set linear solver solve function.
- int setConvTestFn (SUNNonlinSolConvTestFn CTestFn, void ∗ctest_data)

  Set function to test for convergence.
- int setMaxIters (int maxiters)

  Set maximum number of non-linear iterations.
- long int getNumIters () const

  getNumIters
- int getCurIter () const

  getCurIter
- long int getNumConvFails () const

  getNumConvFails

Protected Member Functions

- void initialize ()

  initialize

Protected Attributes

- SUNNonlinearSolver solver = nullptr

18.43.1 Detailed Description

Definition at line 659 of file sundials_linsol_wrapper.h.

18.43.2 Constructor & Destructor Documentation

18.43.2.1 SUNNonLinSolWrapper() [1/3] SUNNonLinSolWrapper (SUNNonlinearSolver *sol*) [explicit]

Parameters

sol	

Definition at line 56 of file sundials_linsol_wrapper.cpp.

18.43.2.2 SUNNonLinSolWrapper() [2/3] SUNNonLinSolWrapper (const SUNNonLinSolWrapper & *other*) [delete]

Parameters

other	

18.43.2.3 SUNNonLinSolWrapper() [3/3] SUNNonLinSolWrapper (SUNNonLinSolWrapper && *other*) [noexcept]

Parameters

other	

Definition at line 64 of file sundials_linsol_wrapper.cpp.

18.43.3 Member Function Documentation

18.43.3.1 operator=() [1/2] SUNNonLinSolWrapper& operator= (const SUNNonLinSolWrapper & *other*) [delete]

Parameters

other	

Returns

18.43.3.2 operator=() [2/2] SUNNonLinSolWrapper & operator= (SUNNonLinSolWrapper && *other*) [noexcept]

Parameters

other	

Returns

Definition at line 68 of file sundials_linsol_wrapper.cpp.

18.43.3.3 get() SUNNonlinearSolver get ( ) const

Returns

SUNNonlinearSolver

Definition at line 74 of file sundials_linsol_wrapper.cpp.

18.43.3.4 getType() SUNNonlinearSolver_Type getType ( ) const

Returns

Definition at line 76 of file sundials_linsol_wrapper.cpp.

18.43.3.5 setup()

```
int setup (
    N_Vector y,
    void * mem )
```

Parameters

y	the initial iteration passed to the nonlinear solver.
mem	the sundials integrator memory structure.

Returns

Definition at line 80 of file sundials_linsol_wrapper.cpp.

18.43.3.6 Solve()

```
int Solve (
    N_Vector y0,
    N_Vector y,
    N_Vector w,
    realtype tol,
    bool callLSetup,
    void * mem )
```

Parameters

y0	the initial iterate for the nonlinear solve. This must remain unchanged throughout the solution process.
y	the solution to the nonlinear system
w	the solution error weight vector used for computing weighted error norms.
tol	the requested solution tolerance in the weighted root-mean- squared norm.
callLSetup	a flag indicating that the integrator recommends for the linear solver setup function to be called.
mem	the sundials integrator memory structure.

Returns

Definition at line 87 of file sundials_linsol_wrapper.cpp.

18.43.3.7 setSysFn()

```
int setSysFn (
    SUNNonlinSolSysFn SysFn )
```

Parameters

SysFn	

Returns

Definition at line 92 of file sundials_linsol_wrapper.cpp.

18.43.3.8 setLSetupFn()

```
int setLSetupFn (
    SUNNonlinSolLSetupFn SetupFn )
```

Parameters

SetupFn	

Returns

Definition at line 96 of file sundials_linsol_wrapper.cpp.

18.43.3.9 setLSolveFn()

```
int setLSolveFn (
    SUNNonlinSolLSolveFn SolveFn )
```

Parameters

SolveFn	

Returns

Definition at line 100 of file sundials_linsol_wrapper.cpp.

18.43.3.10 setConvTestFn()

```
int setConvTestFn (
    SUNNonlinSolConvTestFn CTestFn,
    void * ctest_data )
```

Parameters

CTestFn	
ctest_data	

Returns

Definition at line 104 of file sundials_linsol_wrapper.cpp.

18.43.3.11 setMaxIters()

```
int setMaxIters (
    int maxiters )
```

Parameters

maxiters	

Returns

Definition at line 109 of file sundials_linsol_wrapper.cpp.

18.43.3.12 getNumIters() `long getNumIters ( ) const`

Returns

Definition at line 113 of file sundials_linsol_wrapper.cpp.

18.43.3.13 getCurIter() `int getCurIter ( ) const`

Returns

Definition at line 122 of file sundials_linsol_wrapper.cpp.

18.43.3.14 getNumConvFails() `long getNumConvFails ( ) const`

Returns

Definition at line 131 of file sundials_linsol_wrapper.cpp.

18.43.4 Member Data Documentation

18.43.4.1 solver `SUNNonlinearSolver solver = nullptr [protected]`

the wrapper solver
Definition at line 794 of file sundials_linsol_wrapper.h.

18.44 amidata Class Reference

AMIDATA provides a data container to pass experimental data to the simulation routine for likelihood computation. when any of the properties are updated, the class automatically checks consistency of dimension and updates related properties and initialises them with NaNs.
Inheritance diagram for amidata:

Public Member Functions

- amidata (matlabtypesubstitute varargin)

  amidata creates an amidata container for experimental data with specified dimensions amidata.

Public Attributes

- matlabtypesubstitute nt = 0

  number of timepoints
- matlabtypesubstitute ny = 0

  number of observables
- matlabtypesubstitute nz = 0

  number of event observables
- matlabtypesubstitute ne = 0

  number of events
- matlabtypesubstitute nk = 0

  number of conditions/constants
- matlabtypesubstitute t = double.empty("")

  timepoints of observations
- matlabtypesubstitute Y = double.empty("")

  observations
- matlabtypesubstitute Sigma_Y = double.empty("")

  standard deviation of observations
- matlabtypesubstitute Z = double.empty("")

  event observations
- matlabtypesubstitute Sigma_Z = double.empty("")

  standard deviation of event observations
- matlabtypesubstitute condition = double.empty("")

  experimental condition
- matlabtypesubstitute conditionPreequilibration = double.empty("")

  experimental condition for preequilibration
- matlabtypesubstitute reinitializeStates = false

  reinitialize states based on fixed parameters after preeq.?

18.44.1 Detailed Description

Definition at line 17 of file amidata.m.

18.44.2 Constructor & Destructor Documentation

18.44.2.1 amidata()

```
amidata (
    matlabtypesubstitute varargin )
```

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

fields

t [nt,1] Y [nt,ny] Sigma_Y [nt,ny] Z [ne,nz] Sigma_Z [ne,nz] condition [nk,1] conditionPreequilibration [nk,1] if some fields are missing the function will try to initialise them with NaNs with consistent dimensions

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

Parameters

varargin	

Definition at line 138 of file amidata.m.

18.44.3 Member Data Documentation

18.44.3.1 nt `nt = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 31 of file amidata.m.

18.44.3.2 ny `ny = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 39 of file amidata.m.

18.44.3.3 nz `nz = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 47 of file amidata.m.

18.44.3.4 ne `ne = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 55 of file amidata.m.

18.44.3.5 nk `nk = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 63 of file amidata.m.

18.44.3.6 t `t = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 71 of file amidata.m.

18.44.3.7 Y `Y = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 79 of file amidata.m.

18.44.3.8 Sigma_Y `Sigma_Y = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 87 of file amidata.m.

18.44.3.9 Z `Z = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 95 of file amidata.m.

18.44.3.10 Sigma_Z `Sigma_Z = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 103 of file amidata.m.

18.44.3.11 condition `condition = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 111 of file amidata.m.

18.44.3.12 conditionPreequilibration `conditionPreequilibration = double.empty("")`

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 119 of file amidata.m.

18.44.3.13 reinitializeStates `reinitializeStates = false`

Default: false
Definition at line 127 of file amidata.m.

18.45 amievent Class Reference

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

Public Member Functions

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

  amievent constructs an amievent object from the provided input.
- mlhsInnerSubst< matlabtypesubstitute > setHflag (::double hflag)

  gethflag sets the hflag property.

Public Attributes

- ::symbolic trigger = sym.empty("")

  the trigger function activates the event on every zero crossing
- ::symbolic bolus = sym.empty("")

  the bolus function defines the change in states that is applied on every event occurence
- ::symbolic z = sym.empty("")

  output function for the event
- matlabtypesubstitute hflag = logical.empty("")

  flag indicating that a heaviside function is present, this helps to speed up symbolic computations

18.45.1 Detailed Description

Definition at line 17 of file amievent.m.

18.45.2 Constructor & Destructor Documentation

18.45.2.1 amievent()

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

Parameters

trigger	trigger function, the event will be triggered on at all roots of this function
bolus	the bolus that will be added to all states on every occurence of the event
z	the event output that will be reported on every occurence of the event

Definition at line 75 of file amievent.m.

18.45.3 Member Function Documentation

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

Parameters

hflag	value for the hflag property

Return values

this	updated event definition object

Definition at line 18 of file setHflag.m.
Here is the caller graph for this function:

18.45.4 Member Data Documentation

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

Note

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

`Matlab documentation of property attributes.`
Default: sym.empty("")

Definition at line 27 of file amievent.m.

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

Note

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

`Matlab documentation of property attributes.`
Default: sym.empty("")

Definition at line 38 of file amievent.m.

18.45.4.3 z `z = sym.empty("")`

Note

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

`Matlab documentation of property attributes.`

Default: sym.empty("")

Definition at line 49 of file amievent.m.

18.45.4.4 hflag `hflag = logical.empty("")`

Note

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

`Matlab documentation of property attributes.`

Default: logical.empty("")

Definition at line 60 of file amievent.m.

18.46 amifun Class Reference

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

Public Member Functions

- amifun (matlabtypesubstitute funstr, matlabtypesubstitute model)

  amievent constructs an amifun object from the provided input.
- noret::substitute writeCcode_sensi (::amimodel model,::fileid fid)

  writeCcode_sensi is a wrapper for writeCcode which loops over parameters and reduces overhead by check nonzero values
- noret::substitute writeCcode (::amimodel model,::fileid fid)

  writeCcode is a wrapper for gccode which initialises data and reduces overhead by check nonzero values
- noret::substitute writeMcode (::amimodel model)

  writeMcode generates matlab evaluable code for specific model functions
- noret::substitute gccode (::amimodel model,::fileid fid)

  gccode transforms symbolic expressions into c code and writes the respective expression into a specified file
- mlhsInnerSubst< matlabtypesubstitute > getDeps (::amimodel model)

  getDeps populates the sensiflag for the requested function
- mlhsInnerSubst< matlabtypesubstitute > getArgs (::amimodel model)

  getFArgs populates the fargstr property with the argument string of the respective model function (if applicable). model functions are not wrapped versions of functions which have a model specific name and for which the call is solver specific.
- mlhsInnerSubst< matlabtypesubstitute > getNVecs ()

  getfunargs populates the nvecs property with the names of the N_Vector elements which are required in the execution of the function (if applicable). the information is directly extracted from the argument string
- mlhsInnerSubst< matlabtypesubstitute > getCVar ()

  getCVar populates the cvar property
- mlhsInnerSubst< matlabtypesubstitute > getSensiFlag ()

  getSensiFlag populates the sensiflag property
- mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >,mlhsInnerSubst<::amimodel > > getSyms (↩ ::amimodel model)

  getSyms computes the symbolic expression for the requested function

Public Attributes

- ::symbolic sym = sym("[ ]")

  symbolic definition struct
- ::symbolic sym_noopt = sym("[ ]")

  symbolic definition which was not optimized (no dependencies on w)
- ::symbolic strsym = sym("[ ]")

  short symbolic string which can be used for the reuse of precomputed values
- ::symbolic strsym_old = sym("[ ]")

  short symbolic string which can be used for the reuse of old values
- ::char funstr = char.empty("")

  name of the model
- ::char cvar = char.empty("")

  name of the c variable
- ::char argstr = char.empty("")

  argument string (solver specific)
- ::cell deps = cell.empty("")

  dependencies on other functions
- matlabtypesubstitute nvecs = cell.empty("")

  nvec dependencies
- matlabtypesubstitute sensiflag = logical.empty("")

  indicates whether the function is a sensitivity or derivative with respect to parameters

18.46.1 Detailed Description

Definition at line 17 of file amifun.m.

18.46.2 Constructor & Destructor Documentation

18.46.2.1 amifun()

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

Parameters

funstr	name of the requested function
model	amimodel object which carries all symbolic definitions to construct the function

Definition at line 111 of file amifun.m.

18.46.3 Member Function Documentation

18.46.3.1 writeCcode_sensi()

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

Parameters

model	model defintion object
fid	file id in which the final expression is written

Return values

fid	void

Definition at line 18 of file writeCcode_sensi.m.

18.46.3.2 writeCcode()

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

Parameters

model	model defintion object
fid	file id in which the final expression is written

Return values

fid	void

Definition at line 18 of file writeCcode.m.
Here is the call graph for this function:

18.46.3.3 writeMcode()

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

Parameters

model	model defintion object

Return values

model	void

Definition at line 18 of file writeMcode.m.

Here is the call graph for this function:

18.46.3.4 gccode()

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

Parameters

model	model definition object
fid	file id in which the expression should be written

Return values

this	function definition object

Definition at line 18 of file gccode.m.
Here is the caller graph for this function:

18.46.3.5 getDeps()

```
mlhsInnerSubst<::amifun > getDeps (
    ::amimodel model )
```

Parameters

model	model definition object

Return values

this	updated function definition object

Definition at line 18 of file getDeps.m.

18.46.3.6 getArgs() `mlhsInnerSubst<::amifun > getArgs ( ::amimodel model )`

Parameters

model	model definition object

Return values

this	updated function definition object

Definition at line 18 of file getArgs.m.

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

Return values

this	updated function definition object

Definition at line 18 of file getNVecs.m.

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

Return values

this	updated function definition object

Definition at line 18 of file getCVar.m.

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

Return values

this	updated function definition object

Definition at line 18 of file getSensiFlag.m.

18.46.3.10 getSyms() `mlhsSubst< mlhsInnerSubst<::amifun >,mlhsInnerSubst<::amimodel > > getSyms ( ::amimodel model )`

Parameters

model	model definition object

Return values

this	updated function definition object
model	updated model definition object

Definition at line 18 of file getSyms.m.
Here is the call graph for this function:

18.46.4 Member Data Documentation

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

Default: sym("[]")
Definition at line 27 of file amifun.m.

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

Default: sym("[]")
Definition at line 35 of file amifun.m.

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

Default: sym("[]")
Definition at line 43 of file amifun.m.

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

Default: sym("[]")
Definition at line 51 of file amifun.m.

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

Default: char.empty("")
Definition at line 59 of file amifun.m.

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

Default: char.empty("")
Definition at line 67 of file amifun.m.

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

Default: char.empty("")

Definition at line 75 of file amifun.m.

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

Default: cell.empty("")
Definition at line 83 of file amifun.m.

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

Default: cell.empty("")
Definition at line 91 of file amifun.m.

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

Default: logical.empty("")
Definition at line 99 of file amifun.m.

18.47 amimodel Class Reference

AMIMODEL carries all model definitions including functions and events.
Inheritance diagram for amimodel:

Public Member Functions

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

  amimodel initializes the model object based on the provided symfun and modelname
- noret::substitute updateRHS (matlabtypesubstitute xdot)

  updateRHS updates the private fun property .fun.xdot.sym (right hand side of the differential equation)
- noret::substitute updateModelName (matlabtypesubstitute modelname)

  updateModelName updates the modelname
- noret::substitute updateWrapPath (matlabtypesubstitute wrap_path)

  updateModelName updates the modelname
- noret::substitute parseModel ()

  parseModel parses the model definition and computes all necessary symbolic expressions.
- noret::substitute generateC ()

  generateC generates the c files which will be used in the compilation.
- noret::substitute compileC ()

  compileC compiles the mex simulation file

- noret::substitute generateM (::amimodel amimodelo2)

  generateM generates the matlab wrapper for the compiled C files.
- noret::substitute getFun (::struct HTable,::string funstr)

  getFun generates symbolic expressions for the requested function.
- noret::substitute makeEvents ()

  makeEvents extracts discontiniuties from the model right hand side and converts them into events
- noret::substitute makeSyms ()

  makeSyms extracts symbolic definition from the user provided model and checks them for consistency
- mlhsInnerSubst< matlabtypesubstitute > checkDeps (::struct HTable,::cell deps)

  checkDeps checks the dependencies of functions and populates sym fields if necessary
- mlhsInnerSubst< matlabtypesubstitute > loadOldHashes ()

  loadOldHashes loads information from a previous compilation of the model.
- mlhsInnerSubst< matlabtypesubstitute > augmento2 ()

  augmento2 augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward apporach later on.
- mlhsInnerSubst< matlabtypesubstitute > augmento2vec ()

  augmento2 augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward apporach later on.

Static Public Member Functions

- static noret::substitute compileAndLinkModel (matlabtypesubstitute modelname, matlabtypesubstitute modelSourceFolder, matlabtypesubstitute coptim, matlabtypesubstitute debug, matlabtypesubstitute funs, matlabtypesubstitute cfun)

  compileAndLinkModel compiles the mex simulation file. It does not check if the model files have changed since generating C++ code or whether all files are still present. Use only if you know what you are doing. The safer alternative is rerunning amiwrap().
- static noret::substitute generateMatlabWrapper (matlabtypesubstitute nx, matlabtypesubstitute ny, matlabtypesubstitute np, matlabtypesubstitute nk, matlabtypesubstitute nz, matlabtypesubstitute o2flag,::amimodel amimodelo2, matlabtypesubstitute wrapperFilename, matlabtypesubstitute modelname, matlabtypesubstitute pscale, matlabtypesubstitute forward, matlabtypesubstitute adjoint)

  generateMatlabWrapper generates the matlab wrapper for the compiled C files.

Public Attributes

- ::struct sym = struct.empty("")

  symbolic definition struct
- ::struct fun = struct.empty("")

  struct which stores information for which functions c code needs to be generated
- ::amievent event = amievent.empty("")

  struct which stores information for which functions c code needs to be generated
- ::string modelname = char.empty("")

  name of the model
- ::struct HTable = struct.empty("")

  struct that contains hash values for the symbolic model definitions
- ::bool debug = false

  flag indicating whether debugging symbols should be compiled
- ::bool adjoint = true

  flag indicating whether adjoint sensitivities should be enabled
- ::bool forward = true

  flag indicating whether forward sensitivities should be enabled
- ::double t0 = 0

  default initial time

- ::string wtype = char.empty("")

  type of wrapper (cvodes/idas)
- ::int nx = double.empty("")

  number of states
- ::int nxtrue = double.empty("")

  number of original states for second order sensitivities
- ::int ny = double.empty("")

  number of observables
- ::int nytrue = double.empty("")

  number of original observables for second order sensitivities
- ::int np = double.empty("")

  number of parameters
- ::int nk = double.empty("")

  number of constants
- ::int ng = double.empty("")

  number of objective functions
- ::int nevent = double.empty("")

  number of events
- ::int nz = double.empty("")

  number of event outputs
- ::int nztrue = double.empty("")

  number of original event outputs for second order sensitivities
- ::∗int id = double.empty("")

  flag for DAEs
- ::int ubw = double.empty("")

  upper Jacobian bandwidth
- ::int lbw = double.empty("")

  lower Jacobian bandwidth
- ::int nnz = double.empty("")

  number of nonzero entries in Jacobian
- ::∗int sparseidx = double.empty("")

  dataindexes of sparse Jacobian
- ::∗int rowvals = double.empty("")

  rowindexes of sparse Jacobian
- ::∗int colptrs = double.empty("")

  columnindexes of sparse Jacobian
- ::∗int sparseidxB = double.empty("")

  dataindexes of sparse Jacobian
- ::∗int rowvalsB = double.empty("")

  rowindexes of sparse Jacobian
- ::∗int colptrsB = double.empty("")

  columnindexes of sparse Jacobian
- ::∗cell funs = cell.empty("")

  cell array of functions to be compiled
- ::∗cell mfuns = cell.empty("")

  cell array of matlab functions to be compiled
- ::string coptim = "-O3"

  optimisation flag for compilation
- ::string param = "lin"

  default parametrisation
- matlabtypesubstitute wrap_path = char.empty("")

path to wrapper

- matlabtypesubstitute recompile = false

  flag to enforce recompilation of the model
- matlabtypesubstitute cfun = struct.empty("")

  storage for flags determining recompilation of individual functions
- matlabtypesubstitute o2flag = 0

  flag which identifies augmented models 0 indicates no augmentation 1 indicates augmentation by first order sensitivities (yields second order sensitivities) 2 indicates augmentation by one linear combination of first order sensitivities (yields hessian-vector product)
- matlabtypesubstitute z2event = double.empty("")

  vector that maps outputs to events
- matlabtypesubstitute splineflag = false

  flag indicating whether the model contains spline functions
- matlabtypesubstitute minflag = false

  flag indicating whether the model contains min functions
- matlabtypesubstitute maxflag = false

  flag indicating whether the model contains max functions
- ::int nw = 0

  number of derived variables w, w is used for code optimization to reduce the number of frequently occuring expressions
- ::int ndwdx = 0

  number of derivatives of derived variables w, dwdx
- ::int ndwdp = 0

  number of derivatives of derived variables w, dwdp

18.47.1 Detailed Description

Definition at line 17 of file amimodel.m.

18.47.2 Constructor & Destructor Documentation

18.47.2.1 amimodel()

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

Parameters

symfun	this is the string to the function which generates the modelstruct. You can also directly pass the struct here
modelname	name of the model

Definition at line 515 of file amimodel.m.

Here is the caller graph for this function:

18.47.3 Member Function Documentation

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

Parameters

xdot	new right hand side of the differential equation

Return values

xdot	void

Definition at line 612 of file amimodel.m.
Here is the caller graph for this function:

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

Parameters

modelname	new modelname

Return values

modelname	void

Definition at line 627 of file amimodel.m.

18.47.3.3 updateWrapPath() `noret::substitute updateWrapPath ( matlabtypesubstitute wrap_path )`

Parameters

wrap_path	new wrap_path

Return values

wrap_path	void

Definition at line 640 of file amimodel.m.

18.47.3.4 parseModel() `noret::substitute parseModel ( )`

Return values

void	

Definition at line 18 of file parseModel.m.
Here is the call graph for this function:

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

Return values

void	

Definition at line 18 of file generateC.m.

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

Return values

void	

Definition at line 18 of file compileC.m.
Here is the call graph for this function:

18.47.3.7 generateM()

```
noret::substitute generateM (
    ::amimodel amimodelo2 )
```

Parameters

amimodelo2	this struct must contain all necessary symbolic definitions for second order sensivities

Return values

amimodelo2	void

Definition at line 18 of file generateM.m.
Here is the call graph for this function:

18.47.3.8 getFun()

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

Parameters

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

Return values

funstr	void

Definition at line 18 of file getFun.m.
Here is the call graph for this function:

Here is the caller graph for this function:

18.47.3.9 makeEvents() `noret::substitute makeEvents ( )`

Return values

void	

Definition at line 18 of file makeEvents.m.

Here is the call graph for this function:

18.47.3.10 makeSyms() `noret::substitute makeSyms ( )`

Return values

void	

Definition at line 18 of file makeSyms.m.

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

Parameters

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

Return values

cflag	boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable

Definition at line 18 of file checkDeps.m.
Here is the call graph for this function:

Here is the caller graph for this function:

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

Return values

HTable	struct with hashes of symbolic definition from the previous compilation

Definition at line 18 of file loadOldHashes.m.
Here is the caller graph for this function:

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

Return values

this	augmented system which contains symbolic definition of the original system and its sensitivities

Definition at line 18 of file augmento2.m.

Here is the call graph for this function:

18.47.3.14 augmento2vec() `mlhsInnerSubst<::amimodel > augmento2vec ( )`

Return values

modelo2vec	augmented system which contains symbolic definition of the original system and its sensitivities

Definition at line 18 of file augmento2vec.m.
Here is the call graph for this function:

18.47.3.15 compileAndLinkModel()

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

Parameters

modelname	name of the model as specified for amiwrap()
modelSourceFolder	path to model source directory
coptim	optimization flags
debug	enable debugging
funs	array with names of the model functions, will be guessed from source files if left empty

Parameters

cfun	struct indicating which files should be recompiled

Return values

cfun	void

Definition at line 18 of file compileAndLinkModel.m.
Here is the caller graph for this function:

18.47.3.16 generateMatlabWrapper()

```
noret::substitute generateMatlabWrapper (
    matlabtypesubstitute nx,
    matlabtypesubstitute ny,
    matlabtypesubstitute np,
    matlabtypesubstitute nk,
    matlabtypesubstitute nz,
    matlabtypesubstitute o2flag,
    ::amimodel amimodelo2,
    matlabtypesubstitute wrapperFilename,
    matlabtypesubstitute modelname,
    matlabtypesubstitute pscale,
    matlabtypesubstitute forward,
    matlabtypesubstitute adjoint )  [static]
```

Parameters

nx	number of states
ny	number of observables
np	number of parameters
nk	number of fixed parameters
nz	number of events
o2flag	o2flag
amimodelo2	this struct must contain all necessary symbolic definitions for second order sensivities
wrapperFilename	output filename
modelname	name of the model
pscale	default parameter scaling
forward	has forward sensitivity equations
adjoint	has adjoint sensitivity equations

Return values

adjoint	void

Definition at line 18 of file generateMatlabWrapper.m.
Here is the caller graph for this function:

18.47.4 Member Data Documentation

18.47.4.1 sym `sym = struct.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 27 of file amimodel.m.

18.47.4.2 fun `fun = struct.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 38 of file amimodel.m.

18.47.4.3 event `event = amievent.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: amievent.empty("")

Definition at line 49 of file amimodel.m.

18.47.4.4 modelname `modelname = char.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: char.empty("")

Definition at line 61 of file amimodel.m.

18.47.4.5 HTable `HTable = struct.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 72 of file amimodel.m.

18.47.4.6 debug `debug = false`

Note

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

`Matlab documentation of property attributes.`
Default: false

Definition at line 83 of file amimodel.m.

18.47.4.7 adjoint `adjoint = true`

Note

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

`Matlab documentation of property attributes.`
Default: true

Definition at line 94 of file amimodel.m.

18.47.4.8 forward `forward = true`

Note

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

`Matlab documentation of property attributes.`
Default: true

Definition at line 105 of file amimodel.m.

18.47.4.9 t0 `t0 = 0`

Note

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

`Matlab documentation of property attributes.`
Default: 0

Definition at line 116 of file amimodel.m.

18.47.4.10 wtype `wtype = char.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: char.empty("")

Definition at line 127 of file amimodel.m.

18.47.4.11 nx `nx = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 138 of file amimodel.m.

18.47.4.12 nxtrue `nxtrue = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 149 of file amimodel.m.

18.47.4.13 ny `ny = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 160 of file amimodel.m.

18.47.4.14 nytrue `nytrue = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 171 of file amimodel.m.

18.47.4.15 np `np = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 182 of file amimodel.m.

18.47.4.16 nk `nk = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 193 of file amimodel.m.

18.47.4.17 ng `ng = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 204 of file amimodel.m.

18.47.4.18 nevent `nevent = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 215 of file amimodel.m.

18.47.4.19 nz `nz = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 226 of file amimodel.m.

18.47.4.20 nztrue `nztrue = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 237 of file amimodel.m.

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

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 248 of file amimodel.m.

18.47.4.22 ubw `ubw = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 259 of file amimodel.m.

18.47.4.23 lbw `lbw = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 270 of file amimodel.m.

18.47.4.24 nnz `nnz = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 281 of file amimodel.m.

18.47.4.25 sparseidx `sparseidx = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 292 of file amimodel.m.

18.47.4.26 rowvals `rowvals = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 303 of file amimodel.m.

18.47.4.27 colptrs `colptrs = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 314 of file amimodel.m.

18.47.4.28 sparseidxB `sparseidxB = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 325 of file amimodel.m.

18.47.4.29 rowvalsB `rowvalsB = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 336 of file amimodel.m.

18.47.4.30 colptrsB `colptrsB = double.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 347 of file amimodel.m.

18.47.4.31 funs `funs = cell.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: cell.empty("")

Definition at line 358 of file amimodel.m.

18.47.4.32 mfuns `mfuns = cell.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: cell.empty("")

Definition at line 369 of file amimodel.m.

18.47.4.33 coptim `coptim = "-O3"`

Note

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

`Matlab documentation of property attributes.`
Default: "-O3"

Definition at line 380 of file amimodel.m.

18.47.4.34 param `param = "lin"`

Note

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

`Matlab documentation of property attributes.`
Default: "lin"

Definition at line 391 of file amimodel.m.

18.47.4.35 wrap_path `wrap_path = char.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: char.empty("")

Definition at line 402 of file amimodel.m.

18.47.4.36 recompile `recompile = false`

Note

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

`Matlab documentation of property attributes.`
Default: false

Definition at line 413 of file amimodel.m.

18.47.4.37 cfun `cfun = struct.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 424 of file amimodel.m.

18.47.4.38 o2flag `o2flag = 0`

Note

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

`Matlab documentation of property attributes.`
Default: 0

Definition at line 436 of file amimodel.m.

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

Default: double.empty("")
Definition at line 455 of file amimodel.m.

18.47.4.40 splineflag `splineflag = false`

Default: false
Definition at line 463 of file amimodel.m.

18.47.4.41 minflag `minflag = false`

Default: false
Definition at line 471 of file amimodel.m.

18.47.4.42 maxflag `maxflag = false`

Default: false
Definition at line 479 of file amimodel.m.

18.47.4.43 nw `nw = 0`

Default: 0
Definition at line 487 of file amimodel.m.

18.47.4.44 ndwdx `ndwdx = 0`

Default: 0
Definition at line 496 of file amimodel.m.

18.47.4.45 ndwdp `ndwdp = 0`

Default: 0
Definition at line 504 of file amimodel.m.

18.48 amioption Class Reference

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

Inheritance diagram for amioption:

Public Member Functions

- amioption (matlabtypesubstitute varargin)

  amioptions Construct a new amioptions object OPTS = amioption() creates a set of options with each option set to its default value.

Public Attributes

- matlabtypesubstitute atol = 1e-16

  absolute integration tolerace
- matlabtypesubstitute rtol = 1e-8

  relative integration tolerace
- matlabtypesubstitute maxsteps = 1e4

  maximum number of integration steps
- matlabtypesubstitute quad_atol = 1e-12

  absolute quadrature tolerace
- matlabtypesubstitute quad_rtol = 1e-8

  relative quadrature tolerace
- matlabtypesubstitute maxstepsB = 0

  maximum number of integration steps
- matlabtypesubstitute ss_atol = 1e-16

  absolute steady state tolerace
- matlabtypesubstitute ss_rtol = 1e-8

  relative steady state tolerace
- matlabtypesubstitute sens_ind = double.empty("")

  index of parameters for which the sensitivities are computed
- matlabtypesubstitute tstart = 0

  starting time of the simulation
- matlabtypesubstitute lmm = 2

  linear multistep method.
- matlabtypesubstitute iter = 2

  iteration method for linear multistep.
- matlabtypesubstitute linsol = 9

  linear solver
- matlabtypesubstitute stldet = true

  stability detection flag
- matlabtypesubstitute interpType = 1

interpolation type

- matlabtypesubstitute ism = 1

  forward sensitivity mode
- matlabtypesubstitute sensi_meth = 1

  sensitivity method
- matlabtypesubstitute sensi_meth_preeq = 1

  sensitivity method for preequilibration
- matlabtypesubstitute sensi = 0

  sensitivity order
- matlabtypesubstitute nmaxevent = 10

  number of reported events
- matlabtypesubstitute ordering = 0

  reordering of states
- matlabtypesubstitute ss = 0

  steady state sensitivity flag
- matlabtypesubstitute x0 = double.empty("")

  custom initial state
- matlabtypesubstitute sx0 = double.empty("")

  custom initial sensitivity
- matlabtypesubstitute newton_maxsteps = 40

  newton solver: maximum newton steps
- matlabtypesubstitute newton_maxlinsteps = 100

  newton solver: maximum linear steps
- matlabtypesubstitute newton_preeq = false

  preequilibration of system via newton solver
- matlabtypesubstitute z2event = double.empty("")

  mapping of event ouputs to events
- matlabtypesubstitute pscale = "[ ]"

  parameter scaling Single value or vector matching sens_ind. Valid options are "log","log10" and "lin" for log, log10 or unscaled parameters p. Use [] for default as specified in the model (fallback: `lin`).
- matlabtypesubstitute steadyStateSensitivityMode = 0

  Mode for for computing sensitivities ({0: Newton}, 1: Simulation)

18.48.1 Detailed Description

Definition at line 17 of file amioption.m.

18.48.2 Constructor & Destructor Documentation

18.48.2.1 amioption() `amioption ( matlabtypesubstitute varargin )`

OPTS = amioption(PARAM, VAL, ...) creates a set of options with the named parameters altered with the specified values.
OPTS = amioption(OLDOPTS, PARAM, VAL, ...) creates a copy of OLDOPTS with the named parameters altered with the specified value
Note: to see the parameters, check the documentation page for amioption

Parameters

varargin	input to construct amioption object, see function function description

Definition at line 275 of file amioption.m.

18.48.3 Member Data Documentation

18.48.3.1 atol `atol = 1e-16`

Default: 1e-16
Definition at line 28 of file amioption.m.

18.48.3.2 rtol `rtol = 1e-8`

Default: 1e-8
Definition at line 36 of file amioption.m.

18.48.3.3 maxsteps `maxsteps = 1e4`

Default: 1e4
Definition at line 44 of file amioption.m.

18.48.3.4 quad_atol `quad_atol = 1e-12`

Default: 1e-12
Definition at line 52 of file amioption.m.

18.48.3.5 quad_rtol `quad_rtol = 1e-8`

Default: 1e-8
Definition at line 60 of file amioption.m.

18.48.3.6 maxstepsB `maxstepsB = 0`

Default: 0
Definition at line 68 of file amioption.m.

18.48.3.7 ss_atol `ss_atol = 1e-16`

Default: 1e-16
Definition at line 76 of file amioption.m.

18.48.3.8 ss_rtol `ss_rtol = 1e-8`

Default: 1e-8
Definition at line 84 of file amioption.m.

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

Default: double.empty("")
Definition at line 92 of file amioption.m.

18.48.3.10 tstart `tstart = 0`

Default: 0
Definition at line 100 of file amioption.m.

18.48.3.11 lmm `lmm = 2`

Default: 2
Definition at line 108 of file amioption.m.

18.48.3.12 iter `iter = 2`

Default: 2
Definition at line 116 of file amioption.m.

18.48.3.13 linsol `linsol = 9`

Default: 9
Definition at line 124 of file amioption.m.

18.48.3.14 stldet `stldet = true`

Default: true
Definition at line 132 of file amioption.m.

18.48.3.15 interpType `interpType = 1`

Default: 1
Definition at line 140 of file amioption.m.

18.48.3.16 ism `ism = 1`

Default: 1
Definition at line 148 of file amioption.m.

18.48.3.17 sensi_meth `sensi_meth = 1`

Default: 1

Note

This property has custom functionality when its value is changed.

Definition at line 156 of file amioption.m.

18.48.3.18 sensi_meth_preeq `sensi_meth_preeq = 1`

Default: 1
Definition at line 164 of file amioption.m.

18.48.3.19 sensi `sensi = 0`

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 172 of file amioption.m.

18.48.3.20 nmaxevent `nmaxevent = 10`

Default: 10
Definition at line 180 of file amioption.m.

18.48.3.21 ordering `ordering = 0`

Default: 0
Definition at line 188 of file amioption.m.

18.48.3.22 ss `ss = 0`

Default: 0
Definition at line 196 of file amioption.m.

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

Default: double.empty("")
Definition at line 204 of file amioption.m.

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

Default: double.empty("")
Definition at line 212 of file amioption.m.

18.48.3.25 newton_maxsteps `newton_maxsteps = 40`

Default: 40

Note

This property has custom functionality when its value is changed.

Definition at line 220 of file amioption.m.

18.48.3.26 newton_maxlinsteps `newton_maxlinsteps = 100`

Default: 100

Note

This property has custom functionality when its value is changed.

Definition at line 228 of file amioption.m.

18.48.3.27 newton_preeq `newton_preeq = false`

Default: false

Note

This property has custom functionality when its value is changed.

Definition at line 236 of file amioption.m.

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

Default: double.empty("")
Definition at line 244 of file amioption.m.

18.48.3.29 pscale `pscale = "[]"`

Default: "[]"

Note

This property has custom functionality when its value is changed.

Definition at line 252 of file amioption.m.

18.48.3.30 steadyStateSensitivityMode `steadyStateSensitivityMode = 0`

Default: 0
Definition at line 264 of file amioption.m.

18.49 amised Class Reference

AMISED is a container for SED-ML objects.
Inheritance diagram for amised:

Public Member Functions

- amised (matlabtypesubstitute sedname)

  amised reads in an SEDML document using the JAVA binding of of libSEDML

Public Attributes

- matlabtypesubstitute model = struct("'event',[],'sym',[]")

  amimodel from the specified model
- matlabtypesubstitute modelname = {""}

  cell array of model identifiers
- matlabtypesubstitute sedml = struct.empty("")

  stores the struct tree from the xml definition
- matlabtypesubstitute outputcount = "[]"

  count the number of outputs per model
- matlabtypesubstitute varidx = "[]"

  indexes for dataGenerators
- matlabtypesubstitute varsym = sym("[]")

  symbolic expressions for variables
- matlabtypesubstitute datasym = sym("[]")

  symbolic expressions for data

18.49.1 Detailed Description

Definition at line 17 of file amised.m.

18.49.2 Constructor & Destructor Documentation

18.49.2.1 amised()

```
amised (
    matlabtypesubstitute sedname )
```

Parameters

sedname	name/path of the SEDML document

Definition at line 112 of file amised.m.
Here is the call graph for this function:

18.49.3 Member Data Documentation

18.49.3.1 model

```
model = struct("'event',[],'sym',[]")
```

Note

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

`Matlab documentation of property attributes.`
Default: struct("'event',[],'sym',[]")

Definition at line 27 of file amised.m.

18.49.3.2 modelname `modelname = {""}`

Note

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

`Matlab documentation of property attributes.`
Default: {""}

Definition at line 38 of file amised.m.

18.49.3.3 sedml `sedml = struct.empty("")`

Note

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

`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 49 of file amised.m.

18.49.3.4 outputcount `outputcount = "[]"`

Note

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

`Matlab documentation of property attributes.`
Default: "[]"

Definition at line 60 of file amised.m.

18.49.3.5 varidx `varidx = "[]"`

Note

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

`Matlab documentation of property attributes.`
Default: "[]"

Definition at line 71 of file amised.m.

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

Note

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

`Matlab documentation of property attributes.`

Default: sym("[]")

Definition at line 82 of file amised.m.

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

Note

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

`Matlab documentation of property attributes.`

Default: sym("[]")

Definition at line 93 of file amised.m.

18.50 optsym Class Reference

OPTSYM is an auxiliary class to gain access to the private symbolic property `s` which is necessary to be able to call symobj::optimize on it.
Inheritance diagram for optsym:

Public Member Functions

- optsym (::sym symbol)

  optsym converts the symbolic object into a optsym object
- mlhsInnerSubst<::sym > getoptimized ()

  getoptimized calls symobj::optimize on the optsym object

18.50.1 Detailed Description

Definition at line 17 of file optsym.m.

18.50.2 Constructor & Destructor Documentation

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

Parameters

symbol	symbolic object

Definition at line 32 of file optsym.m.

18.50.3 Member Function Documentation

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

Return values

out	optimized symbolic object

Definition at line 42 of file optsym.m.

19 File Documentation

19.1 am_and.m File Reference

am_and is the amici implementation of the symbolic and function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_and (::sym a,::sym b)

  am_and is the amici implementation of the symbolic and function

19.1.1 Function Documentation

19.1.1.1 am_and()

```
mlhsInnerSubst< matlabtypesubstitute > am_and (
    ::sym a,
    ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_and.m.

Here is the call graph for this function:

Here is the caller graph for this function:

19.2 am_eq.m File Reference

am_eq is currently a placeholder that simply produces an error message

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_eq (matlabtypesubstitute varargin)

  am_eq is currently a placeholder that simply produces an error message

19.2.1 Function Documentation

19.2.1.1 am_eq() `mlhsInnerSubst< matlabtypesubstitute > am_eq ( matlabtypesubstitute varargin )`

Parameters

varargin	elements for chain of equalities

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_eq.m.

19.3 am_ge.m File Reference

am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_ge (::sym varargin)

  am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

19.3.1 Function Documentation

19.3.1.1 am_ge() `mlhsInnerSubst< matlabtypesubstitute > am_ge ( ::sym varargin )`

Parameters

varargin	chain of input parameters

Return values

fun	a >= b logical value, negative for false, positive for true

Definition at line 17 of file am_ge.m.
Here is the call graph for this function:

19.4 am_gt.m File Reference

am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_gt (::sym varargin)

  am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

19.4.1 Function Documentation

19.4.1.1 am_gt() `mlhsInnerSubst< matlabtypesubstitute > am_gt ( ::sym varargin )`

Parameters

varargin	chain of input parameters

Return values

fun	a > b logical value, negative for false, positive for true

Definition at line 17 of file am_gt.m.
Here is the call graph for this function:

19.5 am_if.m File Reference

am_if is the amici implementation of the symbolic if function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_if (::sym condition,::sym truepart,::sym falsepart)

  am_if is the amici implementation of the symbolic if function

19.5.1 Function Documentation

19.5.1.1 am_if() `mlhsInnerSubst< matlabtypesubstitute > am_if ( ::sym condition, ::sym truepart, ::sym falsepart )`

Parameters

condition	logical value
truepart	value if condition is true
falsepart	value if condition is false

Return values

fun	if condition is true truepart, else falsepart

Definition at line 17 of file am_if.m.
Here is the caller graph for this function:

19.6 am_le.m File Reference

am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_le (::sym varargin)

  am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

19.6.1 Function Documentation

19.6.1.1 am_le() `mlhsInnerSubst< matlabtypesubstitute > am_le ( ::sym varargin )`

Parameters

varargin	chain of input parameters

Return values

fun	a <= b logical value, negative for false, positive for true

Definition at line 17 of file am_le.m.
Here is the call graph for this function:

19.7 am_lt.m File Reference

am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_lt (::sym varargin)

  am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

19.7.1 Function Documentation

19.7.1.1 am_lt() `mlhsInnerSubst< matlabtypesubstitute > am_lt ( ::sym varargin )`

Parameters

varargin	chain of input parameters

Return values

fun	a < b logical value, negative for false, positive for true

Definition at line 17 of file am_lt.m.
Here is the call graph for this function:

19.8 am_max.m File Reference

am_max is the amici implementation of the symbolic max function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_max (::sym a,::sym b)

  am_max is the amici implementation of the symbolic max function

19.8.1 Function Documentation

19.8.1.1 am_max() `mlhsInnerSubst< matlabtypesubstitute > am_max ( ::sym a, ::sym b )`

Parameters

a	first input parameter
b	second input parameter

Return values

fun	maximum of a and b

Definition at line 17 of file am_max.m.
Here is the caller graph for this function:

19.9 am_min.m File Reference

am_min is the amici implementation of the symbolic min function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_min (::sym a,::sym b)

  am_min is the amici implementation of the symbolic min function

19.9.1 Function Documentation

19.9.1.1 am_min()

```
mlhsInnerSubst< matlabtypesubstitute > am_min (
    ::sym a,
    ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	minimum of a and b

Definition at line 17 of file am_min.m.

Here is the call graph for this function:

Here is the caller graph for this function:

19.10 am_or.m File Reference

am_or is the amici implementation of the symbolic or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_or (::sym a,::sym b)

  am_or is the amici implementation of the symbolic or function

19.10.1 Function Documentation

19.10.1.1 am_or()

```
mlhsInnerSubst< matlabtypesubstitute > am_or (
    ::sym a,
    ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_or.m.
Here is the call graph for this function:

Here is the caller graph for this function:

19.11 am_piecewise.m File Reference

am_piecewise is the amici implementation of the mathml piecewise function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_piecewise (matlabtypesubstitute piece, matlabtypesubstitute condition, matlabtypesubstitute default)

  am_piecewise is the amici implementation of the mathml piecewise function

19.11.1 Function Documentation

19.11.1.1 am_piecewise()

```
mlhsInnerSubst< matlabtypesubstitute > am_piecewise (
    matlabtypesubstitute piece,
    matlabtypesubstitute condition,
    matlabtypesubstitute default )
```

Parameters

piece	value if condition is true
condition	logical value
default	value if condition is false

Return values

fun	return value, piece if condition is true, default if not

Definition at line 17 of file am_piecewise.m.
Here is the call graph for this function:

19.12 am_stepfun.m File Reference

am_stepfun is the amici implementation of the step function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_stepfun (::sym t, matlabtypesubstitute tstart, matlabtypesubstitute vstart, matlabtypesubstitute tend, matlabtypesubstitute vend)

  am_stepfun is the amici implementation of the step function

19.12.1 Function Documentation

19.12.1.1 am_stepfun()

```
mlhsInnerSubst< matlabtypesubstitute > am_stepfun (
    ::sym t,
    matlabtypesubstitute tstart,
    matlabtypesubstitute vstart,
    matlabtypesubstitute tend,
    matlabtypesubstitute vend )
```

Parameters

t	input variable
tstart	input variable value at which the step starts
vstart	value during the step
tend	input variable value at which the step end
vend	value after the step

Return values

fun	0 before tstart, vstart between tstart and tend and vend after tend

Definition at line 17 of file am_stepfun.m.

19.13 am_xor.m File Reference

am_xor is the amici implementation of the symbolic exclusive or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_xor (::sym a,::sym b)

  am_xor is the amici implementation of the symbolic exclusive or function

19.13.1 Function Documentation

19.13.1.1 am_xor()

```
mlhsInnerSubst< matlabtypesubstitute > am_xor (
    ::sym a,
    ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_xor.m.
Here is the call graph for this function:

19.14 amici.cpp File Reference

core routines for integration

```
#include "amici/amici.h"
#include "amici/steadystateproblem.h"
#include "amici/backwardproblem.h"
#include "amici/forwardproblem.h"
#include "amici/misc.h"
#include <cvodes/cvodes.h>
#include <sundials/sundials_types.h>
#include <cassert>
#include <cstdarg>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <memory>
#include <type_traits>
```

Include dependency graph for amici.cpp:

Functions

- std::unique_ptr< ReturnData > **runAmiciSimulation** (Solver &solver, const ExpData *edata, Model &model, bool rethrow=false)

  Core integration routine. Initializes the solver and runs the forward and backward problem.
- void **printErrMsgIdAndTxt** (std::string const &id, std::string const &message)

  Prints a specified error message associated with the specified identifier.
- void **printWarnMsgIdAndTxt** (std::string const &id, std::string const &message)

  Prints a specified warning message associated with the specified identifier.
- std::vector< std::unique_ptr< ReturnData > > **runAmiciSimulations** (Solver const &solver, const std::vector< ExpData * > &edatas, Model const &model, bool failfast, int num_threads)

  Same as runAmiciSimulation, but for multiple ExpData instances. When compiled with openMP support, this function runs multi-threaded.

19.14.1 Function Documentation

19.14.1.1 runAmiciSimulation()

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

Parameters

solver	Solver instance
edata	pointer to experimental data object
model	model specification object
rethrow	rethrow integration exceptions?

Returns

rdata pointer to return data object

Definition at line 49 of file amici.cpp.

19.14.1.2 printErrMsgIdAndTxt()

```
void printErrMsgIdAndTxt (
    std::string const & id,
    std::string const & message )
```

Parameters

id	error identifier
message	error message

Definition at line 58 of file amici.cpp.

19.14.1.3 printWarnMsgIdAndTxt()

```
void printWarnMsgIdAndTxt (
    std::string const & id,
    std::string const & message )
```

Parameters

id	warning identifier
message	warning message

Definition at line 68 of file amici.cpp.

19.14.1.4 runAmiciSimulations()

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

Parameters

solver	Solver instance
edatas	experimental data objects
model	model specification object
failfast	flag to allow early termination
num_threads	number of threads for parallel execution

Returns

vector of pointers to return data objects

Definition at line 78 of file amici.cpp.

19.15 amiwrap.m File Reference

AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

Functions

- noret::substitute amiwrap (matlabtypesubstitute varargin)

  AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

19.15.1 Function Documentation

19.15.1.1 amiwrap()

```
noret::substitute amiwrap (
    matlabtypesubstitute varargin )
```

Parameters

varargin	`amiwrap ( modelname, symfun, tdir, o2flag )` *Required Parameters for varargin:* • modelname specifies the name of the model which will be later used for the naming of the simulation file • symfun specifies a function which executes model definition see MATLAB Interface for details • tdir target directory where the simulation file should be placed **Default:** $AMICIDIR/models/modelname • o2flag boolean whether second order sensitivities should be enabled **Default:** false

Return values

o2flag	void

Definition at line 17 of file amiwrap.m.

19.16 cblas.cpp File Reference

BLAS routines required by AMICI.
`#include "amici/cblas.h"`
Include dependency graph for cblas.cpp:

Functions

- void **amici_dgemm** (BLASLayout layout, BLASTranspose TransA, BLASTranspose TransB, int M, int N, int K, double alpha, const double ∗A, int lda, const double ∗B, int ldb, double beta, double ∗C, int ldc)
- void **amici_dgemv** (BLASLayout layout, BLASTranspose TransA, int M, int N, double alpha, const double ∗A, int lda, const double ∗X, int incX, double beta, double ∗Y, int incY)

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

  Compute y = a∗x + y.

19.16.1 Function Documentation

19.16.1.1 amici_dgemm()

```
void amici_dgemm (
    BLASLayout layout,
    BLASTranspose TransA,
    BLASTranspose TransB,
    int M,
    int N,
    int K,
    double alpha,
    const double * A,
    int lda,
    const double * B,
    int ldb,
    double beta,
    double * C,
    int ldc )
```

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

Parameters

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
C	matrix C
ldc	leading dimension of C (m or n)

Definition at line 22 of file cblas.cpp.

19.16.1.2 amici_dgemv()

```
void amici_dgemv (
    BLASLayout layout,
    BLASTranspose TransA,
    int M,
    int N,
    double alpha,
    const double * A,
    int lda,
    const double * X,
    int incX,
```

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

amici_dgemm provides an interface to the CBlas matrix vector multiplication routine dgemv. This routines computes y = alpha∗A∗x + beta∗y with A: [MxN] x:[Nx1] y:[Mx1]

Parameters

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
Y	vector Y
incY	increment for entries of Y

Definition at line 32 of file cblas.cpp.

19.16.1.3 amici_daxpy()

```
void amici_daxpy (
    int n,
    double alpha,
    const double * x,
    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 40 of file cblas.cpp.

19.17 interface_matlab.cpp File Reference

core routines for mex interface

```
#include "amici/interface_matlab.h"
#include "amici/model.h"
#include "amici/exception.h"
#include "amici/edata.h"
#include "amici/returndata_matlab.h"
#include <assert.h>
#include <blas.h>
#include <cstring>
#include <memory>
```

Include dependency graph for interface_matlab.cpp:

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

Functions

- int dbl2int (const double x)
- char amici_blasCBlasTransToBlasTrans (BLASTranspose trans)
- void **amici_dgemm** (BLASLayout layout, BLASTranspose TransA, BLASTranspose TransB, int M, int N, int K, double alpha, const double ∗A, int lda, const double ∗B, int ldb, double beta, double ∗C, int ldc)
- void **amici_dgemv** (BLASLayout layout, BLASTranspose TransA, int M, int N, double alpha, const double ∗A, int lda, const double ∗X, int incX, double beta, double ∗Y, int incY)
- void **amici_daxpy** (int n, double alpha, const double ∗x, int incx, double ∗y, int incy)

  Compute y = a∗x + y.
- std::vector< realtype > mxArrayToVector (const mxArray ∗array, int length)
- std::unique_ptr< ExpData > **expDataFromMatlabCall** (const mxArray ∗prhs[ ], const Model &model)

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

19.17.1 Detailed Description

This file defines the function `mexFunction` which is executed upon calling the `.mex` file from Matlab.

19.17.2 Function Documentation

19.17.2.1 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 242 of file interface_matlab.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

19.17.2.2 amici_blasCBlasTransToBlasTrans()

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

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 51 of file interface_matlab.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

19.17.2.3 amici_dgemm()

```
void amici_dgemm (
    BLASLayout layout,
    BLASTranspose TransA,
    BLASTranspose TransB,
    int M,
    int N,
    int K,
    double alpha,
    const double * A,
    int lda,
    const double * B,
    int ldb,
    double beta,
    double * C,
    int ldc )
```

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

Parameters

layout	memory layout.
TransA	flag indicating whether A should be transposed before multiplication
TransB	flag indicating whether B should be transposed before multiplication

Parameters

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
C	matrix C
ldc	leading dimension of C (m or n)

Definition at line 22 of file cblas.cpp.

19.17.2.4 amici_dgemv()

```
void amici_dgemv (
    BLASLayout layout,
    BLASTranspose TransA,
    int M,
    int N,
    double alpha,
    const double * A,
    int lda,
    const double * X,
    int incX,
    double beta,
    double * Y,
    int incY )
```

amici_dgemm provides an interface to the CBlas matrix vector multiplication routine dgemv. This routines computes y = alpha∗A∗x + beta∗y with A: [MxN] x:[Nx1] y:[Mx1]

Parameters

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
Y	vector Y
incY	increment for entries of Y

Definition at line 32 of file cblas.cpp.

19.17.2.5 amici_daxpy()

```
void amici_daxpy (
    int n,
    double alpha,
```

```
const double * x,
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 40 of file cblas.cpp.

19.17.2.6 mxArrayToVector()

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

conversion from mxArray to vector<realtype>

Parameters

array	Matlab array to create vector from
length	Number of elements in array

Returns

std::vector with data from array

Definition at line 114 of file interface_matlab.cpp.
Here is the call graph for this function:

Here is the caller graph for this function:

19.17.2.7 expDataFromMatlabCall()

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

Parameters

prhs	pointer to the array of input arguments
model	pointer to the model object, this is necessary to perform dimension checks **Type**: ∗mxArray

Returns

edata pointer to experimental data object
Type: ∗ExpData

Definition at line 118 of file interface_matlab.cpp.

19.17.2.8 setSolverOptions()

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

Parameters

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

Definition at line 248 of file interface_matlab.cpp.

19.17.2.9 setModelData()

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

Parameters

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

Definition at line 338 of file interface_matlab.cpp.

19.17.2.10 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

Definition at line 477 of file interface_matlab.cpp.
Here is the call graph for this function:

19.18 SBML2AMICI.m File Reference

SBML2AMICI generates AMICI model definition files from SBML.

Functions

- noret::substitute SBML2AMICI (matlabtypesubstitute filename, matlabtypesubstitute modelname)

  SBML2AMICI generates AMICI model definition files from SBML.

19.18.1 Function Documentation

19.18.1.1 SBML2AMICI()

```
noret::substitute SBML2AMICI (
    matlabtypesubstitute filename,
    matlabtypesubstitute modelname )
```

Parameters

filename	name of the SBML file (withouth extension)
modelname	name of the model, this will define the name of the output file (default: input filename)

Return values

modelname	void

Definition at line 17 of file SBML2AMICI.m.
Here is the caller graph for this function:

19.19 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[ ])

19.19.1 Detailed Description

Author

Peter & Nigel, Design Software, 42 Gubberley St, Kenmore, 4069, Australia.

19.19.2 Function Documentation

19.19.2.1 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

19.19.2.2 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.

19.19.2.3 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

Parameters

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.

19.19.2.4 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.

19.20 symbolic_functions.cpp File Reference

definition of symbolic functions

```
#include "amici/symbolic_functions.h"
#include "amici/spline.h"
```

```
#include <algorithm>
#include <cfloat>
#include <cmath>
#include <cstdarg>
#include <cstdlib>
#include <alloca.h>
```

Include dependency graph for symbolic_functions.cpp:

Functions

- int **isNaN** (double what)
- int **isInf** (double what)
- double **getNaN** ()
- double **log** (double x)
- double **dirac** (double x)
- double **heaviside** (double x)
- double **sign** (double x)
- double **max** (double a, double b, double c)
- double **min** (double a, double b, double c)
- double **Dmax** (int id, double a, double b, double c)
- double **Dmin** (int id, double a, double b, double c)
- double **pos_pow** (double base, double exponent)
- double **spline** (double t, int num,...)

  Spline function.
- double **spline_pos** (double t, int num,...)

  Exponentiated spline function.
- double **Dspline** (int id, double t, int num,...)

  Derivation of a spline function.
- double **Dspline_pos** (int id, double t, int num,...)

  Derivation of an exponentiated spline function.
- double **DDspline** (int id1, int id2, double t, int num,...)

  Second derivation of a spline function.
- double **DDspline_pos** (int id1, int id2, double t, int num,...)

  Derivation of an exponentiated spline function.

19.20.1 Detailed Description

This file contains definitions of various symbolic functions which

19.20.2 Function Documentation

19.20.2.1 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.

19.20.2.2 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.

19.20.2.3 getNaN() `double getNaN ( )`

function returning nan

Returns

NaN

Definition at line 52 of file symbolic_functions.cpp.

19.20.2.4 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.

19.20.2.5 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 76 of file symbolic_functions.cpp.

19.20.2.6 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 90 of file symbolic_functions.cpp.

19.20.2.7 sign()

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

c implementation of matlab function sign

Parameters

x	argument

Returns

0

Definition at line 104 of file symbolic_functions.cpp.

19.20.2.8 max()

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

c implementation of matlab function max

Parameters

a	value1
b	value2

Returns

if(a > b) then a else b

Definition at line 122 of file symbolic_functions.cpp.

19.20.2.9 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 143 of file symbolic_functions.cpp.

19.20.2.10 Dmax()

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

parameter derivative of c implementation of matlab function max

Parameters

id	argument index for differentiation
a	value1
b	value2

Returns

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

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

Definition at line 157 of file symbolic_functions.cpp.

19.20.2.11 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 180 of file symbolic_functions.cpp.

19.20.2.12 pos_pow()

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

specialized pow functions that assumes positivity of the first argument

Parameters

base	base
exponent	exponent

Returns

pow(max(base,0.0),exponen)

Definition at line 192 of file symbolic_functions.cpp.

19.20.2.13 spline()

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

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 212 of file symbolic_functions.cpp.

19.20.2.14 spline_pos()

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

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.

19.20.2.15 Dspline()

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

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 317 of file symbolic_functions.cpp.

19.20.2.16 Dspline_pos()

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

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 374 of file symbolic_functions.cpp.

19.20.2.17 DDspline()

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

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

Parameters

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 440 of file symbolic_functions.cpp.

19.20.2.18 DDspline_pos()

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

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 461 of file symbolic_functions.cpp.