1 GeNN Documentation 1

### 1 GeNN Documentation

GeNN is a software package to enable neuronal network simulations on NVIDIA GPUs by code generation. Models are defined in a simple C-style API and the code for running them on either GPU or CPU hardware is generated by GeNN. GeNN can also be used through external interfaces. Currently there are prototype interfaces for Spine Creator and SpineML and for Brian2.

GeNN is currently developed and maintained by

```
Dr James Knight (contact James)
James Turner (contact James)
Dr. Esin Yavuz (contact Esin)
Prof. Thomas Nowotny (contact Thomas)
```

Project homepage is http://genn-team.github.io/genn/.

The development of GeNN is partially supported by the EPSRC (grant number EP/J019690/1 - Green Brain Project).

Note

This documentation is under construction. If you cannot find what you are looking for, please contact the project developers.

Next

### 2 Installation

You can download GeNN either as a zip file of a stable release or a snapshot of the most recent stable version or the unstable development version using the Git version control system.

# 2.1 Downloading a release

Point your browser to https://github.com/genn-team/genn/releases and download a release from the list by clicking the relevant source code button. Note that GeNN is only distributed in the form of source code due to its code generation design. Binary distributions would not make sense in this framework and are not provided. After downloading continue to install GeNN as described in the Installing GeNN section below.

# 2.2 Obtaining a Git snapshot

If it is not yet installed on your system, download and install Git (http://git-scm.com/). Then clone the GeNN repository from Github

```
git clone https://github.com/genn-team/genn.git
```

The github url of GeNN in the command above can be copied from the HTTPS clone URL displayed on the GeNN Github page (https://github.com/genn-team/genn).

This will clone the entire repository, including all open branches. By default git will check out the master branch which contains the source version upon which the latest release is based. If you want the most recent (but unstable) development version (which may or may not be fully functional at any given time), checkout the development branch

```
git checkout development
```

There are other branches in the repository that are used for specific development purposes and are opened and closed without warning.

As an alternative to using git you can also download the full content of GeNN sources clicking on the "Download ZIP" button on the bottom right of the GeNN Github page (https://github.com/genn-team/genn).

## 2.3 Installing GeNN

Installing GeNN comprises a few simple steps to create the GeNN development environment.

- (i) If you have downloaded a zip file, unpack GeNN.zip in a convenient location. Otherwise enter the directory where you downloaded the Git repository.
- (ii) Define the environment variable "GENN\_PATH" to point to the main GeNN directory, e.g. if you extracted/downloaded GeNN to /usr/local/GeNN, then you can add "export GENN\_PATH=/usr/local/GeNN" to your login script (e.g. .profile or .bashrc. If you are using WINDOWS, the path should be a windows path as it will be interpreted by the Visual C++ compiler cl, and environment variables are best set using SETX in a Windows cmd window. To do so, open a Windows cmd window by typing cmd in the search field of the start menu, followed by the enter key. In the cmd window type

```
setx GENN_PATH "C:\Users\me\GeNN"
```

where C:\Users\me\GeNN is the path to your GeNN directory.

(iii) Add \$GENN\_PATH/lib/bin to your PATH variable, e.g.

```
export PATH=$PATH:$GENN_PATH/lib/bin
```

in your login script, or in windows,

```
setx PATH=%GENN_PATH%\lib\bin;%PATH%
```

- (iv) Install the C++ compiler on the machine, if not already present. For Windows, download Microsoft Visual Studio Community Edition from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.4 aspx When installing Visual Studio, one should select "custom install", and ensure that all C++ optional extras are also installed. Mac users should download and set up Xcode from https://developer.apple.com/xcode/index.html Linux users should install the GNU compiler collection gcc and g++ from their Linux distribution repository, or alternatively from https://gcc.gnu.org/index.html Be sure to pick CUDA and C++ compiler versions which are compatible with each other. The latest C++ compiler is not necessarily compatible with the latest CUDA toolkit.
- (v) If you haven't installed CUDA on your machine, obtain a fresh installation of the NVIDIA CUDA toolkit from https://developer.nvidia.com/cuda-downloads Again, be sure to pick CUDA and C++ compiler versions which are compatible with each other. The latest C++ compiler is not necessarily compatible with the latest CUDA toolkit.
- (vi) Set the CUDA\_PATH variable if it is not already set by the system, by putting

```
export CUDA_PATH=/usr/local/cuda
```

in your login script (or, if CUDA is installed in a non-standard location, the appropriate path to the main CUDA directory). For most people, this will be done by the CUDA install script and the default value of /usr/local/cuda is fine. In Windows, CUDA\_PATH is normally already set after installing the CUDA toolkit. If not, set this variable with:

```
setx CUDA_PATH C:\path\to\cuda
```

This normally completes the installation. Windows useres must close and reopen their command window to ensure variables set using SETX are initialised.

Depending on the needs of your own projects, e.g., depencies on other libraries or non-standard installation paths of libraries used by GeNN, you may want to modify Makefile examples under \$GENN\_PATH/userproject/xxxc\_project and \$GENN\_PATH/userproject/xxx\_project/model to add extra linker-, include- and

compiler-flags on a per-project basis, or modify global default flags in \$GENN\_PATH/userproject/include/makefile

\_common\_[win|gnu].mk.

For all makefiles there are separate makefiles for Unix-style operating systems (GNUmakefile) such as Linux or MacOS and for Windows (WINmakefile).

If you are using GeNN in Windows, the Visual Studio development environment must be set up within every instance of the CMD.EXE command window used. One can open an instance of CMD.EXE with the development environment already set up by navigating to Start - All Programs - visual studio - tools - visual studio native command prompt. You may wish to create a shortcut for this tool on the desktop, for convenience. Note that all C++ tools should have been installed during the Visual Studio install process for this to work. Alternatively one can use the make. but scripts to build the example projects, which will attempt to setup your development environment by executing vcvarsall.bat which is part of every Visual Studio distribution, inside the visual studio/VC directory. For this to work properly, GeNN must be able to locate the Visual Studio install directory, which should be contained in the VS\_PATH environment variable. You can set this variable by hand if it is not already set by the Visual C++ installer by typing:

```
setx VS_PATH "C:\Program Files (x86)\Microsoft Visual Studio 10.0"
```

### Note

- The exact path and name of Visual C++ installations will vary between systems.
- Double quotation marks like in the above example are necessary whenever a path contains spaces.

GeNN also has experimental CYGWIN support. However, with the introduction of native Windows support in GeNN 1.1.3, this is not being developed further and should be considered as deprecated.

# 2.4 Testing Your Installation

To test your installation, follow the example in the Quickstart section. Linux and Mac users can perform a more comprehensive test by running:

```
cd $GENN_PATH/userproject && ./testprojects.sh
```

This test script may take a long while to complete, and will terminate if any errors are detected.

### Top | Next

### 3 Quickstart

GeNN is based on the idea of code generation for the involved GPU or CPU simulation code for neuronal network models but leaves a lot of freedom how to use the generated code in the final application. To facilitate the use of Ge-NN on the background of this philosophy, it comes with a number of complete examples containing both the model description code that is used by GeNN for code generation and the "user side code" to run the generated model and safe the results. Running these complete examples should be achievable in a few minutes. The necessary steps are described below.

### 3.1 Running an Example Model in Unix

In order to get a quick start and run a provided model, open a shell, navigate to GeNN/tools and type

make

This will compile additional tools for creating and running example projects. For a first complete test, the system is best used with a full driver program such as in the Insect olfaction model example:

```
./generate_run <0 (CPU) / 1 (GPU) / n (GPU n+2)> <nAL> <nMB> <nLHI> <nLb> <gscale> <outdir> <model name> <OPTIONS>
```

### Possible options:

DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger,
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use,
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run,
CPU ONLY=0 or CPU ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.

To compile  $generate\_run.cc$ , navigate to the  $userproject/MBody1\_project$  directory and type

make

This will generate an executable that you can invoke with, e.g.,

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody1
```

which would generate and simulate a model of the locust olfactory system with 100 projection neurons, 1000 Kenyon cells, 20 lateral horn interneurons and 100 output neurons in the mushroom body lobes.

The tool generate\_run will generate connectivity matrices for the model  $\mathtt{MBody1}$  and store them into files, compile and run the model on an automatically chosen GPU, using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument 1 with the device number of the desired GPU plus 2, e.g., 2 for GPU 0. All input and output files will be prefixed with  $\mathtt{test1}$  and will be created in a sub-directory with the name  $\mathtt{test1\_output}$ . More about the DEBUG flag in the debugging section . The parameter  $\mathtt{FLOAT}$  will run the model in float (single precision floating point), using  $\mathtt{DOUBLE}$  would use double precision. The REU $\leftarrow$  SE parameter regulates whether previously generated files for connectivity and input should be reused (1) or files should be generated anew (0).

The MBody1 example is already a highly integrated example that showcases many of the features of GeNN and how to program the user-side code for a GeNN application. More details in the User Manual.

# 3.2 Running an Example Model in Windows

All interaction with GeNN programs are command-line based and hence are executed within a cmd window. Open a Visual Studio cmd window via Start: All Programs: Visual Studio: Tools: Native Tools Command Prompt, and navigate to the userprojects\tools directory.

```
cd %GENN_PATH%\userprojects\tools
```

### Then type

```
nmake /f WINmakefile
```

to compile a number of tools that are used by the example projects to generate connectivity and inputs to model networks. Then navigate to the userproject/MBody1\_project directory.

```
cd ..\MBody1_project
```

# By typing

```
nmake /f WINmakefile
```

you can compile the <code>generate\_run</code> engine that allows to run a Insect olfaction model of the insect mushroom body:

To invoke generate\_run.exe type, e.g.,

```
generate_run 1 100 1000 20 100 0.0025 test1 MBody1
```

which would generate and simulate a model of the locust olfactory system with 100 projection neurons, 1000 Kenyon cells, 20 lateral horn interneurons and 100 output neurons in the mushroom body lobes.

The tool <code>generate\_run.exe</code> will generate connectivity matrices for the model <code>MBody1</code> and store them into files, compile and run the model on an automatically chosen GPU, using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument 1 with the device number of the desired GPU plus 2, e.g., 2 for GPU 0. All input and output files will be prefixed with <code>test1</code> and will be created in a sub-directory with the name <code>test1\_output</code>. More about the DEBUG flag in the <code>debugging section</code>. The parameter <code>FLOAT</code> will run the model in float (single precision floating point), using <code>DOUBLE</code> would use double precision. The REU $\leftarrow$  SE parameter regulates whether previously generated files for connectivity and input should be reused (1) or files should be generated anew (0).

The MBody1 example is already a highly integrated example that showcases many of the features of GeNN and how to program the user-side code for a GeNN application. More details in the User Manual .

## 3.3 How to use GeNN for New Projects

Creating and running projects in GeNN involves a few steps ranging from defining the fundamentals of the model, inputs to the model, details of the model like specific connectivity matrices or initial values, running the model, and analyzing or saving the data.

GeNN code is generally created by passing the C / C++ model file (see below) directly to the genn-buildmodel script. Another way to use GeNN is to create or modify a script or executable such as  $userproject/MBody1 \leftarrow project/generate_run.cc$  that wraps around the other programs that are used for each of the steps listed above. In more detail, the GeNN workflow consists of:

- 1. Either using tools (programs) to generate connectivity and input matrix files, which are then loaded into the user's simulation code at runtime, or generating these matrices directly inside the user's simulation code.
- 2. Building the source code of a model simulation using <code>genn-buildmodel.sh</code> (On Linux or Mac) or <code>genn-buildmodel.bat</code> (on Windows). In the example of the MBody1\_project this entails writing neuron numbers into <code>userproject/include/sizes.h</code>, and executing

```
genn-buildmodel.sh MBody1.cc
```

The genn-buildmodel script compiles the installed GeNN code generator in conjunction with the user-provided model description model/MBodyl.cc. It then executes the GeNN code generator to generate the complete model simulation code for the MBodyl model.

3. Compiling the generated code, found in model/MBody1\_CODE/, by calling:

```
make clean all
```

It is at this stage that GeNN generated model simulation code is combined with user-side code. In this example, classol\_sim.cu (classify-olfaction-simulation) which uses the map\_classol (map-neuron-based-classifier-olfaction) class.

4. Finally, running the resulting stand-alone simulator executable. In the MBody1 example classol\_sim in the model directory.

The generate\_run tool is only a suggested usage scenario of GeNN. Users have more control by manually executing the four steps above, or integrating GeNN into the development environment of their choice.

Note

The usage scenario described was made explicit for Unix environments. In Windows the setup is essentially the same except for the usual operating system dependent syntax differences, e.g. the build script is named genn-buildmodel.bat, compilation of the generated model simulator would be nmake f WINmakefile clean all, and the resulting executable would be named classol\_sim.exe.

GeNN comes with several example projects which showcase its features. The MBody1 example discussed above is one of the many provided examples that are described in more detail in Example projects.

# 3.4 Defining a New Model in GeNN

According to the work flow outlined above, there are several steps to be completed to define a neuronal network model.

- 1. The neuronal network of interest is defined in a model definition file, e.g. Example1.cc.
- 2. Within the the model definition file Example1.cc, the following tasks need to be completed:
  - a) The GeNN file modelSpec.h needs to be included,

```
#include "modelSpec.h"
```

b) The values for initial variables and parameters for neuron and synapse populations need to be defined, e.g.

would define the (homogeneous) parameters for a population of Poisson neurons.

Note

The number of required parameters and their meaning is defined by the neuron or synapse type. Refer to the User Manual for details. We recommend, however, to use comments like in the above example to achieve maximal clarity of each parameter's meaning.

If heterogeneous parameter values are needed for any particular population of neurons (synapses), a new neuron (synapse) type needs to be defined in which these parameters are defined as "variables" rather than parameters. See the User Manual for how to define new neuron (synapse) types.

c) The actual network needs to be defined in the form of a function modelDefinition, i.e.

```
void modelDefinition(NNmodel &model);
```

Note

The name modelDefinition and its parameter of type NNmodel& are fixed and cannot be changed if GeNN is to recognize it as a model definition.

d) Inside modelDefinition(), The time step DT needs to be defined, e.g.

```
model.setDT(0.1);
```

4 Examples 7

Note

All provided examples and pre-defined model elements in GeNN work with units of mV, ms, nF and muS. However, the choice of units is entirely left to the user if custom model elements are used.

MBody1.cc shows a typical example of a model definition function. In its core it contains calls to NNmodel ∴ addNeuronPopulation and NNmodel::addSynapsePopulation to build up the network. For a full range of options for defining a network, refer to the User Manual.

- 3. The programmer defines their own "user-side" modeling code similar to the code in userproject/M← Body1\_project/model/map\_classol.\* and userproject/MBody1\_project/model/classol← sim.\*. In this code,
  - a) They define the connectivity matrices between neuron groups. (In the MBody1 example those are read from files). Refer to the User Manual for the required format of connectivity matrices for dense or sparse connectivities.
  - b) They define input patterns (e.g. for Poisson neurons like in the MBody1 example) or individual initial values for neuron and / or synapse variables.

Note

The initial values given in the modelDefinition are automatically applied homogeneously to every individual neuron or synapse in each of the neuron or synapse groups.

c) They use stepTimeGPU(...); to run one time step on the GPU or stepTimeCPU(...); to run one on the CPU. (both GPU and CPU versions are always compiled, unless -c is used with genn-buildmodel).

Note

However, mixing CPU and GPU execution does not make too much sense. Among other things, The CPU version uses the same host side memory where to results from the GPU version are copied, which would lead to collisions between what is calculated on the CPU and on the GPU (see next point). However, in certain circumstances, expert users may want to split the calculation and calculate parts (e.g. neurons) on the GPU and parts (e.g. synapses) on the CPU. In such cases the fundamental kernel and function calls contained in stepTimeXXX need to be used and appropriate copies of the data from the CPU to the GPU and vice versa need to be performed.

- d) They use functions like <code>copyStateFromDevice()</code> etc to transfer the results from GPU calculations to the main memory of the host computer for further processing.
- e) They analyze the results. In the most simple case this could just be writing the relevant data to output files.

Previous | Top | Next

# 4 Examples

GeNN comes with a number of complete examples. At the moment, there are seven such example projects provided with GeNN.

# 4.1 Single compartment Izhikevich neuron(s)

This is a minimal example, with only one neuron population (with more or less neurons depending on the command line, but without any synapses). The neurons are Izhikevich neurons with homogeneous parameters across the neuron population. This example project contains a helper executable called "generate\_run", which also

```
prepares additional synapse connectivity and input pattern data, before compiling and
executing the model.
To compile it, navigate to genn/userproject/OneComp_project and type:
nmake /f WINmakefile
for Windows users, or:
make
for Linux, Mac and other UNIX users.
USAGE
{\tt generate\_run} < 0 \; ({\tt CPU}) \; / \; 1 \; ({\tt GPU}) > \; < n > \; < {\tt DIR} > \; < {\tt MODEL} >
Optional arguments:
<code>DEBUG=0</code> or <code>DEBUG=1</code> (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
For a first minimal test, the system may be used with:
generate_run.exe 1 1 outdir OneComp
for Windows users, or:
 ./generate_run 1 1 outdir OneComp
for Linux, Mac and other UNIX users.
This would create a set of tonic spiking Izhikevich neurons with no connectivity,
receiving a constant identical 4 nA input. It is lso possible to use the model
with a sinusoidal input instead, by setting the input to INPRULE.
Another example of an invocation would be:
generate_run.exe 0 1 outdir OneComp FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
 ./generate_run 0 1 outdir OneComp FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users.
Izhikevich neuron model: [1]
```

# 4.2 Izhikevich neurons driven by Poisson input spike trains:

Izhikevich network receiving Poisson input spike trains

In this example project there is again a pool of non-connected Izhikevich model neurons that are connected to a pool of Poisson input neurons with a fixed probability. This example project contains a helper executable called "generate\_run", which also prepares additional synapse connectivity and input pattern data, before compiling and executing the model.

To compile it, navigate to genn/userproject/PoissonIzh\_project and type:

```
nmake /f WINmakefile
for Windows users, or:
make
```

```
for Linux, Mac and other UNIX users.
USAGE
qenerate run <0(CPU)/1(GPU)> <nPoisson> <nIzhikevich> <pconn> <qscale> <DIR> <MODEL>
Optional arguments:
DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger
FTYPE=DOUBLE or FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 100 10 0.5 2 outdir PoissonIzh
for Windows users, or:
./generate_run 1 100 10 0.5 2 outdir PoissonIzh
for Linux, Mac and other UNIX users.
This will generate a network of 100 Poisson neurons with 20 Hz firing rate
connected to 10 Izhikevich neurons with a 0.5 probability.
The same network with sparse connectivity can be used by adding
the synapse population with sparse connectivity in PoissonIzh.cc and by uncommenting
the lines following the "//SPARSE CONNECTIVITY" tag in PoissonIzh.cu and commenting the
lines following '//DENSE CONNECTIVITY'.
Another example of an invocation would be:
generate_run.exe 0 100 10 0.5 2 outdir PoissonIzh FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
./generate_run 0 100 10 0.5 2 outdir PoissonIzh FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users.
Izhikevich neuron model: [1]
```

Pulse-coupled Izhikevich network

Pulse-coupled Izhikevich network

4.3

Pulse-coupled Izhikevich network

This example model is inspired by simple thalamo-cortical network of Izhikevich with an excitatory and an inhibitory population of spiking neurons that are randomly connected. It creates a pulse-coupled network with 80% excitatory 20% inhibitory connections, each connecting to nConn neurons with sparse connectivity.

To compile it, navigate to  $genn/userproject/Izh\_sparse\_project$  and type:

```
nmake /f WINmakefile

for Windows users, or:

make

for Linux, Mac and other UNIX users.

USAGE
----
```

generate\_run <0(CPU)/1(GPU)/n(GPU n-2)> <nNeurons> <nConn> <gScale> <outdir> <model name> <input factor>

Mandatory arguments:

```
CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').
nNeurons: Number of neurons
nConn: Number of connections per neuron
gScale: General scaling of synaptic conductances
outname: The base name of the output location and output files
model name: The name of the model to execute, as provided this would be 'Izh_sparse'
Optional arguments:
DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 10000 1000 1 outdir Izh_sparse 1.0
for Windows users, or:
./generate_run 1 10000 1000 1 outdir Izh_sparse 1.0
for Linux, Mac and other UNIX users.
This would create a pulse coupled network of 8000 excitatory 2000 inhibitory
Izhikevich neurons, each making 1000 connections with other neurons, generating
a mixed alpha and gamma regime. For larger input factor, there is more
input current and more irregular activity, for smaller factors less
and less and more sparse activity. The synapses are of a simple pulse-coupling
type. The results of the simulation are saved in the directory 'outdir_output',
debugging is switched off, and the connectivity is generated afresh (rather than
being read from existing files).
If connectivity were to be read from files, the connectivity files would have to
be in the 'inputfiles' sub-directory and be named according to the names of the
synapse populations involved, e.g., 'gIzh_sparse_ee' (\<variable name>='g'
\model name>='Izh_sparse' \model oppulation>='_ee'). These name conventions
are not part of the core GeNN definitions and it is the privilege (or burden)
of the user to find their own in their own versions of 'generate_run'.
Another example of an invocation would be:
generate_run.exe 0 10000 1000 1 outdir Izh_sparse 1.0 FTYPE=DOUBLE DEBUG=0 CPU_ONLY=1
for Windows users, or:
./generate_run 0 10000 1000 1 outdir Izh_sparse 1.0 FTYPE=DOUBLE DEBUG=0 CPU_ONLY=1
for Linux, Mac and other UNIX users.
Izhikevich neuron model: [1]
```

# 4.4 Izhikevich network with delayed synapses

Izhikevich network with delayed synapses

This example project demonstrates the synaptic delay feature of GeNN. It creates a network of three Izhikevich neuron groups, connected all-to-all with fast, medium and slow synapse groups. Neurons in the output group only spike if they are simultaneously innervated by the input neurons, via slow synapses, and the interneurons, via faster synapses.

```
COMPILE (WINDOWS)
```

To run this example project, first build the model into CUDA code by typing: genn-buildmodel.bat SynDelay.cc

```
then compile the project by typing:

nmake /f WINmakefile

COMPILE (MAC AND LINUX)
------

To run this example project, first build the model into CUDA code by typing:

genn-buildmodel.sh SynDelay.cc

then compile the project by typing:

make

USAGE
-----

syn_delay [CPU = 0 / GPU = 1] [directory to save output]

Izhikevich neuron model: [1]
```

### 4.5 Insect olfaction model

Locust olfactory system (Nowotny et al. 2005)

```
This project implements the insect olfaction model by Nowotny et al. that demonstrates self-organized clustering of odours in a simulation of the insect antennal lobe and mushroom body. As provided the model works with conductance based Hodgkin-Huxley neurons and several different synapse types, conductance based (but pulse-coupled) excitatory synapses, graded inhibitory synapses and synapses with a simplified STDP rule. This example project contains a helper executable called "generate_run", which also prepares additional synapse connectivity and input pattern data, before compiling and executing the model.

To compile it, navigate to genn/userproject/MBodyl_project and type:

nmake /f WINmakefile

for Windows users, or:

make
```

for Linux, Mac and other UNIX users.

USAGE

 $\label{eq:condition} generate\_run <0 (CPU)/1 (GPU)/n (GPU n-2)> <nAL> <nKC> <nLH> <nDN> <gScale> <DIR> <MODEL> <nC-1 (GPU)/1 (GPU)/1$ 

Mandatory parameters:

CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').

nAL: Number of neurons in the antennal lobe (AL), the input neurons to this model

nKC: Number of Kenyon cells (KC) in the "hidden layer"

nLH: Number of lateral horn interneurons, implementing gain control

nDN: Number of decision neurons (DN) in the output layer

gScale: A general rescaling factor for snaptic strength

outname: The base name of the output location and output files

model: The name of the model to execute, as provided this would be 'MBody1'

Optional arguments:

DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger

FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use

REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run

CPU\_ONLY=0 or CPU\_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.

```
An example invocation of generate_run is:
generate_run.exe 1 100 1000 20 100 0.0025 outname MBody1
for Windows users, or:
./generate_run 1 100 1000 20 100 0.0025 outname MBody1
for Linux, Mac and other UNIX users.
Such a command would generate a locust olfaction model with 100 antennal lobe neurons,
1000 mushroom body Kenyon cells, 20 lateral horn interneurons and 100 mushroom body
output neurons, and launch a simulation of it on a CUDA-enabled GPU using single
precision floating point numbers. All output files will be prefixed with "outname"
and will be created under the "outname" directory. The model that is run is defined
in 'model/MBodyl.cc', debugging is switched off, the model would be simulated using
float (single precision floating point) variables and parameters and the connectivity
and input would be generated afresh for this run.
In more details, what generate_run program does is:
a) use some other tools to generate the appropriate connectivity
   matrices and store them in files.
b) build the source code for the model by writing neuron numbers into
   ./model/sizes.h, and executing "genn-buildmodel.sh ./model/MBody1.cc.
c) compile the generated code by invoking "make clean && make"
   running the code, e.g. "./classol_sim r1 1".
Another example of an invocation would be:
generate_run.exe 0 100 1000 20 100 0.0025 outname MBody1 FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
./generate_run 0 100 1000 20 100 0.0025 outname MBody1 FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users, for using double precision floating point
and compiling and running the "CPU only" version.
Note: Optional arguments cannot contain spaces, i.e. "CPU_ONLY= 0" \,
will fail.
As provided, the model outputs a file 'test1.out.st' that contains
the spiking activity observed in the simulation, There are two
columns in this ASCII file, the first one containing the time of
a spike and the second one the ID of the neuron that spiked. Users
of matlab can use the scripts in the 'matlab' directory to plot
the results of a simulation. For more about the model itself and
the scientific insights gained from it see Nowotny et al. referenced below.
MODEL INFORMATION
For information regarding the locust olfaction model implemented in this example project, see:
```

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7

Nowotny insect olfaction model: [3]; Traub-Miles Hodgkin-Huxley neuron model: [5]

### 4.6 Insect olfaction model with user-defined neuron and synapse models

Locust olfactory system (Nowotny et al. 2005) with user-defined synapses

This examples recapitulates the exact same model as MBodyl\_project,

```
but with user-defined model types for neurons and synapses. Also
sparse connectivity is used instead of dense. The way user-defined
types are used should be very instructive to advanced users wishing
to do the same with their models. This example project contains a
helper executable called "generate_run", which also prepares
additional synapse connectivity and input pattern data, before
compiling and executing the model.
To compile it, navigate to genn/userproject/MBody_userdef_project and type:
nmake /f WINmakefile
for Windows users, or:
make
for Linux, Mac and other UNIX users.
    USAGE
\label{eq:condition} \texttt{generate\_run} < 0 \, \texttt{(CPU)} \, / \, \texttt{(GPU)} \, / \, \texttt{(GPU)} \, - 2) > \\ < \texttt{nAL} > \\ < \texttt{nKC} > \\ < \texttt{nLH} > \\ < \texttt{nDN} > \\ < \texttt{gScale} > \\ < \texttt{DIR} > \\ < \texttt{MODEL} > \\ < \texttt{nDN} 
Mandatory parameters:
CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').
nAL: Number of neurons in the antennal lobe (AL), the input neurons to this model
nKC: Number of Kenyon cells (KC) in the "hidden layer"
nLH: Number of lateral horn interneurons, implementing gain control
nDN: Number of decision neurons (DN) in the output layer
gScale: A general rescaling factor for snaptic strength
outname: The base name of the output location and output files
model: The name of the model to execute, as provided this would be 'MBodyl'
Optional arguments:
DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 100 1000 20 100 0.0025 outname MBody_userdef
for Windows users, or:
./generate_run 1 100 1000 20 100 0.0025 outname MBody_userdef
for Linux, Mac and other UNIX users.
Such a command would generate a locust olfaction model with 100\,
antennal lobe neurons, 1000 mushroom body Kenyon cells, 20 lateral
horn interneurons and 100 mushroom body output neurons, and launch
a simulation of it on a CUDA-enabled GPU using single precision
floating point numbers. All output files will be prefixed with
"outname" and will be created under the "outname" directory.
In more details, what generate_run program does is:
a) use some other tools to generate the appropriate connectivity
      matrices and store them in files.
b) build the source code for the model by writing neuron numbers into
       ./model/sizes.h, and executing "genn-buildmodel.sh ./model/MBody_userdef.cc".
c) compile the generated code by invoking "make clean && make"
      running the code, e.g. "./classol_sim r1 1".
Another example of an invocation would be:
generate_run.exe 0 100 1000 20 100 0.0025 outname MBody_userdef FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
```

```
./generate_run 0 100 1000 20 100 0.0025 outname MBody_userdef FTYPE=DOUBLE CPU_ONLY=1 for Linux, Mac and other UNIX users.

MODEL INFORMATION
-----
For information regarding the locust olfaction model implemented in this example project, see:
T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7
```

Nowotny insect olfaction model: [3]; Traub-Miles Hodgkin-Huxley neuron model: [5]

# 4.7 Insect Olfaction Model using INDIVIDUALID connectivity scheme

```
Locust olfactory system (Nowotny et al. 2005)
This example is very similar to the MBody1_project example. The
only difference is that PN to KC connections are defined with
the INDIVIDUALID mechanism.
To compile it, navigate to genn/userproject/MBody_individualID_project and type:
nmake /f WINmakefile
for Windows users, or:
make
for Linux, Mac and other UNIX users.
USAGE
Mandatory parameters:
CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').
nAL: Number of neurons in the antennal lobe (AL), the input neurons to this model
nKC: Number of Kenyon cells (KC) in the "hidden layer"
nLH: Number of lateral horn interneurons, implementing gain control
nDN: Number of decision neurons (DN) in the output layer
gScale: A general rescaling factor for snaptic strength
outname: The base name of the output location and output files
model: The name of the model to execute, as provided this would be 'MBodyl'
Optional arguments:
DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 100 1000 20 100 0.0025 outname MBody_individualID
for Windows users, or:
./generate_run 1 100 1000 20 100 0.0025 outname MBody_individualID
for Linux, Mac and other UNIX users.
Such a command would generate a locust olfaction model with 100
antennal lobe neurons, 1000 mushroom body Kenyon cells, 20 lateral
horn interneurons and 100 mushroom body output neurons, and launch
```

```
a simulation of it on a CUDA-enabled GPU using single precision
floating point numbers. All output files will be prefixed with
"outname" and will be created under the "outname" directory.
In more details, what generate_run program does is:
a) use some other tools to generate the appropriate connectivity
   matrices and store them in files.
b) build the source code for the model by writing neuron numbers into
   ./model/sizes.h, and executing "genn-buildmodel.sh ./model/MBody_individualID.cc".
c) compile the generated code by invoking "make clean && make"
   running the code, e.g. "./classol_sim r1 1".
Another example of an invocation would be:
generate_run.exe 0 100 1000 20 100 0.0025 outname MBody_individualID FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
./generate_run 0 100 1000 20 100 0.0025 outname MBody_individualID FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users.
MODEL INFORMATION
For information regarding the locust olfaction model implemented in this example project, see:
```

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005),

Nowotny insect olfaction model: [3]; Traub-Miles Hodgkin-Huxley neuron model: [5]

# 4.8 Insect Olfaction Model using delayed synapses

doi:10.1007/s00422-005-0019-7

```
Locust olfactory system (Nowotny et al. 2005)
 ______
A variation of the \ensuremath{\texttt{ref}} ex_mbody example using synaptic delays.
  In this example, the Kenyon Cell-Decision Neuron synapses are
 delayed by (5 \star DT) ms, and the Decision Neuron-Decision Neuron
 synapses are delayed by (3 * DT) ms. The example is intended
  to test the operation of synapses which have a combination of
delayed spike propagation and STDP (plasticity). This example % \left( 1\right) =\left( 1\right) \left( 1\right) +\left( 1\right) \left( 1\right) \left( 1\right) +\left( 1\right) \left( 
 project contains a helper executable called "generate_run",
 which also prepares additional synapse connectivity and input
pattern data, before compiling and executing the model.
To compile it, navigate to genn/userproject/MBody_delayedSyn_project and type:
 nmake /f WINmakefile
 for Windows users, or:
make
 for Linux, Mac and other UNIX users.
USAGE
 \label{eq:condition} $$\operatorname{generate\_run} < 0 (CPU) / 1 (GPU) / n (
Mandatory parameters:
 CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').
 nAL: Number of neurons in the antennal lobe (AL), the input neurons to this model
```

```
nKC: Number of Kenyon cells (KC) in the "hidden layer"
nLH: Number of lateral horn interneurons, implementing gain control
nDN: Number of decision neurons (DN) in the output layer
gScale: A general rescaling factor for snaptic strength
outname: The base name of the output location and output files
model: The name of the model to execute, as provided this would be 'MBody1'
Optional arguments:
DEBUG=0 or DEBUG=1 (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 100 1000 20 100 0.0025 outname MBody_delayedSyn
for Windows users, or:
./generate_run 1 100 1000 20 100 0.0025 outname MBody_delayedSyn
for Linux, Mac and other UNIX users.
Such a command would generate a locust olfaction model with 100
antennal lobe neurons, 1000 mushroom body Kenyon cells, 20 lateral
horn interneurons and 100 mushroom body output neurons, and launch
a simulation of it on a CUDA-enabled GPU using single precision
floating point numbers. All output files will be prefixed with
"outname" and will be created under the "outname" directory.
In more details, what generate_run program does is:
a) use some other tools to generate the appropriate connectivity
   matrices and store them in files.
b) build the source code for the model by writing neuron numbers into
   ./model/sizes.h, and executing "genn-buildmodel.sh ./model/MBody_delayedSyn.cc".
c) compile the generated code by invoking "make clean && make"
   running the code, e.g. "./classol_sim r1 1".
Another example of an invocation would be:
generate_run.exe 0 100 1000 20 100 0.0025 outname MBody_delayedSyn FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
./generate_run 0 100 1000 20 100 0.0025 outname MBody_delayedSyn FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users.
MODEL INFORMATION
For information regarding the locust olfaction model implemented in this example project, see:
T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the
olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005),
doi:10.1007/s00422-005-0019-7
```

Nowotny insect olfaction model: [3]; Traub-Miles Hodgkin-Huxley neuron model: [5]

### 4.9 Voltage clamp simulation to estimate Hodgkin-Huxley parameters

Genetic algorithm for tracking parameters in a HH model cell

This example simulates a population of Hodgkin-Huxley neuron models on the GPU and evolves them with a simple guided random search (simple GA) to mimic the dynamics of a separate Hodgkin-Huxley neuron that is simulated on the CPU. The parameters of the CPU simulated "true cell" are drifting

5 Release Notes 17

```
according to a user-chosen protocol: Either one of the parameters gNa, ENa, gKd, EKd, gleak,
Eleak, Cmem are modified by a sinusoidal addition (voltage parameters) or factor (conductance or capacitance)
protocol 0-6. For protocol 7 all 7 parameters undergo a random walk concurrently.
To compile it, navigate to genn/userproject/HHVclampGA_project and type:
nmake /f WINmakefile
for Windows users, or:
make
for Linux, Mac and other UNIX users.
USAGE
generate_run <CPU=0, GPU=1>   <nPop> <totalT> <outdir>
Mandatory parameters:
\mathtt{GPU}/\mathtt{CPU}: Whether to use the GPU (1) or CPU (0) for the model neuron population
protocol: Which changes to apply during the run to the parameters of the "true cell"
nPop: Number of neurons in the tracking population
totalT: Time in ms how long to run the simulation
outdir: The directory in which to save results
Optional arguments:
<code>DEBUG=0</code> or <code>DEBUG=1</code> (default 0): Whether to run in a debugger
FTYPE=DOUBLE of FTYPE=FLOAT (default FLOAT): What floating point type to use
REUSE=0 or REUSE=1 (default 0): Whether to reuse generated connectivity from an earlier run
CPU_ONLY=0 or CPU_ONLY=1 (default 0): Whether to compile in (CUDA independent) "CPU only" mode.
An example invocation of generate_run is:
generate_run.exe 1 -1 12 200000 test1
for Windows users, or:
./generate_run 1 -1 12 200000 test1
for Linux, Mac and other UNIX users.
This will simulate nPop= 5000 Hodgkin-Huxley neurons on the GPU which will for 1000 ms be matched to a
Hodgkin-Huxley neuron where the parameter gKd is sinusoidally modulated. The output files will be
written into a directory of the name test1_output, which will be created if it does not yet exist.
Another example of an invocation would be:
generate_run.exe 0 -1 12 200000 test1 FTYPE=DOUBLE CPU_ONLY=1
for Windows users, or:
./generate_run 0 -1 12 200000 test1 FTYPE=DOUBLE CPU_ONLY=1
for Linux, Mac and other UNIX users.
```

Traub-Miles Hodgkin-Huxley neuron model: [5]

Previous | Top | Next

# 5 Release Notes

### Release Notes for GeNN v3.0.0

This release is the result of some fairly major refactoring of GeNN which we hope will make it more user-friendly and maintainable in the future.

## **User Side Changes**

- 1. Entirely new syntax for defining models hopefully terser and less error-prone (see updated documentation and examples for details).
- 2. Continuous integration testing using Jenkins automated testing and code coverage calculation calculated automatically for Github pull requests etc.
- 3. Support for using Zero-copy memory for model variables. Especially on devices such as NVIDIA Jetson TX1 with no physical GPU memory this can significantly improve performance when recording data or injecting it to the simulation from external sensors.

### Release Notes for GeNN v2.2.3

This release includes minor new features and several bug fixes for certain system configurations.

### **User Side Changes**

- 1. Transitioned feature tests to use Google Test framework.
- 2. Added support for CUDA shader model 6.X

### **Bug fixes:**

- 1. Fixed problem using GeNN on systems running 32-bit Linux kernels on a 64-bit architecture (Nvidia Jetson modules running old software for example).
- 2. Fixed problem linking against CUDA on Mac OS X El Capitan due to SIP (System Integrity Protection).
- 3. Fixed problems with support code relating to its scope and usage in spike-like event threshold code.
- 4. Disabled use of C++ regular expressions on older versions of GCC.

### Release Notes for GeNN v2.2.2

This release includes minor new features and several bug fixes for certain system configurations.

# **User Side Changes**

- 1. Added support for the new version (2.0) of the Brian simulation package for Python.
- Added a mechanism for setting user-defined flags for the C++ compiler and NVCC compiler, via GENN\_PR← EFERENCES.

# Bug fixes:

- 1. Fixed a problem with atomicAdd() redefinitions on certain CUDA runtime versions and GPU configurations.
- 2. Fixed an incorrect bracket placement bug in code generation for certain models.
- 3. Fixed an incorrect neuron group indexing bug in the learning kernel, for certain models.
- 4. The dry-run compile phase now stores temporary files in the current directory, rather than the temp directory, solving issues on some systems.
- 5. The LINK\_FLAGS and INCLUDE\_FLAGS in the common windows makefile include 'makefile\_commin win.mk' are now appended to, rather than being overwritten, fixing issues with custom user makefiles on Windows.

5 Release Notes 19

### Release Notes for GeNN v2.2.1

This bugfix release fixes some critical bugs which occur on certain system configurations.

### **Bug fixes:**

- 1. (important) Fixed a Windows-specific bug where the CL compiler terminates, incorrectly reporting that the nested scope limit has been exceeded, when a large number of device variables need to be initialised.
- 2. (important) Fixed a bug where, in certain circumstances, outdated generateALL objects are used by the Makefiles, rather than being cleaned and replaced by up-to-date ones.
- 3. (important) Fixed an 'atomicAdd' redeclared or missing bug, which happens on certain CUDA architectures when using the newest CUDA 8.0 RC toolkit.
- 4. (minor) The SynDelay example project now correctly reports spike indexes for the input group.

Please refer to the full documentation for further details, tutorials and complete code documentation.

### Release Notes for GeNN v2.2

This release includes minor new features, some core code improvements and several bug fixes on GeNN v2.1.

### **User Side Changes**

- 1. GeNN now analyses automatically which parameters each kernel needs access to and these and only these are passed in the kernel argument list in addition to the global time t. These parameters can be a combination of extraGlobalNeuronKernelParameters and extraGlobalSynapseKernelParameters in either neuron or synapse kernel. In the unlikely case that users wish to call kernels directly, the correct call can be found in the stepTimeGPU() function.
  - Reflecting these changes, the predefined Poisson neurons now simply have two extraGlobalNeuron-Parameter rates and offset which replace the previous custom pointer to the array of input rates and integer offset to indicate the current input pattern. These extraGlobalNeuronKernelParameters are passed to the neuron kernel automatically, but the rates themselves within the array are of course not updated automatically (this is exactly as before with the specifically generated kernel arguments for Poisson neurons).
  - The concept of "directInput" has been removed. Users can easily achieve the same functionality by adding an additional variable (if there are individual inputs to neurons), an extraGlobalNeuronParameter (if the input is homogeneous but time dependent) or, obviously, a simple parameter if it's homogeneous and constant.

### Note

The global time variable "t" is now provided by GeNN; please make sure that you are not duplicating its definition or shadowing it. This could have severe consequences for simulation correctness (e.g. time not advancing in cases of over-shadowing).

- 2. We introduced the namespace GENN\_PREFERENCES which contains variables that determine the behaviour of GeNN.
- 3. We introduced a new code snippet called "supportCode" for neuron models, weightupdate models and post-synaptic models. This code snippet is intended to contain user-defined functions that are used from the other code snippets. We advise where possible to define the support code functions with the CUDA keywords "\_← \_host\_\_ \_device\_\_" so that they are available for both GPU and CPU version. Alternatively one can define separate versions for **host** and **device** in the snippet. The snippets are automatically made available to the relevant code parts. This is regulated through namespaces so that name clashes between different models do not matter. An exception are hash defines. They can in principle be used in the supportCode snippet but need to be protected specifically using ifndef. For example

```
#ifndef clip(x)
#define clip(x) x > 10.0? 10.0 : x
#endif
```

### Note

If there are conflicting definitions for hash defines, the one that appears first in the GeNN generated code will then prevail.

- 4. The new convenience macros spikeCount\_XX and spike\_XX where "XX" is the name of the neuron group are now also available for events: spikeEventCount\_XX and spikeEvent\_XX. They access the values for the current time step even if there are synaptic delays and spikes events are stored in circular queues.
- 5. The old buildmodel.[sh|bat] scripts have been superseded by new genn-buildmodel.[sh|bat] scripts. These scripts accept UNIX style option switches, allow both relative and absolute model file paths, and allow the user to specify the directory in which all output files are placed (-o <path>). Debug (-d), CPU-only (-c) and show help (-h) are also defined.
- 6. We have introduced a CPU-only "-c" genn-buildmodel switch, which, if it's defined, will generate a GeNN version that is completely independent from CUDA and hence can be used on computers without CUDA installation or CUDA enabled hardware. Obviously, this then can also only run on CPU. CPU only mode can either be switched on by defining CPU\_ONLY in the model description file or by passing appropriate parameters during the build, in particular

```
genn-buildmodel.[sh|bat] \<modelfile\> -c
make release CPU_ONLY=1
```

- 7. The new genn-buildmodel "-o" switch allows the user to specify the output directory for all generated files the default is the current directory. For example, a user project could be in '/home/genn\_project', whilst the GeNN directory could be '/usr/local/genn'. The GeNN directory is kept clean, unless the user decides to build the sample projects inside of it without copying them elsewhere. This allows the deployment of GeNN to a read-only directory, like '/usr/local' or 'C:\Program Files'. It also allows multiple users i.e. on a compute cluster to use GeNN simultaneously, without overwriting each other's code-generation files, etcetera.
- 8. The ARM architecture is now supported e.g. the NVIDIA Jetson development platform.
- 9. The NVIDIA CUDA SM\_5\* (Maxwell) architecture is now supported.
- 10. An error is now thrown when the user tries to use double precision floating-point numbers on devices with architecture older than SM\_13, since these devices do not support double precision.
- 11. All GeNN helper functions and classes, such as toString() and NNmodel, are defined in the header files at genn/lib/include/, for example stringUtils.h and modelSpec.h, which should be individually included before the functions and classes may be used. The functions and classes are actually implementated in the static library genn\lib\lib\genn.lib (Windows) or genn/lib/lib/libgenn.a (Mac, Linux), which must be linked into the final executable if any GeNN functions or classes are used.
- 12. In the modelDefinition() file, only the header file modelSpec.h should be included i.e. not the source file modelSpec.cc. This is because the declaration and definition of NNmodel, and associated functions, has been separated into modelSpec.h and modelSpec.cc, respectively. This is to enable NNmodel code to be precompiled separately. Henceforth, only the header file modelSpec.h should be included in model definition files!
- 13. In the modelDefinition() file, DT is now preferrably defined using model.setDT(<val>);, rather than #define DT <val>, in order to prevent problems with DT macro redefinition. For backward-compatibility reasons, the old #define DT <val> method may still be used, however users are advised to adopt the new method.
- 14. In preparation for multi-GPU support in GeNN, we have separated out the compilation of generated code from user-side code. This will eventually allow us to optimise and compile different parts of the model with different CUDA flags, depending on the CUDA device chosen to execute that particular part of the model. As such, we have had to use a header file definitions.h as the generated code interface, rather than the runner.cc file. In practice, this means that user-side code should include myModel\_COD← E/definitions.h, rather than myModel\_CODE/runner.cc. Including runner.cc will likely result in pages of linking errors at best!

5 Release Notes 21

## **Developer Side Changes**

 Blocksize optimization and device choice now obtain the ptxas information on memory usage from a CUDA driver API call rather than from parsing ptxas output of the nvcc compiler. This adds robustness to any change in the syntax of the compiler output.

- 2. The information about device choice is now stored in variables in the namespace GENN\_PREFERENCES.

  This includes chooseDevice, optimiseBlockSize, optimizeCode, debugCode, showPtx

  Info, defaultDevice. asGoodAsZero has also been moved into this namespace.
- 3. We have also introduced the namespace GENN\_FLAGS that contains unsigned int variables that attach names to numeric flags that can be used within GeNN.
- 4. The definitions of all generated variables and functions such as pullXXXStateFromDevice etc, are now generated into definitions.h. This is useful where one wants to compile separate object files that cannot all include the full definitions in e.g. "runnerGPU.cc". One example where this is useful is the brian2genn interface.
- 5. A number of feature tests have been added that can be found in the featureTests directory. They can be run with the respective runTests.sh scripts. The cleanTests.sh scripts can be used to remove all generated code after testing.

### Improvements

- 1. Improved method of obtaining ptxas compiler information on register and shared memory usage and an improved algorithm for estimating shared memory usage requirements for different block sizes.
- 2. Replaced pageable CPU-side memory with page-locked memory. This can significantly speed up simulations in which a lot of data is regularly copied to and from a CUDA device.
- 3. GeNN library objects and the main generateALL binary objects are now compiled separately, and only when a change has been made to an object's source, rather than recompiling all software for a minor change in a single source file. This should speed up compilation in some instances.

# **Bug fixes:**

- 1. Fixed a minor bug with delayed synapses, where delaySlot is declared but not referenced.
- 2. We fixed a bug where on rare occasions a synchronisation problem occurred in sparse synapse populations.
- 3. We fixed a bug where the combined spike event condition from several synapse populations was not assembled correctly in the code generation phase (the parameter values of the first synapse population over-rode the values of all other populations in the combined condition).

Please refer to the full documentation for further details, tutorials and complete code documentation.

### Release Notes for GeNN v2.1

This release includes some new features and several bug fixes on GeNN v2.0.

# **User Side Changes**

- 1. Block size debugging flag and the asGoodAsZero variables are moved into include/global.h.
- 2. NGRADSYNAPSES dynamics have changed (See Bug fix #4) and this change is applied to the example projects. If you are using this synapse model, you may want to consider changing model parameters.
- The delay slots are now such that NO\_DELAY is 0 delay slots (previously 1) and 1 means an actual delay of 1 time step.

- 4. The convenience function convertProbabilityToRandomNumberThreshold(float \*, uint64\_t \*, int) was changed so that it actually converts firing probability/timestep into a threshold value for the GeNN random number generator (as its name always suggested). The previous functionality of converting a *rate* in kHz into a firing threshold number for the GeNN random number generator is now provided with the name convertRateToRandomNumberThreshold(float \*, uint64\_t \*, int)
- 5. Every model definition function modelDefinition() now needs to end with calling NNmodel←::finalize() for the defined network model. This will lock down the model and prevent any further changes to it by the supported methods. It also triggers necessary analysis of the model structure that should only be performed once. If the finalize() function is not called, GeNN will issue an error and exit before code generation.
- 6. To be more consistent in function naming the pull\<SYNAPSENAME\>FromDevice and push\<S\U0048 YNAPSENAME\>ToDevice have been renamed to pull\<SYNAPSENAME\>StateFromDevice and push\<SYNAPSENAME\>StateToDevice. The old versions are still supported through macro definitions to make the transition easier.
- 7. New convenience macros are now provided to access the current spike numbers and identities of neurons that spiked. These are called spikeCount\_XX and spike\_XX where "XX" is the name of the neuron group. They access the values for the current time step even if there are synaptic delays and spikes are stored in circular queues.
- 8. There is now a pre-defined neuron type "SPIKECOURCE" which is empty and can be used to define PyNN style spike source arrays.
- 9. The macros FLOAT and DOUBLE were replaced with GENN\_FLOAT and GENN\_DOUBLE due to name clashes with typedefs in Windows that define FLOAT and DOUBLE.

# **Developer Side Changes**

1. We introduced a file definitions.h, which is generated and filled with useful macros such as spkQuePtrShift which tells users where in the circular spike queue their spikes start.

### Improvements

- 1. Improved debugging information for block size optimisation and device choice.
- 2. Changed the device selection logic so that device occupancy has larger priority than device capability version.
- 3. A new HH model called TRAUBMILES\_PSTEP where one can set the number of inner loops as a parameter is introduced. It uses the TRAUBMILES\_SAFE method.
- 4. An alternative method is added for the insect olfaction model in order to fix the number of connections to a maximum of 10K in order to avoid negative conductance tails.
- 5. We introduced a preprocessor define directive for an "int\_" function that translates floating points to integers.

### Bug fixes:

- 1. AtomicAdd replacement for old GPUs were used by mistake if the model runs in double precision.
- 2. Timing of individual kernels is fixed and improved.
- 3. More careful setting of maximum number of connections in sparse connectivity, covering mixed dense/sparse network scenarios.
- 4. NGRADSYNAPSES was not scaling correctly with varying time step.
- 5. Fixed a bug where learning kernel with sparse connectivity was going out of range in an array.
- 6. Fixed synapse kernel name substitutions where the "dd" prefix was omitted by mistake.

Please refer to the full documentation for further details, tutorials and complete code documentation.

5 Release Notes 23

### Release Notes for GeNN v2.0

Version 2.0 of GeNN comes with a lot of improvements and added features, some of which have necessitated some changes to the structure of parameter arrays among others.

**User Side Changes** 

- 1. Users are now required to call initGeNN () in the model definition function before adding any populations to the neuronal network model.
- 2. glbscnt is now call glbSpkCnt for consistency with glbSpkEvntCnt.
- 3. There is no longer a privileged parameter Epre. Spike type events are now defined by a code string spk EvntThreshold, the same way proper spikes are. The only difference is that Spike type events are specific to a synapse type rather than a neuron type.
- 4. The function setSynapseG has been deprecated. In a GLOBALG scenario, the variables of a synapse group are set to the initial values provided in the modeldefinition function.
- Due to the split of synaptic models into weightUpdateModel and postSynModel, the parameter arrays used during model definition need to be carefully split as well so that each side gets the right parameters. For example, previously

would define the parameter array of three parameters, Erev, Epre, and tau\_S for a synapse of type NSYNAPSE. This now needs to be "split" into

i.e. parameters <code>Erev</code> and <code>tau\_S</code> are moved to the post-synaptic model and its parameter array of two parameters. <code>Epre</code> is discontinued as a parameter for <code>NSYNAPSE</code>. As a consequence the weightupdate model of <code>NSYNAPSE</code> has no parameters and one can pass <code>NULL</code> for the parameter array in <code>addSynapse</code> <code>Population</code>. The correct parameter lists for all defined neuron and synapse model types are listed in the <code>User Manual</code>.

Note

If the parameters are not redefined appropriately this will lead to uncontrolled behaviour of models and likely to segmentation faults and crashes.

- 6. Advanced users can now define variables as type scalar when introducing new neuron or synapse types. This will at the code generation stage be translated to the model's floating point type (ftype), float or double. This works for defining variables as well as in all code snippets. Users can also use the expressions SCALAR\_MAX and SCALAR\_MIN for FLT\_MIN, FLT\_MAX, DBL\_MIN and DBL\_MAX, respectively. Corresponding definitions of scalar, SCALAR\_MIN and SCALAR\_MAX are also available for user-side code whenever the code-generated file runner.cc has been included.
- 7. The example projects have been re-organized so that wrapper scripts of the <code>generate\_run</code> type are now all located together with the models they run instead of in a common <code>tools</code> directory. Generally the structure now is that each example project contains the wrapper script <code>generate\_run</code> and a <code>model</code> subdirectory which contains the model description file and the user side code complete with Makefiles for Unix and Windows operating systems. The generated code will be deposited in the <code>model subdirectory</code> in its <code>own modelname\_CODE</code> folder. Simulation results will always be deposited in a new sub-folder of the main project directory.

- 8. The addSynapsePopulation(...) function has now more mandatory parameters relating to the introduction of separate weightupdate models (pre-synaptic models) and postynaptic models. The correct syntax for the addSynapsePopulation(...) can be found with detailed explanations in teh User Manual.
- We have introduced a simple performance profiling method that users can employ to get an overview over the differential use of time by different kernels. To enable the timers in GeNN generated code, one needs to declare

```
networkmodel.setTiming(TRUE);
```

This will make available and operate GPU-side cudeEvent based timers whose cumulative value can be found in the double precision variables <code>neuron\_tme</code>, <code>synapse\_tme</code> and <code>learning\_tme</code>. They measure the accumulated time that has been spent calculating the neuron kernel, synapse kernel and learning kernel, respectively. CPU-side timers for the simulation functions are also available and their cumulative values can be obtained through

```
float x= sdkGetTimerValue(&neuron_timer);
float y= sdkGetTimerValue(&synapse_timer);
float z= sdkGetTimerValue(&learning_timer);
```

The Insect olfaction model example shows how these can be used in the user-side code. To enable timing profiling in this example, simply enable it for GeNN:

```
model.setTiming(TRUE);
in MBodyl.cc's modelDefinition function and define the macro TIMING in classol_sim.h
#define TIMING
```

This will have the effect that timing information is output into OUTNAME\_output/OUTNAME. ← timingprofile.

# **Developer Side Changes**

- allocateSparseArrays() has been changed to take the number of connections, connN, as an argument rather than expecting it to have been set in the Connection struct before the function is called as was the arrangement previously.
- 2. For the case of sparse connectivity, there is now a reverse mapping implemented with revers index arrays and a remap array that points to the original positions of variable values in teh forward array. By this mechanism, revers lookups from post to pre synaptic indices are possible but value changes in the sparse array values do only need to be done once.
- 3. SpkEvnt code is no longer generated whenever it is not actually used. That is also true on a somewhat finer granularity where variable queues for synapse delays are only maintained if the corresponding variables are used in synaptic code. True spikes on the other hand are always detected in case the user is interested in them.

Please refer to the full documentation for further details, tutorials and complete code documentation.

```
Previous | Top | Next
```

### 6 User Manual

6.1 Contents 25

### 6.1 Contents

- Introduction
- · Defining a network model
- · Neuron models
- · Weight update models
- Synaptic matrix types
- · Postsynaptic integration methods

### 6.2 Introduction

GeNN is a software library for facilitating the simulation of neuronal network models on NVIDIA CUDA enabled GPU hardware. It was designed with computational neuroscience models in mind rather than artificial neural networks. The main philosophy of GeNN is two-fold:

- 1. GeNN relies heavily on code generation to make it very flexible and to allow adjusting simulation code to the model of interest and the GPU hardware that is detected at compile time.
- 2. GeNN is lightweight in that it provides code for running models of neuronal networks on GPU hardware but it leaves it to the user to write a final simulation engine. It so allows maximal flexibility to the user who can use any of the provided code but can fully choose, inspect, extend or otherwise modify the generated code. They can also introduce their own optimisations and in particular control the data flow from and to the GPU in any desired granularity.

This manual gives an overview of how to use GeNN for a novice user and tries to lead the user to more expert use later on. With that we jump right in.

Previous | Top | Next

## 6.3 Defining a network model

A network model is defined by the user by providing the function

```
void modelDefinition(NNmodel &model)
```

in a separate file, such as MyModel.cc. In this function, the following tasks must be completed:

1. The name of the model must be defined:

```
model.setName("MyModel");
```

- 2. Neuron populations (at least one) must be added (see Defining neuron populations). The user may add as many neuron populations as they wish. If resources run out, there will not be a warning but GeNN will fail. However, before this breaking point is reached, GeNN will make all necessary efforts in terms of block size optimisation to accommodate the defined models. All populations must have a unique name.
- 3. Synapse populations (zero or more) can be added (see Defining synapse populations). Again, the number of synaptic connection populations is unlimited other than by resources.

## 6.3.1 Defining neuron populations

Neuron populations are added using the function

```
model.addNeuronPopulation<NeuronModel>(name, num, paramValues, varValues);
```

### where the arguments are:

- NeuronModel: Template argument specifying the type of neuron model These should be derived off NeuronModels::Base and can either be one of the standard models or user-defined (see Neuron models).
- const string &name: Unique name of the neuron population
- unsigned int size: number of neurons in the population
- NeuronModel::ParamValues paramValues: Parameters of this neuron type
- NeuronModel::VarValues varValues: Initial values for variables of this neuron type

The user may add as many neuron populations as the model necessitates. They must all have unique names. The possible values for the arguments, predefined models and their parameters and initial values are detailed Neuron models below.

### 6.3.2 Defining synapse populations

Synapse populations are added with the function

### where the arguments are

- WeightUpdateModel: Template parameter specifying the type of weight update model. These should be
  derived off WeightUpdateModels::Base and can either be one of the standard models or user-defined (see
  Weight update models).
- PostsynapticModel: Template parameter specifying the type of postsynaptic integration model. These should be derived off PostsynapticModels::Base and can either be one of the standard models or user-defined (see Postsynaptic integration methods).
- const string &name: The name of the synapse population
- unsigned int mType: How the synaptic matrix is stored. the options currently are "SPARSE\_GLOBALG" LG", "SPARSE\_INDIVIDUALG", "DENSE\_GLOBALG", "DENSE\_INDIVIDUALG" or "BITMASK\_GLOBALG" (see Synaptic matrix types).
- unsigned int delay: Synaptic delay (in multiples of the simulation time step DT).
- const string preName: Name of the (existing!) pre-synaptic neuron population.
- const string postName: Name of the (existing!) post-synaptic neuron population.
- WeightUpdateModel::ParamValues weightParamValues: The parameter values (common to all synapses of the population) for the weight update model.
- WeightUpdateModel::VarValues weightVarValues: The initial values for the weight update model's state variables
- PostsynapticModel::ParamValues postsynapticParamValues: The parameter values (common to all postsynaptic neurons) for the postsynaptic model.
- PostsynapticModel::VarValues postsynapticVarValues: The initial values for the postsynaptic model's state variables

6.4 Neuron models 27

Note

If the synapse conductance definition type is "GLOBALG" then the global value of the synapse parameters is taken from the initial value provided in weightVarValues.

```
Previous | Top | Next
```

### 6.4 Neuron models

There is a number of predefined models which can be used with the NNmodel::addNeuronGroup function:

• NeuronModels::RulkovMap

• NeuronModels::Izhikevich

NeuronModels::IzhikevichVariable

NeuronModels::SpikeSource

NeuronModels::Poisson

· NeuronModels::TraubMiles

NeuronModels::TraubMilesFast

NeuronModels::TraubMilesAlt

NeuronModels::TraubMilesNStep

### 6.4.1 Defining your own neuron type

In order to define a new neuron type for use in a GeNN application, it is necessary to define a new class derived from NeuronModels::Base. For convenience the methods this class should implement can be implemented using macros:

- DECLARE\_MODEL(TYPE, NUM\_PARAMS, NUM\_VARS): declared the boilerplate code required for the model e.g. the correct specialisations of NewModels::ValueBase used to wrap the neuron model parameters and values.
- SET\_SIM\_CODE(SIM\_CODE): where SIM\_CODE contains the code for executing the integration of the model for one time stepWithin this code string, variables need to be referred to by , where NAME is the name of the variable as defined in the vector varNames. The code may refer to the predefined primitives DT for the time step size and for the total incoming synaptic current. It can also refer to a unique ID (within the population) using .
- SET\_THRESHOLD\_CONDITION\_CODE(THRESHOLD\_CONDITION\_CODE) defines the condition for true spike detection.
- SET\_PARAM\_NAMES() defines the names of the model parameters. If defined as NAME here, they can then be referenced as in the code string. The length of this list should match the NUM\_PARAM specified in DECLARE\_MODEL. Parameters are assumed to be always of type double.
- SET\_VARS() defines the names and type strings (e.g. "float", "double", etc) of the neuron state variables.
   The variables defined here as NAME can then be used in the syntax in the code string.

For example, using these macros, we can define a leaky integrator  $\tau \frac{dV}{dt} = -V + I_{\rm syn}$  solved using Euler's method:

```
class LeakyIntegrator : public NeuronModels::Base
{
public:
    DECLARE_MODEL(LeakyIntegrator, 1, 1);

SET_SIM_CODE("$(V) += (-$(V) +$(Isyn)) *(DT/$(tau));");
```

```
SET_THRESHOLD_CONDITION_CODE("$(V) >= -50.0");
SET_PARAM_NAMES({"tau"});
SET_VARS(({"V", "scalar"}});
};
```

Additionally "dependent parameters" can be defined. Dependent parameters are a mechanism for enhanced efficiency when running neuron models. If parameters with model-side meaning, such as time constants or conductances always appear in a certain combination in the model, then it is more efficient to pre-compute this combination and define it as a dependent parameter.

For example, because the equation defining the previous leaky integrator example has an algebraic solution, it can be more accurately solved as follows - using a derived parameter to calculate  $\exp\left(\frac{-t}{\tau}\right)$ :

GeNN provides two additional features that might be useful when defining more complex neuron models. "Support code" enables a code snippet to be defined that contains supporting code that will be utilized in the other code snippets. Typically, these are functions that are needed in the sim code or threshold condition code. If possible, these should be defined as \_\_\_host\_\_ \_\_\_device\_\_ functions so that both GPU and CPU versions of GeNN code have an appropriate support code function available. The support code is protected with a namespace so that it is exclusively available for the neuron population whose neurons define it. Support code is added to a model using the SET\_SUPPORT\_CODE() macro, for example:

```
SET_SUPPORT_CODE("__device__ __host__ scalar mysin(float x){ return sin(x); }");
```

The final feature is "Extra global parameters". These parameters are common to all neurons in the population and, unlike the standard neuron parameters, can be varied at runtime. This could, for example, be used to provide a global reward signal. These parameters are defined by using the SET\_EXTRA\_GLOBAL\_PARAMS() macro to specify a list of variable names and type strings (like the SET\_VARS() macro). For example:

```
SET_EXTRA_GLOBAL_PARAMS({{"R", "float"}});
```

These variables are available to all neurons in the population. They can also be used in synaptic code snippets; in this case it need to be addressed with a \_pre or \_post postfix.

For example, if the model with the "R" parameter was used for the pre-synaptic neuron population, the weight update model of a synapse population could have simulation code like:

```
SET_SIM_CODE("$(x) = $(x) + $(R_pre);");
```

where we have assumed that the weight update model has a variable x and our synapse type will only be used in conjunction with pre-synaptic neuron populations that do have the extra global parameter R. If the pre-synaptic population does not have the required variable/parameter, GeNN will fail when compiling the kernels.

Once defined in this way, new neuron models classes, can be used in network descriptions by referring to their type e.g.

```
\label{lem:networkModel.addNeuronPopulation<LeakyIntegrator>("Neurons", 1, \\ LeakyIntegrator::ParamValues(20.0 /*tau*/), \\ LeakyIntegrator::VarValues(0.0/*V*/)); \\ \end{aligned}
```

Previous | Top | Next

## 6.5 Weight update models

Currently 3 predefined weight update models are available:

· WeightUpdateModels::StaticPulse

· WeightUpdateModels::StaticGraded

· WeightUpdateModels::PiecewiseSTDP

For more details about these built-in synapse models, see [2].

### 6.5.1 Defining a new weight update model

Like the neuron models discussed in Defining your own neuron type, new weight update models are created by defining a class. Weight update models should all be derived from WeightUpdateModel::Base and, for convenience, the methods a new weight update model should implement can be implemented using macros:

- DECLARE\_MODEL(TYPE, NUM\_PARAMS, NUM\_VARS), SET\_DERIVED\_PARAMS(), SET\_PARAM\_N → AMES(), SET\_VARS() and SET\_EXTRA\_GLOBAL\_PARAMS() perform the same roles as they do in the neuron models discussed in Defining your own neuron type.
- SET\_SIM\_CODE(SIM\_CODE): defines the simulation code that is used when a true spike is detected. The update is performed only in timesteps after a neuron in the presynaptic population has fulfilled its threshold detection condition. Typically, spikes lead to update of synaptic variables that then lead to the activation of input into the post-synaptic neuron. Most of the time these inputs add linearly at the post-synaptic neuron. This is assumed in GeNN and the term to be added to the activation of the post-synaptic neuron should be assigned to the \$(addtoinsyn) variable. For example

```
SET_SIM_CODE(
   "\$(addtoinsyn) = $(inc);\n"
   "\$(updatelinsyn)");
```

where "inc" is a parameter of the weight update model that defines a constant increment of the synaptic input of a post-synaptic neuron for each pre-synaptic spike. Once \$(addtoinsyn) has been assigned, the \$(updatelinsyn) keyword should be used to indicate that the summation of synaptic inputs can now occur. This can then be followed by updates on the internal synapse variables that may have contributed to addtoinSyn. For an example, see WeightUpdateModels::StaticPulse for a simple synapse update model and Weight UpdateModels::PiecewiseSTDP for a more complicated model that uses STDP.

• SET\_EVENT\_THRESHOLD\_CONDITION\_CODE(EVENT\_THRESHOLD\_CONDITION\_CODE) defines a condition for a synaptic event. This typically involves the pre-synaptic variables, e.g. the membrane potential:

```
SET_EVENT_THRESHOLD_CONDITION_CODE("$(V_pre) > -0.02");
```

Whenever this expression evaluates to true, the event code set using the SET\_EVENT\_CODE() macro is executed. For an example, see WeightUpdateModels::StaticGraded.

- SET\_EVENT\_CODE(EVENT\_CODE) defines the code that is used when the event threshold condition is met (as set using the SET\_EVENT\_THRESHOLD\_CONDITION\_CODE() macro).
- SET\_LEARN\_POST\_CODE(LEARN\_POST\_CODE) defines the code which is used in the learnSynapses Post kernel/function, which performs updates to synapses that are triggered by post-synaptic spikes. This is typically used in STDP-like models e.g. WeightUpdateModels::PiecewiseSTDP.
- SET\_SYNAPSE\_DYNAMICS\_CODE(SYNAPSE\_DYNAMICS\_CODE) defines code that is run for each synapse, each timestep i.e. unlike the others it is not event driven. This can be used where synapses have internal variables and dynamics that are described in continuous time, e.g. by ODEs. However using this mechanism is typically computationally very costly because of the large number of synapses in a typical network.

• SET\_NEEDS\_PRE\_SPIKE\_TIME(PRE\_SPIKE\_TIME\_REQUIRED) and SET\_NEEDS\_POST\_SPIKE\_TI

ME(POST\_SPIKE\_TIME\_REQUIRED) define whether the weight update needs to know the times of the spikes emitted from the pre and postsynaptic populations. For example an STDP rule would be likely to require:

```
SET_NEEDS_PRE_SPIKE_TIME(true);
SET_NEEDS_POST_SPIKE_TIME(true);
```

All code snippets can be used to manipulate any synapse variable and so implement both synaptic dynamics and learning processes.

## 6.6 Synaptic matrix types

Synaptic matrix types are made up of two components: SynapseMatrixConnectivity and SynapseMatrixWeight. SynapseMatrixConnectivity defines what data structure is used to store the synaptic matrix:

- SynapseMatrixConnectivity::DENSE stores synaptic matrices as a dense matrix. Large dense matrices require a large amount of memory and if they contain a lot of zeros it may be inefficient.
- SynapseMatrixConnectivity::SPARSE stores synaptic matrices in a Yale format. In general, this is less efficient to traverse using a GPU than the dense matrix format but does result in large memory savings for large matrices. Sparse matrices are stored in a struct named SparseProjection which contains the following members:
  - 1. unsigned int connN: number of connections in the population. This value is needed for allocation of arrays. The indices that correspond to these values are defined in a pre-to-post basis by the subsequent arrays.
  - 2. unsigned int ind (of size connN): Indices of corresponding postsynaptic neurons concatenated for each presynaptic neuron.
  - 3. unsigned int \*indInG with one more entry than there are presynaptic neurons. This array defines from which index in the synapse variable array the indices in ind would correspond to the presynaptic neuron that corresponds to the index of the indlnG array, with the number of connections being the size of ind. More specifically, indIng[i+1]-indlng[i] would give the number of postsynaptic connections for neuron i. For example, consider a network of two presynaptic neurons connected to three postsynaptic neurons: 0th presynaptic neuron connected to 1st and 2nd postsynaptic neurons, the 1st presynaptic neuron connected to 0th and 2nd neurons. The struct SparseProjection should have these members, with indexing from 0:

```
ConnN = 4
ind = [1 2 0 2]
indIng = [0 2 4]
```

Weight update model variables associated with the sparsely connected synaptic population will be kept in an array using this conductance for indexing. For example, a variable called g will be kept in an array such as:  $g=[g\_Pre0-Post1\ g\_pre0-post2\ g\_pre1-post0\ g\_pre1-post2]$  If there are no connections for a presynaptic neuron, then g[indlng[n]]=gp[indlng[n]+1]. See tools/gen\_syns\_sparse\_lzhModel used in lzh\_sparse project to see a working example.

• SynapseMatrixConnectivity::BITMASK is an alternative sparse matrix implementation where which synapses within the matrix are present is specified as a binary array (see Insect olfaction model).

Furthermore the SynapseMatrixWeight defines how

- SynapseMatrixWeight::INDIVIDUAL allows each individual synapse to have unique weight update model variables. Their values must be initialised at runtime and, if running on the GPU, copied across from the user side code, using the pushXXXXXToDevice function, where XXXX is the name of the synapse population.
- SynapseMatrixWeight::GLOBAL saves memory by only maintaining one copy of the weight update model variables. This is automatically initialized to the initial value passed to NNmodel::addSynapsePopulation.

Only certain combinations of SynapseMatrixConnectivity and SynapseMatrixWeight are sensible therefore, to reduce confusion, the SynapseMatrixType enumeration defines the following options which can be passed to N← Nmodel::addSynapsePopulation:

- SynapseMatrixType::SPARSE\_GLOBALG
- SynapseMatrixType::SPARSE\_INDIVIDUALG
- SynapseMatrixType::DENSE GLOBALG
- SynapseMatrixType::DENSE INDIVIDUALG
- SynapseMatrixType::BITMASK GLOBALG

# 6.7 Postsynaptic integration methods

There are currently 2 built-in postsynaptic integration methods:

- · PostsynapticModels::ExpCond
- PostsynapticModels::DeltaCurr

### 6.7.1 Defining a new postsynaptic model

The postsynaptic model defines how synaptic activation translates into an input current (or other input term for models that are not current based). It also can contain equations defining dynamics that are applied to the (summed) synaptic activation, e.g. an exponential decay over time.

In the same manner as to both the neuron and weight update models discussed in Defining your own neuron type and Defining a new weight update model, postsynamic model definitions are encapsulated in a class derived from PostsynapticModels::Base. Again, the methods that a postsynaptic model should implement can be implemented using the following macros:

- DECLARE\_MODEL(TYPE, NUM\_PARAMS, NUM\_VARS), SET\_DERIVED\_PARAMS(), SET\_PARAM\_N
   AMES(), SET\_VARS() perform the same roles as they do in the neuron models discussed in Defining your own neuron type.
- SET\_DECAY\_CODE(DECAY\_CODE) defines the code which provides the continuous time dynamics for the summed presynaptic inputs to the postsynaptic neuron. This usually consists of some kind of decay function.
- SET\_CURRENT\_CONVERTER\_CODE(CURRENT\_CONVERTER\_CODE) defines the code specifying the conversion from synaptic inputs to a postsynaptic neuron input current (Isyn). e.g. for a conductance model:

```
{\tt SET\_CURRENT\_CONVERTER\_CODE("\$(inSyn) * (\$(E) - \$(V))");}
```

where "E" is a postsynaptic model parameter specifying reversal potential and "V" is the variable containing the postsynaptic neuron's membrane potential.

```
Previous | Top | Next
```

# 7 Tutorial 1

In this tutorial we will go through step by step instructions how to create and run a GeNN simulation starting from scratch. Normally, we recommend users to use one of the example projects as a starting point but it can be very instructive to go through the necessary steps one by one once to appreciate what parts make a GeNN simulation.

### 7.1 The Model Definition

In this tutorial we will use a pre-defined neuron model type (TRAUBMILES) and create a simulation of ten Hodgkin-Huxley neurons [5] without any synaptic connections. We will run this simulation on a GPU and save the results to stdout.

The first step is to write a model definition function in a model definition file. Create a new empty file tenHH $\leftarrow$  Model.cc with your favourite editor, e.g.

```
>> emacs tenHHModel.cc &
```

### Note

The ">>" in the example code snippets refers to a shell prompt in a unix shell, do not enter them as part of your shell commands.

The model definition file contains the definition of the network model we want to simulate. First, we need to include the GeNN model specification code modelSpec.h. Then the model definition takes the form of a function named modelDefinition that takes one argument, passed by reference, of type NNmodel. Type in your tenHH Model.cc file:

```
// Model definintion file tenHHModel.cc
#include "modelSpec.h"

void modelDefinition(NNmodel &model)
{
    // definition of tenHHModel
}
```

Now we need to fill the actual model definition. Three standard elements to the 'modelDefinition function are initialising GeNN, setting the simulation step size and setting the name of the model:

```
initGeNN();
model.setDT(0.1);
model.setName("tenHHModel");
```

### Note

With this we have fixed the integration time step to 0.1 in the usual time units. The typical units in GeNN are ms, mV, nF, and  $\mu$ S. Therefore, this defines DT= 0.1 ms. The name of the model given in the setName method does not need to match the file name of the model definition file. However, we strongly recommend it and if conflicting, the file name of the model definition file will prevail.

Making the actual model definition makes use of the NNmodel::addNeuronPopulation and NNmodel::addSynapse Population member functions of the NNmodel object. The arguments to a call to NNmodel::addNeuronPopulation are

- NeuronModel: template parameter specifying the neuron model class to use
- const std::string &name: the name of the population
- unsigned int size: The number of neurons in the population
- const NeuronModel::ParamValues &paramValues: Parameter values for the neurons in the population
- const NeuronModel::VarValues &varValues: Initial values for neuron variables

We first create the parameter and initial variable arrays,

7.1 The Model Definition 33

```
// definition of tenHHModel
NeuronModels::TraubMiles::ParamValues p(
                           // 0 - gNa: Na conductance in muS
// 1 - ENa: Na equi potential in mV
   7.15,
   50.0,
   1.43,
                           // 2 - gK: K conductance in muS
                      // 2 - gK: K conductance in muS
// 3 - EK: K equi potential in mV
// 4 - gl: leak conductance in muS
// 5 - El: leak equi potential in r
// 6 - Cmem: membr. capacity densit
  -95.0,
  0.02672,
                           // 5 - El: leak equi potential in mV
                           // 6 - Cmem: membr. capacity density in nF
  0.143
NeuronModels::TraubMiles::VarValues ini(
                     // 0 - membrane potential V
// 1 - prob. for Na channel activation m
// 2 - prob. for not Na channel blocking h
// 3 - prob. for K channel activation n
   0.0529324,
   0.3176767,
  0.5961207
```

### Note

The comments are obviously only for clarity, they can in principle be omitted. To avoid any confusion about the meaning of parameters and variables, however, we recommend strongly to always include comments of this type.

Having defined the parameter values and initial values we can now create the neuron population,

The model definition then needs to end on calling

```
model.finalize();
```

This completes the model definition in this example. The complete tenHHModel.cc file now should look like this:

```
// Model definintion file tenHHModel.cc
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
  // definition of tenHHModel
  initGeNN();
  model.setDT(0.1);
  model.setName("tenHHModel");
  NeuronModels::TraubMiles::ParamValues p(
              // 0 - gNa: Na conductance in muS // 1 - ENa: Na equi potential in mV
    50.0.
                     // 2 - gK: K conductance in muS
    1.43,
                // 3 - EK: K equi potential in mV

// 4 - gl: leak conductance in muS

// 5 - El: leak equi potential in mV
   -95.0,
    0.02672,
    -63.563,
    0.143
                      // 6 - Cmem: membr. capacity density in nF
  NeuronModels::TraubMiles::VarValues ini(
                // 0 - membrane potential V
     -60.0,
    0.0529324,
                     // 1 - prob. for Na channel activation m
                     // 2 - prob. for not Na channel blocking h // 3 - prob. for K channel activation n
    0.3176767,
0.5961207
 model.addNeuronPopulation<NeuronModels::TraubMiles>("Pop1", 10
       , p, ini);
  model.finalize();
```

This model definition suffices to generate code for simulating the ten Hodgkin-Huxley neurons on the a GPU or CPU. The second part of a GeNN simulation is the user code that sets up the simulation, does the data handling for input and output and generally defines the numerical experiment to be run.

### 7.2 User Code

For the purposes of this tutorial we will initially simply run the model for one simulated second and record the final neuron variables into a file. GeNN provides the code for simulating the model in a function called stepTimeCP U() (execution on CPU only) or stepTimeGPU() (execution on a GPU). To make use of this code, we need to define a minimal C/C++ main function. Open a new empty file tenHHSimulation.cc in an editor and type

```
// tenHHModel simulation code
#include "tenHHModel_CODE/definitions.h"
int main()
{
   allocateMem();
   initialize();
   return 0;
}
```

This boiler plate code includes the relevant model definition file we completed earlier and the header file of entry point to the generated code definitions.h in the subdirectory tenHHModel\_CODE where GeNN deposits all generated code.

Calling allocateMem() allocates the memory structures for all neuron variables and initialize() sets the initial values and copies values to the GPU.

Now we can use the generated code to integrate the neuron equations provided by GeNN for 1000ms (  $\frac{1000}{DT}$  timesteps). To do so, we add after initialize ();

```
for(int i = 0; i < (int)(1000.0 / DT); i++) {
  stepTimeGPU();
}</pre>
```

and we need to copy the result, and output it to stdout,

```
pullPop1StateFromDevice();
for (int j= 0; j < 10; j++) {
  cout << VPop1[j] << " ";
  cout << mPop1[j] << " ";
  cout << hPop1[j] << " ";
  cout << nPop1[j] << endl;
}</pre>
```

 $\verb|pullPop1StateFromDevice|()| copies all relevant state variables of the Pop1 neuron group from the GPU to the CPU main memory. Then we can output the results to stdout by looping through all 10 neurons and outputting the state variables VPop1, mPop1, hPop1, nPop1.$ 

Note

The naming convention for variables in GeNN is the variable name defined by the neuron type, here TraubMiles defining V, m, h, and n, followed by the population name, here Pop1.

This completes the user code. The complete tenHHSimulation.cc file should now look like

```
// tenHHModel simulation code
#include "tenHHModel_CODE/definitions.h"

int main()
{
    allocateMem();
    initialize();
    for(int i = 0; i < (int)(1000.0 / DT); i++) {
        stepTimeGPU();
    }
    pullPoplStateFromDevice();
    for (int j= 0; j < 10; j++) {
        cout << VPopl[j] << " ";
        cout << mPopl[j] << " ";
        cout << nPopl[j] << " ";
        cout << nPopl[j] << " endl;
    }
    return 0;</pre>
```

7.3 Makefile 35

### 7.3 Makefile

A GeNN simulation is built with a simple Makefile. On Unix systems we typically name it GNUmakefile. Create this file and enter

This defines that the final executable of this simulation is named tenHHSimulation and the simulation code is given in the file tenHHSimulation.cc that we completed above.

Now we are ready to compile and run the simulation

## 7.4 Making and Running the Simulation

To build the model and generate the GeNN code, type in a terminal where you are in the directory containing your tenHHModel.cc file,

```
>> genn-buildmodel.sh tenHHModel.cc
```

If your environment variables GENN\_PATH and CUDA\_PATH are correctly configured, you should see some compile output ending in Model build complete .... Now type

make

This should compile your tenHHSimulation executable and you can execute it with

```
./tenHHSimulation
```

The output you obtain should look like

```
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
```

# 7.5 Reading

This is not particularly interesting as we are just observing the final value of the membrane potentials. To see what is going on in the meantime, we need to copy intermediate values from the device and save them into a file. This can be done in many ways but one sensible way of doing this is to replace the calls to stepTimeGPU in tenHHSimulation.cc with something like this:

```
ofstream os("tenHH_output.V.dat");
for (int i= 0; i < 10000; i++) {
    stepTimeGPU();

    pullPop1StateFromDevice();
    os << t << " ";
    for (int j= 0; j < 10; j++) {
        os << VPop1[j] << " ";
    }
    os << endl;
}
os.close();</pre>
```

Note

t is a global variable updated by the GeNN code to keep track of elapsed simulation time in ms.

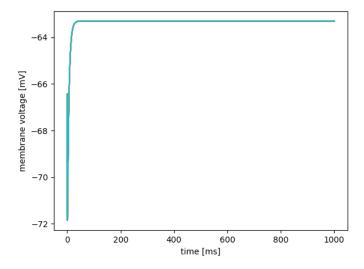
You will also need to add:

```
#include<fstream>
```

to the top of tenHHSimulation.cc. After building, making and executing,

```
genn-builmodel.sh tenHHModel.cc
make clean all
./tenHHSimulation
```

there should be a file tenHH\_output.V.dat in the same directory. If you plot column one (time) against the subsequent 10 columns (voltage of the 10 neurons), you should observe dynamics like this:



However so far, the neurons are not connected and do not receive input. As the NeuronModels::TraubMiles model is silent in such conditions, the membrane voltages of the 10 neurons will simply drift from the -60mV they were initialised at to their resting potential.

Previous | Top | Next

# 8 Tutorial 2

In this tutorial we will learn to add synapsePopulations to connect neurons in neuron groups to each other with synatic models. As an example we will connect the ten Hodgkin-Huxley neurons from tutorial 1 in a ring of excitatory synapses.

First, copy the files from Tutorial 1 into a new directory and rename them to new names, e.g.

```
>> cp -r tenHH_project tenHHRing_project
>> cd tenHHRing_project
>> mv tenHHModel.cc tenHHRingModel.cc
>> mv tenHHSimulation.cc tenHHRingSimulation.cc
```

Now, we need to add a synapse group to the model that allows to connect neurons from the Pop1 group to connect to other neurons of this group. Open tenHHRingModel.cc, change the model name inside,

```
model.setName("tenHHRing");
```

# 8.1 Adding Synaptic connections

Now we need additional initial values and parameters for the synapse and post-synaptic models. We will use the standard WeightUpdateModels::StaticPulse weight update model and PostsynapticModels::ExpCond post-synaptic model. They need the following initial variables and parameters:

#### Note

the WeightUpdateModels::StaticPulse weight update model has no parameters and the PostsynapticModels
::ExpCond post-synaptic model has no state variables.

We can then add a synapse population at the end of the modelDefinition (...) function,

```
model.addSynapsePopulation<WeightUpdateModels::StaticPulse
    , PostsynapticModels::ExpCond>(
    "Poplself", SynapseMatrixType::DENSE_INDIVIDUALG, 10,
    "Popl", "Popl",
    {}, s_ini,
    ps_p, {});
```

The addSynapsePopulation parameters are

- WeightUpdateModel: template parameter specifying the type of weight update model (derived from Weight
   — UpdateModels::Base).
- PostsynapticModel: template parameter specifying the type of postsynaptic model (derived from PostsynapticModels::Base).
- · name string containing unique name of synapse population.
- mtype how the synaptic matrix associated with this synapse population should be represented. Here SynapseMatrixType::DENSE\_INDIVIDUALG means that there will be a dense connectivity matrix with seperate values for each entry.
- delayStep integer specifying number of timesteps of propagation delay that spikes travelling through this synapses population should incur (or NO DELAY for none)
- · src string specifying name of presynaptic (source) population
- · trg string specifying name of postsynaptic (target) population
- weightParamValues parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.
- weightVarValues initial state variable values for weight update model wrapped in WeightUpdateModel::Var
   Values object.
- postsynapticParamValues parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.
- postsynapticVarValues initial state variable values for postsynaptic model wrapped in PostsynapticModel::

  VarValues object. Adding the addSynapsePopulation command to the model definition informs GeNN that there will be synapses between the named neuron populations, here between population Pop1 and itself. The detailed connectivity as defined by the variables g, we have still to define in the setup of our simulation. As always, the modelDefinition function ends on

```
model.finalize();
```

At this point our model definition file tenHHRingModel.cc should look like this

```
// Model definition file tenHHRing.cc
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
  // definition of tenHHRing
  initGeNN();
  model.setDT(0.1);
  model.setName("tenHHRing");
  NeuronModels::TraubMiles::ParamValues p(
               // 0 - gNa: Na conductance in muS
    7.15.
                    // 1 - ENa: Na conductance in muS

// 1 - ENa: Na equi potential in mV

// 2 - gK: K conductance in muS

// 3 - FV: "
    50.0,
    1.43,
    -95.0,
                      // 3 - EK: K equi potential in mV
                   // 3 - EK: K equi potential in muS
// 4 - gl: leak conductance in muS
// 5 - El: leak equi potential in mV
    0.02672,
    -63.563,
    0.143
                   // 6 - Cmem: membr. capacity density in nF
  NeuronModels::TraubMiles::VarValues ini(
                     // 0 - membrane potential V // 1 - prob. for Na channel activation m
    0.0529324.
                      // 2 - prob. for not Na channel blocking h
    0.3176767,
                     // 3 - prob. for K channel activation n
    0.5961207
  model.addNeuronPopulation<NeuronModels::TraubMiles>("Pop1", 10
      , p, ini);
  WeightUpdateModels::StaticPulse::VarValues s_ini(
    0.0 \ // \ 0 - g: the synaptic conductance value
  PostsynapticModels::ExpCond::ParamValues ps_p(
             // 0 - tau_S: decay time constant for S [ms]
// 1 - Erev: Reversal potential
   1.0,
    -80.0
  ):
  model.addSynapsePopulation<WeightUpdateModels::StaticPulse</pre>
       , PostsynapticModels::ExpCond>(
     "Pop1self", SynapseMatrixType::DENSE_INDIVIDUALG,
      NO_DELAY,
     "Pop1", "Pop1", tenHHRingSimulation
    {}, s_ini,
    ps p, {});
  model.finalize();
```

# 8.2 Defining the Detailed Synaptic Connections

Open the tenHHRingSimulation.cc file and update the file names of includes:

```
// tenHHRingModel simulation code
#include "tenHHRingModel.cc"
#include "tenHHRingModel_CODE/definitions.h"
```

Now we need to add code to generate the desired ring connectivity.

```
allocateMem();
initialize();
// define the connectivity
int pre, post;
for (int i= 0; i < 10; i++) {
   pre= i;
   post= (i+1)%10;
   gPoplself[pre*10+post]= -0.2;
}
pushPoplselfStateToDevice();</pre>
```

After memory allocation and initialization gPop1self will contain only zeros. We then assign in the loop a non-zero conductivity of 0.2  $\mu$ S to all synapses from neuron i to i+1 (and 9 to 0 to close the ring).

After adjusting the GNUmakefile to read

```
EXECUTABLE :=tenHHRingSimulation
SOURCES :=tenHHRingSimulation.cc
include $(GENN_PATH)/userproject/include/makefile_common_gnu.mk
```

we can build the model

```
>> genn-buildmodel.sh tenHHRingModel.cc
```

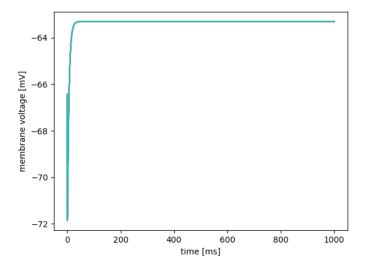
and make it

```
>> make clean all
```

After this there should be an executable tenHHRingSimulation, which can be executed,

```
>> ./tenHHRingSimulation
```

However if we plot the content of columns one against the subsequent 10 columns of tenHHexample.V.dat it looks very similar as in Tutorial 1



This is because none of the neurons are spiking so there are no spikes to propagate around the ring.

# 8.3 Providing initial stimuli

We can use a NeuronModels::SpikeSource to provide an initial spike in the first timestep to begin spikes propagating around the ring. Firstly we need to add it to the network by adding the following to the end of the  $model \leftarrow Definition(...)$  function:

we can then initialise this connection's connectivity matrix in tenHHRingSimulation.cc file

```
// define stimuli connectivity
gStimPop1[0]= -0.2;
pushStimPop1StateToDevice();
```

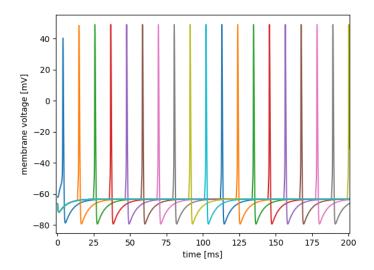
and finally inject a spike in the first timestep

```
if(i == 0) {
  glbSpkStim[0] = 0;
  glbSpkCntStim[0] = 1;
  pushStimSpikesToDevice();
}
```

At this point our model definition file tenHHRingSimulation.cc should look like this

```
// tenHHRing simulation code
#include "tenHHRing_CODE/definitions.h"
#include <fstream>
int main()
  allocateMem();
  initialize();
  // define the connectivity
  int pre, post;
for (int i= 0; i < 10; i++) {
    pre= i;</pre>
      post= (i+1)%10;
      gPop1self[pre*10+post] = -0.2;
  pushPop1selfStateToDevice();
  // define stimuli connectivity
  gStimPop1[0] = -0.2;
  pushStimPop1StateToDevice();
  ofstream os("tenHHRing_output.V.dat");
  for (int i= 0; i < 10000; i++) {
    if(i == 0) {
           glbSpkStim[0] = 0;
           glbSpkCntStim[0] = 1;
           pushStimSpikesToDevice();
      stepTimeGPU();
      pullPop1StateFromDevice();
       if (glbSpkCntPop1[0] > 0)
           printf("GADZOOKS!\n");
      os << t << " ";
for (int j= 0; j < 10; j++) {
    os << VPop1[j] << " ";
        s << endl;
  os.close();
  return 0;
```

Finally if we build, make and run this model; and plot the first 200 ms of the ten neurons' membrane voltages - they now looks like this:



Previous | Top | Next

# 9 Best practices guide

GeNN generates code according to the network model defined by the user, and allows users to include the generated code in their programs as they want. Here we provide a guideline to setup GeNN and use generated functions. We recommend users to also have a look at the Examples, and to follow the tutorials Tutorial 1 and Tutorial 2.

### 9.1 Creating and simulating a network model

The user is first expected to create an object of class NNmodel by creating the function modelDefinition() which includes calls to following methods in correct order:

- initGeNN();
- NNmodel::setDT();
- NNmodel::setName();

Then add neuron populations by:

NNmodel::addNeuronPopulation();

for each neuron population. Add synapse populations by:

NNmodel::addSynapsePopulation();

for each synapse population.

The modelDefinition() needs to end with calling NNmodel::finalize().

Other optional functions are explained in NNmodel class reference. At the end the function should look like this:

```
void modelDefinition(NNModel &model) {
  initGeNN();
  model.setDT(0.5);
  model.setName("YourModelName");
  model.addNeuronPopulation(...);
  ...
  model.addSynapsePopulation(...);
  ...
  model.finalize();
}
```

modelSpec.h should be included in the file where this function is defined.

This function will be called by generateALL.cc to create corresponding CPU and GPU simulation codes under the <YourModelName>\_CODE directory.

These functions can then be used in a .cu file which runs the simulation. This file should include <YourModel← Name>\_CODE/runner.cc. Generated code differ from one model to the other, but core functions are the same and they should be called in correct order. First, the following variables should be defined and initialized:

- NNmodel model // initialized by calling modelDefinition(model)
- · Array containing current input (if any)

The following are declared by GeNN but should be initialized by the user:

Poisson neuron offset and rates (if any)

- · Connectivity matrices (if sparse)
- Neuron and synapse variables (if not initialising to the homogeneous initial value provided during model
   — Definition)

Core functions generated by GeNN to be included in the user code include:

- allocateMem()deviceMemAllocate()
- initialize()
- initializeAllSparseArrays()
- convertProbabilityToRandomNumberThreshold()
- convertRateToRandomNumberThreshold()
- push<neuron or synapse name>StateToDevice()
- pull<neuron or synapse name>StateFromDevice()
- push<neuron name>SpikesToDevice()
- pull<neuron name>SpikesFromDevice()
- push<neuron name>SpikesEventsToDevice()
- pull<neuron name>SpikesEventsFromDevice()
- push<neuron name>CurrentSpikesToDevice()
- pull<neuron name>CurrentSpikesFromDevice()
- push<neuron name>CurrentSpikesEventsToDevice()
- pull<neuron name>CurrentSpikesEventsFromDevice()
- copyStateToDevice()
- copyStateFromDevice()
- copySpikesToDevice()
- copySpikesFromDevice()
- copySpikesEventsToDevice()
- copySpikesEventsFromDevice()
- copyCurrentSpikesToDevice()
- copyCurrentSpikesFromDevice()
- copyCurrentSpikesEventsToDevice()
- copyCurrentSpikesEventsFromDevice()
- stepTimeCPU()
- stepTimeGPU()
- freeMem()

Before calling the kernels, make sure you have copied the initial values of all the neuron and synapse variables in the GPU. You can use the  $push\colon synapse name\colon synapse nam$ 

# 9.2 Floating point precision

Double precision floating point numbers are supported by devices with compute capability 1.3 or higher. If you have an older GPU, you need to use single precision floating point in your models and simulation.

GPUs are designed to work better with single precision while double precision is the standard for CPUs. This difference should be kept in mind while comparing performance.

While setting up the network for GeNN, double precision floating point numbers are used as this part is done on the CPU. For the simulation, GeNN lets users choose between single or double precision. Overall, new variables in the generated code are defined with the precision specified by NNmodel::setPrecision(unsigned int), providing GENN\_FLOAT or GENN\_DOUBLE as argument. GENN\_FLOAT is the default value. The keyword scalar can be used in the user-defined model codes for a variable that could either be single or double precision. This keyword is detected at code generation and substituted with "float" or "double" according to the precision set by NNmodel ::setPrecision(unsigned int).

There may be ambiguities in arithmetic operations using explicit numbers. Standard C compilers presume that any number defined as "X" is an integer and any number defined as "X.Y" is a double. Make sure to use the same precision in your operations in order to avoid performance loss.

# 9.3 Working with variables in GeNN

### 9.3.1 Model variables

User-defined model variables originate from classes derived off the NeuronModels::Base, WeightUpdateModels∴Base or PostsynapticModels::Base classes. The name of model variable is defined in the model type, i.e. with a statement such as

```
SET_VARS({{"V", "scalar"}});
```

When a neuron or synapse population using this model is added to the model, the full GeNN name of the variable will be obtained by concatenating the variable name with the name of the population. For example if we a add a population called Pop using a model which contains our V variable, a variable VPop of type scalar\* will be available in the global namespace of the simulation program. GeNN will pre-allocate this C array to the correct size of elements corresponding to the size of the neuron population. GeNN will also free these variables when the provided function freeMem() is called. Users can otherwise manipulate these variable arrays as they wish. For convenience, GeNN provides functions pullXXStatefromDevice() and pushXXStatetoDevice() to copy the variables associated to a neuron population XX from the device into host memory and vice versa. E.g.

```
pullPopStateFromDevice();
```

would copy the C array VPop from device memory into host memory (and any other variables that the population Pop may have).

The user can also directly use CUDA memory copy commands independent of the provided convenience functions. The relevant device pointers for all variables that exist in host memory have the same name as the host variable but are prefixed with d\_. For example, the copy command that would be contained in pullPopStateFromcDevice() will look like

```
unsigned int size = sizeof(scalar) * nPop;
cudaMemcpy(VPop, d_VPop, size, cudaMemcpyDeviceToHost);
```

where nPop is an integer containing the population size of the Pop population.

Thes conventions also apply to the the variables of postsynaptic and weight update models.

Note

The content of gENIN needs to be interpreted differently for DENSE connectivity and sparse matrix based SPARSE connectivity representations. For DENSE connectivity gENIN would contain "n\_pre" times "n\_post" elements, ordered along the pre-synaptic neurons as the major dimension, i.e. the value of gENIN for the ith pre-synaptic neuron and the jth post-synaptic neuron would be gENIN [ $i*n_post+j$ ]. The arrangement of values in the SPARSE representation is explained in section subsect32

Be aware that the above naming conventions do assume that variables from the weightupdate models and the postSynModels that are used together in a synapse population are unique. If both the weightupdate model and the postSynModel have a variable of the same name, the behaviour is undefined.

#### 9.3.2 Built-in Variables in GeNN

GeNN has no explicitly hard-coded synapse and neuron variables. Users are free to name the variable of their models as they want. However, there are some reserved variables that are used for intermediary calculations and communication between different parts of the generated code. They can be used in the user defined code but no other variables should be defined with these names.

- DT: Time step (typically in ms) for simulation; Neuron integration can be done in multiple sub-steps inside
  the neuron model for numerical stability (see Traub-Miles and Izhikevich neuron model variations in Neuron
  models).
- addtoinSyn: This variable is used by WeightUpdateModels::Base for updating synaptic input. The way it is modified is defined using the SET\_SIM\_CODE or SET\_EVENT\_CODE macros, therefore if a user defines her own model she should update this variable to contain the input to the post-synaptic model.
- updatelinsyn: At the end of the synaptic update by addtoinSyn, final values are copied back to the
  d\_inSyn<synapsePopulation> variables which will be used in the next step of the neuron update to provide
  the input to the postsynaptic neurons. This keyword designated where the changes to addtoinSyn have
  been completed and it is safe to update the summed synaptic input and write back to d\_inSyn<synapse
  Population> in device memory.
- inSyn: This is an intermediary synapse variable which contains the summed input into a postsynaptic neuron (originating from the addtoinSyn variables of the incoming synapses).
- Isyn: This is a local variable which contains the (summed) input current to a neuron. It is typically the sum of any explicit current input and all synaptic inputs. The way its value is calculated during the update of the postsynaptic neuron is defined by the code provided in the postsynaptic model. For example, the standard PostsynapticModels::ExpCond postsynaptic model defines

```
{\tt SET\_CURRENT\_CONVERTER\_CODE} \; ("\$ (inSyn) * (\$ (E) -\$ (V))") \; ; \\
```

which implements a conductance based synapse in which the postsynaptic current is given by  $I_{\rm syn} = g*s*(V_{\rm rev} - V_{\rm post}).$ 

Note

The addtoinSyn variables from all incoming synapses are automatically summed and added to the current value of inSyn.

The value resulting from the current converter code is assigned to Isyn and can then be used in neuron sim code like so:

```
(V) += (-(V) + (Isyn)) *DT
```

sT: This is a neuron variable containing the last spike time of each neuron and is automatically generated for
pre and postsynaptic neuron groups if they are connected using a synapse population with a weight update
model that has SET\_NEEDS\_PRE\_SPIKE\_TIME(true) or SET\_NEEDS\_POST\_SPIKE\_TIME(true) set.

In addition to these variables, neuron variables can be referred to in the synapse models by calling \$(<neuronVar~Name>\_pre) for the presynaptic neuron population, and \$(<neuronVarName>\_post) for the postsynaptic population. For example, \$(sT\_pre), \$(sT\_post), \$(V\_pre), etc.

# 9.4 Debugging suggestions

In Linux, users can call <code>cuda-gdb</code> to debug on the GPU. Example projects in the <code>userproject</code> directory come with a flag to enable debugging (DEBUG=1). genn-buildmodel.sh has a debug flag (-d) to generate debugging data. If you are executing a project with debugging on, the code will be compiled with -g -G flags. In CPU mode the executable will be run in gdb, and in GPU mode it will be run in cuda-gdb in tui mode.

On Mac, some versions of clang aren't supported by the CUDA toolkit. This is a recurring problem on Fedora as well, where CUDA doesn't keep up with GCC releases. You can either hack the CUDA header which checks compiler versions - cuda/include/host\_config.h - or just use an older XCode version (6.4 works fine).

### Note

Do not forget to switch debugging flags -g and -G off after debugging is complete as they may negatively affect performance.

Previous | Top | Next

# 10 Credits

GeNN was created by Thomas Nowotny.

Izhikevich model and sparse connectivity by Esin Yavuz.

Block size optimisations, delayed synapses and page-locked memory by James Turner.

Automatic brackets and dense-to-sparse network conversion helper tools by Alan Diamond.

User-defined synaptic and postsynaptic methods by Alex Cope and Esin Yavuz.

Example projects were provided by Alan Diamond, James Turner, Esin Yavuz and Thomas Nowotny.

# Previous | Top

# 11 Namespace Index

### 11.1 Namespace List

Here is a list of all namespaces with brief descriptions:

GENN_FLAGS	??
GENN_PREFERENCES	??
NeuronModels	??
NewModels	??
PostsynapticModels	??
StandardGeneratedSections	??
StandardSubstitutions	??
WeightUpdateModels	??

# 12 Hierarchical Index

# 12.1 Class Hierarchy

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

NewModels::Base	69
${\bf NewModels:: LegacyWrapper} {<\ } {\bf Base,\ neuronModel,\ nModels} >$	102
NeuronModels::LegacyWrapper	105
${\bf NewModels::} {\bf LegacyWrapper} {<} {\bf Base,postSynModel,postSynModels{>}}$	102
PostsynapticModels::LegacyWrapper	101
${\bf NewModels::} {\bf LegacyWrapper} {\bf < Base, weightUpdateModel, weightUpdateModels} >$	102
WeightUpdateModels::LegacyWrapper	103
NeuronModels::Base	70
МуНН	107
Neuron	111

Neuron	111
Neuron	111
NeuronModels::Izhikevich	98
Mylzhikevich	108
Mylzhikevich	108
NeuronModels::IzhikevichVariable	100
MylzhikevichVariable	110
NeuronModels::Poisson	154
NeuronModels::RulkovMap	163
NeuronModels::SpikeSource	178
NeuronModels::TraubMiles	189
NeuronModels::TraubMilesAlt	191
NeuronModels::TraubMilesFast	193
NeuronModels::TraubMilesNStep	194
PostsynapticModels::Base	72
ExpCondUserDef	95
PostsynapticModels::DeltaCurr	92
PostsynapticModels::ExpCond	94
WeightUpdateModels::Base	73
PiecewiseSTDPUserDef	152
StaticGradedUserDef	181
StaticPulseUserDef	183

	WeightUpdateModel	199
	WeightUpdateModel	199
	WeightUpdateModels::PiecewiseSTDP	149
	WeightUpdateModels::StaticGraded	179
	WeightUpdateModels::StaticPulse	182
	WeightUpdateModelSpikeEvent	213
Baselt		
	irKeyConstIter< BaseIter >	148
classi	zn	76

classol	79
CodeHelper	91
CStopWatch	91
dpclass	93
expDecayDp	96
pwSTDP	158
rulkovdp	163
errTupel	94
inputSpec	97
NamelterCtx< Container >	111
NeuronGroup	131
neuronModel	134
neuronpop	136
NNmodel	137
Parameter	148
postSynModel	156
${\tt QTIsaac} {<\hspace{1mm}{\sf ALPHA}, {\sf T}\hspace{2mm}{>}}$	159
QTIsaac< 8, unsigned long >	159
QTIsaac< ALPHA, T >::randctx	160
randomGauss	161
randomGen	162
Schmuker2014_classifier	165
SimulationNeuronPolicyPrePostVar	172
SimulationNeuronPolicyPreVar	172
SimulationSynapsePolicyDense	173
SimulationSynapsePolicySparse	174
SimulationSynapsePolicy	173
SimulationSynapsePolicyNone	174
SparseProjection	177
stdRG	184
stopWatch	185
SynapseGroup	185

<b>SynDelay</b> TestWithParam	188
SimulationTest	175
SimTest	171
SimulationTestDecoderMatrix	176
SimTest	171
${\bf Simulation Test Vars} < {\bf Neuron Policy, Synapse Policy} >$	176
SingleValueSubstitutionTest	177
NewModels::ValueBase< NumValues >	195
NewModels::ValueBase< 0 >	196
weightUpdateModel ModelBase	196
${\bf NewModels:: LegacyWrapper < ModelBase, LegacyModelType, ModelArray >}$	102
13 Class Index	
13.1 Class List	
Here are the classes, structs, unions and interfaces with brief descriptions:	
NewModels::Base Base class for all models	69
NeuronModels::Base Base class for all neuron models	70
PostsynapticModels::Base Base class for all postsynaptic models	72
WeightUpdateModels::Base Base class for all weight update models	73
classizh	76
classol  This class cpontains the methods for running the MBody1 example model	79
CodeHelper	91
CStopWatch	91
PostsynapticModels::DeltaCurr Simple delta current synapse	92

13.1 Class List 51

dpclass	93
errTupel	94
PostsynapticModels::ExpCond Exponential decay with synaptic input treated as a conductance value	94
ExpCondUserDef	95
expDecayDp Class defining the dependent parameter for exponential decay	96
inputSpec	97
NeuronModels::Izhikevich Izhikevich neuron with fixed parameters [1]	98
NeuronModels::IzhikevichVariable Izhikevich neuron with variable parameters [1]	100
PostsynapticModels::LegacyWrapper	101
NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray > Wrapper around old-style models stored in global arrays and referenced by index	102
WeightUpdateModels::LegacyWrapper Wrapper around legacy weight update models stored in weightUpdateModels array of weight UpdateModel objects	103
NeuronModels::LegacyWrapper Wrapper around legacy weight update models stored in nModels array of neuronModel objects	105
МуНН	107
Mylzhikevich	108
MylzhikevichVariable	110
NamelterCtx < Container >	111
Neuron	111
NeuronGroup	131
neuronModel Class for specifying a neuron model	134
neuronpop	136
NNmodel	137
PairKeyConstIter > BaseIter >	148
Parameter	148
WeightUpdateModels::PiecewiseSTDP  This is a simple STDP rule including a time delay for the finite transmission speed of the synapse	149
PiecewiseSTDPUserDef	152

NeuronModels::Poisson Poisson neurons	154
postSynModel  Class to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn"	156
pwSTDP  TODO This class definition may be code-generated in a future release	158
QTIsaac< ALPHA, T >	159
QTIsaac< ALPHA, T >::randctx	160
randomGauss Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution	161
randomGen Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers	162
rulkovdp Class defining the dependent parameters of the Rulkov map neuron	163
NeuronModels::RulkovMap Rulkov Map neuron	163
Schmuker2014_classifier This class cpontains the methods for running the Schmuker_2014_classifier example model	165
SimTest	171
SimulationNeuronPolicyPrePostVar	172
SimulationNeuronPolicyPreVar	172
SimulationSynapsePolicy	173
SimulationSynapsePolicyDense	173
SimulationSynapsePolicyNone	174
SimulationSynapsePolicySparse	174
SimulationTest	175
SimulationTestDecoderMatrix	176
${\bf Simulation Test Vars} {<} \ {\bf Neuron Policy}, {\bf Synapse Policy} {>}$	176
SingleValueSubstitutionTest	177
SparseProjection Class (struct) for defining a spars connectivity projection	177
NeuronModels::SpikeSource  Empty neuron which allows setting spikes from external sources	178

14 File Index 53

	WeightUpdateModels::StaticGraded	
	Graded-potential, static synapse	179
	StaticGradedUserDef	181
	WeightUpdateModels::StaticPulse	400
	Pulse-coupled, static synapse	182
	StaticPulseUserDef	183
	stdRG	184
	stopWatch	185
	SynapseGroup	185
	SynDelay	188
	NeuronModels::TraubMiles Hodgkin-Huxley neurons with Traub & Miles algorithm	189
	NeuronModels::TraubMilesAlt Hodgkin-Huxley neurons with Traub & Miles algorithm	191
	NeuronModels::TraubMilesFast Hodgkin-Huxley neurons with Traub & Miles algorithm: Original fast implementation, using 25 inner iterations	193
	NeuronModels::TraubMilesNStep Hodgkin-Huxley neurons with Traub & Miles algorithm	194
	NewModels::ValueBase < NumValues >	195
	NewModels::ValueBase < 0 >	196
	weightUpdateModel  Class to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events	196
	WeightUpdateModel	199
	WeightUpdateModelSpikeEvent	213
14	File Index	
14.	1 File List	
Her	re is a list of all files with brief descriptions:	
	MBody1_project/model/classol_sim.cc  Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model	215
	MBody_delayedSyn_project/model/classol_sim.cc  Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody_delayedSyn mushroom body model	215

MBody_individualID_project/model/classol_sim.cc  Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model	216
MBody_map_project/model/classol_sim.cc	216
MBody_map_robot1_project/model/classol_sim.cc	217
MBody_userdef_project/model/classol_sim.cc  Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model	217
MBody1_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody1 model	217
MBody_delayedSyn_project/model/classol_sim.h  Header file containing global variables and macros used in running the classol / MBody_← delayedSyn model	219
MBody_individualID_project/model/classol_sim.h  Header file containing global variables and macros used in running the classol / MBody_← individualID model	220
MBody_map_project/model/classol_sim.h	221
MBody_map_robot1_project/model/classol_sim.h	222
MBody_userdef_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody1 model	223
lib/src/codeGenUtils.cc	224
tests/unit/codeGenUtils.cc	225
codeGenUtils.h	225
CodeHelper.h	227
command_line_processing.h  This file contains some tools for parsing the argv array which contains the command line options	227
dpclass.h	228
experiment.cc	228
experiment.h	230
extra_neurons.h	231
extra_postsynapses.h	233
extra_weightupdates.h	234
GA.cc	234
gauss.cc Contains the implementation of the Gaussian random number generator class randomGauss	234
gauss.h  Pendem number generator for Coursian rendem veriable with mean 0 and standard deviation 1	225

gen_input_structured.cc	
This file is part of a tool chain for running the classol/MBody1 example model	235
gen_kcdn_syns.cc	
This file is part of a tool chain for running the classol/MBody1 example model	236
gen_kcdn_syns_fixto10K.cc	237
gen_pnkc_syns.cc	
This file is part of a tool chain for running the classol/MBody1 example model	237
gen_pnkc_syns_indivID.cc	
This file is part of a tool chain for running the classol/MBody1 example model	238
gen_pnlhi_syns.cc	
This file is part of a tool chain for running the classol/MBody1 example model	239
gen_syns_sparse.cc	
This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity	239
gen_syns_sparse_izhModel.cc	
This file is part of a tool chain for running the Izhikevich network model	240
HHVclampGA_project/generate_run.cc	
This file is used to run the HHVclampGA model with a single command line	242
lzh_sparse_project/generate_run.cc	
This file is part of a tool chain for running the classIzh/Izh_sparse example model	243
MBody1_project/generate_run.cc	
This file is part of a tool chain for running the classol/MBody1 example model	243
MBody_delayedSyn_project/generate_run.cc	
This file is part of a tool chain for running the classol/MBody_delayedSyn example model	244
MBody_individualID_project/generate_run.cc	
This file is part of a tool chain for running the classol/MBody_individualID example model	245
MBody_map_project/generate_run.cc	246
MBody_map_robot1_project/generate_run.cc	246
MBody userdef project/generate run.cc	
This file is part of a tool chain for running the classol/MBody_userdef example model	247
OneComp project/generate run.cc	
This file is part of a tool chain for running the classol/MBody1 example model	247
Poissonlzh_project/generate_run.cc	
This file is part of a tool chain for running the classol/MBody1 example model	248
generateALL.cc	
Main file combining the code for code generation. Part of the code generation section	249
generateALL.h	250
generateCPU.cc	
Functions for generating code that will run the neuron and synapse simulations on the CPU.	
Part of the code generation section	251

generateCPU.h	
Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section	252
generateKernels.cc Contains functions that generate code for CUDA kernels. Part of the code generation section	253
generateKernels.h	
Contains functions that generate code for CUDA kernels. Part of the code generation section	<b>25</b> 4
generateRunner.cc  Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section	255
generateRunner.h  Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section	256
GeNNHelperKrnls.cu	257
GeNNHelperKrnls.h	258
global.cc	258
global.h Global header file containing a few global variables. Part of the code generation section	260
helper.h	262
HHVClamp.cc This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques	264
HHVClampParameters.h	265
hr_time.cc  This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock	265
hr_time.h  This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock	26
isaac.cc Header file and implementation of the ISAAC random number generator	266
lzh_sim_sparse.cc	267
lzh_sparse.cc	267
lzh_sparse_model.cc	268
lzh_sparse_model.h	269
lzh_sparse_sim.h	269
make_input_pats.cc	269
MBody1_project/model/map_classol.cc Implementation of the classol class	270

MBody_delayedSyn_project/model/map_classol.cc Implementation of the classol class	270
MBody_individualID_project/model/map_classol.cc Implementation of the classol class	270
MBody_map_project/model/map_classol.cc	271
MBody_map_robot1_project/model/map_classol.cc	271
MBody_userdef_project/model/map_classol.cc Implementation of the classol class	272
MBody1_project/model/map_classol.h  Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	272
MBody_delayedSyn_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	272
MBody_individualID_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	273
MBody_map_project/model/map_classol.h	273
MBody_map_robot1_project/model/map_classol.h	273
MBody_userdef_project/model/map_classol.h  Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	274
MBody1_project/model/MBody1.cc This file contains the model definition of the mushroom body "MBody1" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol ← _sim)	274
MBody_map_project/model/MBody1.cc	276
MBody_map_robot1_project/model/MBody1.cc	277
MBody_delayedSyn.cc This file contains the model definition of the mushroom body "MBody_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim)	278
MBody_individualID.cc This file contains the model definition of the mushroom body "MBody_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC	280
MBody_userdef.cc This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol_sim)	281

decode_matrix_globalg_bitmask/model.cc	283
decode_matrix_globalg_dense/model.cc	<b>28</b> 4
decode_matrix_globalg_sparse/model.cc	<b>28</b> 4
decode_matrix_individualg_dense/model.cc	285
decode_matrix_individualg_sparse/model.cc	286
extra_global_params_in_sim_code/model.cc	286
extra_global_params_in_sim_code_event_sparse_inv/model.cc	287
extra_global_post_param_in_sim_code/model.cc	287
extra_global_pre_param_in_sim_code/model.cc	288
neuron_support_code_sim/model.cc	289
neuron_support_code_threshold/model.cc	289
post_vars_in_post_learn/model.cc	290
post_vars_in_post_learn_sparse/model.cc	291
post_vars_in_sim_code/model.cc	292
post_vars_in_sim_code_sparse/model.cc	293
post_vars_in_synapse_dynamics/model.cc	293
post_vars_in_synapse_dynamics_sparse/model.cc	294
pre_vars_in_post_learn/model.cc	295
pre_vars_in_post_learn_sparse/model.cc	295
pre_vars_in_sim_code/model.cc	296
pre_vars_in_sim_code_event/model.cc	297
pre_vars_in_sim_code_event_sparse/model.cc	298
pre_vars_in_sim_code_event_sparse_inv/model.cc	299
pre_vars_in_sim_code_sparse/model.cc	299
pre_vars_in_synapse_dynamics/model.cc	300
pre_vars_in_synapse_dynamics_sparse/model.cc	<b>30</b> 1
synapse_support_code_event_sim_code/model.cc	<b>30</b> 1
synapse_support_code_event_threshold/model.cc	302
synapse_support_code_post_learn/model.cc	303
synapse_support_code_sim_code/model.cc	304
synapse_support_code_synapse_dynamics/model.cc	305
decode_matrix_globalg_bitmask/model_new.cc	305

decode_matrix_globalg_dense/model_new.cc	306
decode_matrix_globalg_sparse/model_new.cc	306
decode_matrix_individualg_dense/model_new.cc	307
decode_matrix_individualg_sparse/model_new.cc	307
extra_global_params_in_sim_code/model_new.cc	307
extra_global_params_in_sim_code_event_sparse_inv/model_new.cc	308
extra_global_post_param_in_sim_code/model_new.cc	308
extra_global_pre_param_in_sim_code/model_new.cc	309
neuron_support_code_sim/model_new.cc	309
neuron_support_code_threshold/model_new.cc	309
post_vars_in_post_learn/model_new.cc	310
post_vars_in_post_learn_sparse/model_new.cc	310
post_vars_in_sim_code/model_new.cc	311
post_vars_in_sim_code_sparse/model_new.cc	311
post_vars_in_synapse_dynamics/model_new.cc	312
post_vars_in_synapse_dynamics_sparse/model_new.cc	312
pre_vars_in_post_learn/model_new.cc	313
pre_vars_in_post_learn_sparse/model_new.cc	313
pre_vars_in_sim_code/model_new.cc	313
pre_vars_in_sim_code_event/model_new.cc	314
pre_vars_in_sim_code_event_sparse/model_new.cc	314
pre_vars_in_sim_code_event_sparse_inv/model_new.cc	315
pre_vars_in_sim_code_sparse/model_new.cc	315
pre_vars_in_synapse_dynamics/model_new.cc	316
pre_vars_in_synapse_dynamics_sparse/model_new.cc	316
synapse_support_code_event_sim_code/model_new.cc	317
synapse_support_code_event_threshold/model_new.cc	317
synapse_support_code_post_learn/model_new.cc	317
synapse_support_code_sim_code/model_new.cc	318
synapse_support_code_synapse_dynamics/model_new.cc	318
Model_Schmuker_2014_classifier.cc	319
include/modelSpec.cc	320

src/modelSpec.cc	320
modelSpec.h  Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections	321
neuronGroup.cc	323
neuronGroup.h	323
neuronModels.cc	324
neuronModels.h	325
newModels.h	327
newNeuronModels.cc	328
newNeuronModels.h	329
newPostsynapticModels.cc	330
newPostsynapticModels.h	330
newWeightUpdateModels.cc	331
newWeightUpdateModels.h	331
OneComp.cc	333
OneComp_model.cc	334
OneComp_model.h	334
OneComp_sim.cc	334
OneComp_sim.h	334
parse_options.h	335
Poissonlzh-model.cc	335
Poissonlzh-model.h	336
Poissonlzh.cc	336
Poissonlzh_sim.cc	337
Poissonlzh_sim.h	337
postSynapseModels.cc	338
postSynapseModels.h	338
randomGen.cc Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand()	339
randomGen.h  Header file containing the class definition for a uniform random generator based on the ISAAC random number generator	340

Schmuker2014_classifier.cc	341
Schmuker2014_classifier.h  Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014"	
simulation_neuron_policy_pre_post_var.h	341
simulation_neuron_policy_pre_var.h	342
simulation_synapse_policy_dense.h	342
simulation_synapse_policy_none.h	342
simulation_synapse_policy_sparse.h	342
simulation_test.h	343
simulation_test_decoder_matrix.h	343
simulation_test_vars.h	343
lzh_sparse_project/model/sizes.h	344
MBody1_project/model/sizes.h	344
MBody_delayedSyn_project/model/sizes.h	345
MBody_individualID_project/model/sizes.h	345
MBody_map_project/model/sizes.h	346
MBody_map_robot1_project/model/sizes.h	346
MBody_userdef_project/model/sizes.h	347
OneComp_project/model/sizes.h	348
Poissonlzh_project/model/sizes.h	348
sparseProjection.h	348
sparseUtils.cc	349
sparseUtils.h	350
standardGeneratedSections.cc	351
standardGeneratedSections.h	352
standardSubstitutions.cc	352
standardSubstitutions.h	352
stringUtils.h	353
synapseGroup.cc	354
synapseGroup.h	354
synapseMatrixType.h	354

synapseModels.cc	355
synapseModels.h	356
SynDelay.cc	358
SynDelaySim.cc	358
SynDelaySim.h	358
decode_matrix_globalg_bitmask/test.cc	359
decode_matrix_globalg_dense/test.cc	359
decode_matrix_globalg_sparse/test.cc	360
decode_matrix_individualg_dense/test.cc	360
decode_matrix_individualg_sparse/test.cc	<b>36</b> 1
extra_global_params_in_sim_code/test.cc	362
extra_global_params_in_sim_code_event_sparse_inv/test.cc	362
extra_global_post_param_in_sim_code/test.cc	363
extra_global_pre_param_in_sim_code/test.cc	364
neuron_support_code_sim/test.cc	364
neuron_support_code_threshold/test.cc	365
post_vars_in_post_learn/test.cc	366
post_vars_in_post_learn_sparse/test.cc	366
post_vars_in_sim_code/test.cc	367
post_vars_in_sim_code_sparse/test.cc	368
post_vars_in_synapse_dynamics/test.cc	368
post_vars_in_synapse_dynamics_sparse/test.cc	369
pre_vars_in_post_learn/test.cc	370
pre_vars_in_post_learn_sparse/test.cc	<b>37</b> 1
pre_vars_in_sim_code/test.cc	<b>37</b> 1
pre_vars_in_sim_code_event/test.cc	372
pre_vars_in_sim_code_event_sparse/test.cc	373
pre_vars_in_sim_code_event_sparse_inv/test.cc	373
pre_vars_in_sim_code_sparse/test.cc	374
pre_vars_in_synapse_dynamics/test.cc	375
pre_vars_in_synapse_dynamics_sparse/test.cc	375
synapse_support_code_event_sim_code/test.cc	376

synapse_support_code_event_threshold/test.cc	377
synapse_support_code_post_learn/test.cc	377
synapse_support_code_sim_code/test.cc	378
synapse_support_code_synapse_dynamics/test.cc	379
utils.cc	379
utils.h  This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models	380
VClampGA.cc Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU	382
VClampGA.h Header file containing global variables and macros used in running the HHVClamp/VClampGA	
model	382

# 15 Namespace Documentation

# 15.1 GENN\_FLAGS Namespace Reference

### **Variables**

- unsigned int calcSynapseDynamics = 0
- unsigned int calcSynapses = 1
- unsigned int learnSynapsesPost = 2
- unsigned int calcNeurons = 3

# 15.1.1 Variable Documentation

- 15.1.1.1 unsigned int GENN\_FLAGS::calcNeurons = 3
- 15.1.1.2 unsigned int GENN\_FLAGS::calcSynapseDynamics = 0
- 15.1.1.3 unsigned int GENN\_FLAGS::calcSynapses = 1
- 15.1.1.4 unsigned int GENN\_FLAGS::learnSynapsesPost = 2

# 15.2 GENN\_PREFERENCES Namespace Reference

### **Variables**

• int optimiseBlockSize = 1

Flag for signalling whether or not block size optimisation should be performed.

• int autoChooseDevice = 1

Flag to signal whether the GPU device should be chosen automatically.

• bool optimizeCode = false

Request speed-optimized code, at the expense of floating-point accuracy.

• bool debugCode = false

Request debug data to be embedded in the generated code.

bool showPtxInfo = false

Request that PTX assembler information be displayed for each CUDA kernel during compilation.

double asGoodAsZero = 1e-19

Global variable that is used when detecting close to zero values, for example when setting sparse connectivity from a dense matrix.

- int defaultDevice = 0
- unsigned int neuronBlockSize = 32

default GPU device; used to determine which GPU to use if chooseDevice is 0 (off)

- unsigned int synapseBlockSize = 32
- unsigned int learningBlockSize = 32
- unsigned int synapseDynamicsBlockSize = 32
- unsigned int autoRefractory = 1

Flag for signalling whether spikes are only reported if thresholdCondition changes from false to true (autoRefractory == 1) or spikes are emitted whenever thresholdCondition is true no matter what.%.

std::string userCxxFlagsWIN = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for windows platforms)

std::string userCxxFlagsGNU = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for unix based platforms)

• std::string userNvccFlags = ""

Allows users to set specific nvcc compiler options they may want to use for all GPU code (identical for windows and unix platforms)

### 15.2.1 Variable Documentation

# 15.2.1.1 double GENN\_PREFERENCES::asGoodAsZero = 1e-19

Global variable that is used when detecting close to zero values, for example when setting sparse connectivity from a dense matrix.

15.2.1.2 int GENN PREFERENCES::autoChooseDevice = 1

Flag to signal whether the GPU device should be chosen automatically.

15.2.1.3 unsigned int GENN\_PREFERENCES::autoRefractory = 1

Flag for signalling whether spikes are only reported if thresholdCondition changes from false to true (autoRefractory == 1) or spikes are emitted whenever thresholdCondition is true no matter what.%.

Flag for signalling whether spikes are only reported if thresholdCondition changes from false to true (autoRefractory == 1) or spikes are emitted whenever thresholdCondition is true no matter what.

15.2.1.4 bool GENN\_PREFERENCES::debugCode = false

Request debug data to be embedded in the generated code.

15.2.1.5 int GENN\_PREFERENCES::defaultDevice = 0

15.2.1.6 unsigned int GENN\_PREFERENCES::learningBlockSize = 32

15.2.1.7 unsigned int GENN\_PREFERENCES::neuronBlockSize = 32

default GPU device; used to determine which GPU to use if chooseDevice is 0 (off)

15.2.1.8 int GENN\_PREFERENCES::optimiseBlockSize = 1

Flag for signalling whether or not block size optimisation should be performed.

15.2.1.9 bool GENN\_PREFERENCES::optimizeCode = false

Request speed-optimized code, at the expense of floating-point accuracy.

15.2.1.10 bool GENN\_PREFERENCES::showPtxInfo = false

Request that PTX assembler information be displayed for each CUDA kernel during compilation.

15.2.1.11 unsigned int GENN\_PREFERENCES::synapseBlockSize = 32

15.2.1.12 unsigned int GENN\_PREFERENCES::synapseDynamicsBlockSize = 32

15.2.1.13 std::string GENN\_PREFERENCES::userCxxFlagsGNU = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for unix based platforms)

15.2.1.14 std::string GENN\_PREFERENCES::userCxxFlagsWIN = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for windows platforms)

15.2.1.15 std::string GENN\_PREFERENCES::userNvccFlags = ""

Allows users to set specific nvcc compiler options they may want to use for all GPU code (identical for windows and unix platforms)

### 15.3 NeuronModels Namespace Reference

### Classes

· class Base

Base class for all neuron models.

class Izhikevich

Izhikevich neuron with fixed parameters [1].

· class IzhikevichVariable

Izhikevich neuron with variable parameters [1].

class LegacyWrapper

Wrapper around legacy weight update models stored in nModels array of neuronModel objects.

· class Poisson

Poisson neurons.

class RulkovMap

Rulkov Map neuron.

class SpikeSource

Empty neuron which allows setting spikes from external sources.

class TraubMiles

Hodgkin-Huxley neurons with Traub & Miles algorithm.

class TraubMilesAlt

Hodgkin-Huxley neurons with Traub & Miles algorithm.

· class TraubMilesFast

Hodgkin-Huxley neurons with Traub & Miles algorithm: Original fast implementation, using 25 inner iterations.

class TraubMilesNStep

Hodgkin-Huxley neurons with Traub & Miles algorithm.

### 15.4 NewModels Namespace Reference

#### Classes

· class Base

Base class for all models.

class LegacyWrapper

Wrapper around old-style models stored in global arrays and referenced by index.

- class ValueBase
- class ValueBase< 0 >

### 15.5 PostsynapticModels Namespace Reference

#### Classes

· class Base

Base class for all postsynaptic models.

class DeltaCurr

Simple delta current synapse.

class ExpCond

Exponential decay with synaptic input treated as a conductance value.

class LegacyWrapper

# 15.6 StandardGeneratedSections Namespace Reference

### **Functions**

- void neuronOutputInit (std::ostream &os, const NeuronGroup &ng, const std::string &devPrefix)
- void neuronLocalVarInit (std::ostream &os, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const std::string &devPrefix, const std::string &localID)
- void neuronLocalVarWrite (std::ostream &os, const NeuronGroup &ng, const VarNamelterCtx &nmVars, const std::string &devPrefix, const std::string &localID)
- void neuronSpikeEventTest (std::ostream &os, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &localID, const std::string &ftype)

### 15.6.1 Function Documentation

- 15.6.1.1 void StandardGeneratedSections::neuronLocalVarInit ( std::ostream & os, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const std::string & devPrefix, const std::string & localID )
- 15.6.1.2 void StandardGeneratedSections::neuronLocalVarWrite ( std::ostream & os, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const std::string & devPrefix, const std::string & localID )
- 15.6.1.3 void StandardGeneratedSections::neuronOutputInit ( std::ostream & os, const NeuronGroup & ng, const std::string & devPrefix )
- 15.6.1.4 void StandardGeneratedSections::neuronSpikeEventTest ( std::ostream & os, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & localID, const std::string & ftype )

# 15.7 StandardSubstitutions Namespace Reference

#### **Functions**

- void postSynapseCurrentConverter (std::string &psCode, const SynapseGroup \*sg, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobal
   — ParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void postSynapseDecay (std::string &pdCode, const SynapseGroup \*sg, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParam NameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronSim (std::string &sCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobal← Params, const std::string &ftype)
- void neuronSpikeEventCondition (std::string &eCode, const NeuronGroup &ng, const VarNameIterCtx &nm← Vars, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronReset (std::string &rCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobal← Params, const std::string &ftype)
- void weightUpdateThresholdCondition (std::string &eCode, const SynapseGroup &sg, const DerivedParam
   NameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const string &preIdx, const string &postIdx, const string &devPrefix, const std::string &ftype)
- void weightUpdateSim (std::string &wCode, const SynapseGroup &sg, const VarNameIterCtx &wuVars, const DerivedParamNameIterCtx &wuDerivedParams, const ExtraGlobalParamNameIterCtx &wuExtra
   GlobalParams, const string &postIdx, const string &devPrefix, const std::string &ftype)
- void weightUpdatePostLearn (std::string &code, const SynapseGroup \*sg, const DerivedParamNameIter
   Ctx &wuDerivedParams, const ExtraGlobalParamNameIterCtx &wuExtraGlobalParams, const string &preIdx, const string &devPrefix, const std::string &ftype)

### 15.7.1 Function Documentation

- 15.7.1.1 void StandardSubstitutions::neuronReset ( std::string & rCode, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )
- 15.7.1.2 void StandardSubstitutions::neuronSim ( std::string & sCode, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )
- 15.7.1.3 void StandardSubstitutions::neuronSpikeEventCondition ( std::string & eCode, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )
- 15.7.1.4 void StandardSubstitutions::neuronThresholdCondition ( std::string & thCode, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )
- 15.7.1.5 void StandardSubstitutions::postSynapseCurrentConverter ( std::string & psCode, const SynapseGroup \* sg, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )

### **Parameters**

string to work on	psCode
-------------------	--------

- 15.7.1.6 void StandardSubstitutions::postSynapseDecay ( std::string & pdCode, const SynapseGroup \* sg, const NeuronGroup & ng, const VarNameIterCtx & nmVars, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const std::string & ftype )
- 15.7.1.7 void StandardSubstitutions::weightUpdateDynamics ( std::string & SDcode, const SynapseGroup \* sg, const VarNameIterCtx & wuVars, const DerivedParamNameIterCtx & wuDerivedParams, const string & preldx, const string & postldx, const string & devPrefix, const std::string & ftype )

#### **Parameters**

preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)

15.7.1.8 void StandardSubstitutions::weightUpdatePostLearn ( std::string & code, const SynapseGroup \* sg, const DerivedParamNameIterCtx & wuDerivedParams, const ExtraGlobalParamNameIterCtx & wuExtraGlobalParams, const string & preldx, const string & postldx, const string & devPrefix, const std::string & ftype )

### **Parameters**

preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)

15.7.1.9 void StandardSubstitutions::weightUpdateSim ( std::string & wCode, const SynapseGroup & sg, const VarNameIterCtx & wuVars, const DerivedParamNameIterCtx & wuDerivedParams, const ExtraGlobalParamNameIterCtx & wuExtraGlobalParams, const string & preldx, const string & postldx, const string & devPrefix, const std::string & ftype )

### **Parameters**

preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)

15.7.1.10 void StandardSubstitutions::weightUpdateThresholdCondition ( std::string & eCode, const SynapseGroup & sg, const DerivedParamNameIterCtx & nmDerivedParams, const ExtraGlobalParamNameIterCtx & nmExtraGlobalParams, const string & preldx, const string & postldx, const string & devPrefix, const std::string & ftype )

### **Parameters**

preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)

### 15.8 WeightUpdateModels Namespace Reference

### Classes

class Base

Base class for all weight update models.

class LegacyWrapper

Wrapper around legacy weight update models stored in weightUpdateModels array of weightUpdateModel objects.

• class PiecewiseSTDP

16 Class Documentation 69

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse.

· class StaticGraded

Graded-potential, static synapse.

· class StaticPulse

Pulse-coupled, static synapse.

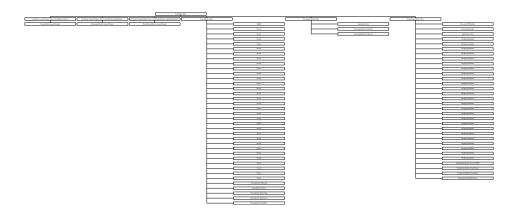
# 16 Class Documentation

### 16.1 NewModels::Base Class Reference

Base class for all models.

#include <newModels.h>

Inheritance diagram for NewModels::Base:



# **Public Types**

- typedef std::function< double(const std::vector< double > &, double)> DerivedParamFunc
- $\bullet \ \ \mathsf{typedef} \ \mathsf{std} : \! \mathsf{vector} \! < \! \mathsf{std} : \! \mathsf{string} \! > \! \mathsf{String} \mathsf{Vec} \\$
- typedef std::vector< std::pair< std::string, std::string >> StringPairVec
- typedef std::vector< std::pair< std::string, DerivedParamFunc >> DerivedParamVec

### **Public Member Functions**

• virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

- virtual DerivedParamVec getDerivedParams () const
- virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

# 16.1.1 Detailed Description

Base class for all models.

- 16.1.2 Member Typedef Documentation
- 16.1.2.1 typedef std::function<double(const std::vector<double> &, double)> NewModels::Base::DerivedParamFunc
- 16.1.2.2 typedef std::vector<std::pair<std::string, DerivedParamFunc> > NewModels::Base::DerivedParamVec
- 16.1.2.3 typedef std::vector<std::pair<std::string, std::string>> NewModels::Base::StringPairVec
- 16.1.2.4 typedef std::vector<std::string> NewModels::Base::StringVec
- 16.1.3 Member Function Documentation
- 16.1.3.1 virtual DerivedParamVec NewModels::Base::getDerivedParams() const [inline], [virtual]

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented in WeightUpdateModels::PiecewiseSTDP, NewModels::LegacyWrapper< Base, neuronModel, nModels >, NewModels::LegacyWrapper< Base, weightUpdateModel, weightUpdateModels >, NewModels $\leftrightarrow$ ::LegacyWrapper< Base, postSynModel, postSynModels >, NeuronModels::RulkovMap, ExpCondUserDef, PiecewiseSTDPUserDef, and PostsynapticModels::ExpCond.

16.1.3.2 virtual StringVec NewModels::Base::getParamNames()const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMiles, NeuronModels::Poisson, WeightUpdateModels::PiecewiseSTDP, NeuronModels::IzhikevichVariable, NeuronModels::Izhikevich, Weight UpdateModels::StaticGraded, NewModels::LegacyWrapper< Base, neuronModel, nModels >, NewModels ::LegacyWrapper< Base, weightUpdateModel, weightUpdateModels >, NewModels::LegacyWrapper< Base, postSynModel, postSynModels >, NeuronModels::RulkovMap, ExpCondUserDef, StaticGradedUserDef, PostsynapticModels::DeltaCurr, PostsynapticModels::ExpCond, WeightUpdateModelSpikeEvent, PiecewiseSTD PuserDef, Mylzhikevich, Mylzhikevich, WeightUpdateModel, Neuron, Neuron, Neuron, Neuron, and Neuron.

16.1.3.3 virtual StringPairVec NewModels::Base::getVars()const [inline],[virtual]

Gets names and types (as strings) of model variables.

Reimplemented in NeuronModels::TraubMiles, NeuronModels::Poisson, WeightUpdateModels::PiecewiseSTDP, NeuronModels::IzhikevichVariable, NeuronModels::Izhikevich, NewModels::LegacyWrapper< Base, neuronModel, nModels >, NewModels::LegacyWrapper< Base, weightUpdateModel, weightUpdateModels >, NewModels::← LegacyWrapper< Base, postSynModel, postSynModels >, WeightUpdateModels::StaticGraded, WeightUpdate← Models::StaticPulse, NeuronModels::RulkovMap, StaticGradedUserDef, StaticPulseUserDef, MyHH, Weight← UpdateModelSpikeEvent, MylzhikevichVariable, PiecewiseSTDPUserDef, WeightUpdateModel, Neuron, Neuron,

The documentation for this class was generated from the following file:

• newModels.h

# 16.2 NeuronModels::Base Class Reference

Base class for all neuron models.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::Base:

NewMod	rls::Base	
NeuronMo	dels::Base	
		МуНН
		Neuron
l		Neuron
		Neuron
ļ		Neuron
		Neuron
		Neuron
l		Neuron
		Neuron
		Neuron
		NeuronModels: Izhikevich
		NeuronModels: Poisson
-		NeuronModels::RulkovMap
		NeuronModels::SpikeSource
Į		NeuronModels::TraubMiles

### **Public Member Functions**

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

• virtual std::string getResetCode () const

Gets code that defines the reset action taken after a spike occurred. This can be empty.

• virtual std::string getSupportCode () const

Gets support code to be made available within the neuron kernel/funcion.

• virtual std::vector< std::pair< std::string, std::string > > getExtraGlobalParams () const

**Additional Inherited Members** 

16.2.1 Detailed Description

Base class for all neuron models.

16.2.2 Member Function Documentation

16.2.2.1 virtual std::vector<std::pair<std::string, std::string> > NeuronModels::Base::getExtraGlobalParams ( ) const [inline], [virtual]

Gets names and types (as strings) of additional per-population parameters for the weight update model.

Reimplemented in NeuronModels::Poisson, MyHH, Neuron, Neuron, and Neuron.

16.2.2.2 virtual std::string NeuronModels::Base::getResetCode() const [inline], [virtual]

Gets code that defines the reset action taken after a spike occurred. This can be empty.

16.2.2.3 virtual std::string NeuronModels::Base::getSimCode( )const [inline], [virtual]

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMilesAlt, NeuronModels::TraubMiles← Fast, NeuronModels::TraubMiles, NeuronModels::Poisson, NeuronModels::Izhikevich, NeuronModels::RulkovMap, MyHH, MylzhikevichVariable, Mylzhikevich, Mylzhikevich, Neuron, Neur

16.2.2.4 virtual std::string NeuronModels::Base::getSupportCode ( ) const [inline], [virtual]

Gets support code to be made available within the neuron kernel/funcion.

This is intended to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "host device "to be available for both GPU and CPU versions.

Reimplemented in Neuron, and Neuron.

16.2.2.5 virtual std::string NeuronModels::Base::getThresholdConditionCode()const [inline], [virtual]

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented in NeuronModels::TraubMiles, NeuronModels::Poisson, NeuronModels::SpikeSource, Neuron ← Models::Izhikevich, NeuronModels::RulkovMap, MyHH, Neuron, Neur

The documentation for this class was generated from the following file:

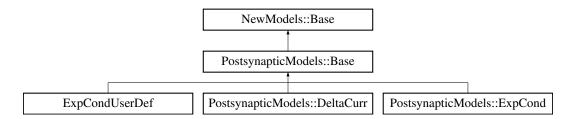
• newNeuronModels.h

### 16.3 PostsynapticModels::Base Class Reference

Base class for all postsynaptic models.

#include <newPostsynapticModels.h>

Inheritance diagram for PostsynapticModels::Base:



**Public Member Functions** 

- virtual std::string getDecayCode () const
- virtual std::string getCurrentConverterCode () const
- virtual std::string getSupportCode () const

**Additional Inherited Members** 

16.3.1 Detailed Description

Base class for all postsynaptic models.

16.3.2 Member Function Documentation

16.3.2.1 virtual std::string PostsynapticModels::Base::getCurrentConverterCode( ) const [inline], [virtual]

Reimplemented in ExpCondUserDef, PostsynapticModels::DeltaCurr, and PostsynapticModels::ExpCond.

16.3.2.2 virtual std::string PostsynapticModels::Base::getDecayCode( )const [inline], [virtual]

Reimplemented in ExpCondUserDef, and PostsynapticModels::ExpCond.

16.3.2.3 virtual std::string PostsynapticModels::Base::getSupportCode( )const [inline], [virtual]

The documentation for this class was generated from the following file:

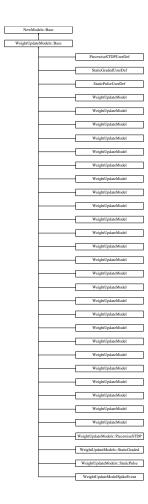
• newPostsynapticModels.h

## 16.4 WeightUpdateModels::Base Class Reference

Base class for all weight update models.

#include <newWeightUpdateModels.h>

 $Inheritance\ diagram\ for\ Weight Update Models:: Base:$ 



## **Public Member Functions**

· virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

• virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

• virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getSimSupportCode () const

Gets support code to be made available within the synapse kernel/function.

virtual std::string getLearnPostSupportCode () const

Gets support code to be made available within learnSynapsesPost kernel/function.

• virtual std::string getSynapseDynamicsSuppportCode () const

Gets support code to be made available within the synapse dynamics kernel/function.

- virtual StringPairVec getExtraGlobalParams () const
- virtual bool isPreSpikeTimeRequired () const

Whether presynaptic spike times are needed or not.

virtual bool isPostSpikeTimeRequired () const

Whether postsynaptic spike times are needed or not.

**Additional Inherited Members** 

16.4.1 Detailed Description

Base class for all weight update models.

16.4.2 Member Function Documentation

```
16.4.2.1 virtual std::string WeightUpdateModels::Base::getEventCode() const [inline], [virtual]
```

Gets code run when events (all the instances where event threshold condition is met) are received.

Reimplemented in WeightUpdateModels::StaticGraded, StaticGradedUserDef, WeightUpdateModelSpikeEvent, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

```
16.4.2.2 virtual std::string WeightUpdateModels::Base::getEventThresholdConditionCode ( ) const [inline], [virtual]
```

Gets codes to test for events.

Reimplemented in WeightUpdateModels::StaticGraded, StaticGradedUserDef, WeightUpdateModelSpikeEvent, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

```
16.4.2.3 virtual StringPairVec WeightUpdateModels::Base::getExtraGlobalParams()const [inline],[virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model.

Reimplemented in WeightUpdateModel.

```
16.4.2.4 virtual std::string WeightUpdateModels::Base::getLearnPostCode() const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented in WeightUpdateModels::PiecewiseSTDP, PiecewiseSTDPUserDef, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

```
16.4.2.5 virtual std::string WeightUpdateModels::Base::getLearnPostSupportCode( )const [inline],[virtual]
```

Gets support code to be made available within learnSynapsesPost kernel/function.

Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

Reimplemented in WeightUpdateModel.

```
16.4.2.6 virtual std::string WeightUpdateModels::Base::getSimCode() const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented in WeightUpdateModels::PiecewiseSTDP, WeightUpdateModels::StaticPulse, StaticPulseUserDef, PiecewiseSTDPUserDef, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

```
16.4.2.7 virtual std::string WeightUpdateModels::Base::getSimSupportCode( ) const [inline], [virtual]
```

Gets support code to be made available within the synapse kernel/function.

This is intended to contain user defined device functions that are used in the weight update code. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be

declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to sim, event threshold and event code

Reimplemented in WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

```
16.4.2.8 virtual std::string WeightUpdateModels::Base::getSynapseDynamicsCode( )const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

Reimplemented in WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, and WeightUpdateModel.

Gets support code to be made available within the synapse dynamics kernel/function.

Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

Reimplemented in WeightUpdateModel.

```
16.4.2.10 virtual bool WeightUpdateModels::Base::isPostSpikeTimeRequired ( ) const [inline], [virtual]
```

Whether postsynaptic spike times are needed or not.

Reimplemented in WeightUpdateModels::PiecewiseSTDP, and PiecewiseSTDPUserDef.

```
16.4.2.11 virtual bool WeightUpdateModels::Base::isPreSpikeTimeRequired( )const [inline],[virtual]
```

Whether presynaptic spike times are needed or not.

Reimplemented in WeightUpdateModels::PiecewiseSTDP, and PiecewiseSTDPUserDef.

The documentation for this class was generated from the following file:

• newWeightUpdateModels.h

# 16.5 classIzh Class Reference

```
#include <Izh_sparse_model.h>
```

# **Public Member Functions**

- · classIzh ()
- ∼classlzh ()
- void init (unsigned int)
- void allocate\_device\_mem\_patterns ()
- void allocate\_device\_mem\_input ()
- void copy\_device\_mem\_input ()
- void read\_sparsesyns\_par (const char \*, struct SparseProjection, FILE \*, FILE \*, FILE \*, scalar \*)

Read sparse connectivity from a file.

void gen\_alltoall\_syns (scalar \*, const char \*, scalar)

Generate random conductivity values for an all to all network.

- void free\_device\_mem ()
- void write\_input\_to\_file (FILE \*)
- void read\_input\_values (FILE \*)
- void create\_input\_values ()
- void run (double, unsigned int)
- void getSpikesFromGPU ()

Method for copying all spikes of the last time step from the GPU.

void getSpikeNumbersFromGPU ()

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

- void output\_state (FILE \*, unsigned int)
- void output spikes (FILE \*, unsigned int)
- void output\_params (FILE \*, FILE \*)
- void sum\_spikes ()
- void setInput (unsigned int)
- void randomizeVar (scalar \*, scalar, const NeuronGroup \*)
- void randomizeVarSq (scalar \*, scalar, const NeuronGroup \*)
- void initializeAllVars (unsigned int)

#### **Public Attributes**

- NNmodel model
- scalar \* input1
- scalar \* input2
- scalar \* d\_input1
- scalar \* d input2
- · unsigned int sumPExc
- unsigned int sumPlnh
- 16.5.1 Constructor & Destructor Documentation

```
16.5.1.2 classlzh::~classlzh( )
```

16.5.1.1 classlzh::classlzh()

16.5.2 Member Function Documentation

```
16.5.2.1 void classlzh::allocate_device_mem_input ( )
```

16.5.2.2 void classlzh::allocate\_device\_mem\_patterns ( )

```
16.5.2.3 void classIzh::copy_device_mem_input ( )
```

16.5.2.4 void classlzh::create\_input\_values ( )

16.5.2.5 void classlzh::free\_device\_mem ( )

16.5.2.6 void classlzh::gen\_alltoall\_syns ( scalar \* g, const char \* synGrpName, scalar gscale )

Generate random conductivity values for an all to all network.

## **Parameters**

g	the resulting synaptic conductances	
synGrpName	synGrpName name of synapse group	
gscale the maximal conductance of generated synaps		

# 16.5.2.7 void classlzh::getSpikeNumbersFromGPU ( )

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.

```
16.5.2.8 void classIzh::getSpikesFromGPU ( )
```

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

```
16.5.2.9 void classIzh::init ( unsigned int which )
16.5.2.10 void classIzh::initializeAllVars ( unsigned int which )
16.5.2.11 void classIzh::output_params ( FILE * f, FILE * f2 )
16.5.2.12 void classIzh::output_spikes ( FILE * f, unsigned int which )
16.5.2.13 void classIzh::output_state ( FILE * f, unsigned int which )
16.5.2.14 void classIzh::randomizeVar ( scalar * Var, scalar strength, const NeuronGroup * neuronGrp )
16.5.2.15 void classIzh::randomizeVarSq ( scalar * Var, scalar strength, const NeuronGroup * neuronGrp )
16.5.2.16 void classIzh::read_input_values ( FILE * )
16.5.2.17 void classIzh::read_sparsesyns_par ( const char * synGrpName, struct SparseProjection C, FILE * f_ind, FILE *
```

Read sparse connectivity from a file.

 $f_{indln}G$ , FILE \*  $f_{g}$ , scalar \* g)

## **Parameters**

synGrpName	index of the synapse population to be worked on	
С	contains teh arrays to be initialized from file	
f_ind	file pointer for the indices of post-synaptic neurons	
f_indInG	file pointer for the summed post-synaptic neurons numbers  File handle for a file containing sparse conductivity values  array to receive the conductance values	
f_g		
g		

```
16.5.2.18 void classlzh::run ( double runtime, unsigned int which )
16.5.2.19 void classlzh::setInput ( unsigned int which )
16.5.2.20 void classlzh::sum_spikes ( )
16.5.2.21 void classlzh::write_input_to_file ( FILE * f )
16.5.3 Member Data Documentation
16.5.3.1 scalar* classlzh::d_input1
16.5.3.2 scalar* classlzh::d_input2
16.5.3.3 scalar* classlzh::input1
16.5.3.4 scalar* classlzh::input2
16.5.3.5 NNmodel classlzh::model
16.5.3.6 unsigned int classlzh::sumPExc
```

## 16.5.3.7 unsigned int classIzh::sumPInh

The documentation for this class was generated from the following files:

- Izh\_sparse\_model.h
- · Izh sparse model.cc

#### 16.6 classol Class Reference

This class coontains the methods for running the MBody1 example model.

```
#include <map_classol.h>
```

#### **Public Member Functions**

- · classol ()
- ∼classol ()

Destructor for olfaction model.

· void init (unsigned int)

Method for initialising variables.

• void allocate\_device\_mem\_patterns ()

Method for allocating memory on the GPU device to hold the input patterns.

• void free\_device\_mem ()

Methods for unallocating the memory for input patterns on the GPU device.

void read\_pnkcsyns (FILE \*)

Method for reading the connectivity between PNs and KCs from a file.

• void read\_sparsesyns\_par (const char \*, struct SparseProjection, scalar \*, FILE \*, FILE \*, FILE \*)

Read sparse connectivity from a file.

void write\_pnkcsyns (FILE \*)

Method for writing the conenctivity between PNs and KCs back into file.

void read\_pnlhisyns (FILE \*)

Method for reading the connectivity between PNs and LHIs from a file.

• void write\_pnlhisyns (FILE \*)

Method for writing the connectivity between PNs and LHIs to a file.

void read\_kcdnsyns (FILE \*)

Method for reading the connectivity between KCs and DNs (detector neurons) from a file.

void write\_kcdnsyns (FILE \*)

Method to write the connectivity between KCs and DNs (detector neurons) to a file.

void read\_input\_patterns (FILE \*)

Method for reading the input patterns from a file.

void generate\_baserates ()

Method for calculating the baseline rates of the Poisson input neurons.

void runGPU (scalar)

Method for simulating the model for a given period of time on the GPU.

void runCPU (scalar)

Method for simulating the model for a given period of time on the CPU.

void output\_state (FILE \*, unsigned int)

Method for copying from device and writing out to file of the entire state of the model.

void getSpikesFromGPU ()

Method for copying all spikes of the last time step from the GPU.

void getSpikeNumbersFromGPU ()

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

```
    void output_spikes (FILE *, unsigned int)

      Method for writing the spikes occurred in the last time step to a file.
• void sum_spikes ()
     Method for summing up spike numbers.

    void get_kcdnsyns ()

      Method for copying the synaptic conductances of the learning synapses between KCs and DNs (detector neurons)
     back to the CPU memory.
· classol ()

    ∼classol ()

· void init (unsigned int)

    void allocate_device_mem_patterns ()

void free_device_mem ()

    void read pnkcsyns (FILE *)

    void read sparsesyns par (const char *, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)

    void write_pnkcsyns (FILE *)

void read_pnlhisyns (FILE *)

    void write pnlhisyns (FILE *)

    void read kcdnsyns (FILE *)

    void write kcdnsyns (FILE *)

    void read input patterns (FILE *)

void generate_baserates ()
· void runGPU (scalar)

    void runCPU (scalar)

· void output state (FILE *, unsigned int)

    void getSpikesFromGPU ()

    void getSpikeNumbersFromGPU ()

    void output_spikes (FILE *, unsigned int)

• void sum_spikes ()
• void get kcdnsyns ()
· classol ()

    ∼classol ()

    void init (unsigned int)

    void allocate_device_mem_patterns ()

• void free_device_mem ()

    void read pnkcsyns (FILE *)

    void read sparsesyns par (const char *, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)

    void write_pnkcsyns (FILE *)

    void read_pnlhisyns (FILE *)

• void write_pnlhisyns (FILE *)

    void read kcdnsyns (FILE *)

    void write_kcdnsyns (FILE *)

    void read_input_patterns (FILE *)

· void generate baserates ()
· void runGPU (scalar)
· void runCPU (scalar)

    void output state (FILE *, unsigned int)

    void getSpikesFromGPU ()

    void getSpikeNumbersFromGPU ()

    void output_spikes (FILE *, unsigned int)

• void sum spikes ()

    void get kcdnsyns ()

· classol ()

    ∼classol ()

· void init (unsigned int)
```

```
    void allocate_device_mem_patterns ()

void free_device_mem ()

    void read pnkcsyns (FILE *)

    void read sparsesyns par (const char *, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)

    void write_pnkcsyns (FILE *)

    void read pnlhisyns (FILE *)

    void write_pnlhisyns (FILE *)

    void read_kcdnsyns (FILE *)

    void write_kcdnsyns (FILE *)

    void read input patterns (FILE *)

· void generate baserates ()

    void runGPU (scalar)

    void runCPU (scalar)

    void output_state (FILE *, unsigned int)

    void getSpikesFromGPU ()

• void getSpikeNumbersFromGPU ()

    void output_spikes (FILE *, unsigned int)

    void outputDN_spikes (FILE *, unsigned int)

      Method for summing up spike numbers.
• void sum spikes ()

    void get kcdnsyns ()

• classol ()

    ∼classol ()

    void init (unsigned int)

• void allocate_device_mem_patterns ()
• void free device mem ()

    void read_pnkcsyns (FILE *)

    void read_sparsesyns_par (const char *, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)

    void write_pnkcsyns (FILE *)

    void read_pnlhisyns (FILE *)

    void write pnlhisyns (FILE *)

void read_kcdnsyns (FILE *)

    void write_kcdnsyns (FILE *)

    void read_input_patterns (FILE *)

· void generate baserates ()
· void runGPU (scalar)

    void runCPU (scalar)

    void output_state (FILE *, unsigned int)

• void getSpikesFromGPU ()

    void getSpikeNumbersFromGPU ()

    void output spikes (FILE *, unsigned int)

    void outputDN spikes (FILE *, unsigned int)

    void sum_spikes ()

    void get_kcdnsyns ()

· classol ()

    ∼classol ()

    void init (unsigned int)

    void allocate device mem patterns ()

void free_device_mem ()

    void read_pnkcsyns (FILE *)

    template < class DATATYPE >

  void read_sparsesyns_par (DATATYPE *, const char *, struct SparseProjection, FILE *, FILE *, FILE *)
      Read sparse connectivity from a file.

    void write_pnkcsyns (FILE *)
```

```
    void read_pnlhisyns (FILE *)

    void write_pnlhisyns (FILE *)

void read_kcdnsyns (FILE *)
• void write_kcdnsyns (FILE *)

    void read_input_patterns (FILE *)

• void generate_baserates ()

    void runGPU (scalar)

    void runCPU (scalar)

    void output_state (FILE *, unsigned int)

    void getSpikesFromGPU ()

· void getSpikeNumbersFromGPU ()

    void output_spikes (FILE *, unsigned int)

• void sum_spikes ()
void get_kcdnsyns ()
• classol ()

    ∼classol ()

· void init (unsigned int)

    void allocate_device_mem_input ()

• void free_device_mem ()

    void read_PNIzh1syns (scalar *, FILE *)

    void read_sparsesyns_par (const char *, struct SparseProjection, FILE *, FILE *, FILE *, double *)

     Read sparse connectivity from a file.
• void generate_baserates ()
· void run (float, unsigned int)

    void output state (FILE *, unsigned int)

    void getSpikesFromGPU ()

· void getSpikeNumbersFromGPU ()

    void output_spikes (FILE *, unsigned int)

• void sum spikes ()
```

#### **Public Attributes**

- NNmodel model
- · unsigned int offset
- uint64\_t \* theRates
- scalar \* p\_pattern
- uint64 t \* pattern
- uint64\_t \* baserates
- uint64\_t \* d\_pattern
- uint64\_t \* d\_baserates
- unsigned int sumPN
- unsigned int sumKC
- unsigned int sumLHI
- · unsigned int sumDN
- · unsigned int size\_g
- · unsigned int sumIzh1

# 16.6.1 Detailed Description

This class coontains the methods for running the MBody1 example model.

This class cpontains the methods for running the MBody\_delayedSyn example model.

```
16.6.2 Constructor & Destructor Documentation
16.6.2.1 classol::classol()
16.6.2.2 classol::∼classol ( )
Destructor for olfaction model.
16.6.2.3 classol::classol()
16.6.2.4 classol:: ∼ classol ( )
16.6.2.5 classol::classol()
16.6.2.6 classol::∼classol ( )
16.6.2.7 classol::classol()
16.6.2.8 classol:: ∼ classol ( )
16.6.2.9 classol::classol()
16.6.2.10 classol::∼classol ( )
16.6.2.11 classol::classol ( )
16.6.2.12 classol::∼classol ( )
16.6.2.13 classol::classol()
16.6.2.14 classol::∼classol ( )
16.6.3 Member Function Documentation
16.6.3.1 void classol::allocate_device_mem_input ( )
16.6.3.2 void classol::allocate_device_mem_patterns ( )
16.6.3.3 void classol::allocate_device_mem_patterns ( )
16.6.3.4 void classol::allocate_device_mem_patterns ( )
16.6.3.5 void classol::allocate_device_mem_patterns ( )
Method for allocating memory on the GPU device to hold the input patterns.
16.6.3.6 void classol::allocate_device_mem_patterns ( )
16.6.3.7 void classol::allocate_device_mem_patterns ( )
16.6.3.8 void classol::free_device_mem ( )
16.6.3.9 void classol::free_device_mem ( )
16.6.3.10 void classol::free_device_mem ( )
```

16.6.3.11 void classol::free\_device\_mem ( )

16.6.3.12 void classol::free\_device\_mem ( )

```
16.6.3.13 void classol::free_device_mem ( )
Methods for unallocating the memory for input patterns on the GPU device.
16.6.3.14 void classol::free_device_mem ( )
16.6.3.15 void classol::generate_baserates ( )
16.6.3.16 void classol::generate_baserates ( )
16.6.3.17 void classol::generate_baserates ( )
16.6.3.18 void classol::generate_baserates ( )
16.6.3.19 void classol::generate_baserates ( )
Method for calculating the baseline rates of the Poisson input neurons.
16.6.3.20 void classol::generate_baserates ( )
16.6.3.21 void classol::generate_baserates ( )
16.6.3.22 void classol::get_kcdnsyns()
Method for copying the synaptic conductances of the learning synapses between KCs and DNs (detector neurons)
back to the CPU memory.
16.6.3.23 void classol::get_kcdnsyns()
16.6.3.24 void classol::get_kcdnsyns()
16.6.3.25 void classol::get_kcdnsyns()
16.6.3.26 void classol::get_kcdnsyns()
16.6.3.27
         void classol::get_kcdnsyns ( )
         void classol::getSpikeNumbersFromGPU ( )
16.6.3.28
16.6.3.29 void classol::getSpikeNumbersFromGPU ( )
16.6.3.30 void classol::getSpikeNumbersFromGPU ( )
Method for copying the number of spikes in all neuron populations that have occurred during the last time step.
This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.
16.6.3.31 void classol::getSpikeNumbersFromGPU ( )
16.6.3.32 void classol::getSpikeNumbersFromGPU ( )
16.6.3.33 void classol::getSpikeNumbersFromGPU ( )
16.6.3.34 void classol::getSpikeNumbersFromGPU ( )
16.6.3.35 void classol::getSpikesFromGPU ( )
16.6.3.36 void classol::getSpikesFromGPU()
```

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

```
16.6.3.37 void classol::getSpikesFromGPU ( )
16.6.3.38 void classol::getSpikesFromGPU ( )
16.6.3.39 void classol::getSpikesFromGPU ( )
16.6.3.40 void classol::getSpikesFromGPU ( )
16.6.3.41 void classol::getSpikesFromGPU ( )
16.6.3.42 void classol::init ( unsigned int )
16.6.3.43 void classol::init ( unsigned int )
16.6.3.44 void classol::init ( unsigned int )
16.6.3.45 void classol::init ( unsigned int )
16.6.3.46 void classol::init ( unsigned int )
```

Method for initialising variables.

#### **Parameters**

	which	Flag defining whether GPU or CPU only version is run
--	-------	--

```
16.6.3.47 void classol::init ( unsigned int )

16.6.3.48 void classol::init ( unsigned int )

16.6.3.49 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.50 void classol::output_spikes ( FILE * f, unsigned int which )
```

Method for writing the spikes occurred in the last time step to a file.

# Parameters

f	File handle for a file to write spike times to
which	Flag determining whether using GPU or CPU only

```
16.6.3.51 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.52 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.53 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.54 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.55 void classol::output_spikes ( FILE * , unsigned int )

16.6.3.56 void classol::output_state ( FILE * , unsigned int )

16.6.3.57 void classol::output_state ( FILE * , unsigned int )

16.6.3.58 void classol::output_state ( FILE * , unsigned int )

16.6.3.59 void classol::output_state ( FILE * , unsigned int )
```

```
16.6.3.60 void classol::output_state ( FILE * f, unsigned int which )
```

Method for copying from device and writing out to file of the entire state of the model.

#### **Parameters**

f	File handle for a file to write the model state to
which	Flag determining whether using GPU or CPU only

```
16.6.3.61 void classol::output_state ( FILE * , unsigned int )

16.6.3.62 void classol::output_state ( FILE * , unsigned int )

16.6.3.63 void classol::outputDN_spikes ( FILE * f, unsigned int which )
```

Method for summing up spike numbers.

## **Parameters**

f	f File handle for a file to write spike times to	
which	Flag determining whether using GPU or CPU only	

```
16.6.3.64 void classol::outputDN_spikes ( FILE * , unsigned int )

16.6.3.65 void classol::read_input_patterns ( FILE * )

16.6.3.66 void classol::read_input_patterns ( FILE * f )
```

Method for reading the input patterns from a file.

#### **Parameters**

f | File handle for a file containing input patterns

```
16.6.3.67 void classol::read_input_patterns ( FILE * )
16.6.3.68 void classol::read_input_patterns ( FILE * )
16.6.3.69 void classol::read_input_patterns ( FILE * )
16.6.3.70 void classol::read_input_patterns ( FILE * )
16.6.3.71 void classol::read_kcdnsyns ( FILE * )
16.6.3.72 void classol::read_kcdnsyns ( FILE * f )
```

Method for reading the connectivity between KCs and DNs (detector neurons) from a file.

## **Parameters**

f | File handle for a file containing KC to DN (detector neuron) conductivity values

```
16.6.3.73 void classol::read_kcdnsyns ( FILE * )

16.6.3.74 void classol::read_kcdnsyns ( FILE * )
```

```
16.6.3.75 void classol::read_kcdnsyns ( FILE * )

16.6.3.76 void classol::read_kcdnsyns ( FILE * )

16.6.3.77 void classol::read_PNlzh1syns ( scalar * gp, FILE * f )

16.6.3.78 void classol::read_pnkcsyns ( FILE * )

16.6.3.79 void classol::read_pnkcsyns ( FILE * f )
```

Method for reading the connectivity between PNs and KCs from a file.

## **Parameters**

f | File handle for a file containing PN to KC conductivity values

```
16.6.3.80 void classol::read_pnkcsyns ( FILE * )

16.6.3.81 void classol::read_pnkcsyns ( FILE * )

16.6.3.82 void classol::read_pnkcsyns ( FILE * )

16.6.3.83 void classol::read_pnkcsyns ( FILE * )

16.6.3.84 void classol::read_pnlhisyns ( FILE * )

16.6.3.85 void classol::read_pnlhisyns ( FILE * )

16.6.3.86 void classol::read_pnlhisyns ( FILE * )

16.6.3.87 void classol::read_pnlhisyns ( FILE * f )
```

Method for reading the connectivity between PNs and LHIs from a file.

#### **Parameters**

f | File handle for a file containing PN to LHI conductivity values

```
16.6.3.88 void classol::read_pnlhisyns ( FILE * )
16.6.3.89 void classol::read_pnlhisyns ( FILE * )
16.6.3.90 void classol::read_sparsesyns_par ( const char * synGrpName, struct SparseProjection C, FILE * f_ind, FILE * f_indlnG, FILE * f_g, double * g )
```

Read sparse connectivity from a file.

## **Parameters**

synGrpName	name of the synapse population to be worked on	
С	contains the arrays to be initialized from file	
f_ind	file pointer for the indices of post-synaptic neurons	
f_indInG	file pointer for the summed post-synaptic neurons numbers	
f_g	File handle for a file containing sparse conductivity values	
g	array to receive the conductance values	

```
16.6.3.91 void classol::read_sparsesyns_par ( const char * , struct SparseProjection , scalar * , FILE * , FILE * , FILE * )
16.6.3.92 void classol::read_sparsesyns_par ( const char * , struct SparseProjection , scalar * , FILE * , FILE * , FILE * )
16.6.3.93 void classol::read_sparsesyns_par ( const char * , struct SparseProjection , scalar * , FILE * , FILE * , FILE * )
16.6.3.94 void classol::read_sparsesyns_par ( const char * sName, struct SparseProjection C, scalar * g, FILE * f_ind, FILE * f_indlnG, FILE * f_g )
```

Read sparse connectivity from a file.

#### **Parameters**

sName	name of the synapse population to be worked on	
С	contains the arrays to be initialized from file	
g	array to receive the conductance values	
f_ind	file pointer for the indices of post-synaptic neurons	
f_indInG	file pointer for the summed post-synaptic neurons numbers	
f_g	File handle for a file containing sparse connectivity values	

```
16.6.3.95 void classol::read_sparsesyns_par ( const char * , struct SparseProjection , scalar * , FILE * , FILE * , FILE * )
```

16.6.3.96 template < class DATATYPE > void classol::read\_sparsesyns\_par ( DATATYPE \* wuvar, const char \* sName, struct SparseProjection C, FILE \* f\_ind, FILE \* f\_indlnG, FILE \* f\_g )

Read sparse connectivity from a file.

## **Parameters**

wuvar	array to receive the conductance values	
sName	name of the synapse population to be worked on	
С	contains the arrays to be initialized from file	
f_ind	file pointer for the indices of post-synaptic neurons	
f_indInG	file pointer for the summed post-synaptic neurons numbers	
f_g	File handle for a file containing sparse conductivity values	

```
16.6.3.97 void classol::run ( float runtime, unsigned int which )
```

16.6.3.98 void classol::runCPU ( scalar runtime )

Method for simulating the model for a given period of time on the CPU.

Method for simulating the model for a given period of time on th CPU.

## **Parameters**

runtime	Duration of time to run the model for

16.6.3.99 void classol::runCPU ( scalar )

```
16.6.3.100 void classol::runCPU ( scalar )
16.6.3.101 void classol::runCPU ( scalar )
16.6.3.102 void classol::runCPU ( scalar )
16.6.3.103 void classol::runCPU ( scalar )
16.6.3.104 void classol::runGPU ( scalar )
16.6.3.105 void classol::runGPU ( scalar )
16.6.3.106 void classol::runGPU ( scalar )
16.6.3.107 void classol::runGPU ( scalar )
```

Method for simulating the model for a given period of time on the GPU.

Method for simulating the model for a given period of time on th GPU.

#### **Parameters**

runtime Duration of time to run	the model for
---------------------------------	---------------

```
16.6.3.108 void classol::runGPU ( scalar )
16.6.3.109 void classol::runGPU ( scalar )
16.6.3.110 void classol::sum_spikes ( )
16.6.3.111 void classol::sum_spikes ( )
Method for summing up spike numbers.
16.6.3.112 void classol::sum_spikes ( )
16.6.3.113 void classol::sum_spikes ( )
16.6.3.114 void classol::sum_spikes ( )
16.6.3.115 void classol::sum_spikes ( )
16.6.3.116 void classol::sum_spikes ( )
16.6.3.117 void classol::write_kcdnsyns ( FILE * )
16.6.3.118 void classol::write_kcdnsyns ( FILE * )
16.6.3.119 void classol::write_kcdnsyns ( FILE * f )
```

Method to write the connectivity between KCs and DNs (detector neurons) to a file.

# **Parameters**

f | File handle for a file to write KC to DN (detectore neuron) conductivity values to

```
16.6.3.120 void classol::write_kcdnsyns ( FILE * )

16.6.3.121 void classol::write_kcdnsyns ( FILE * )
```

```
16.6.3.122 void classol::write_kcdnsyns ( FILE * )

16.6.3.123 void classol::write_pnkcsyns ( FILE * )

16.6.3.124 void classol::write_pnkcsyns ( FILE * )

16.6.3.125 void classol::write_pnkcsyns ( FILE * )

16.6.3.126 void classol::write_pnkcsyns ( FILE * f )
```

Method for writing the conenctivity between PNs and KCs back into file.

## **Parameters**

File handle for a file to write PN to KC conductivity values to

```
16.6.3.127 void classol::write_pnkcsyns ( FILE * )
16.6.3.128 void classol::write_pnkcsyns ( FILE * )
16.6.3.129 void classol::write_pnlhisyns ( FILE * )
16.6.3.130 void classol::write_pnlhisyns ( FILE * )
16.6.3.131 void classol::write_pnlhisyns ( FILE * f )
```

Method for writing the connectivity between PNs and LHIs to a file.

#### **Parameters**

f | File handle for a file to write PN to LHI conductivity values to

```
16.6.3.132 void classol::write_pnlhisyns ( FILE * )
16.6.3.133 void classol::write_pnlhisyns ( FILE * )
16.6.3.134 void classol::write_pnlhisyns ( FILE * )
16.6.4 Member Data Documentation
16.6.4.1 uint64_t * classol::baserates
16.6.4.2 uint64_t * classol::d_baserates
16.6.4.3 uint64_t * classol::d_pattern
16.6.4.4 NNmodel classol::model
16.6.4.5 unsigned int classol::offset
16.6.4.6 scalar * classol::p_pattern
16.6.4.7 uint64_t * classol::pattern
16.6.4.8 unsigned int classol::size_g
16.6.4.9 unsigned int classol::size_g
```

```
16.6.4.10 unsigned int classol::sumIzh1
16.6.4.11 unsigned int classol::sumKC
16.6.4.12 unsigned int classol::sumLHI
16.6.4.13 unsigned int classol::sumPN
16.6.4.14 uint64_t * classol::theRates
```

The documentation for this class was generated from the following files:

- MBody1\_project/model/map\_classol.h
- · PoissonIzh-model.h
- MBody1\_project/model/map\_classol.cc
- · PoissonIzh-model.cc

## 16.7 CodeHelper Class Reference

```
#include <CodeHelper.h>
```

## **Public Member Functions**

- CodeHelper ()
- void setVerbose (bool isVerbose)
- string openBrace (unsigned int level)
- string closeBrace (unsigned int level)
- string endl () const

# 16.7.1 Constructor & Destructor Documentation

```
16.7.1.1 CodeHelper::CodeHelper() [inline]
```

16.7.2 Member Function Documentation

```
16.7.2.1 string CodeHelper::closeBrace (unsigned int level) [inline]
```

16.7.2.2 string CodeHelper::endl() const [inline]

16.7.2.3 string CodeHelper::openBrace (unsigned int level) [inline]

16.7.2.4 void CodeHelper::setVerbose (bool isVerbose) [inline]

The documentation for this class was generated from the following file:

• CodeHelper.h

# 16.8 CStopWatch Class Reference

```
#include <hr_time.h>
```

# **Public Member Functions**

- CStopWatch ()
- void startTimer ()

This method starts the timer.

void stopTimer ()

This method stops the timer.

• double getElapsedTime ()

This method returns the time elapsed between start and stop of the timer in seconds.

```
16.8.1 Constructor & Destructor Documentation
```

```
16.8.1.1 CStopWatch::CStopWatch() [inline]
```

16.8.2 Member Function Documentation

16.8.2.1 double CStopWatch::getElapsedTime ( )

This method returns the time elapsed between start and stop of the timer in seconds.

```
16.8.2.2 void CStopWatch::startTimer ( )
```

This method starts the timer.

```
16.8.2.3 void CStopWatch::stopTimer ( )
```

This method stops the timer.

The documentation for this class was generated from the following files:

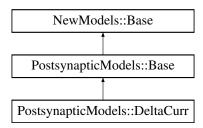
- hr\_time.h
- · hr\_time.cc

# 16.9 PostsynapticModels::DeltaCurr Class Reference

Simple delta current synapse.

```
#include <newPostsynapticModels.h>
```

Inheritance diagram for PostsynapticModels::DeltaCurr:



# **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 0 > VarValues

# **Public Member Functions**

- virtual std::string getCurrentConverterCode () const
- virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

**Static Public Member Functions** 

• static const DeltaCurr \* getInstance ()

#### 16.9.1 Detailed Description

Simple delta current synapse.

Synaptic input provides a direct inject of instantaneous current

- 16.9.2 Member Typedef Documentation
- 16.9.2.1 typedef NewModels::ValueBase< 0 > PostsynapticModels::DeltaCurr::ParamValues
- 16.9.2.2 typedef NewModels::ValueBase< 0 > PostsynapticModels::DeltaCurr::VarValues
- 16.9.3 Member Function Documentation
- 16.9.3.1 virtual std::string PostsynapticModels::DeltaCurr::getCurrentConverterCode( ) const [inline], [virtual]

Reimplemented from PostsynapticModels::Base.

- 16.9.3.2 static const DeltaCurr\* PostsynapticModels::DeltaCurr::getInstance( ) [inline], [static]
- 16.9.3.3 virtual StringVec PostsynapticModels::DeltaCurr::getParamNames( )const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

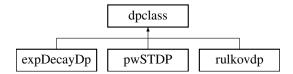
The documentation for this class was generated from the following file:

· newPostsynapticModels.h

# 16.10 dpclass Class Reference

```
#include <dpclass.h>
```

Inheritance diagram for dpclass:



# **Public Member Functions**

• virtual double calculateDerivedParameter (int, vector< double >, double=0.5)

#### 16.10.1 Member Function Documentation

Reimplemented in rulkovdp, pwSTDP, and expDecayDp.

The documentation for this class was generated from the following file:

· dpclass.h

# 16.11 errTupel Struct Reference

**Public Attributes** 

- · unsigned int id
- · double err

## 16.11.1 Member Data Documentation

16.11.1.1 double errTupel::err

16.11.1.2 unsigned int errTupel::id

The documentation for this struct was generated from the following file:

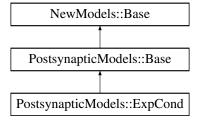
• GA.cc

# 16.12 PostsynapticModels::ExpCond Class Reference

Exponential decay with synaptic input treated as a conductance value.

#include <newPostsynapticModels.h>

Inheritance diagram for PostsynapticModels::ExpCond:



## **Public Types**

- typedef NewModels::ValueBase< 2 > ParamValues
- typedef NewModels::ValueBase< 0 > VarValues

## **Public Member Functions**

- · virtual std::string getDecayCode () const
- virtual std::string getCurrentConverterCode () const
- virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

• virtual DerivedParamVec getDerivedParams () const

# Static Public Member Functions

static const ExpCond \* getInstance ()

## 16.12.1 Detailed Description

Exponential decay with synaptic input treated as a conductance value.

This model has no variables and two parameters:

- tau: Decay time constant
- E : Reversal potential

tau is used by the derived parameter expdecay which returns expf(-dt/tau).

16.12.2 Member Typedef Documentation

```
16.12.2.1 typedef NewModels::ValueBase< 2 > PostsynapticModels::ExpCond::ParamValues
```

16.12.2.2 typedef NewModels::ValueBase< 0 > PostsynapticModels::ExpCond::VarValues

16.12.3 Member Function Documentation

```
16.12.3.1 virtual std::string PostsynapticModels::ExpCond::getCurrentConverterCode ( ) const [inline], [virtual]
```

Reimplemented from PostsynapticModels::Base.

```
16.12.3.2 virtual std::string PostsynapticModels::ExpCond::getDecayCode() const [inline], [virtual]
```

Reimplemented from PostsynapticModels::Base.

```
16.12.3.3 virtual DerivedParamVec PostsynapticModels::ExpCond::getDerivedParams() const [inline], [virtual]
```

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented from NewModels::Base.

```
16.12.3.4 static const ExpCond* PostsynapticModels::ExpCond::getInstance( ) [inline], [static]
```

```
16.12.3.5 virtual StringVec PostsynapticModels::ExpCond::getParamNames() const [inline], [virtual]
```

Gets names of of (independent) model parameters.

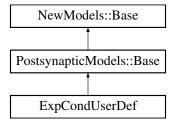
Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• newPostsynapticModels.h

## 16.13 ExpCondUserDef Class Reference

Inheritance diagram for ExpCondUserDef:



## **Public Types**

- typedef NewModels::ValueBase< 2 > ParamValues
- typedef NewModels::ValueBase< 0 > VarValues

#### **Public Member Functions**

- virtual std::string getDecayCode () const
- virtual std::string getCurrentConverterCode () const
- virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual DerivedParamVec getDerivedParams () const

#### **Static Public Member Functions**

static const ExpCondUserDef \* getInstance ()

```
16.13.1 Member Typedef Documentation
```

```
16.13.1.1 typedef NewModels::ValueBase< 2 > ExpCondUserDef::ParamValues
```

16.13.1.2 typedef NewModels::ValueBase< 0 > ExpCondUserDef::VarValues

16.13.2 Member Function Documentation

```
16.13.2.1 virtual std::string ExpCondUserDef::getCurrentConverterCode( ) const [inline], [virtual]
```

Reimplemented from PostsynapticModels::Base.

```
16.13.2.2 virtual std::string ExpCondUserDef::getDecayCode( ) const [inline], [virtual]
```

Reimplemented from PostsynapticModels::Base.

```
16.13.2.3 virtual DerivedParamVec ExpCondUserDef::getDerivedParams() const [inline], [virtual]
```

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented from NewModels::Base.

```
16.13.2.4 static const ExpCondUserDef* ExpCondUserDef::getInstance() [inline], [static]

16.13.2.5 virtual StringVec ExpCondUserDef::getParamNames() const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

MBody\_userdef.cc

# 16.14 expDecayDp Class Reference

Class defining the dependent parameter for exponential decay.

```
#include <postSynapseModels.h>
```

Inheritance diagram for expDecayDp:



## **Public Member Functions**

• double calculateDerivedParameter (int index, vector< double > pars, double dt=1.0)

# 16.14.1 Detailed Description

Class defining the dependent parameter for exponential decay.

## 16.14.2 Member Function Documentation

```
16.14.2.1 double expDecayDp::calculateDerivedParameter ( int index, vector < double > pars, double dt = 1.0 )
[inline], [virtual]
```

Reimplemented from dpclass.

The documentation for this class was generated from the following file:

• postSynapseModels.h

# 16.15 inputSpec Struct Reference

```
#include <helper.h>
```

# **Public Attributes**

- double t
- double baseV
- int N
- vector< double > st
- vector< double > V

## 16.15.1 Member Data Documentation

16.15.1.1 double inputSpec::baseV

16.15.1.2 int inputSpec::N

16.15.1.3 vector<double> inputSpec::st

16.15.1.4 double inputSpec::t

16.15.1.5 vector<double> inputSpec::V

The documentation for this struct was generated from the following file:

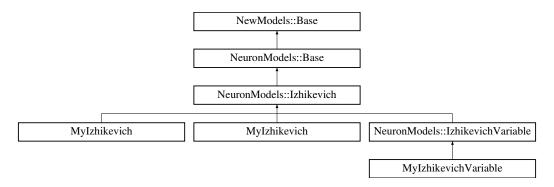
helper.h

## 16.16 NeuronModels::Izhikevich Class Reference

Izhikevich neuron with fixed parameters [1].

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::Izhikevich:



## **Public Types**

- typedef NewModels::ValueBase< 4 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

#### **Public Member Functions**

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

• virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

# **Static Public Member Functions**

static const NeuronModels::Izhikevich \* getInstance ()

## 16.16.1 Detailed Description

Izhikevich neuron with fixed parameters [1].

It is usually described as

$$\begin{array}{lcl} \frac{dV}{dt} & = & 0.04V^2 + 5V + 140 - U + I, \\ \frac{dU}{dt} & = & a(bV - U), \end{array}$$

I is an external input current and the voltage V is reset to parameter c and U incremented by parameter d, whenever V >= 30 mV. This is paired with a particular integration procedure of two 0.5 ms Euler time steps for the V equation followed by one 1 ms time step of the U equation. Because of its popularity we provide this model in this form here event though due to the details of the usual implementation it is strictly speaking inconsistent with the displayed equations.

Variables are:

- ∨ Membrane potential
- U Membrane recovery variable

## Parameters are:

- a time scale of U
- b sensitivity of U
- c after-spike reset value of V
- · d after-spike reset value of U
- 16.16.2 Member Typedef Documentation

```
16.16.2.1 typedef NewModels::ValueBase< 4 > NeuronModels::Izhikevich::ParamValues
```

16.16.2.2 typedef NewModels::ValueBase< 2 > NeuronModels::Izhikevich::VarValues

16.16.3 Member Function Documentation

16.16.3.1 static const NeuronModels::Izhikevich\* NeuronModels::Izhikevich::getInstance( ) [inline], [static]

16.16.3.2 virtual String Vec Neuron Models:: Izhikevich::getParam Names ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

Reimplemented in NeuronModels::IzhikevichVariable, Mylzhikevich, and Mylzhikevich.

```
16.16.3.3 virtual std::string NeuronModels::lzhikevich::getSimCode( )const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

Reimplemented in MylzhikevichVariable, Mylzhikevich, and Mylzhikevich.

```
16.16.3.4 virtual std::string NeuronModels::lzhikevich::getThresholdConditionCode( )const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.16.3.5 virtual StringPairVec NeuronModels::lzhikevich::getVars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

Reimplemented in NeuronModels::IzhikevichVariable, and MylzhikevichVariable.

The documentation for this class was generated from the following file:

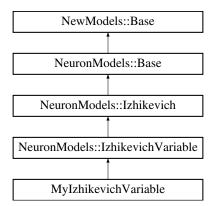
• newNeuronModels.h

## 16.17 NeuronModels::IzhikevichVariable Class Reference

Izhikevich neuron with variable parameters [1].

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::IzhikevichVariable:



# **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 6 > VarValues

#### **Public Member Functions**

- virtual StringVec getParamNames () const
   Gets names of of (independent) model parameters.
- virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

## Static Public Member Functions

• static const NeuronModels::IzhikevichVariable \* getInstance ()

# 16.17.1 Detailed Description

Izhikevich neuron with variable parameters [1].

This is the same model as Izhikevich but parameters are defined as "variables" in order to allow users to provide individual values for each individual neuron instead of fixed values for all neurons across the population.

Accordingly, the model has the Variables:

- $\,\, \lor \,$  Membrane potential
- U Membrane recovery variable
- · a time scale of U
- b sensitivity of U
- c after-spike reset value of V
- d after-spike reset value of U

and no parameters.

- 16.17.2 Member Typedef Documentation
- 16.17.2.1 typedef NewModels::ValueBase< 0 > NeuronModels::IzhikevichVariable::ParamValues
- 16.17.2.2 typedef NewModels::ValueBase< 6 > NeuronModels::IzhikevichVariable::VarValues
- 16.17.3 Member Function Documentation
- 16.17.3.1 static const NeuronModels::IzhikevichVariable\* NeuronModels::IzhikevichVariable::getInstance( ) [inline], [static]
- 16.17.3.2 virtual String Vec Neuron Models::Izhikevich Variable::get Param Names ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

16.17.3.3 virtual StringPairVec NeuronModels::lzhikevichVariable::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NeuronModels::Izhikevich.

Reimplemented in MylzhikevichVariable.

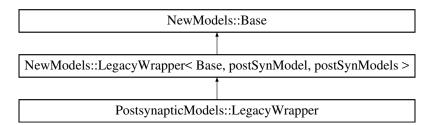
The documentation for this class was generated from the following file:

· newNeuronModels.h

# 16.18 PostsynapticModels::LegacyWrapper Class Reference

#include <newPostsynapticModels.h>

Inheritance diagram for PostsynapticModels::LegacyWrapper:



## **Public Member Functions**

- LegacyWrapper (unsigned int legacyTypeIndex)
- · virtual std::string getDecayCode () const
- virtual std::string getCurrentConverterCode () const
- virtual std::string getSupportCode () const

#### **Additional Inherited Members**

- 16.18.1 Constructor & Destructor Documentation
- 16.18.1.1 PostsynapticModels::LegacyWrapper::LegacyWrapper ( unsigned int legacyTypeIndex ) [inline]
- 16.18.2 Member Function Documentation

```
16.18.2.1 std::string PostsynapticModels::LegacyWrapper::getCurrentConverterCode( )const [virtual]

16.18.2.2 std::string PostsynapticModels::LegacyWrapper::getDecayCode( )const [virtual]
```

16.18.2.3 std::string PostsynapticModels::LegacyWrapper::getSupportCode( )const [virtual]

The documentation for this class was generated from the following files:

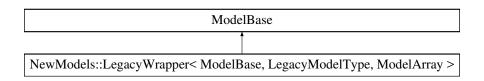
- newPostsynapticModels.h
- newPostsynapticModels.cc

# 16.19 NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray > Class Template Reference

Wrapper around old-style models stored in global arrays and referenced by index.

```
#include <newModels.h>
```

Inheritance diagram for NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray >:



#### **Public Member Functions**

- LegacyWrapper (unsigned int legacyTypeIndex)
- virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

- virtual DerivedParamVec getDerivedParams () const
- virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

# **Static Protected Member Functions**

• static StringPairVec zipStringVectors (const StringVec &a, const StringVec &b)

### **Protected Attributes**

const unsigned int m\_LegacyTypeIndex
 Index into the array of legacy models.

## 16.19.1 Detailed Description

 $template < typename\ ModelBase,\ typename\ LegacyModelType,\ const\ std::vector < LegacyModelType > \&\ ModelArray > \\ class\ NewModels::LegacyWrapper < ModelBase,\ LegacyModelType,\ ModelArray > \\$ 

Wrapper around old-style models stored in global arrays and referenced by index.

- 16.19.2 Constructor & Destructor Documentation
- 16.19.2.1 template < typename ModelBase, typename LegacyModelType, const std::vector < LegacyModelType > & ModelArray > NewModels::LegacyWrapper < ModelBase, LegacyModelType, ModelArray >::LegacyWrapper ( unsigned int legacyTypeIndex ) [inline]
- 16.19.3 Member Function Documentation
- 16.19.3.1 template < typename ModelBase, typename LegacyModelType, const std::vector < LegacyModelType > & ModelArray > virtual DerivedParamVec NewModels::LegacyWrapper < ModelBase, LegacyModelType, ModelArray > ::getDerivedParams() const [inline], [virtual]

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

16.19.3.2 template < typename ModelBase, typename LegacyModelType, const std::vector < LegacyModelType > & ModelArray > virtual StringVec NewModels::LegacyWrapper < ModelBase, LegacyModelType, ModelArray > ::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

16.19.3.3 template<typename ModelBase, typename LegacyModelType, const std::vector< LegacyModelType > & ModelArray> virtual StringPairVec NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray >::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

- 16.19.3.4 template<typename ModelBase, typename LegacyModelType, const std::vector< LegacyModelType > & ModelArray> static StringPairVec NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray >::zipStringVectors ( const StringVec & a, const StringVec & b ) [inline], [static], [protected]
- 16.19.4 Member Data Documentation
- 16.19.4.1 template < typename ModelBase, typename LegacyModelType, const std::vector < LegacyModelType > & ModelArray > const unsigned int NewModels::LegacyWrapper < ModelBase, LegacyModelType, ModelArray > ::m\_LegacyTypeIndex [protected]

Index into the array of legacy models.

The documentation for this class was generated from the following file:

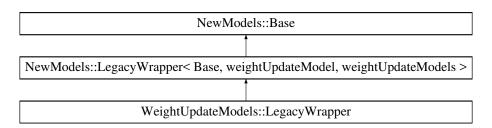
newModels.h

# 16.20 WeightUpdateModels::LegacyWrapper Class Reference

Wrapper around legacy weight update models stored in weightUpdateModels array of weightUpdateModel objects.

#include <newWeightUpdateModels.h>

Inheritance diagram for WeightUpdateModels::LegacyWrapper:



#### **Public Member Functions**

- LegacyWrapper (unsigned int legacyTypeIndex)
- virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getSimSupportCode () const

Gets support code to be made available within the synapse kernel/function.

virtual std::string getLearnPostSupportCode () const

Gets support code to be made available within learnSynapsesPost kernel/function.

virtual std::string getSynapseDynamicsSuppportCode () const

Gets support code to be made available within the synapse dynamics kernel/function.

- virtual NewModels::Base::StringPairVec getExtraGlobalParams () const
- virtual bool isPreSpikeTimeRequired () const

Whether presynaptic spike times are needed or not.

virtual bool isPostSpikeTimeRequired () const

Whether postsynaptic spike times are needed or not.

## **Additional Inherited Members**

#### 16.20.1 Detailed Description

Wrapper around legacy weight update models stored in weightUpdateModels array of weightUpdateModel objects.

```
16.20.2 Constructor & Destructor Documentation
```

```
16.20.2.1 WeightUpdateModels::LegacyWrapper::LegacyWrapper (unsigned int legacyTypeIndex) [inline]
```

16.20.3 Member Function Documentation

```
16.20.3.1 std::string WeightUpdateModels::LegacyWrapper::getEventCode( )const [virtual]
```

Gets code run when events (all the instances where event threshold condition is met) are received.

```
16.20.3.2 std::string WeightUpdateModels::LegacyWrapper::getEventThresholdConditionCode( )const [virtual]
```

Gets codes to test for events.

```
16.20.3.3 NewModels::Base::StringPairVec WeightUpdateModels::LegacyWrapper::getExtraGlobalParams ( ) const [virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model.

```
16.20.3.4 std::string WeightUpdateModels::LegacyWrapper::getLearnPostCode( )const [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

16.20.3.5 std::string WeightUpdateModels::LegacyWrapper::getLearnPostSupportCode( ) const [virtual]

Gets support code to be made available within learnSynapsesPost kernel/function.

Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

16.20.3.6 std::string WeightUpdateModels::LegacyWrapper::getSimCode()const [virtual]

Gets simulation code run when 'true' spikes are received.

16.20.3.7 std::string WeightUpdateModels::LegacyWrapper::getSimSupportCode() const [virtual]

Gets support code to be made available within the synapse kernel/function.

This is intended to contain user defined device functions that are used in the weight update code. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to sim, event threshold and event code

16.20.3.8 std::string WeightUpdateModels::LegacyWrapper::getSynapseDynamicsCode()const [virtual]

Gets code for synapse dynamics which are independent of spike detection.

16.20.3.9 std::string WeightUpdateModels::LegacyWrapper::getSynapseDynamicsSuppportCode()const [virtual]

Gets support code to be made available within the synapse dynamics kernel/function.

Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

16.20.3.10 bool WeightUpdateModels::LegacyWrapper::isPostSpikeTimeRequired( )const [virtual]

Whether postsynaptic spike times are needed or not.

16.20.3.11 bool WeightUpdateModels::LegacyWrapper::isPreSpikeTimeRequired ( ) const [virtual]

Whether presynaptic spike times are needed or not.

The documentation for this class was generated from the following files:

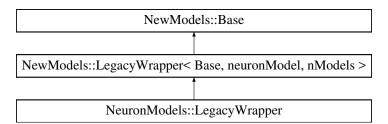
- newWeightUpdateModels.h
- newWeightUpdateModels.cc

# 16.21 NeuronModels::LegacyWrapper Class Reference

Wrapper around legacy weight update models stored in nModels array of neuronModel objects.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::LegacyWrapper:



#### **Public Member Functions**

- LegacyWrapper (unsigned int legacyTypeIndex)
- virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual std::string getResetCode () const

Gets code that defines the reset action taken after a spike occurred. This can be empty.

virtual std::string getSupportCode () const

Gets support code to be made available within the neuron kernel/funcion.

- virtual NewModels::Base::StringPairVec getExtraGlobalParams () const
- · virtual bool isPoisson () const

#### **Additional Inherited Members**

#### 16.21.1 Detailed Description

Wrapper around legacy weight update models stored in nModels array of neuronModel objects.

```
16.21.2 Constructor & Destructor Documentation
```

```
16.21.2.1 NeuronModels::LegacyWrapper::LegacyWrapper ( unsigned int legacyTypeIndex ) [inline]
```

16.21.3 Member Function Documentation

```
16.21.3.1 NewModels::Base::StringPairVec NeuronModels::LegacyWrapper::getExtraGlobalParams ( ) const [virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model.

```
16.21.3.2 std::string NeuronModels::LegacyWrapper::getResetCode()const [virtual]
```

Gets code that defines the reset action taken after a spike occurred. This can be empty.

```
16.21.3.3 std::string NeuronModels::LegacyWrapper::getSimCode( )const [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

```
16.21.3.4 std::string NeuronModels::LegacyWrapper::getSupportCode( )const [virtual]
```

Gets support code to be made available within the neuron kernel/funcion.

This is intended to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

```
16.21.3.5 std::string NeuronModels::LegacyWrapper::getThresholdConditionCode( )const [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

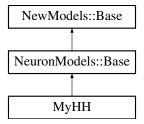
16.21.3.6 bool NeuronModels::LegacyWrapper::isPoisson()const [virtual]

The documentation for this class was generated from the following files:

- newNeuronModels.h
- newNeuronModels.cc

# 16.22 MyHH Class Reference

Inheritance diagram for MyHH:



#### **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 11 > VarValues

## **Public Member Functions**

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual StringPairVec getExtraGlobalParams () const

#### **Static Public Member Functions**

• static const MyHH \* getInstance ()

16.22.1 Member Typedef Documentation

16.22.1.1 typedef NewModels::ValueBase< 0 > MyHH::ParamValues

16.22.1.2 typedef NewModels::ValueBase< 11 > MyHH::VarValues

16.22.2 Member Function Documentation

16.22.2.1 virtual StringPairVec MyHH::getExtraGlobalParams() const [inline], [virtual]

Gets names and types (as strings) of additional per-population parameters for the weight update model.

Reimplemented from NeuronModels::Base.

```
16.22.2.2 static const MyHH* MyHH::getInstance() [inline], [static]
```

```
16.22.2.3 virtual std::string MyHH::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.22.2.4 virtual std::string MyHH::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.22.2.5 virtual StringPairVec MyHH::getVars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

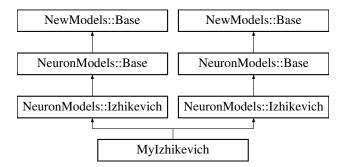
Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• HHVClamp.cc

# 16.23 Mylzhikevich Class Reference

Inheritance diagram for Mylzhikevich:



#### **Public Types**

- typedef NewModels::ValueBase< 5 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 5 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

# **Public Member Functions**

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

#### Static Public Member Functions

- static const Mylzhikevich \* getInstance ()
- static const Mylzhikevich \* getInstance ()

## 16.23.1 Member Typedef Documentation

```
16.23.1.1 typedef NewModels::ValueBase < 5 > Mylzhikevich::ParamValues
```

```
16.23.1.2 typedef NewModels::ValueBase < 5 > Mylzhikevich::ParamValues
```

```
16.23.1.3 typedef NewModels::ValueBase< 2 > Mylzhikevich::VarValues
```

16.23.1.4 typedef NewModels::ValueBase< 2 > Mylzhikevich::VarValues

16.23.2 Member Function Documentation

```
16.23.2.1 static const Mylzhikevich* Mylzhikevich::getInstance( ) [inline], [static]
```

```
16.23.2.2 static const Mylzhikevich* Mylzhikevich::getInstance( ) [inline], [static]
```

16.23.2.3 virtual StringVec Mylzhikevich::getParamNames() const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

```
16.23.2.4 virtual StringVec Mylzhikevich::getParamNames() const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

```
16.23.2.5 virtual std::string Mylzhikevich::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Izhikevich.

```
16.23.2.6 virtual std::string Mylzhikevich::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

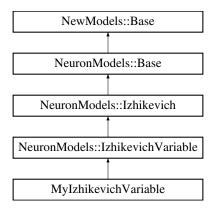
Reimplemented from NeuronModels::Izhikevich.

The documentation for this class was generated from the following files:

- OneComp.cc
- · SynDelay.cc

# 16.24 MylzhikevichVariable Class Reference

Inheritance diagram for MylzhikevichVariable:



## **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 7 > VarValues

### **Public Member Functions**

- virtual std::string getSimCode () const
  - Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

# **Static Public Member Functions**

• static const MylzhikevichVariable \* getInstance ()

## 16.24.1 Member Typedef Documentation

- 16.24.1.1 typedef NewModels::ValueBase< 0 > MylzhikevichVariable::ParamValues
- 16.24.1.2 typedef NewModels::ValueBase< 7 > MylzhikevichVariable::VarValues
- 16.24.2 Member Function Documentation
- **16.24.2.1** static const MylzhikevichVariable\* MylzhikevichVariable::getInstance( ) [inline], [static]
- 16.24.2.2 virtual std::string MylzhikevichVariable::getSimCode ( ) const [inline], [virtual]

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Izhikevich.

16.24.2.3 virtual StringPairVec MylzhikevichVariable::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NeuronModels::IzhikevichVariable.

The documentation for this class was generated from the following file:

· lzh\_sparse.cc

### 16.25 NamelterCtx < Container > Struct Template Reference

#include <standardSubstitutions.h>

## **Public Types**

typedef PairKeyConstIter< typename Container::const\_iterator > NameIter

#### **Public Member Functions**

• NamelterCtx (const Container &c)

#### **Public Attributes**

- · const Container container
- · const Namelter nameBegin
- · const Namelter nameEnd
- 16.25.1 Member Typedef Documentation
- 16.25.2 Constructor & Destructor Documentation
- 16.25.2.1 template < typename Container > NamelterCtx < Container >::NamelterCtx ( const Container & c ) [inline]
- 16.25.3 Member Data Documentation
- 16.25.3.1 template < typename Container > const Container NameIterCtx < Container > :::container
- 16.25.3.2 template < typename Container > const NameIter NameIterCtx < Container > ::nameBegin
- $16.25.3.3 \quad template < typename \ Container > const \ Name \ Iter \ Carte < Container > :: name \ End$

The documentation for this struct was generated from the following file:

· standardSubstitutions.h

## 16.26 Neuron Class Reference

Inheritance diagram for Neuron:

NeuronModels::Base NeuronModels::Base

## **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

### **Public Member Functions**

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

- virtual StringPairVec getExtraGlobalParams () const
- virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

- virtual StringPairVec getExtraGlobalParams () const
- virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

- virtual StringPairVec getExtraGlobalParams () const
- virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSupportCode () const

Gets support code to be made available within the neuron kernel/funcion.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSupportCode () const

Gets support code to be made available within the neuron kernel/funcion.

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

· virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

#### **Static Public Member Functions**

- static const Neuron \* getInstance ()
   static const Neuron \* getInstance ()
- talle series i tearen i getinetanes (
- static const Neuron \* getInstance ()
- static const Neuron \* getInstance ()

# 16.26.1 Member Typedef Documentation

16.26.1.1	typedef NewModels::ValueBase< 0 > Neuron::ParamValues
16.26.1.2	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.3	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.4	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values$
16.26.1.5	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values$
16.26.1.6	$typedef \ NewModels:: ValueBase < 1 > Neuron:: Param Values$
16.26.1.7	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.8	$typedef \ NewModels:: ValueBase < 1 > Neuron:: Param Values$
16.26.1.9	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.10	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.11	$typedef\ NewModels:: ValueBase < 0 > Neuron:: Param Values$
16.26.1.12	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.13	$typedef\ NewModels:: ValueBase < 0 > Neuron:: Param Values$
16.26.1.14	$typedef\ NewModels:: ValueBase < 1 > Neuron:: ParamValues$
16.26.1.15	$typedef\ NewModels:: ValueBase < 0 > Neuron:: Param Values$
16.26.1.16	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.17	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.18	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.19	$typedef \ New Models :: Value Base < 1 > Neuron :: Param Values \\$
16.26.1.20	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.21	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.22	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.23	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.24	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.25	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.26	$typedef \ New Models :: Value Base < 0 > Neuron :: Param Values \\$
16.26.1.27	$typedef\ NewModels:: ValueBase < 1 > Neuron:: ParamValues$
16.26.1.28	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.29	$typedef \ NewModels:: ValueBase < 0 > Neuron:: Param Values \\$
16.26.1.30	typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.31	$\label{typedef} \mbox{ typedef NewModels::ValueBase} < 0 > \mbox{Neuron::ParamValues}$
16.26.1.32	typedef NewModels::ValueBase < 2 > Neuron::VarValues
16.26.1.33	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.34	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.35	typedef NewModels::ValueBase< 1 > Neuron::VarValues
16.26.1.36	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.37	typedef NewModels::ValueBase< 1 > Neuron::VarValues
16.26.1.38	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.39	typedef NewModels::ValueBase < 2 > Neuron::VarValues
16.26.1.40	typedef NewModels::ValueBase< 1 > Neuron::VarValues
16.26.1.41	typedef NewModels::ValueBase < 2 > Neuron::VarValues
16.26.1.42	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.43	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.44	typedef NewModels::ValueBase< 1 > Neuron::VarValues
16.26.1.45	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.46	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.47	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.48	typedef NewModels::ValueBase < 2 > Neuron::VarValues
16.26.1.49	$typedef \ NewModels:: ValueBase < 1 > Neuron:: Var Values \\$
16.26.1.50	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.51	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.52	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.53	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.54	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.55	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.56	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.57	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.58	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.59	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.60	$\label{typedef} \mbox{typedef NewModels::ValueBase} < 2 > \mbox{Neuron::VarValues}$
16.26.1.61	typedef NewModels::ValueBase< 2 > Neuron::VarValues

```
16.26.1.62 typedef NewModels::ValueBase < 2 > Neuron::VarValues
```

16.26.2 Member Function Documentation

```
16.26.2.1 virtual StringPairVec Neuron::getExtraGlobalParams ( ) const [inline], [virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model. Reimplemented from NeuronModels::Base.

```
16.26.2.2 virtual StringPairVec Neuron::getExtraGlobalParams ( ) const [inline], [virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model. Reimplemented from NeuronModels::Base.

```
16.26.2.3 virtual StringPairVec Neuron::getExtraGlobalParams() const [inline], [virtual]
```

Gets names and types (as strings) of additional per-population parameters for the weight update model. Reimplemented from NeuronModels::Base.

```
16.26.2.4 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.5 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.6 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.7
        static const Neuron* Neuron::getInstance( ) [inline], [static]
16.26.2.8
        static const Neuron* Neuron::getInstance( ) [inline], [static]
16.26.2.9
        static const Neuron* Neuron::getInstance( ) [inline],[static]
16.26.2.10 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.11 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.12 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.13 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.14 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.15 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.16 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.17 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.18 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.19 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.20 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.21 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.22 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.23 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.24 static const Neuron* Neuron::getInstance() [inline], [static]
```

```
16.26.2.25
         static const Neuron* Neuron::getInstance( ) [inline], [static]
16.26.2.26 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.27 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.28 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.29 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.30 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.31 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.32 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.33 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.34 static const Neuron* Neuron::getInstance() [inline], [static]
16.26.2.35 virtual String Vec Neuron::getParamNames() const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.26.2.36 virtual String Vec Neuron::getParamNames() const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.26.2.37 virtual StringVec Neuron::getParamNames() const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.26.2.38 virtual String Vec Neuron::getParamNames() const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.26.2.39 virtual String Vec Neuron::getParamNames ( ) const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
```

16.26.2.40 virtual std::string Neuron::getSimCode( )const [inline],[virtual]

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.41 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.42 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.43 virtual std::string Neuron::getSimCode( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.44 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.45 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.46 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.47 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.48 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.49 virtual std::string Neuron::getSimCode( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.50 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.51 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.52 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.53 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.54 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.55 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.56 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.57 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.58 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.59 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.60 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.61 virtual std::string Neuron::getSimCode( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.62 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.63 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.64 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN",

i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.65 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.66 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.67 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.68 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.69 virtual std::string Neuron::getSimCode() const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.70 virtual std::string Neuron::getSimCode ( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.26.2.71 virtual std::string Neuron::getSupportCode( ) const [inline], [virtual]
```

Gets support code to be made available within the neuron kernel/funcion.

This is intended to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "host device "to be available for both GPU and CPU versions.

Reimplemented from NeuronModels::Base.

```
16.26.2.72 virtual std::string Neuron::getSupportCode( ) const [inline], [virtual]
```

Gets support code to be made available within the neuron kernel/funcion.

This is intended to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

Reimplemented from NeuronModels::Base.

```
16.26.2.73 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.74 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.75 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.76 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.77 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.78 virtual std::string Neuron::getThresholdConditionCode() const [inline],[virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.79 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.80 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20"). Reimplemented from NeuronModels::Base. 16.26.2.81 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual] Gets code which defines the condition for a true spike in the described neuron model. This evaluates to a bool (e.g. "V > 20"). Reimplemented from NeuronModels::Base. 16.26.2.82 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual] Gets code which defines the condition for a true spike in the described neuron model. This evaluates to a bool (e.g. "V > 20"). Reimplemented from NeuronModels::Base. 16.26.2.83 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual] Gets code which defines the condition for a true spike in the described neuron model. This evaluates to a bool (e.g. "V > 20"). Reimplemented from NeuronModels::Base. 16.26.2.84 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual] Gets code which defines the condition for a true spike in the described neuron model. This evaluates to a bool (e.g. "V > 20"). Reimplemented from NeuronModels::Base.

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

16.26.2.86 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]

16.26.2.85 virtual std::string Neuron::getThresholdConditionCode()const [inline], [virtual]

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

16.26.2.87 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

16.26.2.88 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.26.2.89 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
Gets code which defines the condition for a true spike in the described neuron model.
This evaluates to a bool (e.g. "V > 20").
Reimplemented from NeuronModels::Base.
16.26.2.90 virtual std::string Neuron::getThresholdConditionCode() const [inline], [virtual]
Gets code which defines the condition for a true spike in the described neuron model.
This evaluates to a bool (e.g. "V > 20").
Reimplemented from NeuronModels::Base.
16.26.2.91 virtual std::string Neuron::getThresholdConditionCode ( ) const [inline], [virtual]
Gets code which defines the condition for a true spike in the described neuron model.
This evaluates to a bool (e.g. "V > 20").
Reimplemented from NeuronModels::Base.
16.26.2.92 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.93 virtual String Pair Vec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.94 virtual String Pair Vec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.95 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.96 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.97 virtual String Pair Vec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.98 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
```

```
16.26.2.99 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.100 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.101 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.102 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.103 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.104 virtual String Pair Vec Neuron::getVars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.105 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.106 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.107 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.108 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.109 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
```

```
16.26.2.110 virtual String Pair Vec Neuron::getVars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.111 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.112 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.113 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.114 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.115 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.116 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.117 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.118 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.119 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.26.2.120 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
```

```
16.26.2.121 virtual StringPairVec Neuron::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.26.2.122 virtual String Pair Vec Neuron::get Vars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

· decode\_matrix\_globalg\_bitmask/model\_new.cc

# 16.27 NeuronGroup Class Reference

```
#include <neuronGroup.h>
```

#### **Public Member Functions**

- NeuronGroup (const std::string &name, int numNeurons, const NeuronModels::Base \*neuronModel, const std::vector< double > &params, const std::vector< double > &initVals)
- void checkNumDelaySlots (unsigned int requiredDelay)
  - < Checks delay slots currently provided by the neuron group against a required delay and extends if required
- void updateVarQueues (const std::string &code)
- void setSpikeTimeRequired (bool req)
- void setTrueSpikeRequired (bool req)
- void setSpikeEventRequired (bool req)
- void setSpikeZeroCopyEnabled (bool enabled)
- void setSpikeEventZeroCopyEnabled (bool enabled)
- void setSpikeTimeZeroCopyEnabled (bool enabled)
- void setVarZeroCopyEnabled (const std::string &varName, bool enabled)
- void setClusterIndex (int hostID, int deviceID)
- void addSpkEventCondition (const std::string &code, const std::string &supportCodeNamespace)
- void addlnSyn (SynapseGroup \*synapseGroup)
- void addOutSyn (SynapseGroup \*synapseGroup)
- · void initDerivedParams (double dt)
- void calcSizes (unsigned int blockSize, unsigned int &idStart, unsigned int &paddedIDStart)
- const std::string & getName () const
- unsigned int getNumNeurons () const
- const std::pair< unsigned int, unsigned int > & getPaddedIDRange () const
- const std::pair< unsigned int, unsigned int > & getIDRange () const
- const NeuronModels::Base \* getNeuronModel () const
- const std::vector< double > & getParams () const
- const std::vector< double > & getDerivedParams () const
- const std::vector< double > & getInitVals () const
- const std::vector< SynapseGroup \* > & getInSyn () const
- const std::vector< SynapseGroup \* > & getOutSyn () const
- bool isSpikeTimeRequired () const
- bool isTrueSpikeRequired () const
- bool isSpikeEventRequired () const
- bool isQueueRequired () const
- bool isVarQueueRequired (const std::string &var) const
- bool isVarQueueRequired () const

- const std::set< std::pair< std::string, std::string > > & getSpikeEventCondition () const
- unsigned int getNumDelaySlots () const
- · bool isDelayRequired () const
- bool isSpikeZeroCopyEnabled () const
- bool isSpikeEventZeroCopyEnabled () const
- bool isSpikeTimeZeroCopyEnabled () const
- bool isZeroCopyEnabled () const
- bool isVarZeroCopyEnabled (const std::string &var) const
- bool isParamRequiredBySpikeEventCondition (const std::string &pnamefull) const
- void addExtraGlobalParams (std::map< std::string, std::string > &kernelParameters) const
- std::string getQueueOffset (const std::string &devPrefix) const

#### 16.27.1 Constructor & Destructor Documentation

- 16.27.1.1 NeuronGroup::NeuronGroup ( const std::string & name, int numNeurons, const NeuronModels::Base \* neuronModel, const std::vector < double > & params, const std::vector < double > & initVals ) [inline]
- 16.27.2 Member Function Documentation
- 16.27.2.1 void NeuronGroup::addExtraGlobalParams ( std::map < std::string, std::string > & kernelParameters ) const
- 16.27.2.2 void NeuronGroup::addlnSyn(SynapseGroup \* synapseGroup) [inline]
- 16.27.2.3 void NeuronGroup::addOutSyn ( SynapseGroup \* synapseGroup ) [inline]
- 16.27.2.4 void NeuronGroup::addSpkEventCondition ( const std::string & code, const std::string & supportCodeNamespace )
- 16.27.2.5 void NeuronGroup::calcSizes ( unsigned int blockSize, unsigned int & idStart, unsigned int & paddedIDStart )
- 16.27.2.6 void NeuronGroup::checkNumDelaySlots (unsigned int requiredDelay)
- < Checks delay slots currently provided by the neuron group against a required delay and extends if required
- 16.27.2.7 const std::vector<double>& NeuronGroup::getDerivedParams ( ) const [inline]
- 16.27.2.8 const std::pair<unsigned int, unsigned int>& NeuronGroup::getIDRange( ) const [inline]
- 16.27.2.9 const std::vector<double>& NeuronGroup::getInitVals( ) const [inline]
- 16.27.2.10 const std::vector<SynapseGroup\*>& NeuronGroup::getInSyn( ) const [inline]
- 16.27.2.11 const std::string& NeuronGroup::getName() const [inline]
- 16.27.2.12 const NeuronModels::Base\* NeuronGroup::getNeuronModel( ) const [inline]
- 16.27.2.13 unsigned int NeuronGroup::getNumDelaySlots ( ) const [inline]
- 16.27.2.14 unsigned int NeuronGroup::getNumNeurons ( ) const [inline]
- 16.27.2.15 const std::vector<SynapseGroup\*>& NeuronGroup::getOutSyn( )const [inline]
- 16.27.2.16 const std::pair < unsigned int, unsigned int> & NeuronGroup::getPaddedIDRange ( ) const [inline]
- 16.27.2.17 const std::vector<double>& NeuronGroup::getParams() const [inline]
- 16.27.2.18 std::string NeuronGroup::getQueueOffset ( const std::string & devPrefix ) const
- 16.27.2.19 const std::set<std::string, std::string> > & NeuronGroup::getSpikeEventCondition ( ) const [inline]

```
16.27.2.20
          void NeuronGroup::initDerivedParams ( double dt )
16.27.2.21
          bool NeuronGroup::isDelayRequired( ) const [inline]
16.27.2.22 bool NeuronGroup::isParamRequiredBySpikeEventCondition (const std::string & pnamefull) const
16.27.2.23 bool NeuronGroup::isQueueRequired() const [inline]
16.27.2.24 bool NeuronGroup::isSpikeEventRequired() const [inline]
16.27.2.25 bool NeuronGroup::isSpikeEventZeroCopyEnabled( )const [inline]
16.27.2.26 bool NeuronGroup::isSpikeTimeRequired( ) const [inline]
16.27.2.27 bool NeuronGroup::isSpikeTimeZeroCopyEnabled( )const [inline]
16.27.2.28 bool NeuronGroup::isSpikeZeroCopyEnabled( ) const [inline]
16.27.2.29 bool NeuronGroup::isTrueSpikeRequired() const [inline]
16.27.2.30 bool NeuronGroup::isVarQueueRequired ( const std::string & var ) const
16.27.2.31 bool NeuronGroup::isVarQueueRequired ( ) const [inline]
16.27.2.32 bool NeuronGroup::isVarZeroCopyEnabled ( const std::string & var ) const
16.27.2.33 bool NeuronGroup::isZeroCopyEnabled ( ) const
16.27.2.34 void NeuronGroup::setClusterIndex (int hostID, int deviceID) [inline]
16.27.2.35 void NeuronGroup::setSpikeEventRequired ( bool req ) [inline]
```

Function to enable the use of zero-copied memory for spikes: May improve IO performance at the expense of kernel performance

```
16.27.2.36 void NeuronGroup::setSpikeEventZeroCopyEnabled (bool enabled) [inline]
```

Function to enable the use of zero-copied memory for spike times: May improve IO performance at the expense of kernel performance

```
16.27.2.37 void NeuronGroup::setSpikeTimeRequired ( bool req ) [inline]

16.27.2.38 void NeuronGroup::setSpikeTimeZeroCopyEnabled ( bool enabled ) [inline]
```

Function to enable the use zero-copied memory for a particular state variable: May improve IO performance at the expense of kernel performance

```
16.27.2.39 void NeuronGroup::setSpikeZeroCopyEnabled (bool enabled) [inline]
```

Function to enable the use of zero-copied memory for spike-like events: May improve IO performance at the expense of kernel performance

```
16.27.2.40 void NeuronGroup::setTrueSpikeRequired ( bool req ) [inline]

16.27.2.41 void NeuronGroup::setVarZeroCopyEnabled ( const std::string & varName, bool enabled )

16.27.2.42 void NeuronGroup::updateVarQueues ( const std::string & code )
```

The documentation for this class was generated from the following files:

- · neuronGroup.h
- neuronGroup.cc

## 16.28 neuronModel Class Reference

class for specifying a neuron model.

#include <neuronModels.h>

#### **Public Member Functions**

• neuronModel ()

Constructor for neuronModel objects.

∼neuronModel ()

Destructor for neuronModel objects.

#### **Public Attributes**

string simCode

Code that defines the execution of one timestep of integration of the neuron model The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

· string thresholdConditionCode

Code evaluating to a bool (e.g. "V > 20") that defines the condition for a true spike in the described neuron model.

string resetCode

Code that defines the reset action taken after a spike occurred. This can be empty.

· string supportCode

Support code is made available within the neuron kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

vector< string > varNames

Names of the variables in the neuron model.

vector< string > tmpVarNames

never used

vector< string > varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

vector< string > tmpVarTypes

never used

vector< string > pNames

Names of (independent) parameters of the model.

vector< string > dpNames

Names of dependent parameters of the model. The dependent parameters are functions of independent parameters that enter into the neuron model. To avoid unecessary computational overhead, these parameters are calculated at compile time and inserted as explicit values into the generated code. See method NNmodel::initDerivedNeuronPara for how this is done.

vector< string > extraGlobalNeuronKernelParameters

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

vector< string > extraGlobalNeuronKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

dpclass \* dps

Derived parameters.

16.28.1 Detailed Description

class for specifying a neuron model.

16.28.2 Constructor & Destructor Documentation

16.28.2.1 neuronModel::neuronModel()

Constructor for neuronModel objects.

16.28.2.2 neuronModel:: ∼neuronModel ( )

Destructor for neuronModel objects.

16.28.3 Member Data Documentation

16.28.3.1 vector<string> neuronModel::dpNames

Names of dependent parameters of the model. The dependent parameters are functions of independent parameters that enter into the neuron model. To avoid unecessary computational overhead, these parameters are calculated at compile time and inserted as explicit values into the generated code. See method NNmodel::initDerivedNeuronPara for how this is done.

16.28.3.2 dpclass\* neuronModel::dps

Derived parameters.

16.28.3.3 vector<string> neuronModel::extraGlobalNeuronKernelParameters

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

16.28.3.4 vector<string> neuronModel::extraGlobalNeuronKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

16.28.3.5 vector<string> neuronModel::pNames

Names of (independent) parameters of the model.

16.28.3.6 string neuronModel::resetCode

Code that defines the reset action taken after a spike occurred. This can be empty.

16.28.3.7 string neuronModel::simCode

Code that defines the execution of one timestep of integration of the neuron model The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

16.28.3.8 string neuronModel::supportCode

Support code is made available within the neuron kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

```
16.28.3.9 string neuronModel::thresholdConditionCode
```

Code evaluating to a bool (e.g. "V > 20") that defines the condition for a true spike in the described neuron model.

```
16.28.3.10 vector<string> neuronModel::tmpVarNames
```

never used

```
16.28.3.11 vector < string > neuronModel::tmpVarTypes
```

never used

```
16.28.3.12 vector < string > neuronModel::varNames
```

Names of the variables in the neuron model.

```
16.28.3.13 vector < string > neuronModel::varTypes
```

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this class was generated from the following files:

- · neuronModels.h
- neuronModels.cc

# 16.29 neuronpop Class Reference

```
#include <OneComp_model.h>
```

# **Public Member Functions**

- neuronpop ()
- ∼neuronpop ()
- void init (unsigned int)
- · void run (float, unsigned int)
- void getSpikesFromGPU ()

Method for copying all spikes of the last time step from the GPU.

void getSpikeNumbersFromGPU ()

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

- void output\_state (FILE \*, unsigned int)
- void output\_spikes (FILE \*, unsigned int)
- void sum\_spikes ()

## **Public Attributes**

- NNmodel model
- · unsigned int sumIzh1

#### 16.29.1 Constructor & Destructor Documentation

```
16.29.1.1 neuronpop::neuronpop()
```

16.29.1.2 neuronpop::∼neuronpop ( )

## 16.29.2 Member Function Documentation

```
16.29.2.1 void neuronpop::getSpikeNumbersFromGPU ( )
```

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.

```
16.29.2.2 void neuronpop::getSpikesFromGPU ( )
```

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

```
16.29.2.3 void neuronpop::init ( unsigned int which )

16.29.2.4 void neuronpop::output_spikes ( FILE * f, unsigned int which )

16.29.2.5 void neuronpop::output_state ( FILE * f, unsigned int which )

16.29.2.6 void neuronpop::run ( float runtime, unsigned int which )

16.29.2.7 void neuronpop::sum_spikes ( )

16.29.3 Member Data Documentation
```

16.29.3.2 unsigned int neuronpop::sumlzh1

16.29.3.1 NNmodel neuronpop::model

The documentation for this class was generated from the following files:

- OneComp\_model.h
- OneComp\_model.cc

## 16.30 NNmodel Class Reference

```
#include <modelSpec.h>
```

# **Public Member Functions**

- NNmodel ()
- ∼NNmodel ()
- void setName (const std::string &)

Method to set the neuronal network model name.

void setPrecision (FloatType)

Set numerical precision for floating point.

void setDT (double)

Set the integration step size of the model.

void setTiming (bool)

Set whether timers and timing commands are to be included.

void setSeed (unsigned int)

Set the random seed (disables automatic seeding if argument not 0).

• void setRNType (const std::string &type)

Sets the underlying type for random number generation (default: uint64\_t)

void setGPUDevice (int)

Sets the underlying type for random number generation (default: uint64 t)

string scalarExpr (const double) const

Get the string literal that should be used to represent a value in the model's floating-point type.

void setPopulationSums ()

Set the accumulated sums of lowest multiple of kernel block size >= group sizes for all simulated groups.

void finalize ()

Declare that the model specification is finalised in modelDefinition().

bool zeroCopyInUse () const

Are any variables in any populations in this model using zero-copy memory?

const std::string & getName () const

Gets the name of the neuronal network model.

· const std::string & getPrecision () const

Gets the floating point numerical precision.

• unsigned int getResetKernel () const

Which kernel should contain the reset logic? Specified in terms of GENN\_FLAGS.

double getDT () const

Gets the model integration step size.

· unsigned int getSeed () const

Get the random seed.

const std::string & getRNType () const

Gets the underlying type for random number generation (default: uint64\_t)

bool isFinalized () const

Is the model specification finalized.

• bool isTimingEnabled () const

Are timers and timing commands enabled.

const map< string, NeuronGroup > & getNeuronGroups () const

Get std::map containing all named NeuronGroup objects in model.

const map< string, string > & getNeuronKernelParameters () const

Gets std::map containing names and types of each parameter that should be passed through to the neuron kernel.

· unsigned int getNeuronGridSize () const

Gets the size of the neuron kernel thread grid.

unsigned int getNumNeurons () const

How many neurons make up the entire model.

const NeuronGroup \* findNeuronGroup (const std::string &name) const

Find a neuron group by name.

NeuronGroup \* findNeuronGroup (const std::string &name)

Find a neuron group by name.

NeuronGroup \* addNeuronPopulation (const string &, unsigned int, unsigned int, const double \*, const double \*)

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

NeuronGroup \* addNeuronPopulation (const string &, unsigned int, unsigned int, const vector< double > &, const vector< double > &)

 $Method \ for \ adding \ a \ neuron \ population \ to \ a \ neuronal \ network \ model, \ using \ C++ \ string \ for \ the \ name \ of \ the \ population.$ 

 $\bullet \ \ \text{template}{<} \text{typename NeuronModel} >$ 

NeuronGroup \* addNeuronPopulation (const string &name, unsigned int size, const typename Neuron← Model::ParamValues &paramValues, const typename NeuronModel::VarValues &varValues)

Adds a new neuron group to the model.

void setNeuronClusterIndex (const string &neuronGroup, int hostID, int deviceID)

Function for setting which host and which device a neuron group will be simulated on.

void activateDirectInput (const string &, unsigned int type)

This function defines the type of the explicit input to the neuron model. Current options are common constant input to all neurons, input from a file and input defines as a rule.

void setConstInp (const string &, double)

This function has been deprecated in GeNN 2.2.

• const map< string, SynapseGroup > & getSynapseGroups () const

Get std::map containing all named SynapseGroup objects in model.

- const map< string, std::pair< unsigned int, unsigned int > > & getSynapsePostLearnGroups () const
- const map< string, std::pair< unsigned int, unsigned int > > & getSynapseDynamicsGroups () const
- const map< string, string > & getSynapseKernelParameters () const

Gets std::map containing names and types of each parameter that should be passed through to the synapse kernel.

const map< string, string > & getSimLearnPostKernelParameters () const

Gets std::map containing names and types of each parameter that should be passed through to the postsynaptic learning kernel.

• const map< string, string > & getSynapseDynamicsKernelParameters () const

Gets std::map containing names and types of each parameter that should be passed through to the synapse dynamics kernel

unsigned int getSynapseKernelGridSize () const

Gets the size of the synapse kernel thread grid.

unsigned int getSynapsePostLearnGridSize () const

Gets the size of the post-synaptic learning kernel thread grid.

unsigned int getSynapseDynamicsGridSize () const

Gets the size of the synapse dynamics kernel thread grid.

const SynapseGroup \* findSynapseGroup (const std::string &name) const

Find a synapse group by name.

SynapseGroup \* findSynapseGroup (const std::string &name)

Find a synapse group by name.

bool isSynapseGroupDynamicsRequired (const std::string &name) const

Does named synapse group have synapse dynamics.

bool isSynapseGroupPostLearningRequired (const std::string &name) const

Does named synapse group have post-synaptic learning.

SynapseGroup \* addSynapsePopulation (const string &name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, const string &src, const string &trg, const double \*p)

This function has been depreciated as of GeNN 2.2.

• SynapseGroup \* addSynapsePopulation (const string &, unsigned int, SynapseConnType, SynapseGType, unsigned int, unsigned int, const string &, const string &, const double \*, const double \*, const double \*)

Overloaded version without initial variables for synapses.

SynapseGroup \* addSynapsePopulation (const string &, unsigned int, SynapseConnType, SynapseGType, unsigned int, unsigned int, const string &, const double \*, const double \*, const double \*, const double \*,

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

SynapseGroup \* addSynapsePopulation (const string &, unsigned int, SynapseConnType, SynapseGType, unsigned int, unsigned int, const string &, const vector< double > &, const vector< double > &, const vector< double > &)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

template<typename WeightUpdateModel , typename PostsynapticModel >

SynapseGroup \* addSynapsePopulation (const string &name, SynapseMatrixType mtype, unsigned int delaySteps, const string &src, const string &trg, const typename WeightUpdateModel::ParamValues &weightParamValues, const typename WeightUpdateModel::VarValues &weightVarValues, const typename PostsynapticModel::ParamValues &postsynapticParamValues, const typename PostsynapticModel::Var Values &postsynapticVarValues)

Adds a new synapse group to the model.

void setSynapseG (const string &, double)

This function has been depreciated as of GeNN 2.2.

void setMaxConn (const string &, unsigned int)

This function defines the maximum number of connections for a neuron in the population.

void setSpanTypeToPre (const string &)

Method for switching the execution order of synapses to pre-to-post.

void setSynapseClusterIndex (const string &synapseGroup, int hostID, int deviceID)

Function for setting which host and which device a synapse group will be simulated on.

#### 16.30.1 Constructor & Destructor Documentation

```
16.30.1.1 NNmodel::NNmodel ( )
```

16.30.1.2 NNmodel::~NNmodel()

16.30.2 Member Function Documentation

16.30.2.1 void NNmodel::activateDirectInput ( const string & , unsigned int type )

This function defines the type of the explicit input to the neuron model. Current options are common constant input to all neurons, input from a file and input defines as a rule.

### **Parameters**

type	Type of input: 1 if common input, 2 if custom input from file, 3 if custom input as a rule
------	--

16.30.2.2 NeuronGroup \* NNmodel::addNeuronPopulation ( const string & name, unsigned int nNo, unsigned int type, const double \* p, const double \* ini )

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This function adds a neuron population to a neuronal network models, assigning the name, the number of neurons in the group, the neuron type, parameters and initial values, the latter two defined as double \*

## **Parameters**

name	The name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
р	Parameters of this neuron type
ini	Initial values for variables of this neuron type

16.30.2.3 NeuronGroup \* NNmodel::addNeuronPopulation ( const string & name, unsigned int nNo, unsigned int type, const vector < double > & p, const vector < double > & ini )

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

This function adds a neuron population to a neuronal network models, assigning the name, the number of neurons in the group, the neuron type, parameters and initial values. The latter two defined as STL vectors of double.

# **Parameters**

name	The name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
р	Parameters of this neuron type
ini	Initial values for variables of this neuron type

16.30.2.4 template < typename NeuronModel > NeuronGroup\* NNmodel::addNeuronPopulation ( const string & name, unsigned int size, const typename NeuronModel::ParamValues & paramValues, const typename NeuronModel::VarValues & varValues ) [inline]

Adds a new neuron group to the model.

# **Template Parameters**

NeuronModel	type of neuron model (derived from NeuronModels::Base).
-------------	---

#### **Parameters**

name	string containing unique name of neuron population.
size	integer specifying how many neurons are in the population.
paramValues	parameters for model wrapped in NeuronModel::ParamValues object.
varValues	initial state variable values for model wrapped in NeuronModel::VarValues object.

#### Returns

pointer to newly created NeuronGroup

16.30.2.5 SynapseGroup \* NNmodel::addSynapsePopulation ( const string & name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, const string & src, const string & trg, const double \* p )

This function has been depreciated as of GeNN 2.2.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This deprecated function is provided for compatibility with the previous release of GeNN. Default values are provide for new parameters, it is strongly recommended these be selected explicity via the new version othe function

# **Parameters**

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.30.2.6 SynapseGroup \* NNmodel::addSynapsePopulation ( const string & name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, unsigned int delaySteps, unsigned int postsyn, const string & src, const string & trg, const double \* p, const double \* PSVini, const double \* ps )

Overloaded version without initial variables for synapses.

Overloaded old version (deprecated)

# **Parameters**

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)

#### **Parameters**

conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.30.2.7 SynapseGroup \* NNmodel::addSynapsePopulation ( const string & name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, unsigned int delaySteps, unsigned int postsyn, const string & src, const string & trg, const double \* synini, const double \* p, const double \* PSVini, const double \* ps )

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population

This function adds a synapse population to a neuronal network model, assigning the name, the synapse type, the connectivity type, the type of conductance specification, the source and destination neuron populations, and the synaptic parameters.

#### **Parameters**

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
synini	A C-type array of doubles that contains the initial values for synapse variables (common to all synapses of the population) which will be used for the defined synapses.
p	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.30.2.8 SynapseGroup \* NNmodel::addSynapsePopulation ( const string & name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, unsigned int delaySteps, unsigned int postsyn, const string & src, const string & trg, const vector< double > & synini, const vector< double > & p, const vector< double > & ps )

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

This function adds a synapse population to a neuronal network model, assigning the name, the synapse type, the connectivity type, the type of conductance specification, the source and destination neuron populations, and the synaptic parameters.

#### **Parameters**

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
synini	A C-type array of doubles that contains the initial values for synapse variables (common to all synapses of the population) which will be used for the defined synapses.
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.30.2.9 template<typename WeightUpdateModel, typename PostsynapticModel > SynapseGroup\*

NNmodel::addSynapsePopulation ( const string & name, SynapseMatrixType mtype, unsigned int delaySteps, const string & src, const string & trg, const typename WeightUpdateModel::ParamValues & weightParamValues, const typename WeightUpdateModel::VarValues & weightVarValues, const typename PostsynapticModel::ParamValues & postsynapticParamValues, const typename PostsynapticModel::VarValues & postsynapticVarValues) [inline]

Adds a new synapse group to the model.

# **Template Parameters**

WeightUpdateModel	type of weight update model (derived from WeightUpdateModels::Base).
PostsynapticModel	type of postsynaptic model (derived from PostsynapticModels::Base).

## **Parameters**

name	string containing unique name of neuron population.
mtype	how the synaptic matrix associated with this synapse population should be represented.
delayStep	integer specifying number of timesteps delay this synaptic connection should incur (or NO_DELAY for none)
src	string specifying name of presynaptic (source) population
trg	string specifying name of postsynaptic (target) population
weightParamValues	parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.
weightVarValues	initial state variable values for weight update model wrapped in WeightUpdateModel::VarValues object.
postsynapticParamValues	parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.

#### **Parameters**

postsynapticVarValues	initial state variable values for postsynaptic model wrapped in
	PostsynapticModel::VarValues object.

#### Returns

pointer to newly created SynapseGroup

```
16.30.2.10 void NNmodel::finalize ( )
```

Declare that the model specification is finalised in modelDefinition().

16.30.2.11 const NeuronGroup \* NNmodel::findNeuronGroup ( const std::string & name ) const

Find a neuron group by name.

16.30.2.12 NeuronGroup \* NNmodel::findNeuronGroup ( const std::string & name )

Find a neuron group by name.

16.30.2.13 const SynapseGroup \* NNmodel::findSynapseGroup ( const std::string & name ) const

Find a synapse group by name.

16.30.2.14 SynapseGroup \* NNmodel::findSynapseGroup ( const std::string & name )

Find a synapse group by name.

16.30.2.15 double NNmodel::getDT( ) const [inline]

Gets the model integration step size.

16.30.2.16 const std::string& NNmodel::getName() const [inline]

Gets the name of the neuronal network model.

16.30.2.17 unsigned int NNmodel::getNeuronGridSize ( ) const

Gets the size of the neuron kernel thread grid.

This is calculated by adding together the number of threads required by each neuron population, padded to be a multiple of GPU's thread block size.

16.30.2.18 const map < string, NeuronGroup > & NNmodel::getNeuronGroups ( ) const [inline]

Get std::map containing all named NeuronGroup objects in model.

16.30.2.19 const map<string>& NNmodel::getNeuronKernelParameters() const [inline]

Gets std::map containing names and types of each parameter that should be passed through to the neuron kernel.

16.30.2.20 unsigned int NNmodel::getNumNeurons ( ) const

How many neurons make up the entire model.

16.30.2.21 const std::string& NNmodel::getPrecision( ) const [inline]

Gets the floating point numerical precision.

```
16.30.2.22 unsigned int NNmodel::getResetKernel( ) const [inline]
```

Which kernel should contain the reset logic? Specified in terms of GENN FLAGS.

```
16.30.2.23 const std::string& NNmodel::getRNType() const [inline]
```

Gets the underlying type for random number generation (default: uint64\_t)

```
16.30.2.24 unsigned int NNmodel::getSeed ( ) const [inline]
```

Get the random seed.

```
16.30.2.25 const map < string > & NNmodel::getSimLearnPostKernelParameters ( ) const [inline]
```

Gets std::map containing names and types of each parameter that should be passed through to the postsynaptic learning kernel.

```
16.30.2.26 unsigned int NNmodel::getSynapseDynamicsGridSize ( ) const
```

Gets the size of the synapse dynamics kernel thread grid.

This is calculated by adding together the number of threads required by each synapse population's synapse dynamics kernel, padded to be a multiple of GPU's thread block size.

```
16.30.2.27 const map<string, std::pair<unsigned int, unsigned int> >& NNmodel::getSynapseDynamicsGroups ( ) const [inline]
```

Get std::map containing names of synapse groups which require synapse dynamics and their thread IDs within the synapse dynamics kernel (padded to multiples of the GPU thread block size)

```
16.30.2.28 const map < string > & NNmodel::getSynapseDynamicsKernelParameters ( ) const [inline]
```

Gets std::map containing names and types of each parameter that should be passed through to the synapse dynamics kernel.

```
16.30.2.29 const map < string, SynapseGroup > & NNmodel::getSynapseGroups ( ) const [inline]
```

Get std::map containing all named SynapseGroup objects in model.

```
16.30.2.30 unsigned int NNmodel::getSynapseKernelGridSize ( ) const
```

Gets the size of the synapse kernel thread grid.

This is calculated by adding together the number of threads required by each synapse population's synapse kernel, padded to be a multiple of GPU's thread block size.

```
16.30.2.31 const map < string, string > & NNmodel::getSynapseKernelParameters ( ) const [inline]
```

Gets std::map containing names and types of each parameter that should be passed through to the synapse kernel.

```
16.30.2.32 unsigned int NNmodel::getSynapsePostLearnGridSize ( ) const
```

Gets the size of the post-synaptic learning kernel thread grid.

This is calculated by adding together the number of threads required by each synapse population's postsynaptic learning kernel, padded to be a multiple of GPU's thread block size.

```
16.30.2.33 const map<string, std::pair<unsigned int, unsigned int>> NNmodel::getSynapsePostLearnGroups ( ) const [inline]
```

Get std::map containing names of synapse groups which require postsynaptic learning and their thread IDs within the postsynaptic learning kernel (padded to multiples of the GPU thread block size)

16.30.2.34 bool NNmodel::isFinalized ( ) const [inline]

Is the model specification finalized.

16.30.2.35 bool NNmodel::isSynapseGroupDynamicsRequired ( const std::string & name ) const

Does named synapse group have synapse dynamics.

16.30.2.36 bool NNmodel::isSynapseGroupPostLearningRequired ( const std::string & name ) const

Does named synapse group have post-synaptic learning.

16.30.2.37 bool NNmodel::isTimingEnabled ( ) const [inline]

Are timers and timing commands enabled.

16.30.2.38 string NNmodel::scalarExpr ( const double val ) const

Get the string literal that should be used to represent a value in the model's floating-point type.

16.30.2.39 void NNmodel::setConstInp ( const string & , double )

This function has been deprecated in GeNN 2.2.

This function sets a global input value to the specified neuron group.

16.30.2.40 void NNmodel::setDT ( double newDT )

Set the integration step size of the model.

This function sets the integration time step DT of the model.

16.30.2.41 void NNmodel::setGPUDevice (int device)

Sets the underlying type for random number generation (default: uint64 t)

This function defines the way how the GPU is chosen. If "AUTODEVICE" (-1) is given as the argument, GeNN will use internal heuristics to choose the device. Otherwise the argument is the device number and the indicated device will be used.

Method to choose the GPU to be used for the model. If "AUTODEVICE' (-1), GeNN will choose the device based on a heuristic rule.

16.30.2.42 void NNmodel::setMaxConn ( const string & sname, unsigned int maxConnP )

This function defines the maximum number of connections for a neuron in the population.

16.30.2.43 void NNmodel::setName ( const std::string & )

Method to set the neuronal network model name.

16.30.2.44 void NNmodel::setNeuronClusterIndex ( const string & neuronGroup, int hostID, int deviceID )

Function for setting which host and which device a neuron group will be simulated on.

This function is for setting which host and which device a neuron group will be simulated on.

# Parameters

neuronGroup	Name of the neuron population
hostID	ID of the host
deviceID	ID of the device

16.30.2.45 void NNmodel::setPopulationSums ( )

Set the accumulated sums of lowest multiple of kernel block size >= group sizes for all simulated groups.

Accumulate the sums and block-size-padded sums of all simulation groups.

This method saves the neuron numbers of the populations rounded to the next multiple of the block size as well as the sums  $s(i) = sum_{1...i} n_i$  of the rounded population sizes. These are later used to determine the branching structure for the generated neuron kernel code.

16.30.2.46 void NNmodel::setPrecision ( FloatType floattype )

Set numerical precision for floating point.

This function sets the numerical precision of floating type variables. By default, it is GENN GENN FLOAT.

16.30.2.47 void NNmodel::setRNType ( const std::string & type )

Sets the underlying type for random number generation (default: uint64\_t)

16.30.2.48 void NNmodel::setSeed ( unsigned int inseed )

Set the random seed (disables automatic seeding if argument not 0).

This function sets the random seed. If the passed argument is > 0, automatic seeding is disabled. If the argument is 0, the underlying seed is obtained from the time() function.

#### **Parameters**

inseed	the new seed
1113000	tile liew seed

16.30.2.49 void NNmodel::setSpanTypeToPre ( const string & sname )

Method for switching the execution order of synapses to pre-to-post.

This function defines the execution order of the synapses in the kernels (0 : execute for every postsynaptic neuron 1: execute for every presynaptic neuron)

#### **Parameters**

sname	name of the synapse group to which to apply the pre-synaptic span type

16.30.2.50 void NNmodel::setSynapseClusterIndex ( const string & synapseGroup, int hostID, int deviceID )

Function for setting which host and which device a synapse group will be simulated on.

This function is for setting which host and which device a synapse group will be simulated on.

#### **Parameters**

synapseGroup	Name of the synapse population
hostID	ID of the host
deviceID	ID of the device

16.30.2.51 void NNmodel::setSynapseG ( const string & , double )

This function has been depreciated as of GeNN 2.2.

This functions sets the global value of the maximal synaptic conductance for a synapse population that was idfentified as conductance specification method "GLOBALG".

16.30.2.52 void NNmodel::setTiming ( bool theTiming )

Set whether timers and timing commands are to be included.

This function sets a flag to determine whether timers and timing commands are to be included in generated code.

16.30.2.53 bool NNmodel::zeroCopyInUse ( ) const

Are any variables in any populations in this model using zero-copy memory?

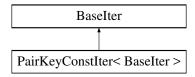
The documentation for this class was generated from the following files:

- · modelSpec.h
- src/modelSpec.cc

# 16.31 PairKeyConstIter < Baselter > Class Template Reference

```
#include <codeGenUtils.h>
```

Inheritance diagram for PairKeyConstIter< BaseIter >:



#### **Public Member Functions**

- PairKeyConstIter ()
- PairKeyConstIter (BaseIter iter)
- const KeyType \* operator-> () const
- const KeyType & operator\* () const
- 16.31.1 Constructor & Destructor Documentation
- 16.31.1.1 template < typename Baselter > PairKeyConstIter < Baselter > ::PairKeyConstIter ( ) [inline]
- 16.31.1.2 template < typename Baselter > PairKeyConstlter < Baselter >::PairKeyConstlter (Baselter iter) [inline]
- 16.31.2 Member Function Documentation
- 16.31.2.1 template < typename Baselter > const KeyType& PairKeyConstIter < Baselter >::operator\* ( ) const [inline]
- 16.31.2.2 template < typename Baselter > const KeyType\* PairKeyConstIter < Baselter >::operator-> ( ) const [inline]

The documentation for this class was generated from the following file:

codeGenUtils.h

# 16.32 Parameter Struct Reference

#### **Public Attributes**

- string name
- · string value

#### 16.32.1 Member Data Documentation

16.32.1.1 string Parameter::name

16.32.1.2 string Parameter::value

The documentation for this struct was generated from the following file:

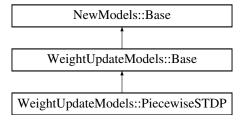
· experiment.cc

# 16.33 WeightUpdateModels::PiecewiseSTDP Class Reference

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse.

#include <newWeightUpdateModels.h>

Inheritance diagram for WeightUpdateModels::PiecewiseSTDP:



## **Public Types**

- typedef NewModels::ValueBase< 10 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

# **Public Member Functions**

· virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

- virtual DerivedParamVec getDerivedParams () const
- virtual bool isPreSpikeTimeRequired () const

Whether presynaptic spike times are needed or not.

virtual bool isPostSpikeTimeRequired () const

Whether postsynaptic spike times are needed or not.

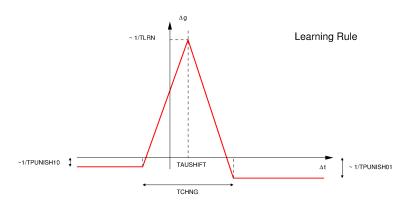
**Static Public Member Functions** 

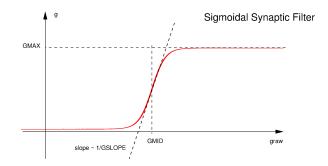
static const PiecewiseSTDP \* getInstance ()

## 16.33.1 Detailed Description

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse.

The STDP window is defined as a piecewise function:





The STDP curve is applied to the raw synaptic conductance gRaw, which is then filtered through the sugmoidal filter displayed above to obtain the value of g.

Note

The STDP curve implies that unpaired pre- and post-synaptic spikes incur a negative increment in gRaw (and hence in g).

The time of the last spike in each neuron, "sTXX", where XX is the name of a neuron population is (somewhat arbitrarily) initialised to -10.0 ms. If neurons never spike, these spike times are used.

It is the raw synaptic conductance gRaw that is subject to the STDP rule. The resulting synaptic conductance is a sigmoid filter of gRaw. This implies that g is initialised but not gRaw, the synapse will revert to the value that corresponds to gRaw.

An example how to use this synapse correctly is given in map\_classol.cc (MBody1 userproject):

```
for (int i= 0; i < model.neuronN[1]*model.neuronN[3]; i++) {
    if (gKCDN[i] < 2.0*SCALAR_MIN) {
        cnt++;
        fprintf(stdout, "Too low conductance value %e detected and set to 2*SCALAR_MIN= %e, at index %d
    \n", gKCDN[i], 2*SCALAR_MIN, i);
        gKCDN[i] = 2.0*SCALAR_MIN; //to avoid log(0)/0 below
    }
    scalar tmp = gKCDN[i] / myKCDN_p[5]*2.0;
    gRawKCDN[i] = 0.5 * log( tmp / (2.0 - tmp)) /myKCDN_p[7] +
    myKCDN_p[6];
}
cerr << "Total number of low value corrections: " << cnt << endl;</pre>
```

Note

One cannot set values of g fully to 0, as this leads to gRaw = -infinity and this is not support. I.e., 'g' needs to be some nominal value > 0 (but can be extremely small so that it acts like it's 0).

The model has 2 variables:

- g: conductance of scalar type
- gRaw: raw conductance of scalar type

Parameters are (compare to the figure above):

- tLrn: Time scale of learning changes
- tChng: Width of learning window
- tDecay: Time scale of synaptic strength decay
- tPunish10: Time window of suppression in response to 1/0
- tPunish01: Time window of suppression in response to 0/1
- gMax: Maximal conductance achievable
- gMid: Midpoint of sigmoid g filter curve
- gSlope: Slope of sigmoid g filter curve
- tauShift: Shift of learning curve
- gSyn0: Value of syn conductance g decays to
- 16.33.2 Member Typedef Documentation
- 16.33.2.1 typedef NewModels::ValueBase< 10 > WeightUpdateModels::PiecewiseSTDP::ParamValues
- 16.33.2.2 typedef NewModels::ValueBase< 2 > WeightUpdateModels::PiecewiseSTDP::VarValues
- 16.33.3 Member Function Documentation
- 16.33.3.1 virtual DerivedParamVec WeightUpdateModels::PiecewiseSTDP::getDerivedParams( )const [inline], [virtual]

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented from NewModels::Base.

```
16.33.3.2 static const PiecewiseSTDP* WeightUpdateModels::PiecewiseSTDP::getInstance( ) [inline], [static]
```

```
16.33.3.3 virtual std::string WeightUpdateModels::PiecewiseSTDP::getLearnPostCode( ) const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

16.33.3.4 virtual String Vec Weight Update Models::Piecewise STDP::getParamNames() const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.33.3.5 virtual std::string WeightUpdateModels::PiecewiseSTDP::getSimCode() const [inline], [virtual]

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

16.33.3.6 virtual StringPairVec WeightUpdateModels::PiecewiseSTDP::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.33.3.7 virtual bool WeightUpdateModels::PiecewiseSTDP::isPostSpikeTimeRequired ( ) const [inline], [virtual]

Whether postsynaptic spike times are needed or not.

Reimplemented from WeightUpdateModels::Base.

```
16.33.3.8 virtual bool WeightUpdateModels::PiecewiseSTDP::isPreSpikeTimeRequired ( ) const [inline], [virtual]
```

Whether presynaptic spike times are needed or not.

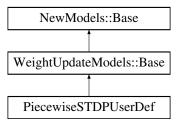
Reimplemented from WeightUpdateModels::Base.

The documentation for this class was generated from the following file:

newWeightUpdateModels.h

# 16.34 PiecewiseSTDPUserDef Class Reference

Inheritance diagram for PiecewiseSTDPUserDef:



# **Public Types**

- typedef NewModels::ValueBase< 10 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

# **Public Member Functions**

• virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

- virtual DerivedParamVec getDerivedParams () const
- virtual bool isPreSpikeTimeRequired () const

Whether presynaptic spike times are needed or not.

virtual bool isPostSpikeTimeRequired () const

Whether postsynaptic spike times are needed or not.

#### Static Public Member Functions

• static const PiecewiseSTDPUserDef \* getInstance ()

16.34.1 Member Typedef Documentation

```
16.34.1.1 typedef NewModels::ValueBase< 10 > PiecewiseSTDPUserDef::ParamValues
```

16.34.1.2 typedef NewModels::ValueBase< 2 > PiecewiseSTDPUserDef::VarValues

16.34.2 Member Function Documentation

```
16.34.2.1 virtual DerivedParamVec PiecewiseSTDPUserDef::getDerivedParams() const [inline], [virtual]
```

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented from NewModels::Base.

```
16.34.2.2 static const PiecewiseSTDPUserDef* PiecewiseSTDPUserDef::getInstance( ) [inline], [static]
```

```
16.34.2.3 virtual std::string PiecewiseSTDPUserDef::getLearnPostCode( ) const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

```
16.34.2.4 virtual String Vec Piecewise STDPUserDef::getParamNames ( ) const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

```
16.34.2.5 virtual std::string PiecewiseSTDPUserDef::getSimCode() const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.34.2.6 virtual StringPairVec PiecewiseSTDPUserDef::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.34.2.7 virtual bool PiecewiseSTDPUserDef::isPostSpikeTimeRequired( ) const [inline], [virtual]

Whether postsynaptic spike times are needed or not.

Reimplemented from WeightUpdateModels::Base.

16.34.2.8 virtual bool PiecewiseSTDPUserDef::isPreSpikeTimeRequired() const [inline], [virtual]

Whether presynaptic spike times are needed or not.

Reimplemented from WeightUpdateModels::Base.

The documentation for this class was generated from the following file:

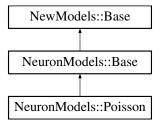
• MBody\_userdef.cc

## 16.35 NeuronModels::Poisson Class Reference

#### Poisson neurons.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::Poisson:



## **Public Types**

- typedef NewModels::ValueBase< 4 > ParamValues
- typedef NewModels::ValueBase< 3 > VarValues

# **Public Member Functions**

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

· virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

- virtual StringPairVec getExtraGlobalParams () const
- · virtual bool isPoisson () const

# **Static Public Member Functions**

static const NeuronModels::Poisson \* getInstance ()

## 16.35.1 Detailed Description

#### Poisson neurons.

Poisson neurons have constant membrane potential (Vrest) unless they are activated randomly to the Vspike value if (t-SpikeTime) > trefract.

It has 3 variables:

- V Membrane potential
- Seed Seed for random number generation
- SpikeTime Time at which the neuron spiked for the last time

#### and 4 parameters:

- therate Firing rate
- trefract Refractory period
- Vspike Membrane potential at spike (mV)
- Vrest Membrane potential at rest (mV)

#### Note

The initial values array for the Poisson type needs three entries for V, Seed and SpikeTime and the parameter array needs four entries for therate, trefract, Vspike and Vrest, in that order. Internally, GeNN uses a linear approximation for the probability of firing a spike in a given time step of size DT, i.e. the probability of firing is therate times DT:  $p=\lambda \Delta t$ . This approximation is usually very good, especially for typical, quite small time steps and moderate firing rates. However, it is worth noting that the approximation becomes poor for very high firing rates and large time steps. An unrelated problem may occur with very low firing rates and small time steps. In that case it can occur that the firing probability is so small that the granularity of the 64 bit integer based random number generator begins to show. The effect manifests itself in that small changes in the firing rate do not seem to have an effect on the behaviour of the Poisson neurons because the numbers are so small that only if the random number is identical 0 a spike will be triggered. GeNN uses a separate random number generator for each Poisson neuron. The seeds (and later states) of these random number generators are stored in the seed variable. GeNN allocates memory for these seeds/states in the generated allocateMem() function. It is, however, currently the responsibility of the user to fill the array of seeds with actual random seeds. Not doing so carries the risk that all random number generators are seeded with the same seed ("0") and produce the same random numbers across neurons at each given time step. When using the GPU, seed also must be copied to the GPU after having been initialized.

```
16.35.2 Member Typedef Documentation
```

- 16.35.2.1 typedef NewModels::ValueBase< 4 > NeuronModels::Poisson::ParamValues
- 16.35.2.2 typedef NewModels::ValueBase< 3 > NeuronModels::Poisson::VarValues
- 16.35.3 Member Function Documentation
- 16.35.3.1 virtual StringPairVec NeuronModels::Poisson::getExtraGlobalParams() const [inline], [virtual]

Gets names and types (as strings) of additional per-population parameters for the weight update model.

Reimplemented from NeuronModels::Base.

```
16.35.3.2 static const NeuronModels::Poisson* NeuronModels::Poisson::getInstance() [inline], [static]
```

16.35.3.3 virtual String Vec Neuron Models::Poisson::getParamNames() const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

```
16.35.3.4 virtual std::string NeuronModels::Poisson::getSimCode()const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.35.3.5 virtual std::string NeuronModels::Poisson::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.35.3.6 virtual StringPairVec NeuronModels::Poisson::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.35.3.7 virtual bool NeuronModels::Poisson::isPoisson() const [inline], [virtual]
```

Is this neuron model the internal Poisson model (which requires a number of special cases)

Reimplemented from NeuronModels::Base.

The documentation for this class was generated from the following file:

newNeuronModels.h

# 16.36 postSynModel Class Reference

Class to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

```
#include <postSynapseModels.h>
```

#### **Public Member Functions**

postSynModel ()

Constructor for postSynModel objects.

~postSynModel ()

Destructor for postSynModel objects.

# **Public Attributes**

string postSyntoCurrent

Code that defines how postsynaptic update is translated to current.

string postSynDecay

Code that defines how postsynaptic current decays.

string supportCode

Support code is made available within the neuron kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

vector< string > varNames

Names of the variables in the postsynaptic model.

vector< string > varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

vector< string > pNames

Names of (independent) parameters of the model.

vector< string > dpNames

Names of dependent parameters of the model.

• dpclass \* dps

Derived parameters.

## 16.36.1 Detailed Description

Class to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

```
16.36.2 Constructor & Destructor Documentation
```

16.36.2.1 postSynModel::postSynModel()

Constructor for postSynModel objects.

16.36.2.2 postSynModel:: $\sim$ postSynModel ( )

Destructor for postSynModel objects.

16.36.3 Member Data Documentation

16.36.3.1 vector<string> postSynModel::dpNames

Names of dependent parameters of the model.

16.36.3.2 dpclass\* postSynModel::dps

Derived parameters.

16.36.3.3 vector<string> postSynModel::pNames

Names of (independent) parameters of the model.

16.36.3.4 string postSynModel::postSynDecay

Code that defines how postsynaptic current decays.

16.36.3.5 string postSynModel::postSyntoCurrent

Code that defines how postsynaptic update is translated to current.

## 16.36.3.6 string postSynModel::supportCode

Support code is made available within the neuron kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

16.36.3.7 vector<string> postSynModel::varNames

Names of the variables in the postsynaptic model.

16.36.3.8 vector<string> postSynModel::varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this class was generated from the following files:

- · postSynapseModels.h
- postSynapseModels.cc

## 16.37 pwSTDP Class Reference

TODO This class definition may be code-generated in a future release.

#include <synapseModels.h>

Inheritance diagram for pwSTDP:



# **Public Member Functions**

• double calculateDerivedParameter (int index, vector< double > pars, double=1.0)

## 16.37.1 Detailed Description

TODO This class definition may be code-generated in a future release.

This class defines derived parameters for the learn1synapse standard weightupdate model

# 16.37.2 Member Function Documentation

16.37.2.1 double pwSTDP::calculateDerivedParameter ( int *index*, vector< double > pars, double = 1.0 ) [inline], [virtual]

Reimplemented from dpclass.

The documentation for this class was generated from the following file:

• synapseModels.h

# 16.38 QTIsaac < ALPHA, T > Class Template Reference

#### Classes

struct randctx

#### **Public Types**

- enum { N = (1<<ALPHA) }</li>
- · typedef unsigned char byte

#### **Public Member Functions**

- QTIsaac (T a=0, T b=0, T c=0)
- virtual ~QTIsaac (void)
- T rand (void)
- virtual void randinit (randctx \*ctx, bool bUseSeed)
- virtual void srand (T a=0, T b=0, T c=0, T \*s=NULL)

# **Protected Member Functions**

- virtual void isaac (randctx \*ctx)
- T ind (T \*mm, T x)
- void rngstep (T mix, T &a, T &b, T \*&mm, T \*&m, T \*&m2, T \*&r, T &x, T &y)
- virtual void shuffle (T &a, T &b, T &c, T &d, T &e, T &f, T &g, T &h)
- 16.38.1 Member Typedef Documentation
- 16.38.1.1 template < int ALPHA = (8), class T = ISAAC\_INT> typedef unsigned char QTIsaac < ALPHA, T >::byte
- 16.38.2 Member Enumeration Documentation
- 16.38.2.1 template < int ALPHA = (8), class T = ISAAC\_INT> anonymous enum

## **Enumerator**

Ν

- 16.38.3 Constructor & Destructor Documentation
- 16.38.3.1 template < int ALPHA, class T> QTIsaac < ALPHA, T>::QTIsaac ( T a = 0, T b = 0, T c = 0 )
- 16.38.3.2 template < int ALPHA, class T > QTIsaac < ALPHA, T >:: ~QTIsaac ( void ) [virtual]
- 16.38.4 Member Function Documentation
- 16.38.4.1 template < int ALPHA, class T > T QTIsaac < ALPHA, T >::ind ( T \* mm, T x ) [inline], [protected]
- 16.38.4.2 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::isaac ( randctx \* ctx ) [protected], [virtual]
- 16.38.4.3 template < int ALPHA, class T > T QTIsaac < ALPHA, T >::rand ( void ) [inline]
- 16.38.4.4 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::randinit ( randctx \* ctx, bool bUseSeed ) [virtual]

```
16.38.4.5 template<int ALPHA, class T> void QTIsaac< ALPHA, T>::rngstep ( T mix, T & a, T & b, T *& mm, T *& m, T *& m2, T *& r, T & x, T & y ) [inline], [protected]
```

```
16.38.4.6 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::shuffle ( T & a, T & b, T & c, T & d, T & e, T & f, T & g, T & h ) [protected], [virtual]
```

```
16.38.4.7 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::srand ( T a = 0, T b = 0, T c = 0, T * s = NULL ) [virtual]
```

The documentation for this class was generated from the following file:

· isaac.cc

## 16.39 QTIsaac < ALPHA, T >::randctx Struct Reference

**Public Member Functions** 

- randctx (void)
- ∼randctx (void)

#### **Public Attributes**

- · T randcnt
- T \* randrsl
- T \* randmem
- T randa
- T randb
- T randc

#### 16.39.1 Constructor & Destructor Documentation

```
16.39.1.1 template < int ALPHA = (8), class T = ISAAC_INT> QTIsaac < ALPHA, T >::randctx::randctx ( void ) [inline]
```

```
16.39.1.2 template < int ALPHA = (8), class T = ISAAC_INT> QTIsaac < ALPHA, T >::randctx::~randctx ( void ) [inline]
```

16.39.2 Member Data Documentation

```
16.39.2.1 \quad template < int ALPHA = (8), class \ T = ISAAC\_INT > T \ QTIsaac < ALPHA, \ T > :: randctx:: randa
```

16.39.2.2 template < int ALPHA = (8), class T = ISAAC\_INT> T QTIsaac < ALPHA, T >::randctx::randb

16.39.2.3 template < int ALPHA = (8), class T = ISAAC\_INT> T QTIsaac < ALPHA, T >::randctx::randc

16.39.2.4 template < int ALPHA = (8), class T = ISAAC\_INT> T QTIsaac < ALPHA, T >::randctx::randcnt

16.39.2.5 template < int ALPHA = (8), class T = ISAAC\_INT> T\* QTIsaac < ALPHA, T >::randctx::randmem

16.39.2.6 template < int ALPHA = (8), class T = ISAAC\_INT> T\* QTIsaac < ALPHA, T >::randctx::randrsl

The documentation for this struct was generated from the following file:

isaac.cc

## 16.40 randomGauss Class Reference

Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

```
#include <gauss.h>
```

#### **Public Member Functions**

randomGauss ()

Constructor for the Gaussian random number generator class without giving explicit seeds.

• randomGauss (unsigned long, unsigned long, unsigned long)

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

- ∼randomGauss ()
- double n ()

Method for obtaining a random number with Gaussian distribution.

void srand (unsigned long, unsigned long, unsigned long)

Function for seeding with fixed seeds.

#### 16.40.1 Detailed Description

Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

A random number from a Gaussian distribution of mean 0 and standard deviation 1 is obtained by calling the method randomGauss::n().

## 16.40.2 Constructor & Destructor Documentation

```
16.40.2.1 randomGauss::randomGauss( ) [explicit]
```

Constructor for the Gaussian random number generator class without giving explicit seeds.

The seeds for random number generation are generated from the internal clock of the computer during execution.

```
16.40.2.2 randomGauss::randomGauss (unsigned long seed1, unsigned long seed2, unsigned long seed3)
```

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

The seeds are three arbitrary unsigned long integers.

```
16.40.2.3 randomGauss::∼randomGauss( ) [inline]
16.40.3 Member Function Documentation
16.40.3.1 double randomGauss::n( )
```

Method for obtaining a random number with Gaussian distribution.

Function for generating a pseudo random number from a Gaussian distribution.

16.40.3.2 void randomGauss::srand ( unsigned long seed1, unsigned long seed2, unsigned long seed3 )

Function for seeding with fixed seeds.

The documentation for this class was generated from the following files:

- · gauss.h
- gauss.cc

# 16.41 randomGen Class Reference

Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.

```
#include <randomGen.h>
```

#### **Public Member Functions**

· randomGen ()

Constructor for the ISAAC random number generator class without giving explicit seeds.

• randomGen (unsigned long, unsigned long, unsigned long)

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

- ∼randomGen ()
- double n ()

Method to obtain a random number from a uniform ditribution on [0,1].

• void srand (unsigned long, unsigned long, unsigned long)

Function for seeding with fixed seeds.

#### 16.41.1 Detailed Description

Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.

The random number generator initializes with system timea or explicit seeds and returns a random number according to a uniform distribution on [0,1]; making use of the ISAAC random number generator; C++ Implementation by Quinn Tyler Jackson of the RG invented by Bob Jenkins Jr.

#### 16.41.2 Constructor & Destructor Documentation

```
16.41.2.1 randomGen::randomGen() [explicit]
```

Constructor for the ISAAC random number generator class without giving explicit seeds.

The seeds for random number generation are generated from the internal clock of the computer during execution.

```
16.41.2.2 randomGen::randomGen (unsigned long seed1, unsigned long seed2, unsigned long seed3)
```

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

The seeds are three arbitrary unsigned long integers.

```
16.41.2.3 randomGen::∼randomGen ( ) [inline]
16.41.3 Member Function Documentation
```

```
16.41.3.1 double randomGen::n()
```

Method to obtain a random number from a uniform ditribution on [0,1].

Function for generating a pseudo random number from a uniform distribution on the interval [0,1].

16.41.3.2 void randomGen::srand (unsigned long seed1, unsigned long seed2, unsigned long seed3)

Function for seeding with fixed seeds.

The documentation for this class was generated from the following files:

- · randomGen.h
- randomGen.cc

# 16.42 rulkovdp Class Reference

Class defining the dependent parameters of the Rulkov map neuron.

#include <neuronModels.h>

Inheritance diagram for rulkovdp:



# **Public Member Functions**

• double calculateDerivedParameter (int index, vector< double > pars, double=1.0)

## 16.42.1 Detailed Description

Class defining the dependent parameters of the Rulkov map neuron.

#### 16.42.2 Member Function Documentation

16.42.2.1 double rulkovdp::calculateDerivedParameter ( int *index*, vector < double > *pars*, double = 1.0 ) [inline], [virtual]

Reimplemented from dpclass.

The documentation for this class was generated from the following file:

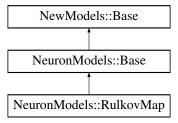
· neuronModels.h

# 16.43 NeuronModels::RulkovMap Class Reference

Rulkov Map neuron.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::RulkovMap:



# **Public Types**

- typedef NewModels::ValueBase< 4 > ParamValues
- typedef NewModels::ValueBase< 2 > VarValues

#### **Public Member Functions**

· virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual DerivedParamVec getDerivedParams () const

# **Static Public Member Functions**

static const NeuronModels::RulkovMap \* getInstance ()

#### 16.43.1 Detailed Description

#### Rulkov Map neuron.

The RulkovMap type is a map based neuron model based on [4] but in the 1-dimensional map form used in [3]:

$$V(t + \Delta t) = \begin{cases} V_{\text{spike}} \left( \frac{\alpha V_{\text{spike}}}{V_{\text{spike}} - V(t) \beta I_{\text{syn}}} + y \right) & V(t) \leq 0 \\ V_{\text{spike}} \left( \alpha + y \right) & V(t) \leq V_{\text{spike}} \left( \alpha + y \right) & V(t + \Delta t) \leq 0 \\ -V_{\text{spike}} & \text{otherwise} \end{cases}$$

Note

The RulkovMap type only works as intended for the single time step size of DT= 0.5.

The RulkovMap type has 2 variables:

- preV the membrane potential at the previous time step

and it has 4 parameters:

- Vspike determines the amplitude of spikes, typically -60mV
- alpha determines the shape of the iteration function, typically  $\alpha$ = 3
- y "shift / excitation" parameter, also determines the iteration function, originally, y= -2.468
- beta roughly speaking equivalent to the input resistance, i.e. it regulates the scale of the input into the neuron, typically  $\beta$ = 2.64 M $\Omega$ .

Note

The initial values array for the RulkovMap type needs two entries for V and Vpre and the parameter array needs four entries for Vspike, alpha, y and beta, in that order.

```
16.43.2 Member Typedef Documentation
```

```
16.43.2.1 typedef NewModels::ValueBase< 4 > NeuronModels::RulkovMap::ParamValues
```

16.43.2.2 typedef NewModels::ValueBase< 2 > NeuronModels::RulkovMap::VarValues

16.43.3 Member Function Documentation

```
16.43.3.1 virtual DerivedParamVec NeuronModels::RulkovMap::getDerivedParams ( ) const [inline], [virtual]
```

Gets names of derived model parameters and the function objects to call to Calculate their value from a vector of model parameter values

Reimplemented from NewModels::Base.

```
16.43.3.2 static const NeuronModels::RulkovMap* NeuronModels::RulkovMap::getInstance( ) [inline], [static]
```

```
16.43.3.3 virtual String Vec NeuronModels::RulkovMap::getParamNames() const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

```
16.43.3.4 virtual std::string NeuronModels::RulkovMap::getSimCode( ) const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

```
16.43.3.5 virtual std::string NeuronModels::RulkovMap::getThresholdConditionCode() const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.43.3.6 virtual String Pair Vec Neuron Models::Rulkov Map::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• newNeuronModels.h

## 16.44 Schmuker2014\_classifier Class Reference

This class cpontains the methods for running the Schmuker\_2014\_classifier example model.

```
#include <Schmuker2014_classifier.h>
```

## **Public Types**

enum data\_type { data\_type\_int, data\_type\_uint, data\_type\_float, data\_type\_double }

#### **Public Member Functions**

- Schmuker2014 classifier ()
- ∼Schmuker2014\_classifier ()
- · void allocateHostAndDeviceMemory ()
- void populateDeviceMemory ()
- void update\_input\_data\_on\_device ()
- void clearDownDevice ()
- void run (float runtime, string filename\_rasterPlot, bool usePlasticity)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void outputSpikes (FILE \*, string delim)
- void initialiseWeights\_SPARSE\_RN\_PN ()
- void initialiseWeights\_WTA\_PN\_PN ()
- void initialiseWeights DENSE PN AN ()
- void initialiseWeights\_WTA\_AN\_AN ()
- void createWTAConnectivity (float \*synapse, UINT populationSize, UINT clusterSize, float synapseWeight, float probability)
- bool randomEventOccurred (float probability)
- void updateWeights\_PN\_AN\_on\_device ()
- · void generate or load inputrates dataset (unsigned int recordingIdx)
- void generate\_inputrates\_dataset (unsigned int recordingldx)
- FILE \* openRecordingFile (UINT recordingIndex)
- void applyLearningRuleSynapses (float \*synapsesPNAN)
- void initialiseInputData ()
- void load VR data ()
- void setCorrectClass (UINT recordingIdx)
- UINT getClassCluster (UINT anldx)
- void loadClassLabels ()
- void addInputRate (float \*samplePoint, UINT timeStep)
- uint64\_t convertToRateCode (float inputRateHz)
- float calculateVrResponse (float \*samplePoint, float \*vrPoint)
- void setMaxMinSampleDistances ()
- void findMaxMinSampleDistances (float \*samples, UINT startAt, UINT totalSamples)
- float getSampleDistance (UINT max0\_min1)
- float getManhattanDistance (float \*pointA, float \*pointB, UINT numElements)
- float getRand0to1 ()
- UINT calculateOverallWinner ()
- UINT calculateWinner (unsigned int \*clusterSpikeCount)
- UINT calculateCurrentWindowWinner ()
- · void updateIndividualSpikeCountPN ()
- · void resetIndividualSpikeCountPN ()
- void updateClusterSpikeCountAN ()
- void resetClusterSpikeCountAN ()
- void resetOverallWinner ()
- void updateWeights\_PN\_AN ()
- UINT getClusterIndex (UINT neuronIndex, UINT clusterSize)
- void generateSimulatedTimeSeriesData ()
- string getRecordingFilename (UINT recordingIdx)
- bool loadArrayFromTextFile (string path, void \*array, string delim, UINT arrayLen, data\_type dataType)
- void checkContents (string title, void \*array, UINT howMany, UINT displayPerLine, data\_type dataType, UINT decimalPoints)
- void checkContents (string title, void \*array, UINT howMany, UINT displayPerLine, data\_type dataType, UINT decimalPoints, string delim)
- void printSeparator ()
- void resetDevice ()
- void startLog ()

#### **Public Attributes**

- double d\_maxRandomNumber
- NNmodel model
- uint64\_t \* inputRates
- unsigned int inputRatesSize
- float \* vrData
- unsigned int \* classLabel
- unsigned int \* individualSpikeCountPN
- unsigned int \* overallWinnerSpikeCountAN
- unsigned int \* clusterSpikeCountAN
- float \* plasticWeights
- uint64\_t \* d\_inputRates
- unsigned int countRN
- · unsigned int countPN
- unsigned int countAN
- unsigned int countPNAN
- float \* sampleDistance
- · string recordingsDir
- string cacheDir
- string outputDir
- · string datasetName
- string uniqueRunId
- UINT correctClass
- · int winningClass
- FILE \* log
- UINT param\_SPIKING\_ACTIVITY\_THRESHOLD\_HZ
- UINT param\_MAX\_FIRING\_RATE\_HZ
- · UINT param MIN FIRING RATE HZ
- float param\_GLOBAL\_WEIGHT\_SCALING
- · float param\_WEIGHT\_RN\_PN
- float param\_CONNECTIVITY\_RN\_PN
- float param\_WEIGHT\_WTA\_PN\_PN
- float param\_WEIGHT\_WTA\_AN\_AN
- float param\_CONNECTIVITY\_PN\_PN
- · float param\_CONNECTIVITY\_AN\_AN
- float param\_CONNECTIVITY\_PN\_AN
- float param\_MIN\_WEIGHT\_PN\_AN
- float param\_MAX\_WEIGHT\_PN\_AN
- float param\_WEIGHT\_DELTA\_PN\_AN
- float param\_PLASTICITY\_INTERVAL\_MS
- · bool clearedDownDevice

## Static Public Attributes

• static const unsigned int timestepsPerRecording = RECORDING\_TIME\_MS / DT

## 16.44.1 Detailed Description

This class cpontains the methods for running the Schmuker\_2014\_classifier example model.

```
16.44.2 Member Enumeration Documentation
16.44.2.1 enum Schmuker2014_classifier::data_type
Enumerator
      data_type_int
      data_type_uint
      data_type_float
      data_type_double
16.44.3 Constructor & Destructor Documentation
16.44.3.1 Schmuker2014_classifier::Schmuker2014_classifier()
16.44.3.2 Schmuker2014_classifier::~Schmuker2014_classifier( )
16.44.4 Member Function Documentation
16.44.4.1 void Schmuker2014_classifier::addInputRate ( float * samplePoint, UINT timeStep )
16.44.4.2 void Schmuker2014_classifier::allocateHostAndDeviceMemory ( )
16.44.4.3 void Schmuker2014_classifier::applyLearningRuleSynapses ( float * synapsesPNAN )
16.44.4.4 UINT Schmuker2014_classifier::calculateCurrentWindowWinner()
16.44.4.5 UINT Schmuker2014_classifier::calculateOverallWinner( )
16.44.4.6 float Schmuker2014 classifier::calculateVrResponse ( float * samplePoint, float * vrPoint )
16.44.4.7 UINT Schmuker2014_classifier::calculateWinner ( unsigned int * clusterSpikeCount )
16.44.4.8 void Schmuker2014_classifier::checkContents ( string title, void * array, UINT howMany, UINT displayPerLine,
          data_type dataType, UINT decimalPoints )
16.44.4.9 void Schmuker2014_classifier::checkContents ( string title, void * array, UINT howMany, UINT displayPerLine,
          data_type dataType, UINT decimalPoints, string delim )
16.44.4.10 void Schmuker2014_classifier::clearDownDevice ( )
16.44.4.11 uint64_t Schmuker2014_classifier::convertToRateCode ( float inputRateHz )
16.44.4.12 void Schmuker2014_classifier::createWTAConnectivity ( float * synapse, UINT populationSize, UINT clusterSize,
           float synapseWeight, float probability )
16.44.4.13 void Schmuker2014_classifier::findMaxMinSampleDistances ( float * samples, UINT startAt, UINT totalSamples )
16.44.4.14 void Schmuker2014_classifier::generate_inputrates_dataset ( unsigned int recordingldx )
16.44.4.15 void Schmuker2014_classifier::generate_or_load_inputrates_dataset ( unsigned int recordingIdx )
16.44.4.16 void Schmuker2014_classifier::generateSimulatedTimeSeriesData ( )
16.44.4.17 UINT Schmuker2014_classifier::getClassCluster ( UINT anldx )
16.44.4.18 UINT Schmuker2014_classifier::getClusterIndex ( UINT neuronIndex, UINT clusterSize )
16.44.4.19 float Schmuker2014_classifier::getManhattanDistance (float * pointA, float * pointB, UINT numElements)
```

```
16.44.4.20
           float Schmuker2014_classifier::getRand0to1 ( )
16.44.4.21
           string Schmuker2014_classifier::getRecordingFilename ( UINT recordingIdx )
16.44.4.22
          float Schmuker2014_classifier::getSampleDistance ( UINT max0_min1 )
16.44.4.23
           void Schmuker2014_classifier::getSpikeNumbersFromGPU ( )
16.44.4.24
           void Schmuker2014_classifier::getSpikesFromGPU ( )
16.44.4.25 void Schmuker2014_classifier::initialiseInputData ( )
16.44.4.26 void Schmuker2014_classifier::initialiseWeights_DENSE_PN_AN ( )
16.44.4.27 void Schmuker2014_classifier::initialiseWeights_SPARSE_RN_PN ( )
16.44.4.28 void Schmuker2014_classifier::initialiseWeights_WTA_AN_AN ( )
16.44.4.29 void Schmuker2014_classifier::initialiseWeights_WTA_PN_PN ( )
16.44.4.30 void Schmuker2014_classifier::load_VR_data( )
           bool Schmuker2014_classifier::loadArrayFromTextFile ( string path, void * array, string delim, UINT arrayLen,
16.44.4.31
           data_type dataType )
16.44.4.32 void Schmuker2014_classifier::loadClassLabels ( )
16.44.4.33 FILE * Schmuker2014_classifier::openRecordingFile ( UINT recordingIndex )
16.44.4.34 void Schmuker2014_classifier::outputSpikes (FILE * f, string delim)
16.44.4.35 void Schmuker2014_classifier::populateDeviceMemory ( )
16.44.4.36 void Schmuker2014_classifier::printSeparator()
16.44.4.37 bool Schmuker2014_classifier::randomEventOccurred ( float probability )
16.44.4.38 void Schmuker2014_classifier::resetClusterSpikeCountAN ( )
16.44.4.39 void Schmuker2014_classifier::resetDevice ( )
16.44.4.40 void Schmuker2014_classifier::resetIndividualSpikeCountPN()
16.44.4.41 void Schmuker2014_classifier::resetOverallWinner()
16.44.4.42 void Schmuker2014_classifier::run ( float runtime, string filename_rasterPlot, bool usePlasticity )
16.44.4.43 void Schmuker2014_classifier::setCorrectClass ( UINT recordingldx )
16.44.4.44 void Schmuker2014_classifier::setMaxMinSampleDistances ( )
16.44.4.45 void Schmuker2014_classifier::startLog ( )
16.44.4.46 void Schmuker2014_classifier::update_input_data_on_device ( )
16.44.4.47 void Schmuker2014_classifier::updateClusterSpikeCountAN ( )
16.44.4.48 void Schmuker2014_classifier::updateIndividualSpikeCountPN ( )
16.44.4.49 void Schmuker2014_classifier::updateWeights_PN_AN ( )
16.44.4.50 void Schmuker2014_classifier::updateWeights_PN_AN_on_device ( )
```

# 16.44.5 Member Data Documentation

6.44.5.1	string Schmuker2014_classifier::cacheDir
6.44.5.2	unsigned int* Schmuker2014_classifier::classLabel
6.44.5.3	bool Schmuker2014_classifier::clearedDownDevice
6.44.5.4	unsigned int* Schmuker2014_classifier::clusterSpikeCountAN
6.44.5.5	UINT Schmuker2014_classifier::correctClass
6.44.5.6	unsigned int Schmuker2014_classifier::countAN
6.44.5.7	unsigned int Schmuker2014_classifier::countPN
6.44.5.8	unsigned int Schmuker2014_classifier::countPNAN
6.44.5.9	unsigned int Schmuker2014_classifier::countRN
6.44.5.10	uint64_t* Schmuker2014_classifier::d_inputRates
6.44.5.11	double Schmuker2014_classifier::d_maxRandomNumber
6.44.5.12	string Schmuker2014_classifier::datasetName
6.44.5.13	unsigned int* Schmuker2014_classifier::individualSpikeCountPN
6.44.5.14	uint64_t* Schmuker2014_classifier::inputRates
6.44.5.15	unsigned int Schmuker2014_classifier::inputRatesSize
6.44.5.16	FILE* Schmuker2014_classifier::log
6.44.5.17	NNmodel Schmuker2014_classifier::model
6.44.5.18	string Schmuker2014_classifier::outputDir
6.44.5.19	unsigned int* Schmuker2014_classifier::overallWinnerSpikeCountAN
6.44.5.20	float Schmuker2014_classifier::param_CONNECTIVITY_AN_AN
6.44.5.21	float Schmuker2014_classifier::param_CONNECTIVITY_PN_AN
6.44.5.22	float Schmuker2014_classifier::param_CONNECTIVITY_PN_PN
6.44.5.23	float Schmuker2014_classifier::param_CONNECTIVITY_RN_PN
6.44.5.24	float Schmuker2014_classifier::param_GLOBAL_WEIGHT_SCALING
6.44.5.25	UINT Schmuker2014_classifier::param_MAX_FIRING_RATE_HZ
6.44.5.26	float Schmuker2014_classifier::param_MAX_WEIGHT_PN_AN
6.44.5.27	UINT Schmuker2014_classifier::param_MIN_FIRING_RATE_HZ
6.44.5.28	float Schmuker2014_classifier::param_MIN_WEIGHT_PN_AN
6.44.5.29	float Schmuker2014_classifier::param_PLASTICITY_INTERVAL_MS
6.44.5.30	UINT Schmuker2014_classifier::param_SPIKING_ACTIVITY_THRESHOLD_HZ

```
16.44.5.31 float Schmuker2014_classifier::param_WEIGHT_DELTA_PN_AN

16.44.5.32 float Schmuker2014_classifier::param_WEIGHT_RN_PN

16.44.5.33 float Schmuker2014_classifier::param_WEIGHT_WTA_AN_AN

16.44.5.34 float Schmuker2014_classifier::param_WEIGHT_WTA_PN_PN

16.44.5.35 float* Schmuker2014_classifier::plasticWeights

16.44.5.36 string Schmuker2014_classifier::recordingsDir

16.44.5.37 float* Schmuker2014_classifier::sampleDistance

16.44.5.38 const unsigned int Schmuker2014_classifier::timestepsPerRecording = RECORDING_TIME_MS / DT [static]

16.44.5.39 string Schmuker2014_classifier::uniqueRunld

16.44.5.40 float* Schmuker2014_classifier::vrData

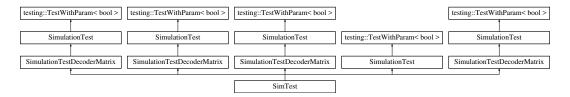
16.44.5.41 int Schmuker2014_classifier::winningClass
```

The documentation for this class was generated from the following files:

- · Schmuker2014 classifier.h
- Schmuker2014\_classifier.cc

#### 16.45 SimTest Class Reference

Inheritance diagram for SimTest:



# **Public Member Functions**

- virtual void Init ()
- virtual void Init ()
- bool Simulate ()
- virtual void Init ()
- virtual void Init ()
- virtual void Init ()

## **Additional Inherited Members**

#### 16.45.1 Member Function Documentation

16.45.1.1 virtual void SimTest::Init() [inline], [virtual]

Implements SimulationTest.

```
16.45.1.2 virtual void SimTest::Init() [inline], [virtual]
Implements SimulationTest.
16.45.1.3 virtual void SimTest::Init() [inline], [virtual]
Implements SimulationTest.
16.45.1.4 virtual void SimTest::Init() [inline], [virtual]
Implements SimulationTest.
16.45.1.5 virtual void SimTest::Init() [inline], [virtual]
Implements SimulationTest.
16.45.1.6 bool SimTest::Simulate ( ) [inline]
The documentation for this class was generated from the following file:

    decode_matrix_globalg_bitmask/test.cc

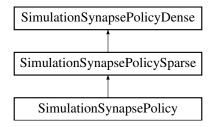
16.46 SimulationNeuronPolicyPrePostVar Class Reference
#include <simulation_neuron_policy_pre_post_var.h>
Public Member Functions
    • void Init ()
16.46.1 Member Function Documentation
16.46.1.1 void SimulationNeuronPolicyPrePostVar::Init() [inline]
The documentation for this class was generated from the following file:
    simulation_neuron_policy_pre_post_var.h
16.47 SimulationNeuronPolicyPreVar Class Reference
#include <simulation_neuron_policy_pre_var.h>
Public Member Functions
    void Init ()
16.47.1 Member Function Documentation
16.47.1.1 void SimulationNeuronPolicyPreVar::Init() [inline]
The documentation for this class was generated from the following file:
```

simulation\_neuron\_policy\_pre\_var.h

Generated on June 7, 2017 for GeNN by Doxygen

# 16.48 SimulationSynapsePolicy Class Reference

Inheritance diagram for SimulationSynapsePolicy:



#### **Public Member Functions**

- void Init ()
- template<typename UpdateFn, typename StepGeNNFn >
  float Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn)

## **Additional Inherited Members**

#### 16.48.1 Member Function Documentation

- 16.48.1.1 void SimulationSynapsePolicy::Init() [inline]
- 16.48.1.2 template < typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicy::Simulate ( UpdateFn updateFn, StepGeNNFn stepGeNNFn ) [inline]

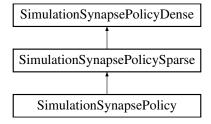
The documentation for this class was generated from the following file:

• extra\_global\_params\_in\_sim\_code\_event\_sparse\_inv/test.cc

# 16.49 SimulationSynapsePolicyDense Class Reference

```
#include <simulation_synapse_policy_dense.h>
```

Inheritance diagram for SimulationSynapsePolicyDense:



## **Public Member Functions**

- void Init ()
- template < typename UpdateFn, typename StepGeNNFn >
  float Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn)

#### **Protected Member Functions**

- float \* GetTheW (unsigned int delay) const
- void SetTheW (unsigned int i, unsigned int j, float value)

#### 16.49.1 Member Function Documentation

```
16.49.1.1 float* SimulationSynapsePolicyDense::GetTheW (unsigned int delay) const [inline], [protected]
```

- 16.49.1.2 void SimulationSynapsePolicyDense::Init() [inline]
- **16.49.1.3 void SimulationSynapsePolicyDense::SetTheW ( unsigned int** *i***, unsigned int** *j***, float** *value* **)** [inline], [protected]
- 16.49.1.4 template < typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicyDense::Simulate ( UpdateFn updateFn, StepGeNNFn stepGeNNFn ) [inline]

The documentation for this class was generated from the following file:

· simulation\_synapse\_policy\_dense.h

# 16.50 SimulationSynapsePolicyNone Class Reference

```
#include <simulation_synapse_policy_none.h>
```

#### **Public Member Functions**

- void Init ()
- template<typename UpdateFn, typename StepGeNNFn>
  float Simulate (UpdateFn updateFn, StepGeNNFn) stepGeNNFn)

## 16.50.1 Member Function Documentation

- 16.50.1.1 void SimulationSynapsePolicyNone::Init() [inline]
- 16.50.1.2 template < typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicyNone::Simulate ( UpdateFn updateFn, StepGeNNFn stepGeNNFn ) [inline]

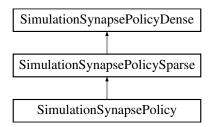
The documentation for this class was generated from the following file:

• simulation\_synapse\_policy\_none.h

# 16.51 SimulationSynapsePolicySparse Class Reference

```
#include <simulation_synapse_policy_sparse.h>
```

Inheritance diagram for SimulationSynapsePolicySparse:



#### **Public Member Functions**

- void Init ()
- template < typename UpdateFn , typename StepGeNNFn > float Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn)

#### Additional Inherited Members

```
16.51.1 Member Function Documentation
```

```
16.51.1.1 void SimulationSynapsePolicySparse::Init() [inline]
```

16.51.1.2 template < typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicySparse::Simulate ( UpdateFn updateFn, StepGeNNFn stepGeNNFn ) [inline]

The documentation for this class was generated from the following file:

· simulation\_synapse\_policy\_sparse.h

## 16.52 SimulationTest Class Reference

```
#include <simulation_test.h>
```

Inheritance diagram for SimulationTest:



# **Protected Member Functions**

- virtual void SetUp ()
- virtual void TearDown ()
- virtual void Init ()=0
- void StepGeNN ()

#### 16.52.1 Member Function Documentation

```
16.52.1.1 virtual void SimulationTest::Init() [protected], [pure virtual]
```

Implemented in SimulationTestVars< NeuronPolicy, SynapsePolicy >, SimTest, SimTest, SimTest, SimTest, and SimTest.

```
16.52.1.2 virtual void SimulationTest::SetUp() [inline], [protected], [virtual]
```

16.52.1.3 void SimulationTest::StepGeNN( ) [inline], [protected]

**16.52.1.4 virtual void SimulationTest::TearDown()** [inline],[protected],[virtual]

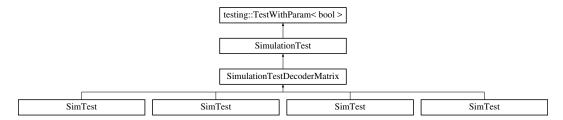
The documentation for this class was generated from the following file:

simulation\_test.h

## 16.53 SimulationTestDecoderMatrix Class Reference

#include <simulation\_test\_decoder\_matrix.h>

Inheritance diagram for SimulationTestDecoderMatrix:



#### **Public Member Functions**

• bool Simulate ()

#### **Additional Inherited Members**

16.53.1 Member Function Documentation

16.53.1.1 bool SimulationTestDecoderMatrix::Simulate() [inline]

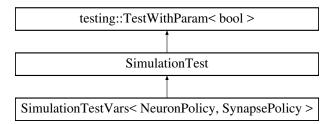
The documentation for this class was generated from the following file:

• simulation\_test\_decoder\_matrix.h

# 16.54 SimulationTestVars < NeuronPolicy, SynapsePolicy > Class Template Reference

#include <simulation\_test\_vars.h>

Inheritance diagram for SimulationTestVars< NeuronPolicy, SynapsePolicy >:



## **Protected Member Functions**

- virtual void Init ()
- template<typename UpdateFn >
  float Simulate (UpdateFn update)

# 16.54.1 Member Function Documentation

16.54.1.1 template<typename NeuronPolicy, typename SynapsePolicy > virtual void SimulationTestVars< NeuronPolicy, SynapsePolicy >::Init() [inline], [protected], [virtual]

Implements SimulationTest.

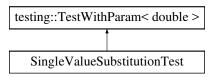
```
16.54.1.2 template < typename NeuronPolicy , typename SynapsePolicy > template < typename UpdateFn > float SimulationTestVars < NeuronPolicy, SynapsePolicy >::Simulate ( UpdateFn update ) [inline], [protected]
```

The documentation for this class was generated from the following file:

• simulation\_test\_vars.h

# 16.55 SingleValueSubstitutionTest Class Reference

Inheritance diagram for SingleValueSubstitutionTest:



#### **Protected Member Functions**

- virtual void SetUp ()
- const std::string & GetCode () const

## 16.55.1 Member Function Documentation

```
16.55.1.1 const std::string& SingleValueSubstitutionTest::GetCode( )const [inline], [protected]
```

16.55.1.2 virtual void SingleValueSubstitutionTest::SetUp() [inline], [protected], [virtual]

The documentation for this class was generated from the following file:

• tests/unit/codeGenUtils.cc

# 16.56 SparseProjection Struct Reference

class (struct) for defining a spars connectivity projection

```
#include <sparseProjection.h>
```

#### **Public Attributes**

- unsigned int \* indlnG
- unsigned int \* ind
- unsigned int \* preInd
- unsigned int \* revIndInG
- unsigned int \* revInd
- unsigned int \* remap
- unsigned int connN

#### 16.56.1 Detailed Description

class (struct) for defining a spars connectivity projection

## 16.56.2 Member Data Documentation

```
16.56.2.1 unsigned int SparseProjection::connN

16.56.2.2 unsigned int* SparseProjection::ind

16.56.2.3 unsigned int* SparseProjection::indlnG

16.56.2.4 unsigned int* SparseProjection::preInd

16.56.2.5 unsigned int* SparseProjection::remap

16.56.2.6 unsigned int* SparseProjection::revInd
```

16.56.2.7 unsigned int \* SparseProjection::revIndInG

The documentation for this struct was generated from the following file:

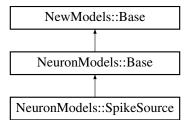
· sparseProjection.h

# 16.57 NeuronModels::SpikeSource Class Reference

Empty neuron which allows setting spikes from external sources.

```
#include <newNeuronModels.h>
```

Inheritance diagram for NeuronModels::SpikeSource:



# **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 0 > VarValues

# **Public Member Functions**

virtual std::string getThresholdConditionCode () const
 Gets code which defines the condition for a true spike in the described neuron model.

## **Static Public Member Functions**

static const NeuronModels::SpikeSource \* getInstance ()

## 16.57.1 Detailed Description

Empty neuron which allows setting spikes from external sources.

This model does not contain any update code and can be used to implement the equivalent of a SpikeGenerator Group in Brian or a SpikeSourceArray in PyNN.

- 16.57.2 Member Typedef Documentation
- 16.57.2.1 typedef NewModels::ValueBase< 0 > NeuronModels::SpikeSource::ParamValues
- 16.57.2.2 typedef NewModels::ValueBase< 0 > NeuronModels::SpikeSource::VarValues
- 16.57.3 Member Function Documentation
- 16.57.3.1 static const NeuronModels::SpikeSource\* NeuronModels::SpikeSource::getInstance( ) [inline], [static]
- **16.57.3.2** virtual std::string NeuronModels::SpikeSource::getThresholdConditionCode ( ) const [inline], [virtual]

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

The documentation for this class was generated from the following file:

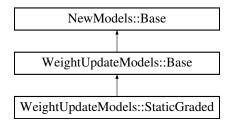
newNeuronModels.h

# 16.58 WeightUpdateModels::StaticGraded Class Reference

Graded-potential, static synapse.

#include <newWeightUpdateModels.h>

Inheritance diagram for WeightUpdateModels::StaticGraded:



# **Public Types**

- typedef NewModels::ValueBase< 2 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

## **Public Member Functions**

- · virtual StringVec getParamNames () const
  - Gets names of of (independent) model parameters.
- virtual StringPairVec getVars () const
  - Gets names and types (as strings) of model variables.
- virtual std::string getEventCode () const
  - Gets code run when events (all the instances where event threshold condition is met) are received.
- virtual std::string getEventThresholdConditionCode () const
  - Gets codes to test for events.

**Static Public Member Functions** 

static const StaticGraded \* getInstance ()

## 16.58.1 Detailed Description

Graded-potential, static synapse.

In a graded synapse, the conductance is updated gradually with the rule:

$$gSyn = g * tanh((V - E_{pre})/V_{slope})$$

whenever the membrane potential V is larger than the threshold  $E_{\it pre}$ . The model has 1 variable:

• g: conductance of scalar type

The parameters are:

- Epre: Presynaptic threshold potential
- Vslope: Activation slope of graded release

#### event code is:

```
 \begin{tabular}{ll} $(addtoinSyn) = $(g) * tanh(($(V_pre)-($(Epre)))*DT*2/$(Vslope)); \\ $(updatelinsyn); \\ \end{tabular}
```

event threshold condition code is:

```
$(V_pre) > $(Epre)
```

Note

The pre-synaptic variables are referenced with the suffix \_pre in synapse related code such as an the event threshold test. Users can also access post-synaptic neuron variables using the suffix \_post.

- 16.58.2 Member Typedef Documentation
- 16.58.2.1 typedef NewModels::ValueBase< 2 > WeightUpdateModels::StaticGraded::ParamValues
- 16.58.2.2 typedef NewModels::ValueBase< 1 > WeightUpdateModels::StaticGraded::VarValues
- 16.58.3 Member Function Documentation
- 16.58.3.1 virtual std::string WeightUpdateModels::StaticGraded::getEventCode() const [inline], [virtual]

Gets code run when events (all the instances where event threshold condition is met) are received.

Reimplemented from WeightUpdateModels::Base.

**16.58.3.2** virtual std::string WeightUpdateModels::StaticGraded::getEventThresholdConditionCode( ) const [inline], [virtual]

Gets codes to test for events.

Reimplemented from WeightUpdateModels::Base.

```
16.58.3.3 static const StaticGraded* WeightUpdateModels::StaticGraded::getInstance() [inline], [static]
```

16.58.3.4 virtual String Vec Weight Update Models::Static Graded::getParamNames() const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.58.3.5 virtual StringPairVec WeightUpdateModels::StaticGraded::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

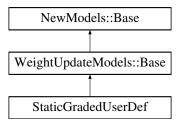
Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

newWeightUpdateModels.h

# 16.59 StaticGradedUserDef Class Reference

Inheritance diagram for StaticGradedUserDef:



### **Public Types**

- typedef NewModels::ValueBase< 2 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

#### **Public Member Functions**

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

### **Static Public Member Functions**

static const StaticGradedUserDef \* getInstance ()

16.59.1 Member Typedef Documentation

16.59.1.1 typedef NewModels::ValueBase< 2 > StaticGradedUserDef::ParamValues

 $16.59.1.2 \quad type def \ New Models:: Value Base < 1 > Static Graded User Def:: Var Values$ 

16.59.2 Member Function Documentation

16.59.2.1 virtual std::string StaticGradedUserDef::getEventCode( ) const [inline], [virtual]

Gets code run when events (all the instances where event threshold condition is met) are received.

16.59.2.2 virtual std::string StaticGradedUserDef::getEventThresholdConditionCode( ) const [inline], [virtual]

Gets codes to test for events.

Reimplemented from WeightUpdateModels::Base.

16.59.2.3 static const StaticGradedUserDef\* StaticGradedUserDef::getInstance() [inline], [static]

16.59.2.4 virtual String Vec StaticGradedUserDef::getParamNames() const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.59.2.5 virtual StringPairVec StaticGradedUserDef::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

MBody\_userdef.cc

### 16.60 WeightUpdateModels::StaticPulse Class Reference

Pulse-coupled, static synapse.

#include <newWeightUpdateModels.h>

Inheritance diagram for WeightUpdateModels::StaticPulse:



### **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

### **Public Member Functions**

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

# Static Public Member Functions

• static const StaticPulse \* getInstance ()

### 16.60.1 Detailed Description

Pulse-coupled, static synapse.

No learning rule is applied to the synapse and for each pre-synaptic spikes, the synaptic conductances are simply added to the postsynaptic input variable. The model has 1 variable:

• g - conductance of scalar type and no other parameters.

sim code is:

```
" (addtoinSyn) = (g); \n\ (updatelinsyn); \n"
```

#### 16.60.2 Member Typedef Documentation

```
16.60.2.1 typedef NewModels::ValueBase< 0 > WeightUpdateModels::StaticPulse::ParamValues
```

```
16.60.2.2 typedef NewModels::ValueBase< 1 > WeightUpdateModels::StaticPulse::VarValues
```

16.60.3 Member Function Documentation

```
16.60.3.1 static const StaticPulse* WeightUpdateModels::StaticPulse::getInstance() [inline], [static]
```

16.60.3.2 virtual std::string WeightUpdateModels::StaticPulse::getSimCode( ) const [inline], [virtual]

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.60.3.3 virtual StringPairVec WeightUpdateModels::StaticPulse::getVars() const [inline],[virtual]
```

Gets names and types (as strings) of model variables.

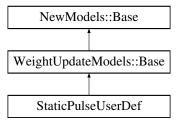
Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• newWeightUpdateModels.h

### 16.61 StaticPulseUserDef Class Reference

Inheritance diagram for StaticPulseUserDef:



# **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

#### **Public Member Functions**

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

· virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

# **Static Public Member Functions**

• static const StaticPulseUserDef \* getInstance ()

# 16.61.1 Member Typedef Documentation

```
16.61.1.1 typedef NewModels::ValueBase< 0 > StaticPulseUserDef::ParamValues
```

```
16.61.1.2 typedef NewModels::ValueBase< 1 > StaticPulseUserDef::VarValues
```

16.61.2 Member Function Documentation

```
16.61.2.1 static const StaticPulseUserDef* StaticPulseUserDef::getInstance( ) [inline], [static]
```

```
16.61.2.2 virtual std::string StaticPulseUserDef::getSimCode() const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.61.2.3 virtual StringPairVec StaticPulseUserDef::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• MBody\_userdef.cc

# 16.62 stdRG Class Reference

```
#include <randomGen.h>
```

# **Public Member Functions**

• stdRG()

Constructor of the standard random number generator class without explicit seed.

stdRG (unsigned int)

Constructor of the standard random number generator class with explicit seed.

- ~stdRG ()
- double n ()

Method to generate a uniform random number.

• unsigned long nlong ()

```
16.62.1 Constructor & Destructor Documentation
```

```
16.62.1.1 stdRG::stdRG() [explicit]
```

Constructor of the standard random number generator class without explicit seed.

The seed is taken from teh internal clock of the computer.

```
16.62.1.2 stdRG::stdRG ( unsigned int seed )
```

Constructor of the standard random number generator class with explicit seed.

The seed is an arbitrary unsigned int

```
16.62.1.3 stdRG::∼stdRG( ) [inline]
16.62.2 Member Function Documentation
16.62.2.1 double stdRG::n( )
```

Method to generate a uniform random number.

The moethod is a wrapper for the C function rand() and returns a pseudo random number in the interval [0,1[

```
16.62.2.2 unsigned long stdRG::nlong ( )
```

The documentation for this class was generated from the following files:

- · randomGen.h
- · randomGen.cc

# 16.63 stopWatch Struct Reference

```
#include <hr_time.h>
```

#### **Public Attributes**

- timeval start
- · timeval stop

# 16.63.1 Member Data Documentation

```
16.63.1.1 timeval stopWatch::start
```

16.63.1.2 timeval stopWatch::stop

The documentation for this struct was generated from the following file:

· hr time.h

# 16.64 SynapseGroup Class Reference

```
#include <synapseGroup.h>
```

# **Public Types**

enum SpanType { SpanType::POSTSYNAPTIC, SpanType::PRESYNAPTIC }

#### **Public Member Functions**

- SynapseGroup (const std::string name, SynapseMatrixType matrixType, unsigned int delaySteps, const WeightUpdateModels::Base \*wu, const std::vector< double > &wuParams, const std::vector< double > &wuInitVals, const PostsynapticModels::Base \*ps, const std::vector< double > &psParams, const std::vector< double > &psPa
- NeuronGroup \* getSrcNeuronGroup ()
- NeuronGroup \* getTrgNeuronGroup ()
- void setTrueSpikeRequired (bool req)
- void setSpikeEventRequired (bool req)
- void setEventThresholdReTestRequired (bool req)
- void setWUVarZeroCopyEnabled (const std::string &varName, bool enabled)
- void setPSVarZeroCopyEnabled (const std::string &varName, bool enabled)
- · void setClusterIndex (int hostID, int deviceID)
- void setMaxConnections (unsigned int maxConnections)
- void setSpanType (SpanType spanType)
- · void initDerivedParams (double dt)
- · void calcKernelSizes (unsigned int blockSize, unsigned int &paddedKernelIDStart)
- std::pair< unsigned int, unsigned int > getPaddedKernelIDRange () const
- const std::string & getName () const
- SpanType getSpanType () const
- unsigned int getDelaySteps () const
- unsigned int getMaxConnections () const
- SynapseMatrixType getMatrixType () const
- unsigned int getPaddedDynKernelSize (unsigned int blockSize) const
- unsigned int getPaddedPostLearnKernelSize (unsigned int blockSize) const
- const NeuronGroup \* getSrcNeuronGroup () const
- const NeuronGroup \* getTrgNeuronGroup () const
- · bool isTrueSpikeRequired () const
- bool isSpikeEventRequired () const
- bool isEventThresholdReTestRequired () const
- const WeightUpdateModels::Base \* getWUModel () const
- const std::vector< double > & getWUParams () const
- const std::vector< double > & getWUDerivedParams () const
- const std::vector< double > & getWUInitVals () const
- const PostsynapticModels::Base \* getPSModel () const
- const std::vector< double > & getPSParams () const
- const std::vector< double > & getPSDerivedParams () const
- const std::vector< double > & getPSInitVals () const
- bool isZeroCopyEnabled () const
- bool isWUVarZeroCopyEnabled (const std::string &var) const
- bool isPSVarZeroCopyEnabled (const std::string &var) const

Is this synapse group too large to use shared memory for combining postsynaptic output.

- bool isPSAtomicAddRequired (unsigned int blockSize) const
- void addExtraGlobalSynapseParams (std::map< string, string > &kernelParameters) const
- void addExtraGlobalNeuronParams (std::map< string, string > &kernelParameters) const
- std::string getOffsetPre () const
- std::string getOffsetPost (const std::string &devPrefix) const

#### 16.64.1 Member Enumeration Documentation

**16.64.1.1 enum SynapseGroup::SpanType** [strong]

**Enumerator** 

POSTSYNAPTIC PRESYNAPTIC

```
16.64.2 Constructor & Destructor Documentation
```

```
16.64.2.1 SynapseGroup::SynapseGroup ( const std::string name, SynapseMatrixType matrixType, unsigned int delaySteps, const WeightUpdateModels::Base * wu, const std::vector< double > & wuParams, const std::vector< double > & wulnitVals, const PostsynapticModels::Base * ps, const std::vector< double > & psParams, const std::vector< double > & psInitVals, NeuronGroup * srcNeuronGroup, NeuronGroup * trgNeuronGroup) [inline]
```

# 16.64.3 Member Function Documentation

```
16.64.3.1 void SynapseGroup::addExtraGlobalNeuronParams ( std::map< string, string > & kernelParameters ) const
16.64.3.2 void SynapseGroup::addExtraGlobalSynapseParams ( std::map < string, string > & kernelParameters ) const
16.64.3.3 void SynapseGroup::calcKernelSizes ( unsigned int blockSize, unsigned int & paddedKernelIDStart )
16.64.3.4 unsigned int SynapseGroup::getDelaySteps ( ) const [inline]
16.64.3.5 SynapseMatrixType SynapseGroup::getMatrixType ( ) const [inline]
16.64.3.6 unsigned int SynapseGroup::getMaxConnections ( ) const [inline]
16.64.3.7 const std::string& SynapseGroup::getName() const [inline]
16.64.3.8 std::string SynapseGroup::getOffsetPost ( const std::string & devPrefix ) const
16.64.3.9 std::string SynapseGroup::getOffsetPre ( ) const
16.64.3.10 unsigned int SynapseGroup::getPaddedDynKernelSize ( unsigned int blockSize ) const
16.64.3.11 std::pair<unsigned int, unsigned int> SynapseGroup::getPaddedKernellDRange()const [inline]
16.64.3.12 unsigned int SynapseGroup::getPaddedPostLearnKernelSize ( unsigned int blockSize ) const
16.64.3.13 const std::vector<double>& SynapseGroup::getPSDerivedParams() const [inline]
16.64.3.14 const std::vector < double > & Synapse Group::getPSInitVals ( ) const [inline]
16.64.3.15 const PostsynapticModels::Base* SynapseGroup::getPSModel( ) const [inline]
16.64.3.16 const std::vector < double > & Synapse Group::getPSParams ( ) const [inline]
16.64.3.17 SpanType SynapseGroup::getSpanType() const [inline]
16.64.3.18 NeuronGroup * SynapseGroup::getSrcNeuronGroup() [inline]
16.64.3.19 const NeuronGroup* SynapseGroup::getSrcNeuronGroup( ) const [inline]
16.64.3.20 NeuronGroup * SynapseGroup::getTrgNeuronGroup( ) [inline]
16.64.3.21 const NeuronGroup* SynapseGroup::getTrgNeuronGroup( ) const [inline]
16.64.3.22 const std::vector < double > & Synapse Group::getWUDerived Params ( ) const [inline]
16.64.3.23 const std::vector<double>& SynapseGroup::getWUInitVals ( ) const [inline]
16.64.3.24 const WeightUpdateModels::Base*SynapseGroup::getWUModel()const [inline]
```

16.64.3.25 const std::vector<double>& SynapseGroup::getWUParams( ) const [inline]

16.64.3.26 void SynapseGroup::initDerivedParams ( double dt )

```
bool SynapseGroup::isEventThresholdReTestRequired( ) const [inline]
16.64.3.28
          bool SynapseGroup::isPSAtomicAddRequired ( unsigned int blockSize ) const
16.64.3.29 bool SynapseGroup::isPSVarZeroCopyEnabled ( const std::string & var ) const
Is this synapse group too large to use shared memory for combining postsynaptic output.
          bool SynapseGroup::isSpikeEventRequired( ) const [inline]
16.64.3.30
16.64.3.31 bool SynapseGroup::isTrueSpikeRequired ( ) const [inline]
16.64.3.32 bool SynapseGroup::isWUVarZeroCopyEnabled ( const std::string & var ) const
16.64.3.33 bool SynapseGroup::isZeroCopyEnabled ( ) const
16.64.3.34 void SynapseGroup::setClusterIndex (int hostID, int deviceID) [inline]
16.64.3.35 void SynapseGroup::setEventThresholdReTestRequired ( bool req ) [inline]
Function to enable the use of zero-copied memory for a particular weight update model state variable: May improve
IO performance at the expense of kernel performance
16.64.3.36 void SynapseGroup::setMaxConnections (unsigned int maxConnections)
16.64.3.37 void SynapseGroup::setPSVarZeroCopyEnabled ( const std::string & varName, bool enabled )
16.64.3.38 void SynapseGroup::setSpanType ( SpanType spanType )
16.64.3.39 void SynapseGroup::setSpikeEventRequired (bool req) [inline]
16.64.3.40 void SynapseGroup::setTrueSpikeRequired ( bool req ) [inline]
16.64.3.41 void SynapseGroup::setWUVarZeroCopyEnabled ( const std::string & varName, bool enabled )
```

Function to enable the use zero-copied memory for a particular postsynaptic model state variable May improve IO performance at the expense of kernel performance

The documentation for this class was generated from the following files:

- synapseGroup.h
- synapseGroup.cc

### 16.65 SynDelay Class Reference

```
#include <SynDelaySim.h>
```

# **Public Member Functions**

- SynDelay (bool usingGPU)
- ∼SynDelay ()
- void run (float t)

### 16.65.1 Constructor & Destructor Documentation

```
16.65.1.1 SynDelay::SynDelay ( bool usingGPU )
```

16.65.1.2 SynDelay:: $\sim$ SynDelay ( )

#### 16.65.2 Member Function Documentation

# 16.65.2.1 void SynDelay::run ( float t )

The documentation for this class was generated from the following files:

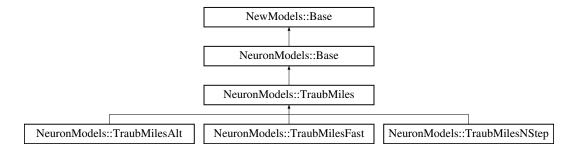
- SynDelaySim.h
- SynDelaySim.cc

#### 16.66 NeuronModels::TraubMiles Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

```
#include <newNeuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMiles:



# **Public Types**

- typedef NewModels::ValueBase< 7 > ParamValues
- typedef NewModels::ValueBase< 4 > VarValues

### **Public Member Functions**

• virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

• virtual std::string getThresholdConditionCode () const

Gets code which defines the condition for a true spike in the described neuron model.

• virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

# **Static Public Member Functions**

static const NeuronModels::TraubMiles \* getInstance ()

# 16.66.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

This conductance based model has been taken from [5] and can be described by the equations:

$$C\frac{dV}{dt} = -I_{Na} - I_K - I_{leak} - I_M - I_{i,DC} - I_{i,syn} - I_i,$$

$$I_{Na}(t) = g_{Na}m_i(t)^3 h_i(t)(V_i(t) - E_{Na})$$

$$I_K(t) = g_K n_i(t)^4 (V_i(t) - E_K)$$

$$\frac{dy(t)}{dt} = \alpha_y(V(t))(1 - y(t)) - \beta_y(V(t))y(t),$$

where  $y_i = m, h, n$ , and

$$\begin{array}{lcl}
\alpha_n & = & 0.032(-50-V)/\big(\exp((-50-V)/5)-1\big) \\
\beta_n & = & 0.5\exp((-55-V)/40) \\
\alpha_m & = & 0.32(-52-V)/\big(\exp((-52-V)/4)-1\big) \\
\beta_m & = & 0.28(25+V)/\big(\exp((25+V)/5)-1\big) \\
\alpha_h & = & 0.128\exp((-48-V)/18) \\
\beta_h & = & 4/\big(\exp((-25-V)/5)+1\big).
\end{array}$$

and typical parameters are C=0.143 nF,  $g_{\rm leak}=0.02672~\mu$ S,  $E_{\rm leak}=-63.563$  mV,  $g_{\rm Na}=7.15~\mu$ S,  $E_{\rm Na}=50$  mV,  $g_{\rm K}=1.43~\mu$ S,  $E_{\rm K}=-95$  mV.

It has 4 variables:

- V membrane potential E
- m probability for Na channel activation m
- · h probability for not Na channel blocking h
- n probability for K channel activation n

and 7 parameters:

- gNa Na conductance in 1/(mOhms \* cm<sup>2</sup>)
- ENa Na equi potential in mV
- gK K conductance in 1/(mOhms \* cm<sup>2</sup>)
- EK K equi potential in mV
- gl Leak conductance in  $1/(mOhms * cm^{^{^{^{^{^{}}}}}}2)$
- El Leak equi potential in mV
- Cmem Membrane capacity density in muF/cm<sup>2</sup>

Note

Internally, the ordinary differential equations defining the model are integrated with a linear Euler algorithm and GeNN integrates 25 internal time steps for each neuron for each network time step. I.e., if the network is simulated at DT = 0.1 ms, then the neurons are integrated with a linear Euler algorithm with 1DT = 0.004 ms. This variant uses IF statements to check for a value at which a singularity would be hit. If so, value calculated by L'Hospital rule is used.

```
16.66.2 Member Typedef Documentation
```

```
16.66.2.1 typedef NewModels::ValueBase< 7 > NeuronModels::TraubMiles::ParamValues
```

16.66.2.2 typedef NewModels::ValueBase< 4 > NeuronModels::TraubMiles::VarValues

16.66.3 Member Function Documentation

```
16.66.3.1 static const NeuronModels::TraubMiles* NeuronModels::TraubMiles::getInstance( ) [inline], [static]
```

```
16.66.3.2 virtual StringVec NeuronModels::TraubMiles::getParamNames() const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

Reimplemented in NeuronModels::TraubMilesNStep.

```
16.66.3.3 virtual std::string NeuronModels::TraubMiles::getSimCode()const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::Base.

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMilesAlt, and NeuronModels::Traub← MilesFast.

```
16.66.3.4 virtual std::string NeuronModels::TraubMiles::getThresholdConditionCode( ) const [inline], [virtual]
```

Gets code which defines the condition for a true spike in the described neuron model.

This evaluates to a bool (e.g. "V > 20").

Reimplemented from NeuronModels::Base.

```
16.66.3.5 virtual StringPairVec NeuronModels::TraubMiles::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

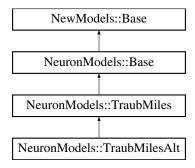
• newNeuronModels.h

# 16.67 NeuronModels::TraubMilesAlt Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

```
#include <newNeuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMilesAlt:



### **Public Types**

- typedef NewModels::ValueBase< 7 > ParamValues
- typedef NewModels::ValueBase< 4 > VarValues

#### **Public Member Functions**

virtual std::string getSimCode () const

Gets the code that defines the execution of one timestep of integration of the neuron model.

#### Static Public Member Functions

static const NeuronModels::TraubMilesAlt \* getInstance ()

### 16.67.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

Using a workaround to avoid singularity: adding the munimum numerical value of the floating point precision used.

- 16.67.2 Member Typedef Documentation
- 16.67.2.1 typedef NewModels::ValueBase< 7 > NeuronModels::TraubMilesAlt::ParamValues
- 16.67.2.2 typedef NewModels::ValueBase< 4 > NeuronModels::TraubMilesAlt::VarValues
- 16.67.3 Member Function Documentation
- 16.67.3.1 static const NeuronModels::TraubMilesAlt\* NeuronModels::TraubMilesAlt::getInstance() [inline], [static]
- 16.67.3.2 virtual std::string NeuronModels::TraubMilesAlt::getSimCode( )const [inline], [virtual]

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain , if it is to receive input.

Reimplemented from NeuronModels::TraubMiles.

The documentation for this class was generated from the following file:

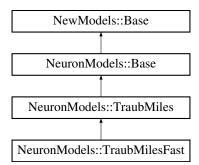
• newNeuronModels.h

### 16.68 NeuronModels::TraubMilesFast Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm: Original fast implementation, using 25 inner iterations.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::TraubMilesFast:



### **Public Types**

- typedef NewModels::ValueBase< 7 > ParamValues
- typedef NewModels::ValueBase< 4 > VarValues

#### **Public Member Functions**

virtual std::string getSimCode () const
 Gets the code that defines the execution of one timestep of integration of the neuron model.

# **Static Public Member Functions**

static const NeuronModels::TraubMilesFast \* getInstance ()

### 16.68.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm: Original fast implementation, using 25 inner iterations.

There are singularities in this model, which can be easily hit in float precision

- 16.68.2 Member Typedef Documentation
- 16.68.2.1 typedef NewModels::ValueBase< 7 > NeuronModels::TraubMilesFast::ParamValues
- 16.68.2.2 typedef NewModels::ValueBase< 4 > NeuronModels::TraubMilesFast::VarValues
- 16.68.3 Member Function Documentation
- 16.68.3.1 static const NeuronModels::TraubMilesFast\* NeuronModels::TraubMilesFast::getInstance() [inline], [static]
- 16.68.3.2 virtual std::string NeuronModels::TraubMilesFast::getSimCode( )const [inline], [virtual]

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::TraubMiles.

The documentation for this class was generated from the following file:

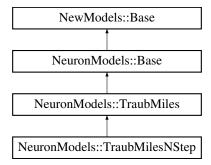
newNeuronModels.h

# 16.69 NeuronModels::TraubMilesNStep Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

#include <newNeuronModels.h>

Inheritance diagram for NeuronModels::TraubMilesNStep:



# **Public Types**

- typedef NewModels::ValueBase< 8 > ParamValues
- typedef NewModels::ValueBase< 4 > VarValues

#### **Public Member Functions**

- virtual std::string getSimCode () const
  - Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

# Static Public Member Functions

• static const NeuronModels::TraubMilesNStep \* getInstance ()

### 16.69.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

Same as standard TraubMiles model but number of inner loops can be set using a parameter

- 16.69.2 Member Typedef Documentation
- 16.69.2.1 typedef NewModels::ValueBase< 8 > NeuronModels::TraubMilesNStep::ParamValues
- $16.69.2.2 \quad typedef \ New Models:: Value Base < 4 > Neuron Models:: Traub Miles NS tep:: Var Values$
- 16.69.3 Member Function Documentation

```
16.69.3.1 static const NeuronModels::TraubMilesNStep* NeuronModels::TraubMilesNStep::getInstance ( ) [inline], [static]
```

16.69.3.2 virtual String Vec Neuron Models::Traub Miles N Step::get Param Names ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::TraubMiles.

```
16.69.3.3 virtual std::string NeuronModels::TraubMilesNStep::getSimCode( )const [inline], [virtual]
```

Gets the code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain, if it is to receive input.

Reimplemented from NeuronModels::TraubMiles.

The documentation for this class was generated from the following file:

• newNeuronModels.h

# 16.70 NewModels::ValueBase < NumValues > Class Template Reference

```
#include <newModels.h>
```

#### **Public Member Functions**

- template<typename... T>
   ValueBase (T &&...vals)
- std::vector< double > getValues () const

Gets values as a vector of doubles.

• double operator[] (size t pos) const

#### 16.70.1 Detailed Description

```
template < size_t NumValues > class NewModels::ValueBase < NumValues >
```

Wrapper to ensure at compile time that correct number of values are used when specifying the values of a model's parameters and initial state.

# 16.70.2 Constructor & Destructor Documentation

```
16.70.2.1 template < size_t NumValues > template < typename... T > NewModels::ValueBase < NumValues > ::ValueBase ( T &&... vals ) [inline]
```

#### 16.70.3 Member Function Documentation

```
16.70.3.1 template < size_t NumValues > std::vector < double > NewModels::ValueBase < NumValues > ::getValues ( ) const [inline]
```

Gets values as a vector of doubles.

```
16.70.3.2 template < size_t NumValues > double NewModels::ValueBase < NumValues > ::operator[]( size_t pos ) const [inline]
```

The documentation for this class was generated from the following file:

newModels.h

# 16.71 NewModels::ValueBase < 0 > Class Template Reference

```
#include <newModels.h>
```

#### **Public Member Functions**

- template<typename... T>
   ValueBase (T &&...vals)
- std::vector< double > getValues () const

Gets values as a vector of doubles.

#### 16.71.1 Detailed Description

```
template<> class NewModels::ValueBase< 0 >
```

Template specialisation of ValueBase to avoid compiler warnings in the case when a model requires no parameters or state variables

16.71.2 Constructor & Destructor Documentation

```
16.71.2.1 template<typename... T> NewModels::ValueBase<0>::ValueBase(T &&... vals) [inline]
```

16.71.3 Member Function Documentation

```
16.71.3.1 std::vector<double> NewModels::ValueBase< 0 >::getValues ( ) const [inline]
```

Gets values as a vector of doubles.

The documentation for this class was generated from the following file:

· newModels.h

# 16.72 weightUpdateModel Class Reference

Class to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

```
#include <synapseModels.h>
```

#### **Public Member Functions**

• weightUpdateModel ()

Constructor for weightUpdateModel objects.

∼weightUpdateModel ()

Destructor for weightUpdateModel objects.

#### **Public Attributes**

string simCode

Simulation code that is used for true spikes (only one time step after spike detection)

string simCodeEvnt

Simulation code that is used for spike events (all the instances where event threshold condition is met)

string simLearnPost

Simulation code which is used in the learnSynapsesPost kernel/function, where postsynaptic neuron spikes before the presynaptic neuron in the STDP window.

string evntThreshold

Simulation code for spike event detection.

string synapseDynamics

Simulation code for synapse dynamics independent of spike detection.

string simCode supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to simCode, evntThreshold and simCodeEvnt.

string simLearnPost\_supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

• string synapseDynamics\_supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

vector< string > varNames

Names of the variables in the postsynaptic model.

vector< string > varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

vector< string > pNames

Names of (independent) parameters of the model.

vector< string > dpNames

Names of dependent parameters of the model.

 $\hbox{-} \ \ vector < string > extraGlobalSynapseKernelParameters$ 

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

vector< string > extraGlobalSynapseKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

- · dpclass \* dps
- bool needPreSt

Whether presynaptic spike times are needed or not.

bool needPostSt

Whether postsynaptic spike times are needed or not.

# 16.72.1 Detailed Description

Class to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

16.72.2 Constructor & Destructor Documentation

16.72.2.1 weightUpdateModel::weightUpdateModel()

Constructor for weightUpdateModel objects.

16.72.2.2 weightUpdateModel::~weightUpdateModel()

Destructor for weightUpdateModel objects.

16.72.3 Member Data Documentation

16.72.3.1 vector<string> weightUpdateModel::dpNames

Names of dependent parameters of the model.

16.72.3.2 dpclass\* weightUpdateModel::dps

16.72.3.3 string weightUpdateModel::evntThreshold

Simulation code for spike event detection.

16.72.3.4 vector<string> weightUpdateModel::extraGlobalSynapseKernelParameters

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

16.72.3.5 vector<string> weightUpdateModel::extraGlobalSynapseKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

16.72.3.6 bool weightUpdateModel::needPostSt

Whether postsynaptic spike times are needed or not.

16.72.3.7 bool weightUpdateModel::needPreSt

Whether presynaptic spike times are needed or not.

16.72.3.8 vector<string> weightUpdateModel::pNames

Names of (independent) parameters of the model.

16.72.3.9 string weightUpdateModel::simCode

Simulation code that is used for true spikes (only one time step after spike detection)

16.72.3.10 string weightUpdateModel::simCode\_supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to simCode, evntThreshold and simCodeEvnt.

16.72.3.11 string weightUpdateModel::simCodeEvnt

Simulation code that is used for spike events (all the instances where event threshold condition is met)

16.72.3.12 string weightUpdateModel::simLearnPost

Simulation code which is used in the learnSynapsesPost kernel/function, where postsynaptic neuron spikes before the presynaptic neuron in the STDP window.

16.72.3.13 string weightUpdateModel::simLearnPost\_supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_device\_\_" to be available for both GPU and CPU versions.

16.72.3.14 string weightUpdateModel::synapseDynamics

Simulation code for synapse dynamics independent of spike detection.

16.72.3.15 string weightUpdateModel::synapseDynamics\_supportCode

Support code is made available within the synapse kernel definition file and is meant to contain user defined device functions that are used in the neuron codes. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions.

16.72.3.16 vector < string > weight Update Model::varNames

Names of the variables in the postsynaptic model.

16.72.3.17 vector < string > weightUpdateModel::varTypes

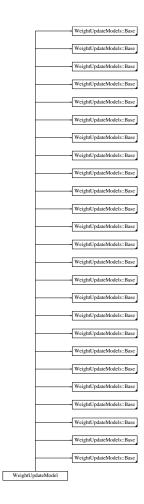
Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this class was generated from the following files:

- synapseModels.h
- synapseModels.cc

# 16.73 WeightUpdateModel Class Reference

Inheritance diagram for WeightUpdateModel:



### **Public Types**

- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- $\bullet \ \ typedef \ NewModels:: ValueBase < 0 > ParamValues \\$
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- $\bullet \ \ typedef \ NewModels:: ValueBase < 1 > VarValues \\$
- $\hbox{ typedef NewModels::} ValueBase < 0 > ParamValues \\$
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

- typedef NewModels::ValueBase< 0 > ParamValues
- $\bullet \ \ typedef \ NewModels:: ValueBase < 1 > VarValues \\$
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues
- typedef NewModels::ValueBase< 0 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

#### **Public Member Functions**

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

- virtual StringPairVec getExtraGlobalParams () const
- virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

· virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

• virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimCode () const

Gets simulation code run when 'true' spikes are received.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSynapseDynamicsCode () const

Gets code for synapse dynamics which are independent of spike detection.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

· virtual std::string getSimSupportCode () const

Gets support code to be made available within the synapse kernel/function.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual StringVec getParamNames () const

Gets names of of (independent) model parameters.

· virtual std::string getSimSupportCode () const

Gets support code to be made available within the synapse kernel/function.

virtual std::string getEventThresholdConditionCode () const

Gets codes to test for events.

virtual std::string getEventCode () const

Gets code run when events (all the instances where event threshold condition is met) are received.

• virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getLearnPostSupportCode () const

Gets support code to be made available within learnSynapsesPost kernel/function.

virtual std::string getLearnPostCode () const

Gets code to include in the learnSynapsesPost kernel/function.

virtual StringPairVec getVars () const

Gets names and types (as strings) of model variables.

virtual std::string getSimSupportCode () const

Gets support code to be made available within the synapse kernel/function.

- virtual std::string getSimCode () const
  - Gets simulation code run when 'true' spikes are received.
- virtual StringPairVec getVars () const
  - Gets names and types (as strings) of model variables.
- virtual std::string getSynapseDynamicsSuppportCode () const
  - Gets support code to be made available within the synapse dynamics kernel/function.
- virtual std::string getSynapseDynamicsCode () const
  - Gets code for synapse dynamics which are independent of spike detection.

#### **Static Public Member Functions**

- static const WeightUpdateModel \* getInstance ()
- Statio const Weight opaciomodel in getinotarios (
- static const WeightUpdateModel \* getInstance ()
   static const WeightUpdateModel \* getInstance ()
- 16.73.1 Member Typedef Documentation
- 16.73.1.1 typedef NewModels::ValueBase < 0 > WeightUpdateModel::ParamValues
- 16.73.1.2 typedef NewModels::ValueBase< 0 > WeightUpdateModel::ParamValues

- 16.73.1.6 typedef NewModels::ValueBase< 0 > WeightUpdateModel::ParamValues
- 16.73.1.7 typedef NewModels::ValueBase< 0 > WeightUpdateModel::ParamValues

16.73.1.9	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: ParamValues$
16.73.1.10	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.11	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.12	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: ParamValues$
16.73.1.13	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: ParamValues$
16.73.1.14	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.15	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.16	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.17	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: ParamValues$
16.73.1.18	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: ParamValues$
16.73.1.19	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.20	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.21	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.22	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.23	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.24	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.25	$typedef\ NewModels:: ValueBase < 0 > WeightUpdateModel:: ParamValues$
16.73.1.26	typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.27	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.28	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.29	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.30	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.31	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.32	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.33	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.34	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.35	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.36	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$
16.73.1.37	typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.38	typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.39	$typedef\ NewModels:: ValueBase < 1 > WeightUpdateModel:: VarValues$

```
typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.41
         typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.42 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.43 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.44 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.45 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.46 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.47 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.48 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.49 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.1.50 typedef NewModels::ValueBase< 1 > WeightUpdateModel::VarValues
16.73.2 Member Function Documentation
16.73.2.1 virtual std::string WeightUpdateModel::getEventCode( )const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.2 virtual std::string WeightUpdateModel::getEventCode( )const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.3 virtual std::string WeightUpdateModel::getEventCode() const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.4 virtual std::string WeightUpdateModel::getEventCode()const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.5 virtual std::string WeightUpdateModel::getEventCode() const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.6 virtual std::string WeightUpdateModel::getEventCode() const [inline], [virtual]
Gets code run when events (all the instances where event threshold condition is met) are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.7 virtual std::string WeightUpdateModel::getEventThresholdConditionCode() const [inline], [virtual]
Gets codes to test for events.
```

```
16.73.2.8 virtual std::string WeightUpdateModel::getEventThresholdConditionCode() const [inline], [virtual]
Gets codes to test for events.
Reimplemented from WeightUpdateModels::Base.
16.73.2.9 virtual std::string WeightUpdateModel::getEventThresholdConditionCode() const [inline],[virtual]
Gets codes to test for events.
Reimplemented from WeightUpdateModels::Base.
16.73.2.10 virtual std::string WeightUpdateModel::getEventThresholdConditionCode ( ) const [inline], [virtual]
Gets codes to test for events.
Reimplemented from WeightUpdateModels::Base.
16.73.2.11 virtual std::string WeightUpdateModel::getEventThresholdConditionCode ( ) const [inline], [virtual]
Gets codes to test for events.
Reimplemented from WeightUpdateModels::Base.
16.73.2.12 virtual std::string WeightUpdateModel::getEventThresholdConditionCode ( ) const [inline], [virtual]
Gets codes to test for events.
Reimplemented from WeightUpdateModels::Base.
16.73.2.13 virtual StringPairVec WeightUpdateModel::getExtraGlobalParams() const [inline], [virtual]
Gets names and types (as strings) of additional per-population parameters for the weight update model.
Reimplemented from WeightUpdateModels::Base.
16.73.2.14 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.15 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.16 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.17 static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline], [static]
16.73.2.18 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.19 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.20 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.21 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.22 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.23 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.24 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.25 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.26 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.27 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
```

```
16.73.2.28
         static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline], [static]
16.73.2.29
         static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline], [static]
16.73.2.30
         static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline], [static]
16.73.2.31
         static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline],[static]
16.73.2.32 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.33 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.34
         static const WeightUpdateModel* WeightUpdateModel::getInstance( ) [inline], [static]
16.73.2.35 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.36 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.37 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.38 static const WeightUpdateModel* WeightUpdateModel::getInstance() [inline], [static]
16.73.2.39 virtual std::string WeightUpdateModel::getLearnPostCode( )const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.40 virtual std::string WeightUpdateModel::getLearnPostCode() const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.41 virtual std::string WeightUpdateModel::getLearnPostCode( ) const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.42 virtual std::string WeightUpdateModel::getLearnPostCode ( ) const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.43 virtual std::string WeightUpdateModel::getLearnPostCode() const [inline], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

For examples when modelling STDP, this is where the effect of postsynaptic spikes which occur *after* presynaptic spikes are applied.

```
16.73.2.44 virtual std::string WeightUpdateModel::getLearnPostSupportCode( ) const [inline], [virtual]
Gets support code to be made available within learnSynapsesPost kernel/function.
Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; func-
tions should be declared as "__host__ __device__" to be available for both GPU and CPU versions.
Reimplemented from WeightUpdateModels::Base.
16.73.2.45 virtual StringVec WeightUpdateModel::getParamNames( )const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.73.2.46 virtual String Vec Weight Update Model::get Param Names ( ) const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.73.2.47 virtual String Vec Weight Update Model::get Param Names ( ) const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.73.2.48 virtual String Vec Weight Update Model::get Param Names ( ) const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.73.2.49 virtual String Vec Weight Update Model::get Param Names ( ) const [inline], [virtual]
Gets names of of (independent) model parameters.
Reimplemented from NewModels::Base.
16.73.2.50 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
Gets simulation code run when 'true' spikes are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.51 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
Gets simulation code run when 'true' spikes are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.52 virtual std::string WeightUpdateModel::getSimCode()const [inline], [virtual]
Gets simulation code run when 'true' spikes are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.53 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
Gets simulation code run when 'true' spikes are received.
Reimplemented from WeightUpdateModels::Base.
16.73.2.54 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
Gets simulation code run when 'true' spikes are received.
```

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.55 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.56 virtual std::string WeightUpdateModel::getSimCode()const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.57 virtual std::string WeightUpdateModel::getSimCode() const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.58 virtual std::string WeightUpdateModel::getSimCode()const [inline], [virtual]
```

Gets simulation code run when 'true' spikes are received.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.59 virtual std::string WeightUpdateModel::getSimSupportCode() const [inline], [virtual]
```

Gets support code to be made available within the synapse kernel/function.

This is intended to contain user defined device functions that are used in the weight update code. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to sim, event threshold and event code

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.60 virtual std::string WeightUpdateModel::getSimSupportCode() const [inline], [virtual]
```

Gets support code to be made available within the synapse kernel/function.

This is intended to contain user defined device functions that are used in the weight update code. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to sim, event threshold and event code

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.61 virtual std::string WeightUpdateModel::getSimSupportCode() const [inline], [virtual]
```

Gets support code to be made available within the synapse kernel/function.

This is intended to contain user defined device functions that are used in the weight update code. Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as "\_\_host\_\_ \_\_device\_\_" to be available for both GPU and CPU versions; note that this support code is available to sim, event threshold and event code

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.62 virtual std::string WeightUpdateModel::getSynapseDynamicsCode() const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

```
16.73.2.63 virtual std::string WeightUpdateModel::getSynapseDynamicsCode( ) const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.64 virtual std::string WeightUpdateModel::getSynapseDynamicsCode( )const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.65 virtual std::string WeightUpdateModel::getSynapseDynamicsCode( )const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.66 virtual std::string WeightUpdateModel::getSynapseDynamicsCode( )const [inline], [virtual]
```

Gets code for synapse dynamics which are independent of spike detection.

Reimplemented from WeightUpdateModels::Base.

Gets support code to be made available within the synapse dynamics kernel/function.

Preprocessor defines are also allowed if appropriately safeguarded against multiple definition by using ifndef; functions should be declared as " host device " to be available for both GPU and CPU versions.

Reimplemented from WeightUpdateModels::Base.

```
16.73.2.68 virtual StringPairVec WeightUpdateModel::getVars( )const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.73.2.69 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.73.2.70 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.73.2.71 virtual StringPairVec WeightUpdateModel::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.73.2.72 virtual StringPairVec WeightUpdateModel::getVars( ) const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

```
16.73.2.73 virtual StringPairVec WeightUpdateModel::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

```
Reimplemented from NewModels::Base.
16.73.2.74 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.75 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.76 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.77 virtual StringPairVec WeightUpdateModel::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.78 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.79 virtual StringPairVec WeightUpdateModel::getVars() const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.80 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.81 virtual StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.82 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.83 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
16.73.2.84 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]
Gets names and types (as strings) of model variables.
Reimplemented from NewModels::Base.
```

16.73.2.85 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.86 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.87 virtual StringPairVec WeightUpdateModel::getVars( )const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.88 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.89 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.90 virtual StringPairVec WeightUpdateModel::getVars( )const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.91 virtual StringPairVec WeightUpdateModel::getVars() const [inline], [virtual]

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

16.73.2.92 virtual String Pair Vec Weight Update Model::get Vars ( ) const [inline], [virtual]

Gets names and types (as strings) of model variables.

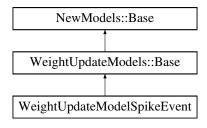
Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

· extra global params in sim code event sparse inv/model new.cc

# 16.74 WeightUpdateModelSpikeEvent Class Reference

Inheritance diagram for WeightUpdateModelSpikeEvent:



### **Public Types**

- typedef NewModels::ValueBase< 1 > ParamValues
- typedef NewModels::ValueBase< 1 > VarValues

#### **Public Member Functions**

- virtual std::string getEventCode () const
  - Gets code run when events (all the instances where event threshold condition is met) are received.
- virtual std::string getEventThresholdConditionCode () const
  - Gets codes to test for events.
- virtual StringVec getParamNames () const
  - Gets names of of (independent) model parameters.
- virtual StringPairVec getVars () const
  - Gets names and types (as strings) of model variables.

### **Static Public Member Functions**

- static const WeightUpdateModelSpikeEvent \* getInstance ()
- 16.74.1 Member Typedef Documentation
- 16.74.1.1 typedef NewModels::ValueBase< 1 > WeightUpdateModelSpikeEvent::ParamValues
- 16.74.1.2 typedef NewModels::ValueBase< 1 > WeightUpdateModelSpikeEvent::VarValues
- 16.74.2 Member Function Documentation
- **16.74.2.1** virtual std::string WeightUpdateModelSpikeEvent::getEventCode( ) const [inline], [virtual]

Gets code run when events (all the instances where event threshold condition is met) are received.

Reimplemented from WeightUpdateModels::Base.

```
16.74.2.2 virtual std::string WeightUpdateModelSpikeEvent::getEventThresholdConditionCode( ) const [inline], [virtual]
```

Gets codes to test for events.

Reimplemented from WeightUpdateModels::Base.

```
16.74.2.3 static const WeightUpdateModelSpikeEvent* WeightUpdateModelSpikeEvent::getInstance( ) [inline], [static]
```

```
16.74.2.4 virtual String Vec Weight Update Model Spike Event::get Param Names ( ) const [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

```
16.74.2.5 virtual StringPairVec WeightUpdateModelSpikeEvent::getVars() const [inline], [virtual]
```

Gets names and types (as strings) of model variables.

Reimplemented from NewModels::Base.

The documentation for this class was generated from the following file:

• Model\_Schmuker\_2014\_classifier.cc

17 File Documentation 215

# 17 File Documentation

- 17.1 00\_MainPage.dox File Reference
- 17.2 01\_Installation.dox File Reference
- 17.3 02\_Quickstart.dox File Reference
- 17.4 03\_Examples.dox File Reference
- 17.5 09\_ReleaseNotes.dox File Reference
- 17.6 10\_UserManual.dox File Reference
- 17.7 11 Tutorial.dox File Reference
- 17.8 12\_Tutorial.dox File Reference
- 17.9 13\_UserGuide.dox File Reference
- 17.10 14\_Credits.dox File Reference
- 17.11 classol\_sim.cc File Reference

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

```
#include "classol_sim.h"
```

# **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody1 model network.

#### 17.11.1 Detailed Description

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

17.11.2 Function Documentation

17.11.2.1 int main ( int argc, char \* argv[] )

This function is the entry point for running the simulation of the MBody1 model network.

#### 17.12 classol sim.cc File Reference

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody\_delayedSyn mushroom body model.

```
#include "classol_sim.h"
```

#### **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody\_delayedSyn model network.

### 17.12.1 Detailed Description

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody delayedSyn mushroom body model.

#### 17.12.2 Function Documentation

```
17.12.2.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the simulation of the MBody\_delayedSyn model network.

### 17.13 classol sim.cc File Reference

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

```
#include "classol_sim.h"
```

### **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody1 model network.

### 17.13.1 Detailed Description

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

### 17.13.2 Function Documentation

```
17.13.2.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the simulation of the MBody1 model network.

# 17.14 classol\_sim.cc File Reference

```
#include "classol_sim.h"
```

### **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody1 model network.

#### 17.14.1 Function Documentation

```
17.14.1.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the simulation of the MBody1 model network.

# 17.15 classol\_sim.cc File Reference

```
#include "classol_sim.h"
```

#### **Functions**

int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody1 model network.

#### 17.15.1 Function Documentation

```
17.15.1.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the simulation of the MBody1 model network.

# 17.16 classol\_sim.cc File Reference

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

```
#include "classol_sim.h"
```

# **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the simulation of the MBody1 model network.

# 17.16.1 Detailed Description

Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model.

# 17.16.2 Function Documentation

```
17.16.2.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the simulation of the MBody1 model network.

# 17.17 classol\_sim.h File Reference

Header file containing global variables and macros used in running the classol / MBody1 model.

```
#include <cassert>
#include "MBody1.cc"
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "map_classol.cc"
```

#### Macros

- #define MYRAND(Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);
- #define PATTERNNO 100
- #define T\_REPORT\_TME 10000.0
- #define SYN\_OUT\_TME 20000.0
- #define PAT TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL\_TME 5000.0

#### **Variables**

- scalar InputBaseRate = 2e-04
- int patSetTime
- · int patFireTime
- · CStopWatch timer

# 17.17.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody1 model.

### 17.17.2 Macro Definition Documentation

- 17.17.2.1 #define MYRAND( Y, X) Y = Y \* 1103515245 + 12345; <math>X = (Y >> 16);
- 17.17.2.2 #define PAT\_TIME 100.0
- 17.17.2.3 #define PATFTIME 1.5
- 17.17.2.4 #define PATTERNNO 100
- 17.17.2.5 #define SYN\_OUT\_TME 20000.0
- 17.17.2.6 #define T\_REPORT\_TME 10000.0
- 17.17.2.7 #define TOTAL\_TME 5000.0
- 17.17.3 Variable Documentation
- 17.17.3.1 scalar InputBaseRate = 2e-04
- 17.17.3.2 int patFireTime
- 17.17.3.3 int patSetTime
- 17.17.3.4 CStopWatch timer

# 17.18 classol\_sim.h File Reference

Header file containing global variables and macros used in running the classol / MBody\_delayedSyn model.

```
#include <cassert>
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "MBody_delayedSyn.cc"
#include "map_classol.cc"
```

#### Macros

- #define MYRAND(Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);
- #define DBG\_SIZE 10000
- #define PATTERNNO 100
- #define T\_REPORT\_TME 10000.0
- #define SYN\_OUT\_TME 20000.0
- #define PAT\_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL\_TME 5000.0

#### **Variables**

- scalar InputBaseRate = 2e-04
- int patSetTime
- · int patFireTime
- · CStopWatch timer

# 17.18.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody\_delayedSyn model.

### 17.18.2 Macro Definition Documentation

```
17.18.2.1 #define DBG_SIZE 10000

17.18.2.2 #define MYRAND( Y, X ) Y = Y * 1103515245 +12345; X= (Y >> 16);

17.18.2.3 #define PAT_TIME 100.0

17.18.2.4 #define PATFTIME 1.5

17.18.2.5 #define PATTERNNO 100

17.18.2.6 #define SYN_OUT_TME 20000.0

17.18.2.7 #define T_REPORT_TME 10000.0

17.18.2.8 #define TOTAL_TME 5000.0

17.18.3 Variable Documentation
```

17.18.3.1 scalar InputBaseRate = 2e-04

```
17.18.3.2 int patFireTime
```

17.18.3.3 int patSetTime

17.18.3.4 CStopWatch timer

# 17.19 classol\_sim.h File Reference

Header file containing global variables and macros used in running the classol / MBody individualID model.

```
#include <cassert>
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "MBody_individualID.cc"
#include "map_classol.cc"
```

#### Macros

- #define MYRAND(Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);
- #define DBG\_SIZE 10000
- #define PATTERNNO 100
- #define T REPORT TME 10000.0
- #define SYN\_OUT\_TME 20000.0
- #define PAT TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL\_TME 5000.0

### **Variables**

- scalar InputBaseRate = 2e-04
- · int patSetTime
- · int patFireTime
- · CStopWatch timer

### 17.19.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody\_individualID model.

## 17.19.2 Macro Definition Documentation

```
17.19.2.1 #define DBG_SIZE 10000
```

17.19.2.2 #define MYRAND( Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);

17.19.2.3 #define PAT\_TIME 100.0

17.19.2.4 #define PATFTIME 1.5

17.19.2.5 #define PATTERNNO 100

17.19.2.6 #define SYN\_OUT\_TME 20000.0

17.19.2.7 #define T\_REPORT\_TME 10000.0

```
17.20 classol_sim.h File Reference
17.19.2.8 #define TOTAL_TME 5000.0
17.19.3 Variable Documentation
17.19.3.1 scalar InputBaseRate = 2e-04
17.19.3.2 int patFireTime
17.19.3.3 int patSetTime
17.19.3.4 CStopWatch timer
17.20 classol sim.h File Reference
#include <cassert>
#include "MBody1.cc"
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "map_classol.cc"
Macros
   • #define MYRAND(Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);
   • #define PATTERNNO 100
   • #define T_REPORT_TME 10000.0
   #define SYN_OUT_TME 20000.0

    #define PAT_TIME 100.0

   • #define PATFTIME 1.5
   • #define TOTAL TME 1000000.0
```

# **Variables**

- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- · CStopWatch timer

### 17.20.1 Macro Definition Documentation

```
17.20.1.1 #define MYRAND( Y, X) Y = Y * 1103515245 + 12345; X = (Y >> 16);
17.20.1.2 #define PAT_TIME 100.0
17.20.1.3 #define PATFTIME 1.5
17.20.1.4 #define PATTERNNO 100
17.20.1.5 #define SYN_OUT_TME 20000.0
17.20.1.6 #define T_REPORT_TME 10000.0
```

17.20.1.7 #define TOTAL\_TME 1000000.0

```
17.20.2.1 scalar InputBaseRate = 2e-04
17.20.2.2 int patFireTime
17.20.2.3 int patSetTime
17.20.2.4 CStopWatch timer
17.21 classol_sim.h File Reference
#include <cassert>
#include "MBody1.cc"
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "map_classol.cc"
Macros
   • #define MYRAND(Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);
   • #define PATTERNNO 100
   • #define T_REPORT_TME 10000.0
   #define SYN_OUT_TME 20000.0
   • #define PAT_TIME 100.0
    • #define PATFTIME 1.5
    • #define TOTAL_TME 1000000.0
Variables
   • scalar InputBaseRate = 2e-04

    int patSetTime

    int patFireTime

    CStopWatch timer

17.21.1 Macro Definition Documentation
17.21.1.1 #define MYRAND( Y, X) Y = Y * 1103515245 + 12345; X = (Y >> 16);
17.21.1.2 #define PAT_TIME 100.0
17.21.1.3 #define PATFTIME 1.5
17.21.1.4 #define PATTERNNO 100
17.21.1.5 #define SYN_OUT_TME 20000.0
17.21.1.6 #define T_REPORT_TME 10000.0
17.21.1.7 #define TOTAL_TME 1000000.0
17.21.2 Variable Documentation
17.21.2.1 scalar InputBaseRate = 2e-04
```

17.21.2.2 int patFireTime

```
17.21.2.3 int patSetTime
```

# 17.21.2.4 CStopWatch timer

# 17.22 classol\_sim.h File Reference

Header file containing global variables and macros used in running the classol / MBody1 model.

```
#include <cassert>
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include <cuda_runtime.h>
#include "MBody_userdef.cc"
#include "map_classol.cc"
```

#### Macros

- #define MYRAND(Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);
- #define PATTERNNO 100
- #define T\_REPORT\_TME 10000.0
- #define SYN\_OUT\_TME 20000.0
- #define PAT\_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL\_TME 5000.0

### Variables

- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- · CStopWatch timer

### 17.22.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody1 model.

### 17.22.2 Macro Definition Documentation

```
17.22.2.1 #define MYRAND( Y, X ) Y = Y * 1103515245 +12345; X= (Y >> 16);
17.22.2.2 #define PAT_TIME 100.0
17.22.2.3 #define PATFTIME 1.5
17.22.2.4 #define PATTERNNO 100
17.22.2.5 #define SYN_OUT_TME 20000.0
17.22.2.6 #define T_REPORT_TME 10000.0
```

17.22.3 Variable Documentation

17.22.2.7 #define TOTAL\_TME 5000.0

```
17.22.3.1 scalar InputBaseRate = 2e-04
```

17.22.3.2 int patFireTime

17.22.3.3 int patSetTime

17.22.3.4 CStopWatch timer

### 17.23 codeGenUtils.cc File Reference

```
#include "codeGenUtils.h"
#include <regex>
#include "modelSpec.h"
#include "standardSubstitutions.h"
#include "utils.h"
```

#### Macros

#define REGEX OPERATIONAL

#### **Functions**

void substitute (string &s, const string &trg, const string &rep)

Tool for substituting strings in the neuron code strings or other templates.

string ensureFtype (const string &oldcode, const string &type)

This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

• void checkUnreplacedVariables (const string &code, const string &codeName)

This function checks for unknown variable definitions and returns a gennError if any are found.

void neuron\_substitutions\_in\_synaptic\_code (string &wCode, const SynapseGroup \*sg, const string &preldx, const string &devPrefix)

Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

### 17.23.1 Macro Definition Documentation

17.23.1.1 #define REGEX\_OPERATIONAL

17.23.2 Function Documentation

17.23.2.1 void checkUnreplacedVariables ( const string & code, const string & codeName )

This function checks for unknown variable definitions and returns a gennError if any are found.

17.23.2.2 string ensureFtype (const string & oldcode, const string & type)

This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

17.23.2.3 void neuron\_substitutions\_in\_synaptic\_code ( string & wCode, const SynapseGroup \* sg, const string & preldx, const string & devPrefix )

Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

#### **Parameters**

wCode	the code string to work on
preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)
devPrefix	device prefix, "dd_" for GPU, nothing for CPU

17.23.2.4 void substitute ( string & s, const string & trg, const string & rep )

Tool for substituting strings in the neuron code strings or other templates.

### 17.24 codeGenUtils.cc File Reference

```
#include <limits>
#include <tuple>
#include <cstdlib>
#include "gtest/gtest.h"
#include "codeGenUtils.h"
```

#### Classes

class SingleValueSubstitutionTest

### **Functions**

- TEST\_P (SingleValueSubstitutionTest, CorrectGeneratedValue)
- INSTANTIATE\_TEST\_CASE\_P (DoubleValues, SingleValueSubstitutionTest,::testing::Values(std::numeric
   —limits < double >::max(), 1.0,-1.0))

# 17.24.1 Function Documentation

```
17.24.1.1 INSTANTIATE_TEST_CASE_P ( DoubleValues , SingleValueSubstitutionTest , ::testing::Values(std::numeric_limits< double >::min(), std::numeric_limits< double >::max(), 1.0,-1.0) )
```

17.24.1.2 TEST\_P ( SingleValueSubstitutionTest , CorrectGeneratedValue )

# 17.25 codeGenUtils.h File Reference

```
#include <limits>
#include <string>
#include <sstream>
#include <vector>
```

#### Classes

class PairKeyConstIter < BaseIter >

# **Namespaces**

NeuronModels

#### **Functions**

- template<typename Baselter >
   PairKeyConstlter < Baselter > GetPairKeyConstlter (Baselter iter)
- void substitute (string &s, const string &trg, const string &rep)

Tool for substituting strings in the neuron code strings or other templates.

template<typename Namelter >
 void name\_substitutions (string &code, const string &prefix, Namelter namesBegin, Namelter namesEnd, const string &postfix="", const string &ext="")

This function performs a list of name substitutions for variables in code snippets.

- void name\_substitutions (string &code, const string &prefix, const vector< string > &names, const string &postfix="", const string &ext="")
- template<typename Namelter >
   void value\_substitutions (string &code, Namelter namesBegin, Namelter namesEnd, const vector< double
   > &values, const string &ext="")

This function performs a list of value substitutions for parameters in code snippets.

- void value\_substitutions (string &code, const vector< string > &names, const vector< double > &values, const string &ext="")
- string ensureFtype (const string &oldcode, const string &type)

This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

void checkUnreplacedVariables (const string &code, const string &codeName)

This function checks for unknown variable definitions and returns a gennError if any are found.

# 17.25.1 Function Documentation

17.25.1.1 void checkUnreplacedVariables ( const string & code, const string & codeName )

This function checks for unknown variable definitions and returns a gennError if any are found.

17.25.1.2 string ensureFtype (const string & oldcode, const string & type)

This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

```
17.25.1.3 template<typename Baselter > PairKeyConstlter<Baselter> GetPairKeyConstlter ( Baselter iter ) [inline]
```

17.25.1.4 template<typename Namelter > void name\_substitutions ( string & code, const string & prefix, Namelter namesBegin, Namelter namesEnd, const string & postfix = " ", const string & ext = " " ) [inline]

This function performs a list of name substitutions for variables in code snippets.

- 17.25.1.5 void name\_substitutions ( string & code, const string & prefix, const vector < string > & names, const string & postfix = "", const string & ext = "") [inline]
- 17.25.1.6 void substitute ( string & s, const string & trg, const string & rep )

Tool for substituting strings in the neuron code strings or other templates.

17.25.1.7 template<typename Namelter > void value\_substitutions ( string & code, Namelter namesBegin, Namelter namesEnd, const vector< double > & values, const string & ext = " " ) [inline]

This function performs a list of value substitutions for parameters in code snippets.

17.25.1.8 void value\_substitutions ( string & code, const vector < string > & names, const vector < double > & values, const string & ext = "" ) [inline]

# 17.26 CodeHelper.h File Reference

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <string>
#include <stream>
#include <vector>
```

#### Classes

class CodeHelper

#### Macros

- #define SAVEP(X) "(" << X << ")"</li>
- #define OB(X) hlp.openBrace(X)
- #define CB(X) hlp.closeBrace(X)
- #define ENDL hlp.endl()

# Variables

· CodeHelper hlp

```
17.26.1 Macro Definition Documentation
```

```
17.26.1.1 #define CB( X ) hlp.closeBrace(X)
```

17.26.1.2 #define ENDL hlp.endl()

17.26.1.3 #define OB( X ) hlp.openBrace(X)

17.26.1.4 #define SAVEP( X ) "(" << X << ")"

17.26.2 Variable Documentation

17.26.2.1 CodeHelper hlp

# 17.27 command\_line\_processing.h File Reference

This file contains some tools for parsing the argv array which contains the command line options.

```
#include <string>
```

#### **Functions**

- string toUpper (string s)
- string toLower (string s)
- int extract\_option (char \*op, string &option)

- int extract\_bool\_value (char \*op, unsigned int &val)
- int extract\_string\_value (char \*op, string &val)

### 17.27.1 Detailed Description

This file contains some tools for parsing the argy array which contains the command line options.

```
17.27.2 Function Documentation
```

```
17.27.2.1 int extract_bool_value ( char * op, unsigned int & val )
17.27.2.2 int extract_option ( char * op, string & option )
17.27.2.3 int extract_string_value ( char * op, string & val )
17.27.2.4 string toLower ( string s )
17.27.2.5 string toUpper ( string s )
17.28 dpclass.h File Reference
```

```
#include <vector>
```

### Classes

· class dpclass

# 17.29 experiment.cc File Reference

```
#include "experiment.h"
#include <time.h>
#include <algorithm>
#include <sys/types.h>
#include <sys/stat.h>
```

# Classes

• struct Parameter

## Macros

- #define RAND(Y, X) Y = Y \* 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767
- #define S\_ISDIR(mode) (((mode) & S\_IFMT) == S\_IFDIR)

# Typedefs

• typedef struct Parameter Parameter

#### **Functions**

- · bool directoryExists (string const &path)
- bool createDirectory (string path)
- float getAverage (vector< float > &v)
- float getStdDev (vector< float > &v, float avg)
- bool printTextFile (string path)
- string getUniqueRunId ()
- void outputRunParameters ()
- bool applyInputToClassifier (UINT recordingIdx, bool usePlasticity)
- bool vectorContains (vector< int > &vec, int lookingFor)
- void setDefaultParamValues ()
- int main (int argc, char \*argv[])

#### **Variables**

• Schmuker2014\_classifier classifier

```
17.29.1 Macro Definition Documentation
```

```
17.29.1.1 #define RAND( Y, X) Y = Y * 1103515245 + 12345; X = (unsigned int)(Y >> 16) & 32767
```

- 17.29.1.2 #define S\_ISDIR( *mode* ) (((mode) & S\_IFMT) == S\_IFDIR)
- 17.29.2 Typedef Documentation
- 17.29.2.1 typedef struct Parameter Parameter
- 17.29.3 Function Documentation
- 17.29.3.1 bool applyInputToClassifier ( UINT recordingIdx, bool usePlasticity )
- 17.29.3.2 bool createDirectory ( string path )
- 17.29.3.3 bool directoryExists ( string const & path )
- 17.29.3.4 float getAverage (vector < float > & v)
- 17.29.3.5 float getStdDev ( vector< float > & v, float avg )
- 17.29.3.6 string getUniqueRunId ( )
- 17.29.3.7 int main ( int argc, char \* argv[] )
- 17.29.3.8 void outputRunParameters ( )
- 17.29.3.9 bool printTextFile ( string path )
- 17.29.3.10 void setDefaultParamValues ( )
- 17.29.3.11 bool vectorContains ( vector< int > & vec, int lookingFor )
- 17.29.4 Variable Documentation
- 17.29.4.1 Schmuker2014\_classifier classifier

# 17.30 experiment.h File Reference

```
#include <cassert>
#include "hr_time.h"
#include "utils.h"
#include "Schmuker2014_classifier.cc"
```

#### Macros

- #define divi "/"
- #define D\_MAX\_RANDOM\_NUM 32767
- #define RECORDINGS DIR "recordings iris data"
- #define CACHE\_DIR "cached\_iris\_data"
- #define OUTPUT DIR "output iris"
- #define VR\_DATA\_FILENAME "VR-recordings-iris.data"
- #define DATASET\_NAME "Iris"
- #define TOTAL\_RECORDINGS 150
- #define N FOLDING 5
- #define RECORDING\_TIME\_MS 1000
- #define REPEAT\_LEARNING\_SET 2
- #define SPIKING\_ACTIVITY\_THRESHOLD\_HZ 5
- #define FLAG\_RUN\_ON\_CPU 1
- #define MAX\_FIRING\_RATE\_HZ 70
- #define MIN FIRING RATE HZ 20
- #define GLOBAL WEIGHT SCALING 1.0
- #define WEIGHT\_RN\_PN 0.5
- #define CONNECTIVITY RN PN 0.5
- #define WEIGHT\_WTA\_PN\_PN 0.01
- #define WEIGHT\_WTA\_AN\_AN 0.01
- #define CONNECTIVITY PN PN 0.5
- #define CONNECTIVITY\_AN\_AN 0.5
- #define CONNECTIVITY\_PN\_AN 0.5
- #define MIN WEIGHT PN AN 0.1
- #define MAX\_WEIGHT\_PN\_AN 0.4
- #define WEIGHT\_DELTA\_PN\_AN 0.04
- #define PLASTICITY INTERVAL MS 330

# **Typedefs**

typedef unsigned int UINT

# **Variables**

CStopWatch timer

# 17.30.1 Macro Definition Documentation

- 17.30.1.1 #define CACHE\_DIR "cached\_iris\_data"
- 17.30.1.2 #define CONNECTIVITY\_AN\_AN 0.5
- 17.30.1.3 #define CONNECTIVITY\_PN\_AN 0.5

17.30.1.4 #define CONNECTIVITY\_PN\_PN 0.5 17.30.1.5 #define CONNECTIVITY\_RN\_PN 0.5 17.30.1.6 #define D\_MAX\_RANDOM\_NUM 32767 17.30.1.7 #define DATASET\_NAME "Iris" 17.30.1.8 #define divi "/" 17.30.1.9 #define FLAG\_RUN\_ON\_CPU 1 17.30.1.10 #define GLOBAL\_WEIGHT\_SCALING 1.0 17.30.1.11 #define MAX\_FIRING\_RATE\_HZ 70 17.30.1.12 #define MAX\_WEIGHT\_PN\_AN 0.4 17.30.1.13 #define MIN\_FIRING\_RATE\_HZ 20 17.30.1.14 #define MIN\_WEIGHT\_PN\_AN 0.1 17.30.1.15 #define N\_FOLDING 5 17.30.1.16 #define OUTPUT\_DIR "output\_iris" 17.30.1.17 #define PLASTICITY\_INTERVAL\_MS 330 17.30.1.18 #define RECORDING\_TIME\_MS 1000 17.30.1.19 #define RECORDINGS\_DIR "recordings\_iris\_data" 17.30.1.20 #define REPEAT\_LEARNING\_SET 2 17.30.1.21 #define SPIKING\_ACTIVITY\_THRESHOLD\_HZ 5 17.30.1.22 #define TOTAL\_RECORDINGS 150 17.30.1.23 #define VR\_DATA\_FILENAME "VR-recordings-iris.data" 17.30.1.24 #define WEIGHT\_DELTA\_PN\_AN 0.04 17.30.1.25 #define WEIGHT\_RN\_PN 0.5 17.30.1.26 #define WEIGHT\_WTA\_AN\_AN 0.01 17.30.1.27 #define WEIGHT\_WTA\_PN\_PN 0.01 17.30.2 Typedef Documentation 17.30.2.1 typedef unsigned int UINT 17.30.3 Variable Documentation 17.30.3.1 CStopWatch timer

#### **Functions**

- n varNames clear ()
- n varNames push\_back ("V")

17.31 extra\_neurons.h File Reference

- n varTypes push\_back ("float")
- n varNames push\_back ("V\_NB")
- n varNames push\_back ("tSpike\_NB")
- n varNames push\_back ("\_\_regime\_val")
- n varTypes push\_back ("int")
- n pNames push\_back ("VReset\_NB")
- n pNames push\_back ("VThresh\_NB")
- n pNames push\_back ("tRefrac\_NB")
- n pNames push\_back ("VRest\_NB")
- n pNames push\_back ("TAUm\_NB")
- n pNames push\_back ("Cm\_NB")
- nModels push\_back (n)
- n varNames push back ("count t NB")
- n pNames push\_back ("max\_t\_NB")

#### **Variables**

• n simCode

```
17.31.1 Function Documentation
```

```
17.31.1.1 ps dpNames clear ( )
```

- 17.31.1.2 n varNames push\_back ( "V" )
- 17.31.1.3 ps varTypes push\_back ( "float" )
- 17.31.1.4 n varNames push\_back ( "V\_NB" )
- 17.31.1.5 n varNames push\_back ( "tSpike\_NB" )
- 17.31.1.6 n varNames push\_back ( "\_\_regime\_val" )
- 17.31.1.7 n varTypes push\_back ( "int" )
- 17.31.1.8 n pNames push\_back ( "VReset\_NB" )
- 17.31.1.9 n pNames push\_back ( "VThresh\_NB" )
- 17.31.1.10 n pNames push\_back ( "tRefrac\_NB" )
- 17.31.1.11 n pNames push\_back ( "VRest\_NB" )
- 17.31.1.12 n pNames push\_back ( "TAUm\_NB" )
- 17.31.1.13 n pNames push\_back ( "Cm\_NB" )
- 17.31.1.14 nModels push\_back ( n )
- 17.31.1.15 n varNames push\_back ( "count\_t\_NB" )
- 17.31.2 Variable Documentation
- 17.31.2.1 n simCode

# Initial value:

# 17.32 extra\_postsynapses.h File Reference

#### **Functions**

- ps varNames clear ()
- ps varNames push\_back ("g\_PS")
- ps varTypes push\_back ("float")
- ps pNames push\_back ("tau\_syn\_PS")
- ps pNames push\_back ("E\_PS")
- postSynModels push\_back (ps)

### **Variables**

- ps postSyntoCurrent
- ps postSynDecay

### 17.32.1 Function Documentation

```
17.32.1.1 ps varNames clear ( )
```

- 17.32.1.2 ps varNames push\_back ( "g\_PS" )
- 17.32.1.3 ps varTypes push\_back ( "float" )
- 17.32.1.4 ps pNames push\_back ( "tau\_syn\_PS" )
- 17.32.1.5 ps pNames push\_back ( "E\_PS" )
- 17.32.1.6 postSynModels push\_back (ps )
- 17.32.2 Variable Documentation
- 17.32.2.1 ps postSynDecay

# Initial value:

```
$ (g_PS) += (-$(g_PS)/$(tau_syn_PS))*DT; \n \
$ (inSyn) = 0; \
```

### 17.32.2.2 ps postSyntoCurrent

### Initial value:

# 17.33 extra\_weightupdates.h File Reference

### 17.34 GA.cc File Reference

```
#include <algorithm>
```

### Classes

struct errTupel

### **Functions**

- int compareErrTupel (const void \*x, const void \*y)
- void procreatePop (FILE \*osb)

# 17.34.1 Function Documentation

```
17.34.1.1 int compareErrTupel ( const void *x, const void *y )
```

```
17.34.1.2 void procreatePop ( FILE * osb )
```

# 17.35 gauss.cc File Reference

Contains the implementation of the Gaussian random number generator class randomGauss.

```
#include "gauss.h"
```

### Macros

• #define GAUSS\_CC

macro for avoiding multiple inclusion during compilation

# 17.35.1 Detailed Description

Contains the implementation of the Gaussian random number generator class randomGauss.

#### 17.35.2 Macro Definition Documentation

#### 17.35.2.1 #define GAUSS\_CC

macro for avoiding multiple inclusion during compilation

# 17.36 gauss.h File Reference

Random number generator for Gaussian random variable with mean 0 and standard deviation 1.

```
#include <cmath>
#include "randomGen.h"
#include "randomGen.cc"
#include "gauss.cc"
```

#### Classes

· class randomGauss

Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

#### **Macros**

#define GAUSS\_H
 macro for avoiding multiple inclusion during compilation

### 17.36.1 Detailed Description

Random number generator for Gaussian random variable with mean 0 and standard deviation 1.

This random number generator is based on the ratio of uniforms method by A.J. Kinderman and J.F. Monahan and improved with quadratic boundind curves by J.L. Leva. Taken from Algorithm 712 ACM Trans. Math. Softw. 18 p. 454. (the necessary uniform random variables are obtained from the ISAAC random number generator; C++ Implementation by Quinn Tyler Jackson of the RG invented by Bob Jenkins Jr.).

#### 17.36.2 Macro Definition Documentation

```
17.36.2.1 #define GAUSS_H
```

macro for avoiding multiple inclusion during compilation

# 17.37 gen\_input\_structured.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "randomGen.cc"
```

# **Functions**

• int main (int argc, char \*argv[])

### **Variables**

randomGen R

### 17.37.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate input patterns for the antennal lobe in the model. The triple "fix" in the filename refers to a three-fold control for having the same number of active inputs for each pattern, even if changing patterns by adding noise.

```
17.37.2 Function Documentation17.37.2.1 int main (int argc, char * argv[])17.37.3 Variable Documentation
```

#### 17.37.3.1 randomGen R

# 17.38 gen\_kcdn\_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

### **Functions**

• int main (int argc, char \*argv[])

### Variables

- randomGen R
- · randomGauss RG

# 17.38.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between KCs and DNs (detector neurons) in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

```
17.38.2 Function Documentation
17.38.2.1 int main (int argc, char * argv[])
17.38.3 Variable Documentation
17.38.3.1 randomGen R
```

#### 17.38.3.2 randomGauss RG

# 17.39 gen\_kcdn\_syns\_fixto10K.cc File Reference

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

#### **Functions**

• int main (int argc, char \*argv[])

#### Variables

- · randomGen R
- randomGen R2
- · randomGauss RG

#### 17.39.1 Function Documentation

```
17.39.1.1 int main ( int argc, char * argv[] )
```

17.39.2 Variable Documentation

17.39.2.1 randomGen R

17.39.2.2 randomGen R2

17.39.2.3 randomGauss RG

# 17.40 gen\_pnkc\_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

### **Functions**

• int main (int argc, char \*argv[])

# Variables

- randomGen R
- · randomGauss RG

# 17.40.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and KCs in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

```
17.40.2 Function Documentation

17.40.2.1 int main ( int argc, char * argv[] )

17.40.3 Variable Documentation

17.40.3.1 randomGen R

17.40.3.2 randomGauss RG

17.41 gen_pnkc_syns_indivID.cc File Reference
```

```
This file is part of a tool chain for running the classol/MBody1 example model.
```

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <cstdint>
#include "gauss.h"
#include "randomGen.h"
#include "randomGen.cc"
```

#### Macros

```
#define B(x, i) ((x) & (0x80000000 >> (i)))

Extract the bit at the specified position i from x.
#define setB(x, i) x= ((x) | (0x800000000 >> (i)))

Set the bit at the specified position i in x to 1.
#define delB(x, i) x= ((x) & (~(0x800000000 >> (i))))

Set the bit at the specified position i in x to 0.
```

### **Functions**

• int main (int argc, char \*argv[])

### **Variables**

- randomGen R
- · randomGauss RG

# 17.41.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and KCs in the model. In contrast to the gen\_pnkc\_syns.cc tool, here the output is in a format that is suited for the "INDIVIDUALID" method for specifying connectivity. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

#### 17.41.2 Macro Definition Documentation

```
17.41.2.1 #define B( x, i) ((x) & (0x800000000 >> (i)))
```

Extract the bit at the specified position i from x.

```
17.41.2.2 #define delB( x, i) x= ((x) & (\sim(0x800000000 >> (i))))
```

Set the bit at the specified position i in x to 0.

```
17.41.2.3 #define setB( x, i) x= ((x) | (0x800000000 >> (i)))
```

Set the bit at the specified position i in x to 1.

#### 17.41.3 Function Documentation

```
17.41.3.1 int main ( int argc, char * argv[] )
```

17.41.4 Variable Documentation

17.41.4.1 randomGen R

17.41.4.2 randomGauss RG

### 17.42 gen\_pnlhi\_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
```

# **Functions**

• int main (int argc, char \*argv[])

### 17.42.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and LHIs (lateral horn interneurons) in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

# 17.42.2 Function Documentation

```
17.42.2.1 int main ( int argc, char * argv[] )
```

# 17.43 gen\_syns\_sparse.cc File Reference

This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity.

```
#include <iostream>
#include <fstream>
#include <string.h>
#include "randomGen.h"
#include "gauss.h"
#include <vector>
```

#### **Functions**

• int main (int argc, char \*argv[])

#### **Variables**

- randomGen R
- · randomGauss RG

# 17.43.1 Detailed Description

This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity.

```
17.43.2 Function Documentation
```

```
17.43.2.1 int main ( int argc, char * argv[])
```

17.43.3 Variable Documentation

17.43.3.1 randomGen R

17.43.3.2 randomGauss RG

# 17.44 gen\_syns\_sparse\_izhModel.cc File Reference

This file is part of a tool chain for running the Izhikevich network model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <cmath>
#include <vector>
#include "randomGen.h"
#include "randomGen.cc"
```

# **Functions**

- int printVector (vector< unsigned int > &)
- int printVector (vector< double > &)
- int main (int argc, char \*argv[])

### **Variables**

- · randomGen R
- randomGen Rind
- · double gsyn
- double \* garray
- · unsigned int \* ind
- double \* garray\_ee
- std::vector< double > g\_ee
- std::vector< unsigned int > indlnG\_ee
- std::vector< unsigned int > ind\_ee
- double \* garray\_ei
- std::vector< double > g ei
- std::vector< unsigned int > indlnG\_ei
- std::vector< unsigned int > ind\_ei
- double \* garray\_ie
- std::vector< double > g\_ie
- std::vector< unsigned int > indlnG\_ie
- $std::vector < unsigned int > ind_ie$
- double \* garray\_ii
- std::vector< double > g\_ii
- std::vector< unsigned int > indlnG\_ii
- std::vector< unsigned int > ind\_ii

### 17.44.1 Detailed Description

This file is part of a tool chain for running the Izhikevich network model.

```
17.44.2 Function Documentation
```

```
17.44.2.1 int main ( int argc, char * argv[] )
```

- 17.44.2.2 int printVector ( vector< unsigned int > &  $\nu$  )
- 17.44.2.3 int printVector (vector < double > & v)
- 17.44.3 Variable Documentation
- 17.44.3.1 std::vector<double> g\_ee
- 17.44.3.2 std::vector<double> g\_ei
- 17.44.3.3 std::vector<double> g\_ie
- 17.44.3.4 std::vector<double> g\_ii
- 17.44.3.5 double\* garray
- 17.44.3.6 double\* garray\_ee
- 17.44.3.7 double\* garray\_ei
- 17.44.3.8 double\* garray\_ie
- 17.44.3.9 double\* garray\_ii
- 17.44.3.10 double gsyn

```
17.44.3.11 unsigned int* ind

17.44.3.12 std::vector<unsigned int> ind_ee

17.44.3.13 std::vector<unsigned int> ind_ei

17.44.3.14 std::vector<unsigned int> ind_ie

17.44.3.15 std::vector<unsigned int> ind_ii

17.44.3.16 std::vector<unsigned int> indlnG_ee

17.44.3.17 std::vector<unsigned int> indlnG_ei

17.44.3.18 std::vector<unsigned int> indlnG_ei

17.44.3.19 std::vector<unsigned int> indlnG_ie

17.44.3.20 randomGen R

17.44.3.21 randomGen Rind
```

# 17.45 generate\_run.cc File Reference

This file is used to run the HHVclampGA model with a single command line.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

### **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate\_run.

# 17.45.1 Detailed Description

This file is used to run the HHVclampGA model with a single command line.

# 17.45.2 Function Documentation

17.45.2.1 int main ( int argc, char \* argv[] )

Main entry point for generate\_run.

# 17.46 generate\_run.cc File Reference

This file is part of a tool chain for running the classIzh/Izh\_sparse example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <locale>
#include <stringUtils.h>
#include <sys/stat.h>
#include "command_line_processing.h"
#include "parse_options.h"
```

#### **Functions**

- unsigned int openFileGetMax (unsigned int \*array, unsigned int size, string name)
- int main (int argc, char \*argv[])

Main entry point for generate\_run.

# 17.46.1 Detailed Description

This file is part of a tool chain for running the classIzh/Izh sparse example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_ \* tools for generating connectivity, providing the population size information through ./model/sizes.h to the model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

```
17.46.2 Function Documentation
```

```
17.46.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

17.46.2.2 unsigned int openFileGetMax ( unsigned int \* array, unsigned int size, string name )

# 17.47 generate\_run.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

#### **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate run.

### 17.47.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_\* tools for generating connectivity, providing the population size information through ./model/sizes.h to the MBody1 model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

```
17.47.2 Function Documentation

17.47.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

# 17.48 generate run.cc File Reference

This file is part of a tool chain for running the classol/MBody\_delayedSyn example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

# **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate\_run.

# 17.48.1 Detailed Description

This file is part of a tool chain for running the classol/MBody\_delayedSyn example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_\* tools for generating connectivity, providing the population size information through ./model/sizes.h to the MBody—delayedSyninition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

#### 17.48.2 Function Documentation

```
17.48.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate run.

### 17.49 generate run.cc File Reference

This file is part of a tool chain for running the classol/MBody\_individualID example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

# **Functions**

• int main (int argc, char \*argv[])

Main entry point for generate\_run.

# 17.49.1 Detailed Description

This file is part of a tool chain for running the classol/MBody\_individualID example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_\* tools for generating connectivity, providing the population size information through ./model/sizes.h to the MBody—individualID model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

```
17.49.2 Function Documentation
```

```
17.49.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

# 17.50 generate\_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

#### **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate\_run.

### 17.50.1 Function Documentation

```
17.50.1.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

# 17.51 generate\_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

### **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate\_run.

# 17.51.1 Function Documentation

17.51.1.1 int main ( int argc, char \* argv[] )

Main entry point for generate\_run.

# 17.52 generate\_run.cc File Reference

This file is part of a tool chain for running the classol/MBody\_userdef example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

#### **Functions**

int main (int argc, char \*argv[])
 Main entry point for generate\_run.

#### 17.52.1 Detailed Description

This file is part of a tool chain for running the classol/MBody\_userdef example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_\* tools for generating connectivity, providing the population size information through ./model/sizes.h to the MBody\_userdef model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

```
17.52.2 Function Documentation
```

```
17.52.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

# 17.53 generate\_run.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

### **Functions**

```
    int main (int argc, char *argv[])
    Main entry point for generate_run.
```

### 17.53.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_\* tools for generating connectivity, providing the population size information through ./model/sizes.h to the MBody1 model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using GeNN as it only requires a single command line to execute all necessary tasks.

```
17.53.2 Function Documentation17.53.2.1 int main ( int argc, char * argv[] )Main entry point for generate_run.
```

### 17.54 generate\_run.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
#include "stringUtils.h"
#include "command_line_processing.h"
#include "parse_options.h"
```

# **Functions**

```
    int main (int argc, char *argv[])
    Main entry point for generate run.
```

# 17.54.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool that wraps all the other tools into one chain of tasks, including running all the gen\_ \* tools for generating connectivity, providing the population size information through ./model/sizes.h to the model definition, running the GeNN code generation and compilation steps, executing the model and collecting some timing information. This tool is the recommended way to quickstart using Poisson-Izhikevich example in GeNN as it only requires a single command line to execute all necessary tasks.

#### 17.54.2 Function Documentation

```
17.54.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate\_run.

# 17.55 generateALL.cc File Reference

Main file combining the code for code generation. Part of the code generation section.

```
#include "global.h"
#include "generateALL.h"
#include "generateRunner.h"
#include "generateCPU.h"
#include "generateKernels.h"
#include "modelSpec.h"
#include "utils.h"
#include "codeGenUtils.h"
#include "CodeHelper.h"
#include <algorithm>
#include <iterator>
#include <sys/stat.h>
```

### **Functions**

• void generate\_model\_runner (const NNmodel &model, const string &path)

This function will call the necessary sub-functions to generate the code for simulating a model.

void chooseDevice (NNmodel &model, const string &path)

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

• int main (int argc, char \*argv[])

Main entry point for the generateALL executable that generates the code for GPU and CPU.

### **Variables**

· CodeHelper hlp

# 17.55.1 Detailed Description

Main file combining the code for code generation. Part of the code generation section.

The file includes separate files for generating kernels (generateKernels.cc), generating the CPU side code for running simulations on either the CPU or GPU (generateRunner.cc) and for CPU-only simulation code (generateCP U.cc).

### 17.55.2 Function Documentation

```
17.55.2.1 void chooseDevice ( NNmodel & model, const string & path )
```

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

The main tasks in this function are the detection and characterization of the GPU device present (if any), choosing which GPU device to use, finding and appropriate block size, taking note of the major and minor version of the  $C \leftarrow UDA$  enabled device chosen for use, and populating the list of standard neuron models. The chosen device number is returned.

#### **Parameters**

model	the nn model we are generating code for
path	path the generated code will be deposited

17.55.2.2 void generate\_model\_runner ( const NNmodel & model, const string & path )

This function will call the necessary sub-functions to generate the code for simulating a model.

### **Parameters**

model	Model description
path	Path where the generated code will be deposited

17.55.2.3 int main ( int argc, char \* argv[] )

Main entry point for the generateALL executable that generates the code for GPU and CPU.

The main function is the entry point for the code generation engine. It prepares the system and then invokes generate\_model\_runner to initiate the different parts of actual code generation.

### **Parameters**

argc	number of arguments; expected to be 2
argv	Arguments; expected to contain the target directory for code generation.

# 17.55.3 Variable Documentation

# 17.55.3.1 CodeHelper hlp

# 17.56 generateALL.h File Reference

```
#include "modelSpec.h"
#include <string>
```

# **Functions**

• void generate\_model\_runner (const NNmodel &model, const string &path)

This function will call the necessary sub-functions to generate the code for simulating a model.

• void chooseDevice (NNmodel &model, const string &path)

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

### 17.56.1 Function Documentation

# 17.56.1.1 void chooseDevice ( NNmodel & model, const string & path )

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

The main tasks in this function are the detection and characterization of the GPU device present (if any), choosing which GPU device to use, finding and appropriate block size, taking note of the major and minor version of the  $C \leftarrow UDA$  enabled device chosen for use, and populating the list of standard neuron models. The chosen device number is returned.

#### **Parameters**

model	the nn model we are generating code for
path	path the generated code will be deposited

### 17.56.1.2 void generate\_model\_runner ( const NNmodel & model, const string & path )

This function will call the necessary sub-functions to generate the code for simulating a model.

#### **Parameters**

model	Model description
path	Path where the generated code will be deposited

# 17.57 generateCPU.cc File Reference

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

```
#include "generateCPU.h"
#include "global.h"
#include "utils.h"
#include "codeGenUtils.h"
#include "standardGeneratedSections.h"
#include "standardSubstitutions.h"
#include "CodeHelper.h"
#include <algorithm>
#include <typeinfo>
```

# **Functions**

• void genNeuronFunction (const NNmodel &model, const string &path)

Function that generates the code of the function the will simulate all neurons on the CPU.

• void genSynapseFunction (const NNmodel &model, const string &path)

Function that generates code that will simulate all synapses of the model on the CPU.

# 17.57.1 Detailed Description

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

#### 17.57.2 Function Documentation

17.57.2.1 void genNeuronFunction (const NNmodel & model, const string & path)

Function that generates the code of the function the will simulate all neurons on the CPU.

### **Parameters**

model	Model description
path	Path for code generation

17.57.2.2 void genSynapseFunction (const NNmodel & model, const string & path)

Function that generates code that will simulate all synapses of the model on the CPU.

#### **Parameters**

model	Model description
path	Path for code generation

# 17.58 generateCPU.h File Reference

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

```
#include "modelSpec.h"
#include <string>
#include <fstream>
```

### **Functions**

- void genNeuronFunction (const NNmodel &model, const string &path)
   Function that generates the code of the function the will simulate all neurons on the CPU.
- void genSynapseFunction (const NNmodel &model, const string &path)

Function that generates code that will simulate all synapses of the model on the CPU.

# 17.58.1 Detailed Description

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

### 17.58.2 Function Documentation

17.58.2.1 void genNeuronFunction (const NNmodel & model, const string & path)

Function that generates the code of the function the will simulate all neurons on the CPU.

### **Parameters**

model	Model description
path	Path for code generation

17.58.2.2 void genSynapseFunction (const NNmodel & model, const string & path)

Function that generates code that will simulate all synapses of the model on the CPU.

#### **Parameters**

model	Model description
path	Path for code generation

# 17.59 generateKernels.cc File Reference

Contains functions that generate code for CUDA kernels. Part of the code generation section.

```
#include "generateKernels.h"
#include "global.h"
#include "utils.h"
#include "standardGeneratedSections.h"
#include "standardSubstitutions.h"
#include "codeGenUtils.h"
#include "CodeHelper.h"
#include <algorithm>
```

#### **Functions**

• void genNeuronKernel (const NNmodel &model, const string &path)

Function for generating the CUDA kernel that simulates all neurons in the model.

• void genSynapseKernel (const NNmodel &model, const string &path)

Function for generating a CUDA kernel for simulating all synapses.

### 17.59.1 Detailed Description

Contains functions that generate code for CUDA kernels. Part of the code generation section.

# 17.59.2 Function Documentation

17.59.2.1 void genNeuronKernel (const NNmodel & model, const string & path)

Function for generating the CUDA kernel that simulates all neurons in the model.

The code generated upon execution of this function is for defining GPU side global variables that will hold model state in the GPU global memory and for the actual kernel function for simulating the neurons for one time step.

# Parameters

model	Model description
path	Path for code generation

17.59.2.2 void genSynapseKernel (const NNmodel & model, const string & path)

Function for generating a CUDA kernel for simulating all synapses.

This functions generates code for global variables on the GPU side that are synapse-related and the actual CUDA kernel for simulating one time step of the synapses. < "id" if first synapse group, else "lid". lid =(thread index- last

thread of the last synapse group)

#### **Parameters**

model	Model description
path	Path for code generation

# 17.60 generateKernels.h File Reference

Contains functions that generate code for CUDA kernels. Part of the code generation section.

```
#include "modelSpec.h"
#include <string>
#include <fstream>
```

#### **Functions**

• void genNeuronKernel (const NNmodel &model, const string &path)

Function for generating the CUDA kernel that simulates all neurons in the model.

void genSynapseKernel (const NNmodel &model, const string &path)

Function for generating a CUDA kernel for simulating all synapses.

### 17.60.1 Detailed Description

Contains functions that generate code for CUDA kernels. Part of the code generation section.

#### 17.60.2 Function Documentation

17.60.2.1 void genNeuronKernel ( const NNmodel & model, const string & path )

Function for generating the CUDA kernel that simulates all neurons in the model.

The code generated upon execution of this function is for defining GPU side global variables that will hold model state in the GPU global memory and for the actual kernel function for simulating the neurons for one time step.

# Parameters

model	Model description
path	Path for code generation

17.60.2.2 void genSynapseKernel (const NNmodel & model, const string & path)

Function for generating a CUDA kernel for simulating all synapses.

This functions generates code for global variables on the GPU side that are synapse-related and the actual CUDA kernel for simulating one time step of the synapses. < "id" if first synapse group, else "lid". lid =(thread index- last thread of the last synapse group)

## **Parameters**

model	Model description
path	Path for code generation

# 17.61 generateRunner.cc File Reference

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

```
#include "generateRunner.h"
#include "global.h"
#include "utils.h"
#include "codeGenUtils.h"
#include "CodeHelper.h"
#include <stdint.h>
#include <algorithm>
#include <cfloat>
```

#### **Functions**

void genRunner (const NNmodel &model, const string &path)

A function that generates predominantly host-side code.

void genRunnerGPU (const NNmodel &model, const string &path)

A function to generate the code that simulates the model on the GPU.

void genMakefile (const NNmodel &model, const string &path)

A function that generates the Makefile for all generated GeNN code.

## 17.61.1 Detailed Description

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

# 17.61.2 Function Documentation

17.61.2.1 void genMakefile (const NNmodel & model, const string & path)

A function that generates the Makefile for all generated GeNN code.

# Parameters

model	Model description
path	Path for code generation

17.61.2.2 void genRunner (const NNmodel & model, const string & path)

A function that generates predominantly host-side code.

In this function host-side functions and other code are generated, including: Global host variables, "allocatedMem()" function for allocating memories, "freeMem" function for freeing the allocated memories, "initialize" for initializing host variables, "gFunc" and "initGRaw()" for use with plastic synapses if such synapses exist in the model. Method for cleaning up and resetting device while quitting GeNN

### **Parameters**

model	Model description
path	Path for code generationn

17.61.2.3 void genRunnerGPU (const NNmodel & model, const string & path)

A function to generate the code that simulates the model on the GPU.

The function generates functions that will spawn kernel grids onto the GPU (but not the actual kernel code which is generated in "genNeuronKernel()" and "genSynpaseKernel()"). Generated functions include "copyGToDevice()", "copyGFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()". The last mentioned function is the function that will initialize the execution on the GPU in the generated simulation engine. All other generated functions are "convenience functions" to handle data transfer from and to the GPU.

#### **Parameters**

model	Model description
path	Path for code generation

# 17.62 generateRunner.h File Reference

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

```
#include "modelSpec.h"
#include <string>
#include <fstream>
```

#### **Functions**

- void genRunner (const NNmodel &model, const string &path)
  - A function that generates predominantly host-side code.
- void genRunnerGPU (const NNmodel &model, const string &path)
  - A function to generate the code that simulates the model on the GPU.
- void genMakefile (const NNmodel &model, const string &path)

A function that generates the Makefile for all generated GeNN code.

## 17.62.1 Detailed Description

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

### 17.62.2 Function Documentation

17.62.2.1 void genMakefile (const NNmodel & model, const string & path)

A function that generates the Makefile for all generated GeNN code.

### **Parameters**

model	Model description
path	Path for code generation

17.62.2.2 void genRunner (const NNmodel & model, const string & path)

A function that generates predominantly host-side code.

In this function host-side functions and other code are generated, including: Global host variables, "allocatedMem()" function for allocating memories, "freeMem" function for freeing the allocated memories, "initialize" for initializing host variables, "gFunc" and "initGRaw()" for use with plastic synapses if such synapses exist in the model. Method for cleaning up and resetting device while quitting GeNN

#### **Parameters**

model	Model description
path	Path for code generationn

17.62.2.3 void genRunnerGPU (const NNmodel & model, const string & path)

A function to generate the code that simulates the model on the GPU.

The function generates functions that will spawn kernel grids onto the GPU (but not the actual kernel code which is generated in "genNeuronKernel()" and "genSynpaseKernel()"). Generated functions include "copyGToDevice()", "copyGFromDevice()", "copySpikesFromDevice()", "copySpi

#### **Parameters**

model	Model description
path	Path for code generation

# 17.63 GeNNHelperKrnls.cu File Reference

#include "GeNNHelperKrnls.h"

#### **Functions**

- \_\_global\_\_ void setup\_kernel (curandState \*state, unsigned long seed, int sizeofResult)
- void xorwow setup (curandState \*devStates, long int sampleSize, long long int seed)
- template<typename T >

\_\_global\_\_ void generate\_random\_gpuInput\_xorwow (curandState \*state, T \*result, int sizeofResult, T Rstrength, T Rshift)

- template<typename T >
  - void generate\_random\_gpuInput\_xorwow (curandState \*state, T \*result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads)
- template void <a href="mailto:generate\_random\_gpulnput\_xorwow">gpulnput\_xorwow</a> double > (curandState \*state, double \*result, int sizeofResult, double Rstrength, double Rshift, dim3 sGrid, dim3 sThreads)

#### 17.63.1 Function Documentation

17.63.1.1 template<typename T > \_\_global\_\_ void generate\_random\_gpulnput\_xorwow ( curandState \* state, T \* result, int sizeofResult, T Rstrength, T Rshift )

- 17.63.1.2 template < typename T > void generate\_random\_gpulnput\_xorwow ( curandState \* state, T \* result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads )
- 17.63.1.3 template void generate\_random\_gpuInput\_xorwow< double > ( curandState \* state, double \* result, int sizeofResult, double Rstrength, double Rshift, dim3 sGrid, dim3 sThreads )
- 17.63.1.4 template void generate\_random\_gpuInput\_xorwow< float > ( curandState \* state, float \* result, int sizeofResult, float Rstrength, float Rshift, dim3 sGrid, dim3 sThreads )
- 17.63.1.5 \_\_global\_\_ void setup\_kernel ( curandState \* state, unsigned long seed, int sizeofResult )
- 17.63.1.6 void xorwow\_setup ( curandState \* devStates, long int sampleSize, long long int seed )

# 17.64 GeNNHelperKrnls.h File Reference

```
#include <curand_kernel.h>
```

#### **Functions**

- \_\_global\_\_ void setup\_kernel (curandState \*state, unsigned long seed, int sizeofResult)
- void xorwow\_setup (curandState \*devStates, long int sampleSize, long long int seed)
- template<typename T >
   \_\_global\_\_ void generate\_random\_gpulnput\_xorwow (curandState \*state, T \*result, int sizeofResult, T Rstrength, T Rshift)
- template<typename T >
   void generate\_random\_gpuInput\_xorwow (curandState \*state, T \*result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads)

#### **Variables**

- const int BlkSz = 256
- 17.64.1 Function Documentation
- 17.64.1.1 template < typename T > \_\_global\_\_ void generate\_random\_gpulnput\_xorwow ( curandState \* state, T \* result, int sizeofResult, T Rstrength, T Rshift )
- 17.64.1.2 template < typename T > void generate\_random\_gpulnput\_xorwow ( curandState \* state, T \* result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads )
- 17.64.1.3 \_\_global\_\_ void setup\_kernel ( curandState \* state, unsigned long seed, int sizeofResult )
- 17.64.1.4 void xorwow\_setup ( curandState \* devStates, long int sampleSize, long long int seed )
- 17.64.2 Variable Documentation
- 17.64.2.1 const int BlkSz = 256

# 17.65 global.cc File Reference

```
#include "global.h"
```

# Namespaces

• GENN FLAGS

GENN\_PREFERENCES

#### Macros

• #define GLOBAL CC

#### **Variables**

· unsigned int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

unsigned int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

· unsigned int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

• unsigned int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

- cudaDeviceProp \* deviceProp
- · int theDevice

Global variable containing the currently selected CUDA device's number.

int deviceCount

Global variable containing the number of CUDA devices on this host.

· int hostCount

Global variable containing the number of hosts within the local compute cluster.

17.65.1 Macro Definition Documentation

17.65.1.1 #define GLOBAL\_CC

17.65.2 Variable Documentation

17.65.2.1 int deviceCount

Global variable containing the number of CUDA devices on this host.

17.65.2.2 cudaDeviceProp\* deviceProp

17.65.2.3 int hostCount

Global variable containing the number of hosts within the local compute cluster.

17.65.2.4 unsigned int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

17.65.2.5 unsigned int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

17.65.2.6 unsigned int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

17.65.2.7 unsigned int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

#### 17.65.2.8 int the Device

Global variable containing the currently selected CUDA device's number.

## 17.66 global.h File Reference

Global header file containing a few global variables. Part of the code generation section.

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <string>
```

## Namespaces

- GENN FLAGS
- GENN PREFERENCES

#### Variables

- unsigned int GENN\_FLAGS::calcSynapseDynamics = 0
- unsigned int GENN\_FLAGS::calcSynapses = 1
- unsigned int GENN\_FLAGS::learnSynapsesPost = 2
- unsigned int GENN FLAGS::calcNeurons = 3
- int GENN\_PREFERENCES::optimiseBlockSize = 1

Flag for signalling whether or not block size optimisation should be performed.

• int GENN\_PREFERENCES::autoChooseDevice = 1

Flag to signal whether the GPU device should be chosen automatically.

• bool GENN\_PREFERENCES::optimizeCode = false

Request speed-optimized code, at the expense of floating-point accuracy.

• bool GENN\_PREFERENCES::debugCode = false

Request debug data to be embedded in the generated code.

• bool GENN PREFERENCES::showPtxInfo = false

Request that PTX assembler information be displayed for each CUDA kernel during compilation.

• double GENN\_PREFERENCES::asGoodAsZero = 1e-19

Global variable that is used when detecting close to zero values, for example when setting sparse connectivity from a dense matrix.

- int GENN\_PREFERENCES::defaultDevice = 0
- unsigned int GENN\_PREFERENCES::neuronBlockSize = 32

default GPU device; used to determine which GPU to use if chooseDevice is 0 (off)

- unsigned int GENN PREFERENCES::synapseBlockSize = 32
- unsigned int GENN\_PREFERENCES::learningBlockSize = 32
- unsigned int GENN\_PREFERENCES::synapseDynamicsBlockSize = 32
- unsigned int GENN\_PREFERENCES::autoRefractory = 1

Flag for signalling whether spikes are only reported if thresholdCondition changes from false to true (autoRefractory == 1) or spikes are emitted whenever thresholdCondition is true no matter what.%.

std::string GENN\_PREFERENCES::userCxxFlagsWIN = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for windows platforms)

• std::string GENN\_PREFERENCES::userCxxFlagsGNU = ""

Allows users to set specific C++ compiler options they may want to use for all host side code (used for unix based platforms)

• std::string GENN\_PREFERENCES::userNvccFlags = ""

Allows users to set specific nvcc compiler options they may want to use for all GPU code (identical for windows and unix platforms)

• unsigned int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

unsigned int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

· unsigned int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

unsigned int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

- cudaDeviceProp \* deviceProp
- · int theDevice

Global variable containing the currently selected CUDA device's number.

· int deviceCount

Global variable containing the number of CUDA devices on this host.

· int hostCount

Global variable containing the number of hosts within the local compute cluster.

## 17.66.1 Detailed Description

Global header file containing a few global variables. Part of the code generation section.

17.66.2 Variable Documentation

17.66.2.1 int deviceCount

Global variable containing the number of CUDA devices on this host.

17.66.2.2 cudaDeviceProp\* deviceProp

17.66.2.3 int hostCount

Global variable containing the number of hosts within the local compute cluster.

17.66.2.4 unsigned int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

17.66.2.5 unsigned int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

17.66.2.6 unsigned int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

17.66.2.7 unsigned int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

17.66.2.8 int theDevice

Global variable containing the currently selected CUDA device's number.

# 17.67 helper.h File Reference

```
#include <vector>
```

#### Classes

struct inputSpec

#### **Functions**

- ostream & operator<< (ostream &os, inputSpec &I)</li>
- · void write\_para ()
- void single\_var\_init\_fullrange (int n)
- void single\_var\_reinit (int n, double fac)
- void copy\_var (int src, int trg)
- void var\_init\_fullrange ()
- void var\_reinit (double fac)
- void truevar\_init ()
- void initexpHH ()
- void truevar\_initexpHH ()
- void runexpHH (float t)
- void initl (inputSpec &I)

# Variables

- double sigGNa = 0.1
- double sigENa = 10.0
- double sigGK = 0.1
- double sigEK = 10.0
- double sigGl = 0.1
- double sigEl = 10.0
- double sigC = 0.1
- const double limit [7][2]
- · double Vexp
- · double mexp
- double hexp
- double nexp
- double gNaexp
- double ENaexp
- double gKexp
- double EKexp
- double glexp
- double Elexp
- double Cexp

```
17.67.1 Function Documentation
17.67.1.1 void copy_var ( int src, int trg )
17.67.1.2 void initexpHH ( )
17.67.1.3 void initl (inputSpec & I)
17.67.1.4 ostream & operator << ( ostream & os, inputSpec & I )
17.67.1.5 void runexpHH ( float t )
17.67.1.6 void single_var_init_fullrange ( int n )
17.67.1.7 void single_var_reinit (int n, double fac )
17.67.1.8 void truevar_init ( )
17.67.1.9 void truevar_initexpHH ( )
17.67.1.10 void var_init_fullrange ( )
17.67.1.11 void var_reinit ( double fac )
17.67.1.12 void write_para ( )
17.67.2 Variable Documentation
17.67.2.1 double Cexp
17.67.2.2 double EKexp
17.67.2.3 double Elexp
17.67.2.4 double ENaexp
17.67.2.5 double gKexp
17.67.2.6 double glexp
17.67.2.7 double gNaexp
17.67.2.8 double hexp
17.67.2.9 const double limit[7][2]
Initial value:
= {{1.0, 200.0},
                               {0.0, 100.0},
{1.0, 100.0},
                                \{-100.0, -20.0\},\
                               {1.0, 50.0},
{-100.0, -20.0},
{1e-1, 10.0}}
17.67.2.10 double mexp
17.67.2.11 double nexp
17.67.2.12 double sigC = 0.1
17.67.2.13 double sigEK = 10.0
```

```
17.67.2.14 double sigEl = 10.0

17.67.2.15 double sigENa = 10.0

17.67.2.16 double sigGK = 0.1

17.67.2.17 double sigGl = 0.1

17.67.2.18 double sigGNa = 0.1

17.67.2.19 double Vexp
```

# 17.68 HHVClamp.cc File Reference

This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques.

```
#include "modelSpec.h"
#include "global.h"
#include "HHVClampParameters.h"
```

#### Classes

· class MyHH

#### **Functions**

- IMPLEMENT\_MODEL (MyHH)
- void modelDefinition (NNmodel &model)

This function defines the HH model with variable parameters.

# Variables

• MyHH::VarValues myHH\_ini (-60.0, 0.0529324, 0.3176767, 0.5961207, 120.0, 55.0, 36.0,-72.0, 0.3,-50.0, 1.0)

#### 17.68.1 Detailed Description

This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques.

```
17.68.2 Function Documentation
17.68.2.1 IMPLEMENT_MODEL ( MyHH )
17.68.2.2 void modelDefinition ( NNmodel & model )
This function defines the HH model with variable parameters.
```

17.68.3 Variable Documentation

17.68.3.1 MyHH::VarValues myHH\_ini(-60.0,0.0529324,0.3176767,0.5961207,120.0,55.0,36.0,-72.0,0.3,-50.0,1.0)

# 17.69 HHVClampParameters.h File Reference

#### **Macros**

- #define NPOP 12
- #define TOTALT 200000
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR\_MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

#### 17.69.1 Macro Definition Documentation

```
17.69.1.1 #define _FTYPE GENN_FLOAT
```

17.69.1.2 #define NPOP 12

17.69.1.3 #define scalar float

17.69.1.4 #define SCALAR\_MAX 3.40282e+38f

17.69.1.5 #define SCALAR\_MIN 1.17549e-38f

17.69.1.6 #define TOTALT 200000

# 17.70 hr\_time.cc File Reference

This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock.

```
#include <cstdio>
#include "hr_time.h"
```

### Macros

• #define HR\_TIMER

# 17.70.1 Detailed Description

This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock.

17.70.2 Macro Definition Documentation

17.70.2.1 #define HR\_TIMER

## 17.71 hr\_time.h File Reference

This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock.

```
#include <sys/time.h>
```

#### Classes

- struct stopWatch
- · class CStopWatch

## 17.71.1 Detailed Description

This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock.

# 17.72 isaac.cc File Reference

Header file and implementation of the ISAAC random number generator.

```
#include <stdlib.h>
```

#### Classes

- class QTIsaac< ALPHA, T >
- struct QTIsaac< ALPHA, T >::randctx

#### Macros

#define \_\_ISAAC\_HPP
 macro for avoiding multiple inclusion during compilation

## **Typedefs**

typedef unsigned long int ISAAC\_INT

# Variables

const ISAAC\_INT GOLDEN\_RATIO = ISAAC\_INT(0x9e3779b9)

# 17.72.1 Detailed Description

Header file and implementation of the ISAAC random number generator.

C++ TEMPLATE VERSION OF Robert J. Jenkins Jr.'s ISAAC Random Number Generator.

Ported from vanilla C to to template C++ class by Quinn Tyler Jackson on 16-23 July 1998.

```
qjackson@wave.home.com
```

The function for the expected period of this random number generator, according to Jenkins is:

```
f(a,b) = 2**((a+b*(3+2^a)-1)
(where a is ALPHA and b is bitwidth)
```

So, for a bitwidth of 32 and an ALPHA of 8, the expected period of ISAAC is:

```
2^{(8+32)} = 2^{820} = 2^{8295}
```

Jackson has been able to run implementations with an ALPHA as high as 16, or

```
2^^2097263
```

```
17.72.2 Macro Definition Documentation
```

```
17.72.2.1 #define __ISAAC_HPP
```

macro for avoiding multiple inclusion during compilation

# 17.72.3 Typedef Documentation

```
17.72.3.1 typedef unsigned long int ISAAC_INT
```

17.72.4 Variable Documentation

```
17.72.4.1 const ISAAC_INT GOLDEN_RATIO = ISAAC_INT(0x9e3779b9)
```

# 17.73 Izh\_sim\_sparse.cc File Reference

```
#include <iostream>
#include <fstream>
#include "Izh_sparse_sim.h"
#include <cuda_runtime.h>
#include "GeNNHelperKrnls.h"
```

#### **Functions**

int main (int argc, char \*argv[])

# 17.73.1 Function Documentation

```
17.73.1.1 int main ( int argc, char * argv[] )
```

# 17.74 Izh\_sparse.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "stringUtils.h"
#include 
vector>
#include "sizes.h"
```

## Classes

• class MylzhikevichVariable

### **Functions**

- IMPLEMENT\_MODEL (MylzhikevichVariable)
- void modelDefinition (NNmodel &model)

### Variables

- std::vector< unsigned int > neuronPSize
- std::vector< unsigned int > neuronVSize
- std::vector< unsigned int > synapsePSize

- scalar meanInpExc = 5.0\*inputFac
- scalar meanInpInh = 2.0\*inputFac
- MylzhikevichVariable::VarValues lzhExc\_ini (-65.0, 0.0, 0.02, 0.2,-65.0, 8.0, 0.0)
- MylzhikevichVariable::VarValues lzhInh\_ini (-65.0, 0.0, 0.02, 0.25,-65.0, 2.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues SynIzh\_ini (0.0)

#### 17.74.1 Function Documentation

- 17.74.1.1 IMPLEMENT\_MODEL ( MylzhikevichVariable )
- 17.74.1.2 void modelDefinition ( NNmodel & model )
- 17.74.2 Variable Documentation
- 17.74.2.1 MylzhikevichVariable::VarValues lzhExc\_ini(-65.0,0.0,0.02,0.2,-65.0,8.0,0.0)
- 17.74.2.2 MylzhikevichVariable::VarValues lzhlnh\_ini(-65.0,0.0,0.02,0.25,-65.0,2.0,0.0)
- 17.74.2.3 scalar meanInpExc = 5.0\*inputFac
- 17.74.2.4 scalar meanInplnh = 2.0\*inputFac
- 17.74.2.5 std::vector<unsigned int> neuronPSize
- 17.74.2.6 std::vector<unsigned int> neuronVSize
- 17.74.2.7 std::vector<unsigned int> synapsePSize
- 17.74.2.8 WeightUpdateModels::StaticPulse::VarValues Synlzh\_ini(0.0)

# 17.75 Izh\_sparse\_model.cc File Reference

```
#include "Izh_sparse_CODE/definitions.h"
#include "randomGen.h"
#include "gauss.h"
#include "Izh_sparse_model.h"
```

### Macros

• #define IZH SPARSE MODEL CC

### **Variables**

- · randomGauss RG
- randomGen R
- 17.75.1 Macro Definition Documentation
- 17.75.1.1 #define \_IZH\_SPARSE\_MODEL\_CC\_
- 17.75.2 Variable Documentation
- 17.75.2.1 randomGen R
- 17.75.2.2 randomGauss RG

# 17.76 Izh\_sparse\_model.h File Reference

## Classes

· class classIzh

# 17.77 Izh\_sparse\_sim.h File Reference

```
#include <cassert>
#include "hr_time.h"
#include "utils.h"
#include <cuda_runtime.h>
#include "Izh_sparse.cc"
#include "Izh_sparse_model.cc"
```

#### Macros

- #define DBG\_SIZE 5000
- #define T\_REPORT\_TME 5000.0
- #define TOTAL\_TME 1000.0

## **Variables**

CStopWatch timer

# 17.77.1 Macro Definition Documentation

```
17.77.1.1 #define DBG_SIZE 5000
```

17.77.1.2 #define T\_REPORT\_TME 5000.0

17.77.1.3 #define TOTAL\_TME 1000.0

17.77.2 Variable Documentation

17.77.2.1 CStopWatch timer

# 17.78 make\_input\_pats.cc File Reference

```
#include <iostream>
#include <fstream>
#include <cstdlib>
```

# **Functions**

• int main (int argc, char \*argv[])

# 17.78.1 Function Documentation

17.78.1.1 int main ( int argc, char \* argv[] )

# 17.79 map\_classol.cc File Reference

```
Implementation of the classol class.
```

```
#include "map_classol.h"
#include "MBody1_CODE/definitions.h"
```

#### Macros

• #define \_MAP\_CLASSOL\_CC\_

macro for avoiding multiple inclusion during compilation

17.79.1 Detailed Description

Implementation of the classol class.

17.79.2 Macro Definition Documentation

```
17.79.2.1 #define _MAP_CLASSOL_CC_
```

macro for avoiding multiple inclusion during compilation

# 17.80 map\_classol.cc File Reference

Implementation of the classol class.

```
#include "map_classol.h"
#include "MBody_delayedSyn_CODE/definitions.h"
```

### Macros

#define \_MAP\_CLASSOL\_CC\_

macro for avoiding multiple inclusion during compilation

17.80.1 Detailed Description

Implementation of the classol class.

17.80.2 Macro Definition Documentation

17.80.2.1 #define \_MAP\_CLASSOL\_CC\_

macro for avoiding multiple inclusion during compilation

# 17.81 map\_classol.cc File Reference

Implementation of the classol class.

```
#include "map_classol.h"
#include "MBody_individualID_CODE/definitions.h"
```

#### **Macros**

 #define \_MAP\_CLASSOL\_CC\_ macro for avoiding multiple inclusion during compilation

## 17.81.1 Detailed Description

Implementation of the classol class.

### 17.81.2 Macro Definition Documentation

```
17.81.2.1 #define _MAP_CLASSOL_CC_
```

macro for avoiding multiple inclusion during compilation

# 17.82 map\_classol.cc File Reference

```
#include "map_classol.h"
#include "MBody1_CODE/definitions.h"
```

#### Macros

 #define \_MAP\_CLASSOL\_CC\_ macro for avoiding multiple inclusion during compilation

## 17.82.1 Macro Definition Documentation

```
17.82.1.1 #define _MAP_CLASSOL_CC_
```

macro for avoiding multiple inclusion during compilation

## 17.83 map\_classol.cc File Reference

```
#include "map_classol.h"
#include "MBody1_CODE/definitions.h"
```

### Macros

 #define \_MAP\_CLASSOL\_CC\_ macro for avoiding multiple inclusion during compilation

### 17.83.1 Macro Definition Documentation

17.83.1.1 #define \_MAP\_CLASSOL\_CC\_

macro for avoiding multiple inclusion during compilation

# 17.84 map\_classol.cc File Reference

Implementation of the classol class.

```
#include "MBody_userdef_CODE/definitions.h"
#include "global.h"
#include "sparseUtils.h"
#include "map_classol.h"
```

#### **Macros**

• #define \_MAP\_CLASSOL\_CC\_ macro for avoiding multiple inclusion during compilation

## 17.84.1 Detailed Description

Implementation of the classol class.

17.84.2 Macro Definition Documentation

```
17.84.2.1 #define _MAP_CLASSOL_CC_
```

macro for avoiding multiple inclusion during compilation

# 17.85 map\_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

```
#include <stdint.h>
```

## Classes

· class classol

This class coontains the methods for running the MBody1 example model.

### 17.85.1 Detailed Description

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

The "classol" class is provided as part of a complete example of using GeNN in a user application. The model is a reimplementation of the model in

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7

### 17.86 map classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

```
#include <stdint.h>
```

#### Classes

· class classol

This class coontains the methods for running the MBody1 example model.

#### 17.86.1 Detailed Description

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

The "classol" class is provided as part of a complete example of using GeNN in a user application. The model is a reimplementation of the model in

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7

# 17.87 map\_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

```
#include <stdint.h>
```

#### Classes

· class classol

This class cpontains the methods for running the MBody1 example model.

## 17.87.1 Detailed Description

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

The "classol" class is provided as part of a complete example of using GeNN in a user application. The model is a reimplementation of the model in

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7

## 17.88 map classol.h File Reference

```
#include <stdint.h>
```

## Classes

· class classol

This class coontains the methods for running the MBody1 example model.

## 17.89 map\_classol.h File Reference

```
#include <stdint.h>
```

#### Classes

class classol

This class coontains the methods for running the MBody1 example model.

# 17.90 map\_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

```
#include <stdint.h>
```

#### Classes

· class classol

This class coontains the methods for running the MBody1 example model.

#### 17.90.1 Detailed Description

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

The "classol" class is provided as part of a complete example of using GeNN in a user application. The model is a reimplementation of the model in

T. Nowotny, R. Huerta, H. D. I. Abarbanel, and M. I. Rabinovich Self-organization in the olfactory system: One shot odor recognition in insects, Biol Cyber, 93 (6): 436-446 (2005), doi:10.1007/s00422-005-0019-7

# 17.91 MBody1.cc File Reference

This file contains the model definition of the mushroom body "MBody1" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol\_sim).

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

### **Functions**

- WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini (1.0/\_NLHI)
- WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini (5.0/\_NLB)
- void modelDefinition (NNmodel &model)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

### **Variables**

- NeuronModels::Poisson::ParamValues myPOI\_p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI\_ini (-60.0, 0,-10.0)
- NeuronModels::TraubMiles::ParamValues stdTM\_p (7.15, 50.0, 1.43,-95.0, 0.02672,-63.563, 0.143)
- NeuronModels::TraubMiles::VarValues stdTM\_ini (-60.0, 0.0529324, 0.3176767, 0.5961207)
- WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini (0.01)
- PostsynapticModels::ExpCond::ParamValues postExpPNKC (1.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues myPNLHI\_ini (0.0)

- PostsynapticModels::ExpCond::ParamValues postExpPNLHI (1.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p (-40.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpLHIKC (1.5,-92.0)
- WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p (50.0, 50.0, 50000.0, 100000.0, 200.0, 0.015, 0.0075, 33.33, 10.0, 0.00006)
- WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini (0.01, 0.01)
- PostsynapticModels::ExpCond::ParamValues postExpKCDN (5.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p (-30.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpDNDN (2.5,-92.0)

## 17.91.1 Detailed Description

This file contains the model definition of the mushroom body "MBody1" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol\_sim).

- 17.91.2 Function Documentation
- 17.91.2.1 void modelDefinition ( NNmodel & model )

- 17.91.2.2 WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini ( 5.0/\_NLB )
- 17.91.2.3 WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini ( 1.0/\_NLHI )
- 17.91.3 Variable Documentation
- 17.91.3.1 WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p(-30.0,50.0)
- 17.91.3.2 WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini(0.01,0.01)
- 17.91.3.3 WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p(50.0,50.0,50000.0,100000.0,200.0,0. ← 015,0.0075,33.33,10.0,0.00006)
- 17.91.3.4 WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p(-40.0,50.0)
- 17.91.3.5 WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini(0.01)
- 17.91.3.6 WeightUpdateModels::StaticPulse::VarValues myPNLHI\_ini(0.0)
- 17.91.3.7 NeuronModels::Poisson::VarValues myPOI\_ini(-60.0,0,-10.0)
- 17.91.3.8 NeuronModels::Poisson::ParamValues myPOI\_p(0.1,2.5,20.0,-60.0)
- 17.91.3.9 PostsynapticModels::ExpCond::ParamValues postExpDNDN(2.5,-92.0)
- 17.91.3.10 PostsynapticModels::ExpCond::ParamValues postExpKCDN(5.0,0.0)
- $17.91.3.11 \quad \textbf{PostsynapticModels::ExpCond::ParamValues postExpLHIKC} (1.5,-92.0)$
- 17.91.3.12 PostsynapticModels::ExpCond::ParamValues postExpPNKC(1.0,0.0)
- 17.91.3.13 PostsynapticModels::ExpCond::ParamValues postExpPNLHI(1.0,0.0)
- 17.91.3.14 NeuronModels::TraubMiles::VarValues stdTM\_ini(-60.0,0.0529324,0.3176767,0.5961207)
- 17.91.3.15 NeuronModels::TraubMiles::ParamValues stdTM\_p(7.15,50.0,1.43,-95.0,0.02672,-63.563,0.143)

## 17.92 MBody1.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

#### **Functions**

- WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini (1.0/\_NLHI)
- void modelDefinition (NNmodel &model)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

#### **Variables**

- NeuronModels::Poisson::ParamValues myPOI\_p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI ini (-60.0, 0,-10.0)
- NeuronModels::RulkovMap::ParamValues stdRMP p (60.0, 3.0,-2.468, 2.64)
- NeuronModels::RulkovMap::VarValues stdRMP\_ini (-60.0,-60.0)
- WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini (1.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNKC (1.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues myPNLHI\_ini (0.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNLHI (1.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p (-40.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpLHIKC (1.5,-92.0)
- WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p (100.0, 50.0, 50000.0, 100000.0, 200.0, 0.0015, 0.00075, 333.3, 10.0, 0.000006)
- WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini (0.01, 0.01)
- PostsynapticModels::ExpCond::ParamValues postExpKCDN (5.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p (-30.0, 50.0)
- WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini (0.01)
- PostsynapticModels::ExpCond::ParamValues postExpDNDN (2.5,-92.0)

# 17.92.1 Function Documentation

17.92.1.1 void modelDefinition ( NNmodel & model )

- 17.92.1.2 WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini ( 1.0/\_NLHI )
- 17.92.2 Variable Documentation
- 17.92.2.1 WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini(0.01)
- 17.92.2.2 WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p(-30.0,50.0)
- 17.92.2.3 WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini(0.01,0.01)
- 17.92.2.5 WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p(-40.0,50.0)
- 17.92.2.6 WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini(1.0)

```
17.92.2.7 WeightUpdateModels::StaticPulse::VarValues myPNLHI_ini(0.0)
17.92.2.8 NeuronModels::Poisson::VarValues myPOI_ini(-60.0,0,-10.0)
17.92.2.9 NeuronModels::Poisson::ParamValues myPOI_p(0.1,2.5,20.0,-60.0)
17.92.2.10 PostsynapticModels::ExpCond::ParamValues postExpDNDN(2.5,-92.0)
17.92.2.11 PostsynapticModels::ExpCond::ParamValues postExpKCDN(5.0,0.0)
17.92.2.12 PostsynapticModels::ExpCond::ParamValues postExpLHIKC(1.5,-92.0)
17.92.2.13 PostsynapticModels::ExpCond::ParamValues postExpPNKC(1.0,0.0)
17.92.2.14 PostsynapticModels::ExpCond::ParamValues postExpPNLHI(1.0,0.0)
17.92.2.15 NeuronModels::RulkovMap::VarValues stdRMP_ini(-60.0,-60.0)
17.92.2.16 NeuronModels::RulkovMap::ParamValues stdRMP_p(60.0,3.0,-2.468,2.64)
17.93 MBody1.cc File Reference
#include "modelSpec.h"
#include "global.h"
```

#include "sizes.h"

- WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini (1.0/\_NLHI)
- void modelDefinition (NNmodel &model)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

### Variables

- NeuronModels::Poisson::ParamValues myPOI\_p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI\_ini (-60.0, 0,-10.0)
- NeuronModels::RulkovMap::ParamValues stdRMP\_p (60.0, 3.0,-2.468, 2.64)
- NeuronModels::RulkovMap::VarValues stdRMP\_ini (-60.0,-60.0)
- WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini (1.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNKC (1.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues myPNLHI ini (0.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNLHI (1.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p (-40.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpLHIKC (1.5,-92.0)
- WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p (100.0, 50.0, 50000.0, 100000.0, 200.0, 0.0015, 0.00075, 333.3, 10.0, 0.000006)
- WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini (0.01, 0.01)
- PostsynapticModels::ExpCond::ParamValues postExpKCDN (5.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myDNDN p (-30.0, 50.0)
- WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini (0.01)
- PostsynapticModels::ExpCond::ParamValues postExpDNDN (2.5,-92.0)

#### 17.93.1 Function Documentation

# 17.93.1.1 void modelDefinition ( NNmodel & model )

```
17.93.1.2 WeightUpdateModels::StaticGraded::VarValues myLHIKC_ini ( 1.0/_NLHI )
17.93.2 Variable Documentation
17.93.2.1 WeightUpdateModels::StaticGraded::VarValues myDNDN_ini(0.01)
17.93.2.2 WeightUpdateModels::StaticGraded::ParamValues myDNDN_p(-30.0,50.0)
17.93.2.3 WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN_ini(0.01,0.01)
17.93.2.4 WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN_p(100.0,50.0,50000.0,100000.0,200.0,0. ←
         0015,0.00075,333.3,10.0,0.000006)
17.93.2.5 WeightUpdateModels::StaticGraded::ParamValues myLHIKC_p(-40.0,50.0)
17.93.2.6 WeightUpdateModels::StaticPulse::VarValues myPNKC_ini(1.0)
17.93.2.7 WeightUpdateModels::StaticPulse::VarValues myPNLHI_ini(0.0)
17.93.2.8 NeuronModels::Poisson::VarValues myPOl_ini(-60.0,0,-10.0)
17.93.2.9 NeuronModels::Poisson::ParamValues myPOl_p(0.1,2.5,20.0,-60.0)
17.93.2.10 PostsynapticModels::ExpCond::ParamValues postExpDNDN(2.5,-92.0)
17.93.2.11 PostsynapticModels::ExpCond::ParamValues postExpKCDN(5.0,0.0)
17.93.2.12 PostsynapticModels::ExpCond::ParamValues postExpLHIKC(1.5,-92.0)
17.93.2.13 PostsynapticModels::ExpCond::ParamValues postExpPNKC(1.0,0.0)
17.93.2.14 PostsynapticModels::ExpCond::ParamValues postExpPNLHI(1.0,0.0)
17.93.2.15 NeuronModels::RulkovMap::VarValues stdRMP_ini(-60.0,-60.0)
17.93.2.16 NeuronModels::RulkovMap::ParamValues stdRMP_p(60.0,3.0,-2.468,2.64)
```

# 17.94 MBody\_delayedSyn.cc File Reference

This file contains the model definition of the mushroom body "MBody\_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol sim).

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

### **Functions**

- WeightUpdateModels::StaticGraded::VarValues myLHIKC ini (1.0/ NLHI)
- WeightUpdateModels::StaticGraded::VarValues myDNDN ini (5.0/ NLB)
- void modelDefinition (NNmodel &model)

This function defines the MBody delayedSyn model, and it is a good example of how networks should be defined.

#### **Variables**

- NeuronModels::Poisson::ParamValues myPOI p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI ini (-60.0, 0,-10.0)
- NeuronModels::TraubMiles::ParamValues stdTM\_p (7.15, 50.0, 1.43,-95.0, 0.02672,-63.563, 0.143)

- NeuronModels::TraubMiles::VarValues stdTM\_ini (-60.0, 0.0529324, 0.3176767, 0.5961207)
- WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini (0.01)
- PostsynapticModels::ExpCond::ParamValues postExpPNKC (1.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues myPNLHI ini (0.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNLHI (1.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p (-40.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpLHIKC (1.5,-92.0)
- WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p (50.0, 50.0, 50000.0, 100000.0, 200.0, 0.015, 0.0075, 33.33, 10.0, 0.00006)
- WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini (0.01, 0.01)
- PostsynapticModels::ExpCond::ParamValues postExpKCDN (5.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p (-30.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpDNDN (2.5,-92.0)

#### 17.94.1 Detailed Description

This file contains the model definition of the mushroom body "MBody\_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol sim).

- 17.94.2 Function Documentation
- 17.94.2.1 void modelDefinition ( NNmodel & model )

- 17.94.2.2 WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini ( 5.0/ \_NLB )
- 17.94.2.3 WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini ( 1.0/\_NLHI )
- 17.94.3 Variable Documentation
- 17.94.3.1 WeightUpdateModels::StaticGraded::ParamValues myDNDN\_p(-30.0,50.0)
- 17.94.3.2 WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini(0.01,0.01)
- 17.94.3.4 WeightUpdateModels::StaticGraded::ParamValues myLHIKC\_p(-40.0,50.0)
- 17.94.3.5 WeightUpdateModels::StaticPulse::VarValues myPNKC\_ini(0.01)
- 17.94.3.6 WeightUpdateModels::StaticPulse::VarValues myPNLHI\_ini(0.0)
- 17.94.3.7 NeuronModels::Poisson::VarValues myPOl\_ini(-60.0,0,-10.0)
- 17.94.3.8 NeuronModels::Poisson::ParamValues myPOI\_p(0.1,2.5,20.0,-60.0)
- 17.94.3.9 PostsynapticModels::ExpCond::ParamValues postExpDNDN(2.5,-92.0)
- 17.94.3.10 PostsynapticModels::ExpCond::ParamValues postExpKCDN(5.0,0.0)
- 17.94.3.11 PostsynapticModels::ExpCond::ParamValues postExpLHIKC(1.5,-92.0)
- 17.94.3.12 PostsynapticModels::ExpCond::ParamValues postExpPNKC(1.0,0.0)
- 17.94.3.13 PostsynapticModels::ExpCond::ParamValues postExpPNLHI(1.0,0.0)

17.94.3.14 NeuronModels::TraubMiles::VarValues stdTM\_ini(-60.0,0.0529324,0.3176767,0.5961207)

17.94.3.15 NeuronModels::TraubMiles::ParamValues stdTM\_p(7.15,50.0,1.43,-95.0,0.02672,-63.563,0.143)

# 17.95 MBody\_individualID.cc File Reference

This file contains the model definition of the mushroom body "MBody\_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol\_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC.

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

#### **Functions**

- WeightUpdateModels::StaticGraded::VarValues myLHIKC\_ini (1.0/\_NLHI)
- WeightUpdateModels::StaticGraded::VarValues myDNDN\_ini (5.0/\_NLB)
- void modelDefinition (NNmodel &model)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

## **Variables**

- NeuronModels::Poisson::ParamValues myPOI\_p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI\_ini (-60.0, 0,-10.0)
- NeuronModels::TraubMiles::ParamValues stdTM\_p (7.15, 50.0, 1.43,-95.0, 0.02672,-63.563, 0.143)
- NeuronModels::TraubMiles::VarValues stdTM\_ini (-60.0, 0.0529324, 0.3176767, 0.5961207)
- WeightUpdateModels::StaticPulse::VarValues myPNKC ini (gPNKC GLOBAL)
- PostsynapticModels::ExpCond::ParamValues postExpPNKC (1.0, 0.0)
- WeightUpdateModels::StaticPulse::VarValues myPNLHI\_ini (0.0)
- PostsynapticModels::ExpCond::ParamValues postExpPNLHI (1.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myLHIKC p (-40.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpLHIKC (1.5,-92.0)
- WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN\_p (50.0, 50.0, 50000.0, 100000.0, 200.0, 0.015, 0.0075, 33.33, 10.0, 0.00006)
- WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN\_ini (0.01, 0.01)
- PostsynapticModels::ExpCond::ParamValues postExpKCDN (5.0, 0.0)
- WeightUpdateModels::StaticGraded::ParamValues myDNDN p (-30.0, 50.0)
- PostsynapticModels::ExpCond::ParamValues postExpDNDN (2.5,-92.0)

# 17.95.1 Detailed Description

This file contains the model definition of the mushroom body "MBody\_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol\_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC.

### 17.95.2 Function Documentation

17.95.2.1 void modelDefinition ( NNmodel & model )

```
17.95.2.2 WeightUpdateModels::StaticGraded::VarValues myDNDN_ini ( 5.0/ _NLB )
17.95.2.3 WeightUpdateModels::StaticGraded::VarValues myLHIKC_ini ( 1.0/_NLHI )
17.95.3 Variable Documentation
17.95.3.1 WeightUpdateModels::StaticGraded::ParamValues myDNDN_p(-30.0,50.0)
17.95.3.2 WeightUpdateModels::PiecewiseSTDP::VarValues myKCDN_ini(0.01,0.01)
17.95.3.3 WeightUpdateModels::PiecewiseSTDP::ParamValues myKCDN_p(50.0,50.0,50000.0,100000.0,200.0,0...
         015,0.0075,33.33,10.0,0.00006)
17.95.3.4 WeightUpdateModels::StaticGraded::ParamValues myLHIKC_p(-40.0,50.0)
17.95.3.5 WeightUpdateModels::StaticPulse::VarValues myPNKC_ini(gPNKC_GLOBAL)
17.95.3.6 WeightUpdateModels::StaticPulse::VarValues myPNLHI_ini(0.0)
17.95.3.7 NeuronModels::Poisson::VarValues myPOl_ini(-60.0,0,-10.0)
17.95.3.8 NeuronModels::Poisson::ParamValues myPOl_p(0.1,2.5,20.0,-60.0)
17.95.3.9 PostsynapticModels::ExpCond::ParamValues postExpDNDN(2.5,-92.0)
17.95.3.10 PostsynapticModels::ExpCond::ParamValues postExpKCDN(5.0,0.0)
17.95.3.11 PostsynapticModels::ExpCond::ParamValues postExpLHIKC(1.5,-92.0)
17.95.3.12 PostsynapticModels::ExpCond::ParamValues postExpPNKC(1.0,0.0)
17.95.3.13 PostsynapticModels::ExpCond::ParamValues postExpPNLHI(1.0,0.0)
17.95.3.14 NeuronModels::TraubMiles::VarValues stdTM_ini(-60.0,0.0529324,0.3176767,0.5961207)
17.95.3.15 NeuronModels::TraubMiles::ParamValues stdTM_p(7.15,50.0,1.43,-95.0,0.02672,-63.563,0.143)
```

# 17.96 MBody\_userdef.cc File Reference

This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol\_sim).

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

### Classes

- · class PiecewiseSTDPUserDef
- class StaticPulseUserDef
- · class StaticGradedUserDef
- class ExpCondUserDef

#### Macros

#define TIMING

- IMPLEMENT MODEL (PiecewiseSTDPUserDef)
- IMPLEMENT\_MODEL (StaticPulseUserDef)
- IMPLEMENT\_MODEL (StaticGradedUserDef)
- IMPLEMENT MODEL (ExpCondUserDef)
- StaticGradedUserDef::VarValues myLHIKC ini (1.0/ NLHI)
- StaticGradedUserDef::VarValues myDNDN ini (5.0/ NLB)
- void modelDefinition (NNmodel &model)

This function defines the MBody1 model with user defined synapses.

#### **Variables**

- NeuronModels::Poisson::ParamValues myPOI\_p (0.1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI\_ini (-60.0, 0,-10.0)
- NeuronModels::TraubMiles::ParamValues stdTM p (7.15, 50.0, 1.43,-95.0, 0.02672,-63.563, 0.143)
- NeuronModels::TraubMiles::VarValues stdTM ini (-60.0, 0.0529324, 0.3176767, 0.5961207)
- StaticPulseUserDef::VarValues myPNKC ini (0.01)
- ExpCondUserDef::ParamValues postExpPNKC (1.0, 0.0)
- StaticPulseUserDef::VarValues myPNLHI\_ini (0.0)
- ExpCondUserDef::ParamValues postExpPNLHI (1.0, 0.0)
- StaticGradedUserDef::ParamValues myLHIKC\_p (-40.0, 50.0)
- ExpCondUserDef::ParamValues postExpLHIKC (1.5,-92.0)
- PiecewiseSTDPUserDef::ParamValues myKCDN\_p (50.0, 50.0, 50000.0, 100000.0, 200.0, 0.015, 0.0075, 33.33, 10.0, 0.00006)
- PiecewiseSTDPUserDef::VarValues myKCDN ini (0.01, 0.01)
- ExpCondUserDef::ParamValues postExpKCDN (5.0, 0.0)
- StaticGradedUserDef::ParamValues myDNDN\_p (-30.0, 50.0)
- ExpCondUserDef::ParamValues postExpDNDN (2.5,-92.0)
- scalar \* gpPNKC = new scalar[\_NAL\*\_NMB]
- scalar \* gpKCDN = new scalar[ NMB\* NLB]

### 17.96.1 Detailed Description

This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol\_sim).

# 17.96.2 Macro Definition Documentation

```
17.96.2.1 #define TIMING
```

17.96.3 Function Documentation

17.96.3.1 IMPLEMENT\_MODEL ( PiecewiseSTDPUserDef )

17.96.3.2 IMPLEMENT\_MODEL ( StaticPulseUserDef )

17.96.3.3 IMPLEMENT\_MODEL ( StaticGradedUserDef )

17.96.3.4 IMPLEMENT\_MODEL ( ExpCondUserDef )

17.96.3.5 void modelDefinition ( NNmodel & model )

This function defines the MBody1 model with user defined synapses.

```
17.96.3.6 StaticGradedUserDef::VarValues myDNDN_ini ( 5.0/ _NLB )
17.96.3.7 StaticGradedUserDef::VarValues myLHIKC_ini ( 1.0/ _NLHI )
17.96.4 Variable Documentation
17.96.4.1 scalar* gpKCDN = new scalar[_NMB*_NLB]
17.96.4.2 scalar* gpPNKC = new scalar[_NAL*_NMB]
17.96.4.3 StaticGradedUserDef::ParamValues myDNDN_p(-30.0,50.0)
17.96.4.4 PiecewiseSTDPUserDef::VarValues myKCDN_ini(0.01,0.01)
17.96.4.5 PiecewiseSTDPUserDef::ParamValues myKCDN_p(50.0,50.0,50000.0,100000.0,200.0,0.015,0.0075,33.33,10. ←
         0,0.00006)
17.96.4.6 StaticGradedUserDef::ParamValues myLHIKC_p(-40.0,50.0)
17.96.4.7 StaticPulseUserDef::VarValues myPNKC_ini(0.01)
17.96.4.8 StaticPulseUserDef::VarValues myPNLHI_ini(0.0)
17.96.4.9 NeuronModels::Poisson::VarValues myPOl_ini(-60.0,0,-10.0)
17.96.4.10 NeuronModels::Poisson::ParamValues myPOl_p(0.1,2.5,20.0,-60.0)
17.96.4.11 ExpCondUserDef::ParamValues postExpDNDN(2.5,-92.0)
17.96.4.12 ExpCondUserDef::ParamValues postExpKCDN(5.0,0.0)
17.96.4.13 ExpCondUserDef::ParamValues postExpLHIKC(1.5,-92.0)
17.96.4.14 ExpCondUserDef::ParamValues postExpPNKC(1.0,0.0)
17.96.4.15 ExpCondUserDef::ParamValues postExpPNLHI(1.0,0.0)
17.96.4.16 NeuronModels::TraubMiles::VarValues stdTM_ini(-60.0,0.0529324,0.3176767,0.5961207)
17.96.4.17 NeuronModels::TraubMiles::ParamValues stdTM_p(7.15,50.0,1.43,-95.0,0.02672,-63.563,0.143)
17.97 model.cc File Reference
#include "modelSpec.h"
```

void modelDefinition (NNmodel &model)

# **Variables**

- double neuron\_ini [1]
- double synapses\_ini [1]

### 17.97.1 Function Documentation

17.97.1.1 void modelDefinition ( NNmodel & model )

```
17.97.2 Variable Documentation
17.97.2.1 double neuron_ini[1]
Initial value:
= {
17.97.2.2 double synapses_ini[1]
Initial value:
= {
1.0
17.98 model.cc File Reference
#include "modelSpec.h"
Functions
    • void modelDefinition (NNmodel &model)
Variables
    • double neuron_ini [1]
    • double synapses_ini [1]
17.98.1 Function Documentation
17.98.1.1 void modelDefinition ( NNmodel & model )
17.98.2 Variable Documentation
17.98.2.1 double neuron_ini[1]
Initial value:
= {
17.98.2.2 double synapses_ini[1]
Initial value:
= {
1.0
17.99 model.cc File Reference
#include "modelSpec.h"
```

• void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_ini [1]
- double synapses\_ini [1]

## 17.99.1 Function Documentation

```
17.99.1.1 void modelDefinition ( NNmodel & model )
```

17.99.2 Variable Documentation

17.99.2.1 double neuron\_ini[1]

## Initial value:

17.99.2.2 double synapses\_ini[1]

## Initial value:

```
= {
 1.0
}
```

# 17.100 model.cc File Reference

```
#include "modelSpec.h"
```

### **Functions**

· void modelDefinition (NNmodel &model)

### Variables

- double neuron\_ini [1]
- double synapses\_ini [1]

# 17.100.1 Function Documentation

17.100.1.1 void modelDefinition ( NNmodel & model )

17.100.2 Variable Documentation

17.100.2.1 double neuron\_ini[1]

# Initial value:

```
= { 0.0
```

```
17.100.2.2 double synapses_ini[1]
```

## Initial value:

```
= {
 1.0
```

## 17.101 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

# **Variables**

- double neuron\_ini [1]
- double synapses\_ini [1]

# 17.101.1 Function Documentation

17.101.1.1 void modelDefinition ( NNmodel & model )

17.101.2 Variable Documentation

17.101.2.1 double neuron\_ini[1]

## Initial value:

```
= {
```

17.101.2.2 double synapses\_ini[1]

# Initial value:

```
= {
 1.0
```

# 17.102 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

# Variables

• double neuron\_ini [2]

```
17.102.1 Function Documentation
17.102.1.1 void modelDefinition ( NNmodel & model )
17.102.2 Variable Documentation
17.102.2.1 double neuron_ini[2]
Initial value:
    0.0,
    0.0
17.103 model.cc File Reference
#include "modelSpec.h"
Functions

    void modelDefinition (NNmodel &model)

Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.103.1 Function Documentation
17.103.1.1 void modelDefinition ( NNmodel & model )
17.103.2 Variable Documentation
17.103.2.1 double neuron_ini[2]
Initial value:
    0.0,
    0.0
17.103.2.2 double synapses_ini[1]
Initial value:
   0.0
17.104 model.cc File Reference
```

#include "modelSpec.h"

void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_ini [2]
- double synapses\_ini [1]

## 17.104.1 Function Documentation

```
17.104.1.1 void modelDefinition ( NNmodel & model )
```

17.104.2 Variable Documentation

17.104.2.1 double neuron\_ini[2]

## Initial value:

17.104.2.2 double synapses\_ini[1]

### Initial value:

```
= {
 0.0
}
```

# 17.105 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

## Variables

- double neuron\_ini [2]
- double synapses\_ini [1]

# 17.105.1 Function Documentation

17.105.1.1 void modelDefinition ( NNmodel & model )

17.105.2 Variable Documentation

17.105.2.1 double neuron\_ini[2]

# Initial value:

```
= {
0.0,
^ ^ 0
   0.0
17.105.2.2 double synapses_ini[1]
Initial value:
= {
17.106 model.cc File Reference
#include "modelSpec.h"
Functions
    • void modelDefinition (NNmodel &model)
Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.106.1 Function Documentation
17.106.1.1 void modelDefinition ( NNmodel & model )
17.106.2 Variable Documentation
17.106.2.1 double neuron_ini[2]
Initial value:
   0.0,
17.106.2.2 double synapses_ini[1]
```

# Initial value:

= { 0.0 }

## 17.107 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

#### **Variables**

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

## 17.107.1 Function Documentation

```
17.107.1.1 void modelDefinition ( NNmodel & model )
```

17.107.2 Variable Documentation

17.107.2.1 double neuron\_ini[2]

#### Initial value:

```
= {
    0.0,
    0.0
```

17.107.2.2 double synapses\_ini[1]

## Initial value:

```
= {
 0.0
```

## 17.108 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

#### **Variables**

```
• double neuron_p [1]
```

- double neuron\_p2 [1]
- double neuron\_ini [2]
- double synapses\_ini [1]

## 17.108.1 Function Documentation

17.108.1.1 void modelDefinition ( NNmodel & model )

17.108.2 Variable Documentation

17.108.2.1 double neuron\_ini[2]

## Initial value:

```
17.108.2.2 double neuron_p[1]
```

```
Initial value:
```

```
={
1.0
```

17.108.2.3 double neuron\_p2[1]

## Initial value:

```
={
2.0
```

17.108.2.4 double synapses\_ini[1]

#### Initial value:

```
= {
 0.0
```

## 17.109 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

#### **Variables**

- double neuron\_p [1]
- double neuron\_p2 [1]
- double neuron\_ini [2]
- double synapses\_ini [1]

# 17.109.1 Function Documentation

17.109.1.1 void modelDefinition ( NNmodel & model )

17.109.2 Variable Documentation

17.109.2.1 double neuron\_ini[2]

# Initial value:

= { 0.0

```
17.109.2.2 double neuron_p[1]
Initial value:
={
1.0
17.109.2.3 double neuron_p2[1]
Initial value:
={
2.0
17.109.2.4 double synapses_ini[1]
Initial value:
= {
17.110 model.cc File Reference
#include "modelSpec.h"
Functions

    void modelDefinition (NNmodel &model)

Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.110.1 Function Documentation
17.110.1.1 void modelDefinition ( NNmodel & model )
17.110.2 Variable Documentation
17.110.2.1 double neuron_ini[2]
Initial value:
= {
0.0,
   0.0
17.110.2.2 double synapses_ini[1]
Initial value:
```

## 17.111 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_ini [2]
- double synapses\_ini [1]

#### 17.111.1 Function Documentation

```
17.111.1.1 void modelDefinition ( NNmodel & model )
```

17.111.2 Variable Documentation

17.111.2.1 double neuron\_ini[2]

## Initial value:

# 17.111.2.2 double synapses\_ini[1]

# Initial value:

```
= {
```

# 17.112 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

# Variables

- double neuron\_ini [2]
- double synapses\_ini [1]

```
17.112.1 Function Documentation
17.112.1.1 void modelDefinition ( NNmodel & model )
17.112.2 Variable Documentation
17.112.2.1 double neuron_ini[2]
Initial value:
    0.0,
17.112.2.2 double synapses_ini[1]
Initial value:
= { 0.0
17.113 model.cc File Reference
#include "modelSpec.h"
Functions
    • void modelDefinition (NNmodel &model)
Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.113.1 Function Documentation
17.113.1.1 void modelDefinition ( NNmodel & model )
17.113.2 Variable Documentation
17.113.2.1 double neuron_ini[2]
Initial value:
   0.0,
    0.0
17.113.2.2 double synapses_ini[1]
Initial value:
= {
```

## 17.114 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_p [1]
- double neuron\_p2 [1]
- double neuron\_ini [2]
- double synapses\_ini [1]

## 17.114.1 Function Documentation

```
17.114.1.1 void modelDefinition ( NNmodel & model )
```

17.114.2 Variable Documentation

17.114.2.1 double neuron\_ini[2]

#### Initial value:

## 17.114.2.2 double neuron\_p[1]

# Initial value:

```
= {
```

# 17.114.2.3 double neuron\_p2[1]

#### Initial value:

```
={
2.0
}
```

# 17.114.2.4 double synapses\_ini[1]

## Initial value:

```
= { 0.0
```

# 17.115 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_p [1]
- double neuron\_p2 [1]
- double neuron\_ini [2]
- double synapses\_ini [1]

#### 17.115.1 Function Documentation

```
17.115.1.1 void modelDefinition ( NNmodel & model )
```

17.115.2 Variable Documentation

17.115.2.1 double neuron\_ini[2]

#### Initial value:

```
= {
    0.0,
    0.0
```

17.115.2.2 double neuron\_p[1]

## Initial value:

```
={
1.0
}
```

17.115.2.3 double neuron\_p2[1]

#### Initial value:

```
= { 2.0
```

17.115.2.4 double synapses\_ini[1]

#### Initial value:

```
= {
 0.0
```

# 17.116 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

## Variables

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

#### 17.116.1 Function Documentation

```
17.116.1.1 void modelDefinition ( NNmodel & model )
```

17.116.2 Variable Documentation

17.116.2.1 double neuron\_ini[2]

## Initial value:

## 17.116.2.2 double synapses\_ini[1]

#### Initial value:

```
= {
 0.0
```

#### 17.117 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

#### Variables

- double neuron\_ini [2]
- double synapses\_p [1]
- double synapses\_ini [1]

## 17.117.1 Function Documentation

17.117.1.1 void modelDefinition ( NNmodel & model )

17.117.2 Variable Documentation

17.117.2.1 double neuron\_ini[2]

# Initial value:

```
17.117.2.2 double synapses_ini[1]
Initial value:
= { 0.0
17.117.2.3 double synapses_p[1]
Initial value:
= { 0.0
17.118 model.cc File Reference
#include "modelSpec.h"
Functions
    • void modelDefinition (NNmodel &model)
Variables
    • double neuron_ini [2]
    • double synapses_p [1]
    • double synapses_ini [1]
17.118.1 Function Documentation
17.118.1.1 void modelDefinition ( NNmodel & model )
17.118.2 Variable Documentation
17.118.2.1 double neuron_ini[2]
Initial value:
= {
    0.0,
    0.0
17.118.2.2 double synapses_ini[1]
Initial value:
= {
17.118.2.3 double synapses_p[1]
```

Initial value:

= { 0.0

# 17.119 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_ini [2]
- double synapses\_p [1]
- double synapses\_ini [1]

# 17.119.1 Function Documentation

```
17.119.1.1 void modelDefinition ( NNmodel & model )
```

17.119.2 Variable Documentation

17.119.2.1 double neuron\_ini[2]

## Initial value:

```
= {
    0.0,
    0.0
```

17.119.2.2 double synapses\_ini[1]

# Initial value:

```
= {
```

17.119.2.3 double synapses\_p[1]

## Initial value:

## 17.120 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

#### **Variables**

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

# 17.120.1 Function Documentation

```
17.120.1.1 void modelDefinition ( NNmodel & model )
```

17.120.2 Variable Documentation

17.120.2.1 double neuron\_ini[2]

#### Initial value:

17.120.2.2 double synapses\_ini[1]

## Initial value:

```
= {
 0.0
```

## 17.121 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

• void modelDefinition (NNmodel &model)

## Variables

- double neuron\_ini [2]
- double synapses\_ini [1]

#### 17.121.1 Function Documentation

17.121.1.1 void modelDefinition ( NNmodel & model )

17.121.2 Variable Documentation

17.121.2.1 double neuron\_ini[2]

## Initial value:

17.121.2.2 double synapses\_ini[1]

Initial value:

```
= {
 0.0
```

## 17.122 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

· void modelDefinition (NNmodel &model)

#### Variables

- double neuron\_ini [2]
- double synapses\_ini [1]
- 17.122.1 Function Documentation
- 17.122.1.1 void modelDefinition ( NNmodel & model )
- 17.122.2 Variable Documentation
- 17.122.2.1 double neuron\_ini[2]

## Initial value:

17.122.2.2 double synapses\_ini[1]

#### Initial value:

```
= {
 0.0
```

# 17.123 model.cc File Reference

```
#include "modelSpec.h"
```

# Functions

• void modelDefinition (NNmodel &model)

## **Variables**

```
• double neuron_ini [2]
```

- double synapses\_p [1]
- double synapses\_ini [1]

#### 17.123.1 Function Documentation

```
17.123.1.1 void modelDefinition ( NNmodel & model )
```

17.123.2 Variable Documentation

17.123.2.1 double neuron\_ini[2]

## Initial value:

17.123.2.2 double synapses\_ini[1]

#### Initial value:

17.123.2.3 double synapses\_p[1]

# Initial value:

```
= {
```

#### 17.124 model.cc File Reference

```
#include "modelSpec.h"
```

## **Functions**

void modelDefinition (NNmodel &model)

## **Variables**

- double neuron\_ini [2]
- double synapses\_p [1]
- double synapses\_ini [1]

```
17.124.1 Function Documentation
17.124.1.1 void modelDefinition ( NNmodel & model )
17.124.2 Variable Documentation
17.124.2.1 double neuron_ini[2]
Initial value:
   0.0,
17.124.2.2 double synapses_ini[1]
Initial value:
= {
17.124.2.3 double synapses_p[1]
Initial value:
= { 0.0
17.125 model.cc File Reference
#include "modelSpec.h"
Functions
    · void modelDefinition (NNmodel &model)
Variables
    • double neuron_p [1]
    • double neuron_p2 [1]
    • double neuron_ini [2]
    • double synapses_ini [1]
17.125.1 Function Documentation
17.125.1.1 void modelDefinition ( NNmodel & model )
17.125.2 Variable Documentation
```

# Initial value:

17.125.2.1 double neuron\_ini[2]

```
= {
0.0,
   0.0
17.125.2.2 double neuron_p[1]
Initial value:
    1.0
17.125.2.3 double neuron_p2[1]
Initial value:
   2.0
17.125.2.4 double synapses_ini[1]
Initial value:
= { 0.0
17.126 model.cc File Reference
#include "modelSpec.h"
Functions

    void modelDefinition (NNmodel &model)

Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.126.1 Function Documentation
17.126.1.1 void modelDefinition ( NNmodel & model )
17.126.2 Variable Documentation
17.126.2.1 double neuron_ini[2]
Initial value:
= {
0.0,
0.0
```

17.126.2.2 double synapses\_ini[1]

Initial value:

```
= {
 0.0
```

## 17.127 model.cc File Reference

```
#include "modelSpec.h"
```

#### **Functions**

· void modelDefinition (NNmodel &model)

#### Variables

- double neuron\_ini [2]
- double synapses\_ini [1]
- 17.127.1 Function Documentation
- 17.127.1.1 void modelDefinition ( NNmodel & model )
- 17.127.2 Variable Documentation
- 17.127.2.1 double neuron\_ini[2]

## Initial value:

17.127.2.2 double synapses\_ini[1]

#### Initial value:

```
= {
 0.0
```

# 17.128 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

• class Neuron

#### **Functions**

```
• IMPLEMENT_MODEL (Neuron)
```

```
• void modelDefinition (NNmodel &model)
```

```
17.128.1 Function Documentation

17.128.1.1 IMPLEMENT_MODEL ( Neuron )

17.128.1.2 void modelDefinition ( NNmodel & model )

17.129 model_new.cc File Reference

#include "modelSpec.h"
```

#### Classes

• class Neuron

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- void modelDefinition (NNmodel &model)

```
17.129.1 Function Documentation

17.129.1.1 IMPLEMENT_MODEL ( Neuron )

17.129.1.2 void modelDefinition ( NNmodel & model )
```

```
#include "modelSpec.h"
```

17.130 model\_new.cc File Reference

## Classes

• class Neuron

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- void modelDefinition (NNmodel &model)

```
17.130.1 Function Documentation
```

```
17.130.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.130.1.2 void modelDefinition ( NNmodel & model )

# 17.131 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

• class Neuron

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- void modelDefinition (NNmodel &model)

```
17.131.1 Function Documentation
```

```
17.131.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.131.1.2 void modelDefinition ( NNmodel & model )

17.132 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

· class Neuron

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- void modelDefinition (NNmodel &model)

```
17.132.1 Function Documentation
```

```
17.132.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.132.1.2 void modelDefinition ( NNmodel & model )

# 17.133 model\_new.cc File Reference

```
#include "modelSpec.h"
```

## Classes

class Neuron

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- void modelDefinition (NNmodel &model)

```
17.133.1 Function Documentation
17.133.1.1 IMPLEMENT_MODEL ( Neuron )
17.133.1.2 void modelDefinition ( NNmodel & model )
17.134 model new.cc File Reference
#include "modelSpec.h"
Classes

    class Neuron

    • class WeightUpdateModel
Functions
   • IMPLEMENT MODEL (Neuron)

    IMPLEMENT_MODEL (WeightUpdateModel)

    void modelDefinition (NNmodel &model)

17.134.1 Function Documentation
17.134.1.1 IMPLEMENT_MODEL ( Neuron )
17.134.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.134.1.3 void modelDefinition ( NNmodel & model )
17.135 model_new.cc File Reference
#include "modelSpec.h"
Classes
    · class Neuron
    · class WeightUpdateModel
Functions

    IMPLEMENT_MODEL (Neuron)

    IMPLEMENT MODEL (WeightUpdateModel)

   • void modelDefinition (NNmodel &model)
17.135.1 Function Documentation
17.135.1.1 IMPLEMENT_MODEL ( Neuron )
17.135.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.135.1.3 void modelDefinition ( NNmodel & model )
```

## 17.136 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- · class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.136.1 Function Documentation

17.136.1.1 IMPLEMENT_MODEL ( Neuron )

17.136.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )

17.136.1.3 void modelDefinition ( NNmodel & model )
```

17.137 model\_new.cc File Reference

```
#include "modelSpec.h"
```

# Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.137.1 Function Documentation

17.137.1.1 IMPLEMENT_MODEL ( Neuron )

17.137.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )

17.137.1.3 void modelDefinition ( NNmodel & model )

17.138 model_new.cc File Reference
```

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.138.1 Function Documentation

17.138.1.1 IMPLEMENT_MODEL( Neuron )

17.138.1.2 IMPLEMENT_MODEL( WeightUpdateModel )

17.138.1.3 void modelDefinition( NNmodel & model )

17.139 model_new.cc File Reference

#include "modelSpec.h"
```

## Classes

- class Neuron
- class WeightUpdateModel

# **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.139.1 Function Documentation

17.139.1.1 IMPLEMENT_MODEL ( Neuron )

17.139.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )

17.139.1.3 void modelDefinition ( NNmodel & model )

17.140 model_new.cc File Reference

#include "modelSpec.h"
```

## Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

#### 17.140.1 Function Documentation

```
17.140.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.140.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.140.1.3 void modelDefinition ( NNmodel & model )

#### 17.141 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

#### 17.141.1 Function Documentation

```
17.141.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.141.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.141.1.3 void modelDefinition ( NNmodel & model )

#### 17.142 model\_new.cc File Reference

```
#include "modelSpec.h"
```

# Classes

- class Neuron
- · class WeightUpdateModel

## **Functions**

- IMPLEMENT MODEL (Neuron)
- IMPLEMENT MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.142.1 Function Documentation
17.142.1.1 IMPLEMENT_MODEL ( Neuron )
17.142.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.142.1.3 void modelDefinition ( NNmodel & model )
17.143 model new.cc File Reference
#include "modelSpec.h"
Classes

    class Neuron

   • class WeightUpdateModel
Functions
   • IMPLEMENT_MODEL (Neuron)
   • IMPLEMENT_MODEL (WeightUpdateModel)

    void modelDefinition (NNmodel &model)

17.143.1 Function Documentation
17.143.1.1 IMPLEMENT_MODEL ( Neuron )
17.143.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.143.1.3 void modelDefinition ( NNmodel & model )
17.144 model_new.cc File Reference
#include "modelSpec.h"
Classes
    · class Neuron
    · class WeightUpdateModel
Functions

    IMPLEMENT MODEL (Neuron)

    • IMPLEMENT_MODEL (WeightUpdateModel)
   • void modelDefinition (NNmodel &model)
17.144.1 Function Documentation
17.144.1.1 IMPLEMENT_MODEL ( Neuron )
17.144.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
```

```
17.144.1.3 void modelDefinition ( NNmodel & model )
```

## 17.145 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.145.1 Function Documentation
```

```
17.145.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.145.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.145.1.3 void modelDefinition ( NNmodel & model )

## 17.146 model\_new.cc File Reference

```
#include "modelSpec.h"
```

# Classes

- class Neuron
- · class WeightUpdateModel

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

## 17.146.1 Function Documentation

```
17.146.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.146.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.146.1.3 void modelDefinition ( NNmodel & model )

## 17.147 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.147.1 Function Documentation
```

```
17.147.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.147.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.147.1.3 void modelDefinition ( NNmodel & model )

17.148 model\_new.cc File Reference

```
#include "modelSpec.h"
```

## Classes

- class Neuron
- class WeightUpdateModel

# Functions

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.148.1 Function Documentation
```

```
17.148.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.148.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.148.1.3 void modelDefinition ( NNmodel & model )

17.149 model\_new.cc File Reference

```
#include "modelSpec.h"
```

## Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.149.1 Function Documentation
```

```
17.149.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.149.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.149.1.3 void modelDefinition ( NNmodel & model )

17.150 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.150.1 Function Documentation
```

```
17.150.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.150.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.150.1.3 void modelDefinition ( NNmodel & model )

17.151 model\_new.cc File Reference

```
#include "modelSpec.h"
```

# Classes

- class Neuron
- · class WeightUpdateModel

## **Functions**

- IMPLEMENT MODEL (Neuron)
- IMPLEMENT MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.151.1 Function Documentation
17.151.1.1 IMPLEMENT_MODEL ( Neuron )
17.151.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.151.1.3 void modelDefinition ( NNmodel & model )
17.152 model new.cc File Reference
#include "modelSpec.h"
Classes

    class Neuron

   • class WeightUpdateModel
Functions
   • IMPLEMENT_MODEL (Neuron)
   • IMPLEMENT_MODEL (WeightUpdateModel)

    void modelDefinition (NNmodel &model)

17.152.1 Function Documentation
17.152.1.1 IMPLEMENT_MODEL ( Neuron )
17.152.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
17.152.1.3 void modelDefinition ( NNmodel & model )
17.153 model_new.cc File Reference
#include "modelSpec.h"
Classes
    · class Neuron
    · class WeightUpdateModel
Functions

    IMPLEMENT MODEL (Neuron)

    • IMPLEMENT_MODEL (WeightUpdateModel)
   • void modelDefinition (NNmodel &model)
17.153.1 Function Documentation
17.153.1.1 IMPLEMENT_MODEL ( Neuron )
17.153.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )
```

```
17.153.1.3 void modelDefinition ( NNmodel & model )
```

## 17.154 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.154.1 Function Documentation
```

```
17.154.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.154.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.154.1.3 void modelDefinition ( NNmodel & model )

# 17.155 model\_new.cc File Reference

```
#include "modelSpec.h"
```

# Classes

- class Neuron
- · class WeightUpdateModel

## **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

## 17.155.1 Function Documentation

```
17.155.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.155.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.155.1.3 void modelDefinition ( NNmodel & model )

## 17.156 model\_new.cc File Reference

```
#include "modelSpec.h"
```

#### Classes

- class Neuron
- · class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.156.1 Function Documentation

17.156.1.1 IMPLEMENT_MODEL ( Neuron )

17.156.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )

17.156.1.3 void modelDefinition ( NNmodel & model )

17.157 model_new.cc File Reference
```

## Classes

- class Neuron
- class WeightUpdateModel

#include "modelSpec.h"

# Functions

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

```
17.157.1 Function Documentation

17.157.1.1 IMPLEMENT_MODEL ( Neuron )

17.157.1.2 IMPLEMENT_MODEL ( WeightUpdateModel )

17.157.1.3 void modelDefinition ( NNmodel & model )

17.158 model_new.cc File Reference

#include "modelSpec.h"
```

## Classes

- class Neuron
- class WeightUpdateModel

#### **Functions**

- IMPLEMENT\_MODEL (Neuron)
- IMPLEMENT\_MODEL (WeightUpdateModel)
- void modelDefinition (NNmodel &model)

#### 17.158.1 Function Documentation

```
17.158.1.1 IMPLEMENT_MODEL ( Neuron )
```

17.158.1.2 IMPLEMENT\_MODEL ( WeightUpdateModel )

17.158.1.3 void modelDefinition ( NNmodel & model )

## 17.159 Model\_Schmuker\_2014\_classifier.cc File Reference

```
#include "modelSpec.h"
#include <iostream>
```

#### Classes

· class WeightUpdateModelSpikeEvent

#### Macros

- #define DT 0.5
- #define NUM VR 10
- #define NUM FEATURES 4
- #define NUM\_CLASSES 3
- #define NETWORK\_SCALE 10
- #define CLUST\_SIZE\_AN (NETWORK\_SCALE \* 6)
- #define CLUST\_SIZE\_PN (NETWORK\_SCALE \* 6)
- #define CLUST SIZE RN (NETWORK SCALE \* 6)
- #define SYNAPSE\_TAU\_RNPN 1.0
- #define SYNAPSE\_TAU\_PNPN 5.5
- #define SYNAPSE TAU PNAN 1.0
- #define SYNAPSE\_TAU\_ANAN 8.0

#### **Functions**

- IMPLEMENT\_MODEL (WeightUpdateModelSpikeEvent)
- void modelDefinition (NNmodel &model)

#### 17.159.1 Macro Definition Documentation

```
17.159.1.1 #define CLUST_SIZE_AN (NETWORK_SCALE * 6)
```

17.159.1.2 #define CLUST\_SIZE\_PN (NETWORK SCALE \* 6)

17.159.1.3 #define CLUST\_SIZE\_RN (NETWORK\_SCALE \* 6)

17.159.1.4 #define DT 0.5

17.159.1.5 #define NETWORK\_SCALE 10

```
17.159.1.6 #define NUM_CLASSES 3
17.159.1.7 #define NUM_FEATURES 4
17.159.1.8 #define NUM_VR 10
17.159.1.9 #define SYNAPSE_TAU_ANAN 8.0
17.159.1.10 #define SYNAPSE_TAU_PNAN 1.0
17.159.1.11 #define SYNAPSE_TAU_PNPN 5.5
17.159.1.12 #define SYNAPSE_TAU_RNPN 1.0
17.159.2 Function Documentation
17.159.2.1 IMPLEMENT_MODEL ( WeightUpdateModelSpikeEvent )
17.159.2.2 void modelDefinition ( NNmodel & model )
17.160 modelSpec.cc File Reference
17.161 modelSpec.cc File Reference
#include "codeGenUtils.h"
#include "global.h"
#include "modelSpec.h"
#include "standardSubstitutions.h"
#include "utils.h"
#include <cstdio>
#include <cmath>
#include <cassert>
#include <algorithm>
Macros

    #define MODELSPEC_CC

Functions

    void initGeNN ()

        Method for GeNN initialisation (by preparing standard models)
Variables

    unsigned int GeNNReady = 0

17.161.1 Macro Definition Documentation
17.161.1.1 #define MODELSPEC_CC
17.161.2 Function Documentation
```

```
17.161.2.1 void initGeNN ( )
```

Method for GeNN initialisation (by preparing standard models)

17.161.3 Variable Documentation

17.161.3.1 unsigned int GeNNReady = 0

#### 17.162 modelSpec.h File Reference

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections.

```
#include "neuronGroup.h"
#include "synapseGroup.h"
#include "utils.h"
#include <map>
#include <set>
#include <string>
#include <vector>
```

#### Classes

· class NNmodel

#### Macros

#define MODELSPEC H

macro for avoiding multiple inclusion during compilation

#define NO\_DELAY 0

Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

• #define NOLEARNING 0

Macro attaching the label "NOLEARNING" to flag 0.

• #define LEARNING 1

Macro attaching the label "LEARNING" to flag 1.

• #define EXITSYN 0

Macro attaching the label "EXITSYN" to flag 0 (excitatory synapse)

• #define INHIBSYN 1

Macro attaching the label "INHIBSYN" to flag 1 (inhibitory synapse)

#define CPU 0

Macro attaching the label "CPU" to flag 0.

• #define GPU 1

Macro attaching the label "GPU" to flag 1.

#define AUTODEVICE -1

Macro attaching the label AUTODEVICE to flag -1. Used by setGPUDevice.

#### **Enumerations**

- enum SynapseConnType { ALLTOALL, DENSE, SPARSE }
- enum SynapseGType { INDIVIDUALG, GLOBALG, INDIVIDUALID }
- enum FloatType { , GENN\_LONG\_DOUBLE }

#### **Functions**

• void initGeNN ()

Method for GeNN initialisation (by preparing standard models)

#### **Variables**

· unsigned int GeNNReady

#### 17.162.1 Detailed Description

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections.

17.162.2 Macro Definition Documentation

17.162.2.1 #define \_MODELSPEC\_H\_

macro for avoiding multiple inclusion during compilation

17.162.2.2 #define AUTODEVICE -1

Macro attaching the label AUTODEVICE to flag -1. Used by setGPUDevice.

17.162.2.3 #define CPU 0

Macro attaching the label "CPU" to flag 0.

17.162.2.4 #define EXITSYN 0

Macro attaching the label "EXITSYN" to flag 0 (excitatory synapse)

17.162.2.5 #define GPU 1

Macro attaching the label "GPU" to flag 1.

17.162.2.6 #define INHIBSYN 1

Macro attaching the label "INHIBSYN" to flag 1 (inhibitory synapse)

17.162.2.7 #define LEARNING 1

Macro attaching the label "LEARNING" to flag 1.

17.162.2.8 #define NO\_DELAY 0

Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

17.162.2.9 #define NOLEARNING 0

Macro attaching the label "NOLEARNING" to flag 0.

17.162.3 Enumeration Type Documentation

```
17.162.3.1 enum FloatType
Enumerator
     GENN_LONG_DOUBLE
17.162.3.2 enum SynapseConnType
Enumerator
     ALLTOALL
     DENSE
     SPARSE
17.162.3.3 enum SynapseGType
Enumerator
     INDIVIDUALG
     GLOBALG
     INDIVIDUALID
17.162.4 Function Documentation
17.162.4.1 void initGeNN ( )
Method for GeNN initialisation (by preparing standard models)
17.162.5 Variable Documentation
17.162.5.1 unsigned int GeNNReady
17.163 neuronGroup.cc File Reference
#include "neuronGroup.h"
#include <algorithm>
#include <cmath>
#include "codeGenUtils.h"
#include "standardSubstitutions.h"
#include "utils.h"
```

# 17.164 neuronGroup.h File Reference

```
#include <map>
#include <set>
#include <string>
#include <vector>
#include "newNeuronModels.h"
```

#### Classes

class NeuronGroup

#### 17.165 neuronModels.cc File Reference

```
#include "codeGenUtils.h"
#include "neuronModels.h"
#include "extra_neurons.h"
```

#### Macros

• #define NEURONMODELS CC

#### **Functions**

· void prepareStandardModels ()

Function that defines standard neuron models.

#### **Variables**

vector< neuronModel > nModels

Global C++ vector containing all neuron model descriptions.

unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

unsigned int POISSONNEURON

variable attaching the name "POISSONNEURON"

· unsigned int TRAUBMILES FAST

variable attaching the name "TRAUBMILES\_FAST"

unsigned int TRAUBMILES\_ALTERNATIVE

variable attaching the name "TRAUBMILES\_ALTERNATIVE"

• unsigned int TRAUBMILES SAFE

variable attaching the name "TRAUBMILES\_SAFE"

• unsigned int TRAUBMILES

variable attaching the name "TRAUBMILES"

unsigned int TRAUBMILES PSTEP

variable attaching the name "TRAUBMILES\_PSTEP"

• unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

unsigned int IZHIKEVICH V

variable attaching the name "IZHIKEVICH\_V"

• unsigned int SPIKESOURCE

variable attaching the name "SPIKESOURCE"

17.165.1 Macro Definition Documentation

17.165.1.1 #define NEURONMODELS\_CC

17.165.2 Function Documentation

17.165.2.1 void prepareStandardModels ( )

Function that defines standard neuron models.

The neuron models are defined and added to the C++ vector nModels that is holding all neuron model descriptions. User defined neuron models can be appended to this vector later in (a) separate function(s).

```
17.165.3 Variable Documentation
17.165.3.1 unsigned int IZHIKEVICH
variable attaching the name "IZHIKEVICH"
17.165.3.2 unsigned int IZHIKEVICH_V
variable attaching the name "IZHIKEVICH_V"
17.165.3.3 unsigned int MAPNEURON
variable attaching the name "MAPNEURON"
17.165.3.4 vector<neuronModel> nModels
Global C++ vector containing all neuron model descriptions.
17.165.3.5 unsigned int POISSONNEURON
variable attaching the name "POISSONNEURON"
17.165.3.6 unsigned int SPIKESOURCE
variable attaching the name "SPIKESOURCE"
17.165.3.7 unsigned int TRAUBMILES
variable attaching the name "TRAUBMILES"
17.165.3.8 unsigned int TRAUBMILES_ALTERNATIVE
variable attaching the name "TRAUBMILES_ALTERNATIVE"
17.165.3.9 unsigned int TRAUBMILES_FAST
variable attaching the name "TRAUBMILES_FAST"
17.165.3.10 unsigned int TRAUBMILES_PSTEP
variable attaching the name "TRAUBMILES_PSTEP"
17.165.3.11 unsigned int TRAUBMILES_SAFE
variable attaching the name "TRAUBMILES_SAFE"
```

# 17.166 neuronModels.h File Reference

```
#include "dpclass.h"
#include <string>
#include <vector>
```

#### Classes

· class neuronModel

class for specifying a neuron model.

· class rulkovdp

Class defining the dependent parameters of the Rulkov map neuron.

#### **Functions**

· void prepareStandardModels ()

Function that defines standard neuron models.

#### Variables

vector< neuronModel > nModels

Global C++ vector containing all neuron model descriptions.

unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

unsigned int POISSONNEURON

variable attaching the name "POISSONNEURON"

• unsigned int TRAUBMILES\_FAST

variable attaching the name "TRAUBMILES\_FAST"

unsigned int TRAUBMILES\_ALTERNATIVE

variable attaching the name "TRAUBMILES\_ALTERNATIVE"

unsigned int TRAUBMILES\_SAFE

variable attaching the name "TRAUBMILES\_SAFE"

• unsigned int TRAUBMILES

variable attaching the name "TRAUBMILES"

• unsigned int TRAUBMILES\_PSTEP

variable attaching the name "TRAUBMILES\_PSTEP"

• unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

unsigned int IZHIKEVICH\_V

variable attaching the name "IZHIKEVICH\_V"

• unsigned int SPIKESOURCE

variable attaching the name "SPIKESOURCE"

• const unsigned int MAXNRN = 7

17.166.1 Function Documentation

17.166.1.1 void prepareStandardModels ( )

Function that defines standard neuron models.

The neuron models are defined and added to the C++ vector nModels that is holding all neuron model descriptions. User defined neuron models can be appended to this vector later in (a) separate function(s).

17.166.2 Variable Documentation

17.166.2.1 unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

17.166.2.2 unsigned int IZHIKEVICH\_V

variable attaching the name "IZHIKEVICH\_V"

17.166.2.3 unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

```
17.166.2.4 const unsigned int MAXNRN = 7
17.166.2.5 vector<neuronModel> nModels
Global C++ vector containing all neuron model descriptions.
17.166.2.6 unsigned int POISSONNEURON
variable attaching the name "POISSONNEURON"
17.166.2.7 unsigned int SPIKESOURCE
variable attaching the name "SPIKESOURCE"
17.166.2.8 unsigned int TRAUBMILES
variable attaching the name "TRAUBMILES"
17.166.2.9 unsigned int TRAUBMILES_ALTERNATIVE
variable attaching the name "TRAUBMILES_ALTERNATIVE"
17.166.2.10 unsigned int TRAUBMILES_FAST
variable attaching the name "TRAUBMILES_FAST"
17.166.2.11 unsigned int TRAUBMILES_PSTEP
variable attaching the name "TRAUBMILES PSTEP"
17.166.2.12 unsigned int TRAUBMILES_SAFE
```

#### 17.167 newModels.h File Reference

variable attaching the name "TRAUBMILES\_SAFE"

```
#include <array>
#include <functional>
#include <string>
#include <vector>
#include <cassert>
```

# Classes

- class NewModels::ValueBase< NumValues >
- class NewModels::ValueBase< 0 >
- · class NewModels::Base

Base class for all models.

class NewModels::LegacyWrapper< ModelBase, LegacyModelType, ModelArray >

Wrapper around old-style models stored in global arrays and referenced by index.

# Namespaces

NewModels

#### Macros

```
    #define DECLARE_MODEL(TYPE, NUM_PARAMS, NUM_VARS)
    #define IMPLEMENT_MODEL(TYPE) TYPE *TYPE::s_Instance = NULL
```

- #define SET\_PARAM\_NAMES(...) virtual StringVec getParamNames() const{ return \_\_VA\_ARGS\_\_; }
- #define SET\_DERIVED\_PARAMS(...) virtual DerivedParamVec getDerivedParams() const{ return \_\_VA\_A↔
   RGS ;}
- #define SET\_VARS(...) virtual StringPairVec getVars() const{ return \_\_VA\_ARGS\_\_; }

#### 17.167.1 Macro Definition Documentation

17.167.1.1 #define DECLARE\_MODEL( TYPE, NUM\_PARAMS, NUM\_VARS )

#### Value:

```
private:
    static TYPE *s_Instance;
public:
    static const TYPE *getInstance()
    {
        if(s_Instance == NULL)
        {
             s_Instance = new TYPE;
        }
        return s_Instance;
    }
    typedef NewModels::ValueBase<NUM_PARAMS> ParamValues;
    typedef NewModels::ValueBase<NUM_VARS> VarValues;
```

```
17.167.1.2 #define IMPLEMENT_MODEL( TYPE ) TYPE *TYPE::s_Instance = NULL
```

```
17.167.1.3 #define SET_DERIVED_PARAMS( ... ) virtual DerivedParamVec getDerivedParams() const{ return __VA_ARGS__; }
```

17.167.1.4 #define SET\_PARAM\_NAMES( ... ) virtual StringVec getParamNames() const{ return \_\_VA\_ARGS\_\_; }

17.167.1.5 #define SET\_VARS( ... ) virtual StringPairVec getVars() const{ return \_\_VA\_ARGS\_\_; }

# 17.168 newNeuronModels.cc File Reference

```
#include "newNeuronModels.h"
```

# **Functions**

- IMPLEMENT\_MODEL (NeuronModels::RulkovMap)
- IMPLEMENT\_MODEL (NeuronModels::Izhikevich)
- IMPLEMENT\_MODEL (NeuronModels::IzhikevichVariable)
- IMPLEMENT\_MODEL (NeuronModels::SpikeSource)
- IMPLEMENT\_MODEL (NeuronModels::Poisson)
- IMPLEMENT MODEL (NeuronModels::TraubMiles)
- IMPLEMENT\_MODEL (NeuronModels::TraubMilesFast)
- IMPLEMENT\_MODEL (NeuronModels::TraubMilesAlt)
- IMPLEMENT\_MODEL (NeuronModels::TraubMilesNStep)

#### 17.168.1 Function Documentation

```
17.168.1.1 IMPLEMENT_MODEL ( NeuronModels::RulkovMap )
```

17.168.1.2 IMPLEMENT\_MODEL ( NeuronModels::Izhikevich )

```
17.168.1.3 IMPLEMENT_MODEL ( NeuronModels::IzhikevichVariable )
17.168.1.4 IMPLEMENT_MODEL ( NeuronModels::SpikeSource )
17.168.1.5 IMPLEMENT_MODEL ( NeuronModels::Poisson )
17.168.1.6 IMPLEMENT_MODEL ( NeuronModels::TraubMiles )
17.168.1.7 IMPLEMENT_MODEL ( NeuronModels::TraubMilesFast )
17.168.1.8 IMPLEMENT_MODEL ( NeuronModels::TraubMilesAlt )
17.168.1.9 IMPLEMENT_MODEL ( NeuronModels::TraubMilesNStep )
      newNeuronModels.h File Reference
#include <array>
#include <functional>
#include <string>
#include <tuple>
#include <vector>
#include "codeGenUtils.h"
#include "neuronModels.h"
#include "newModels.h"
```

#### Classes

· class NeuronModels::Base

Base class for all neuron models.

· class NeuronModels::LegacyWrapper

Wrapper around legacy weight update models stored in nModels array of neuronModel objects.

• class NeuronModels::RulkovMap

Rulkov Map neuron.

· class NeuronModels::Izhikevich

Izhikevich neuron with fixed parameters [1].

class NeuronModels::IzhikevichVariable

Izhikevich neuron with variable parameters [1].

• class NeuronModels::SpikeSource

Empty neuron which allows setting spikes from external sources.

• class NeuronModels::Poisson

Poisson neurons.

class NeuronModels::TraubMiles

Hodgkin-Huxley neurons with Traub & Miles algorithm.

· class NeuronModels::TraubMilesFast

Hodgkin-Huxley neurons with Traub & Miles algorithm: Original fast implementation, using 25 inner iterations.

· class NeuronModels::TraubMilesAlt

Hodgkin-Huxley neurons with Traub & Miles algorithm.

• class NeuronModels::TraubMilesNStep

Hodgkin-Huxley neurons with Traub & Miles algorithm.

# Namespaces

NeuronModels

#### **Macros**

- #define SET\_SIM\_CODE(SIM\_CODE) virtual std::string getSimCode() const{ return SIM\_CODE; }
- #define SET\_THRESHOLD\_CONDITION\_CODE(THRESHOLD\_CONDITION\_CODE) virtual std::string getThresholdConditionCode() const{ return THRESHOLD\_CONDITION\_CODE; }
- #define SET\_RESET\_CODE(RESET\_CODE) virtual std::string getResetCode() const{ return RESET\_CO
   DE; }
- #define SET\_SUPPORT\_CODE(SUPPORT\_CODE) virtual std::string getSupportCode() const{ return SU← PPORT\_CODE;}
- #define SET\_EXTRA\_GLOBAL\_PARAMS(...) virtual StringPairVec getExtraGlobalParams() const{ return 
   \_\_VA\_ARGS\_\_;}

#### 17.169.1 Macro Definition Documentation

```
17.169.1.1 #define SET_EXTRA_GLOBAL_PARAMS( ... ) virtual StringPairVec getExtraGlobalParams() const{ return __VA_ARGS__; }
```

- 17.169.1.2 #define SET\_RESET\_CODE( RESET\_CODE ) virtual std::string getResetCode() const{ return RESET\_CODE; }
- 17.169.1.3 #define SET\_SIM\_CODE( SIM\_CODE ) virtual std::string getSimCode() const{ return SIM\_CODE; }
- 17.169.1.4 #define SET\_SUPPORT\_CODE( SUPPORT\_CODE ) virtual std::string getSupportCode() const{ return SUPPORT\_CODE; }
- 17.169.1.5 #define SET\_THRESHOLD\_CONDITION\_CODE( THRESHOLD\_CONDITION\_CODE ) virtual std::string getThresholdConditionCode() const{ return THRESHOLD\_CONDITION\_CODE; }

# 17.170 newPostsynapticModels.cc File Reference

```
#include "newPostsynapticModels.h"
```

# **Functions**

- IMPLEMENT\_MODEL (PostsynapticModels::ExpCond)
- IMPLEMENT\_MODEL (PostsynapticModels::DeltaCurr)

#### 17.170.1 Function Documentation

```
17.170.1.1 IMPLEMENT_MODEL ( PostsynapticModels::ExpCond )
```

17.170.1.2 IMPLEMENT\_MODEL ( PostsynapticModels::DeltaCurr )

# 17.171 newPostsynapticModels.h File Reference

```
#include "newModels.h"
#include "postSynapseModels.h"
```

# Classes

· class PostsynapticModels::Base

Base class for all postsynaptic models.

- class PostsynapticModels::LegacyWrapper
- · class PostsynapticModels::ExpCond

Exponential decay with synaptic input treated as a conductance value.

· class PostsynapticModels::DeltaCurr

Simple delta current synapse.

## Namespaces

PostsynapticModels

#### Macros

- #define SET\_DECAY\_CODE(DECAY\_CODE) virtual std::string getDecayCode() const{ return DECAY\_C
  ODE; }
- #define SET\_SUPPORT\_CODE(SUPPORT\_CODE) virtual std::string getSupportCode() const{ return SU← PPORT\_CODE; }

#### 17.171.1 Macro Definition Documentation

- 17.171.1.1 #define SET\_CURRENT\_CONVERTER\_CODE( CURRENT\_CONVERTER\_CODE ) virtual std::string getCurrentConverterCode() const{ return CURRENT\_CONVERTER\_CODE; }
- 17.171.1.2 #define SET\_DECAY\_CODE( DECAY\_CODE ) virtual std::string getDecayCode() const{ return DECAY\_CODE; }
- 17.171.1.3 #define SET\_SUPPORT\_CODE( SUPPORT\_CODE ) virtual std::string getSupportCode() const{ return SUPPORT\_CODE; }

# 17.172 newWeightUpdateModels.cc File Reference

```
#include "newWeightUpdateModels.h"
```

## **Functions**

- IMPLEMENT\_MODEL (WeightUpdateModels::StaticPulse)
- IMPLEMENT\_MODEL (WeightUpdateModels::StaticGraded)
- IMPLEMENT MODEL (WeightUpdateModels::PiecewiseSTDP)

# 17.172.1 Function Documentation

```
17.172.1.1 IMPLEMENT_MODEL ( WeightUpdateModels::StaticPulse )
17.172.1.2 IMPLEMENT_MODEL ( WeightUpdateModels::StaticGraded )
17.172.1.3 IMPLEMENT_MODEL ( WeightUpdateModels::PiecewiseSTDP )
```

## 17.173 newWeightUpdateModels.h File Reference

```
#include "newModels.h"
#include "synapseModels.h"
```

#### Classes

· class WeightUpdateModels::Base

Base class for all weight update models.

class WeightUpdateModels::LegacyWrapper

Wrapper around legacy weight update models stored in weightUpdateModels array of weightUpdateModel objects.

· class WeightUpdateModels::StaticPulse

Pulse-coupled, static synapse.

• class WeightUpdateModels::StaticGraded

Graded-potential, static synapse.

class WeightUpdateModels::PiecewiseSTDP

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse.

#### **Namespaces**

WeightUpdateModels

#### **Macros**

- #define SET\_SIM\_CODE(SIM\_CODE) virtual std::string getSimCode() const{ return SIM\_CODE; }
- #define SET\_EVENT\_CODE(EVENT\_CODE) virtual std::string getEventCode() const{ return EVENT\_CO
   DE; }
- #define SET\_LEARN\_POST\_CODE(LEARN\_POST\_CODE) virtual std::string getLearnPostCode() const{ return LEARN\_POST\_CODE; }
- #define SET\_SYNAPSE\_DYNAMICS\_CODE(SYNAPSE\_DYNAMICS\_CODE) virtual std::string get
   SynapseDynamicsCode() const{ return SYNAPSE\_DYNAMICS\_CODE; }
- #define SET\_EVENT\_THRESHOLD\_CONDITION\_CODE(EVENT\_THRESHOLD\_CONDITION\_CODE) virtual std::string getEventThresholdConditionCode() const{ return EVENT\_THRESHOLD\_CONDITION\_CO← DE; }
- #define SET\_SIM\_SUPPORT\_CODE(SIM\_SUPPORT\_CODE) virtual std::string getSimSupportCode() const{ return SIM\_SUPPORT\_CODE; }
- #define SET\_LEARN\_POST\_SUPPORT\_CODE(LEARN\_POST\_SUPPORT\_CODE) virtual std::string get
   LearnPostSupportCode() const{ return LEARN\_POST\_SUPPORT\_CODE; }
- #define SET\_SYNAPSE\_DYNAMICS\_SUPPORT\_CODE(SYNAPSE\_DYNAMICS\_SUPPORT\_CODE) virtual std::string getSynapseDynamicsSuppportCode() const{ return SYNAPSE\_DYNAMICS\_SUPPORT\_C
  ODE; }
- #define SET\_EXTRA\_GLOBAL\_PARAMS(...) virtual StringPairVec getExtraGlobalParams() const{ return ←
   \_\_VA\_ARGS\_\_;}
- #define SET\_NEEDS\_PRE\_SPIKE\_TIME(PRE\_SPIKE\_TIME\_REQUIRED) virtual bool isPreSpikeTime ← Required() const{ return PRE\_SPIKE\_TIME\_REQUIRED; }
- 17.173.1 Macro Definition Documentation
- 17.173.1.1 #define SET\_EVENT\_CODE( EVENT\_CODE ) virtual std::string getEventCode() const{ return EVENT\_CODE; }
- 17.173.1.2 #define SET\_EVENT\_THRESHOLD\_CONDITION\_CODE( EVENT\_THRESHOLD\_CONDITION\_CODE ) virtual std::string getEventThresholdConditionCode() const{ return EVENT\_THRESHOLD\_CONDITION\_CODE; }
- 17.173.1.3 #define SET\_EXTRA\_GLOBAL\_PARAMS( ... ) virtual StringPairVec getExtraGlobalParams() const{ return \_\_VA\_ARGS\_\_; }

- 17.173.1.4 #define SET\_LEARN\_POST\_CODE( LEARN\_POST\_CODE ) virtual std::string getLearnPostCode() const{ return LEARN\_POST\_CODE; }
- 17.173.1.5 #define SET\_LEARN\_POST\_SUPPORT\_CODE( LEARN\_POST\_SUPPORT\_CODE ) virtual std::string getLearnPostSupportCode() const{ return LEARN\_POST\_SUPPORT\_CODE; }
- 17.173.1.6 #define SET\_NEEDS\_POST\_SPIKE\_TIME( POST\_SPIKE\_TIME\_REQUIRED ) virtual bool isPostSpikeTimeRequired() const{ return POST\_SPIKE\_TIME\_REQUIRED; }
- 17.173.1.7 #define SET\_NEEDS\_PRE\_SPIKE\_TIME( PRE\_SPIKE\_TIME\_REQUIRED ) virtual bool isPreSpikeTimeRequired() const{ return PRE\_SPIKE\_TIME\_REQUIRED; }
- 17.173.1.8 #define SET\_SIM\_CODE( SIM\_CODE ) virtual std::string getSimCode() const{ return SIM\_CODE; }
- 17.173.1.9 #define SET\_SIM\_SUPPORT\_CODE( SIM\_SUPPORT\_CODE ) virtual std::string getSimSupportCode() const{ return SIM\_SUPPORT\_CODE; }
- 17.173.1.10 #define SET\_SYNAPSE\_DYNAMICS\_CODE( SYNAPSE\_DYNAMICS\_CODE ) virtual std::string getSynapseDynamicsCode() const{ return SYNAPSE\_DYNAMICS\_CODE; }
- 17.173.1.11 #define SET\_SYNAPSE\_DYNAMICS\_SUPPORT\_CODE( SYNAPSE\_DYNAMICS\_SUPPORT\_CODE ) virtual std::string getSynapseDynamicsSuppportCode() const{ return SYNAPSE\_DYNAMICS\_SUPPORT\_CODE; }

# 17.174 OneComp.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

#### Classes

· class Mylzhikevich

#### **Functions**

- IMPLEMENT MODEL (Mylzhikevich)
- void modelDefinition (NNmodel &model)

# Variables

- Mylzhikevich::ParamValues exlzh\_p (0.02, 0.2,-65, 6, 4.0)
- Mylzhikevich::VarValues exlzh ini (-65,-20)
- 17.174.1 Function Documentation
- 17.174.1.1 IMPLEMENT\_MODEL ( Mylzhikevich )
- 17.174.1.2 void modelDefinition ( NNmodel & model )
- 17.174.2 Variable Documentation
- 17.174.2.1 Mylzhikevich::VarValues exlzh\_ini(-65,-20)
- 17.174.2.2 Mylzhikevich::ParamValues exlzh\_p(0.02,0.2,-65,6,4.0)

# 17.175 OneComp\_model.cc File Reference

```
#include "OneComp_CODE/definitions.h"
```

# Macros

```
• #define _ONECOMP_MODEL_CC_
```

#### 17.175.1 Macro Definition Documentation

```
17.175.1.1 #define _ONECOMP_MODEL_CC_
```

# 17.176 OneComp\_model.h File Reference

```
#include "OneComp.cc"
```

#### Classes

• class neuronpop

# 17.177 OneComp\_sim.cc File Reference

```
#include "OneComp_sim.h"
```

# **Functions**

• int main (int argc, char \*argv[])

# 17.177.1 Function Documentation

```
17.177.1.1 int main ( int argc, char * argv[] )
```

# 17.178 OneComp\_sim.h File Reference

```
#include "utils.h"
#include "stringUtils.h"
#include "hr_time.h"
#include <cuda_runtime.h>
#include <cassert>
#include "OneComp_model.h"
#include "OneComp_model.cc"
```

# Macros

- #define DBG SIZE 10000
- #define T\_REPORT\_TME 100.0
- #define TOTAL\_TME 5000

# **Variables**

CStopWatch timer

```
17.178.1 Macro Definition Documentation
17.178.1.1 #define DBG_SIZE 10000
17.178.1.2 #define T_REPORT_TME 100.0
17.178.1.3 #define TOTAL_TME 5000
17.178.2 Variable Documentation
17.178.2.1 CStopWatch timer
17.179 parse_options.h File Reference
Functions

    for (int i=argStart;i< argc;i++)</li>

    • if (cpu_only &&(which==1))
Variables
    • unsigned int dbgMode = 0
    • string ftype = "FLOAT"
    • unsigned int fixsynapse = 0
    • unsigned int cpu_only = 0

    string option

17.179.1 Function Documentation
17.179.1.1 for ( )
17.179.1.2 if (cpu_only && which==1)
17.179.2 Variable Documentation
17.179.2.1 unsigned int cpu_only = 0
17.179.2.2 unsigned int dbgMode = 0
17.179.2.3 unsigned int fixsynapse = 0
17.179.2.4 string ftype = "FLOAT"
17.179.2.5 string option
17.180 Poissonlzh-model.cc File Reference
#include "PoissonIzh-model.h"
#include "PoissonIzh_CODE/definitions.h"
```

#include "modelSpec.h"

#### **Macros**

#define \_POISSONIZHMODEL\_CC\_

# 17.180.1 Macro Definition Documentation

```
17.180.1.1 #define _POISSONIZHMODEL_CC_
```

# 17.181 Poissonlzh-model.h File Reference

```
#include <stdint.h>
```

#### Classes

· class classol

This class coontains the methods for running the MBody1 example model.

# 17.182 PoissonIzh.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

## **Functions**

• void modelDefinition (NNmodel &model)

# Variables

- NeuronModels::Poisson::ParamValues myPOI p (1, 2.5, 20.0,-60.0)
- NeuronModels::Poisson::VarValues myPOI ini (-60.0, 0,-10.0)
- NeuronModels::Izhikevich::ParamValues exIzh\_p (0.02, 0.2,-65, 6)
- NeuronModels::Izhikevich::VarValues exIzh\_ini (-65,-20)
- WeightUpdateModels::StaticPulse::VarValues mySyn\_ini (0.0)

## 17.182.1 Function Documentation

- 17.182.1.1 void modelDefinition ( NNmodel & model )
- 17.182.2 Variable Documentation
- 17.182.2.1 NeuronModels::Izhikevich::VarValues exizh\_ini(-65,-20)
- 17.182.2.2 NeuronModels::lzhikevich::ParamValues exlzh\_p(0.02,0.2,-65,6)
- 17.182.2.3 NeuronModels::Poisson::VarValues myPOI\_ini(-60.0,0,-10.0)
- 17.182.2.4 NeuronModels::Poisson::ParamValues myPOl\_p(1,2.5,20.0,-60.0)
- 17.182.2.5 WeightUpdateModels::StaticPulse::VarValues mySyn\_ini(0.0)

# 17.183 PoissonIzh\_sim.cc File Reference

```
#include "PoissonIzh_sim.h"
```

#### **Functions**

int main (int argc, char \*argv[])

# 17.183.1 Function Documentation

```
17.183.1.1 int main ( int argc, char * argv[] )
```

# 17.184 Poissonlzh\_sim.h File Reference

```
#include "utils.h"
#include "stringUtils.h"
#include "hr_time.h"
#include <cuda_runtime.h>
#include "PoissonIzh.cc"
#include <cassert>
#include "PoissonIzh-model.h"
#include "PoissonIzh-model.cc"
```

#### Macros

- #define MYRAND(Y, X) Y = Y \* 1103515245 +12345; X= (Y >> 16);
- #define T\_REPORT\_TME 1000.0
- #define SYN\_OUT\_TME 2000.0
- #define TOTAL\_TME 5000

# **Variables**

- scalar InputBaseRate = 2e-02
- CStopWatch timer

# 17.184.1 Macro Definition Documentation

```
17.184.1.1 #define MYRAND( Y, X) Y = Y * 1103515245 + 12345; <math>X = (Y >> 16);
```

17.184.1.2 #define SYN\_OUT\_TME 2000.0

17.184.1.3 #define T\_REPORT\_TME 1000.0

17.184.1.4 #define TOTAL\_TME 5000

17.184.2 Variable Documentation

17.184.2.1 scalar InputBaseRate = 2e-02

17.184.2.2 CStopWatch timer

# 17.185 postSynapseModels.cc File Reference

```
#include "codeGenUtils.h"
#include "postSynapseModels.h"
#include "extra_postsynapses.h"
```

#### Macros

#define POSTSYNAPSEMODELS\_CC

#### **Functions**

void preparePostSynModels ()

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

#### **Variables**

vector< postSynModel > postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

- unsigned int EXPDECAY
- unsigned int IZHIKEVICH\_PS

```
17.185.1 Macro Definition Documentation
```

17.185.1.1 #define POSTSYNAPSEMODELS\_CC

17.185.2 Function Documentation

17.185.2.1 void preparePostSynModels ( )

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

```
17.185.3 Variable Documentation
```

17.185.3.1 unsigned int EXPDECAY

17.185.3.2 unsigned int IZHIKEVICH\_PS

17.185.3.3 vector<postSynModel> postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

# 17.186 postSynapseModels.h File Reference

```
#include "dpclass.h"
#include <string>
#include <vector>
#include <cmath>
```

#### Classes

class postSynModel

Class to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

class expDecayDp

Class defining the dependent parameter for exponential decay.

#### **Functions**

void preparePostSynModels ()

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

# **Variables**

vector< postSynModel > postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

- · unsigned int EXPDECAY
- unsigned int IZHIKEVICH\_PS
- const unsigned int MAXPOSTSYN = 2

#### 17.186.1 Function Documentation

```
17.186.1.1 void preparePostSynModels ( )
```

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

```
17.186.2 Variable Documentation
```

17.186.2.1 unsigned int EXPDECAY

17.186.2.2 unsigned int IZHIKEVICH\_PS

17.186.2.3 const unsigned int MAXPOSTSYN = 2

 $17.186.2.4 \quad vector < postSynModel > postSynModels$ 

Global C++ vector containing all post-synaptic update model descriptions.

# 17.187 randomGen.cc File Reference

Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand().

```
#include "randomGen.h"
```

## **Macros**

• #define RANDOMGEN\_CC

macro for avoiding multiple inclusion during compilation

# 17.187.1 Detailed Description

Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand().

#### 17.187.2 Macro Definition Documentation

# 17.187.2.1 #define RANDOMGEN\_CC

macro for avoiding multiple inclusion during compilation

# 17.188 randomGen.h File Reference

header file containing the class definition for a uniform random generator based on the ISAAC random number generator

```
#include <time.h>
#include <limits.h>
#include <stdlib.h>
#include <assert.h>
#include "isaac.cc"
```

#### Classes

class randomGen

Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.

class stdRG

# Macros

• #define RANDOMGEN H

macro for avoiding multiple inclusion during compilation

# 17.188.1 Detailed Description

header file containing the class definition for a uniform random generator based on the ISAAC random number generator

17.188.2 Macro Definition Documentation

# 17.188.2.1 #define RANDOMGEN\_H

macro for avoiding multiple inclusion during compilation

# 17.189 Schmuker2014\_classifier.cc File Reference

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <cstdlib>
#include <cstring>
#include <time.h>
#include "Schmuker2014_classifier.h"
#include "schmuker_2014_classifier_CODE/definitions.h"
#include "sparseUtils.h"
#include "sparseUtils.h"
```

#### Macros

• #define \_SCHMUKER2014\_CLASSIFIER\_

macro for avoiding multiple inclusion during compilation

17.189.1 Macro Definition Documentation

```
17.189.1.1 #define _SCHMUKER2014_CLASSIFIER_
```

macro for avoiding multiple inclusion during compilation

# 17.190 Schmuker2014\_classifier.h File Reference

Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014".

```
#include <stdint.h>
#include "Model_Schmuker_2014_classifier.cc"
```

#### Classes

· class Schmuker2014\_classifier

This class cpontains the methods for running the Schmuker\_2014\_classifier example model.

# 17.190.1 Detailed Description

Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014".

# 17.191 simulation\_neuron\_policy\_pre\_post\_var.h File Reference

#### Classes

• class SimulationNeuronPolicyPrePostVar

# 17.192 simulation\_neuron\_policy\_pre\_var.h File Reference

# Classes

• class SimulationNeuronPolicyPreVar

# 17.193 simulation\_synapse\_policy\_dense.h File Reference

```
#include <functional>
#include <numeric>
```

#### Classes

• class SimulationSynapsePolicyDense

# 17.194 simulation\_synapse\_policy\_none.h File Reference

```
#include <numeric>
```

# Classes

• class SimulationSynapsePolicyNone

# 17.195 simulation\_synapse\_policy\_sparse.h File Reference

```
#include "simulation_synapse_policy_dense.h"
#include <functional>
#include <numeric>
```

# Classes

• class SimulationSynapsePolicySparse

## Macros

• #define SETUP\_THE\_C(I)

# 17.195.1 Macro Definition Documentation

```
17.195.1.1 #define SETUP_THE_C( I )
```

#### Value:

# 17.196 simulation\_test.h File Reference

```
#include "gtest/gtest.h"
```

#### Classes

· class SimulationTest

#### Macros

- #define WRAPPED\_INSTANTIATE\_TEST\_CASE\_P(prefix, test\_case\_name, generator) INSTANTIATE\_T

  EST\_CASE\_P(prefix, test\_case\_name, generator)
- #define TOKENPASTE(x, y) x ## y
- #define INIT\_SPARSE(N) TOKENPASTE(init, N)()

```
17.196.1 Macro Definition Documentation
```

```
17.196.1.1 #define INIT_SPARSE( N ) TOKENPASTE(init, N)()
```

```
17.196.1.2 #define TOKENPASTE( x, y ) x ## y
```

17.196.1.3 #define WRAPPED\_INSTANTIATE\_TEST\_CASE\_P( prefix, test\_case\_name, generator ) INSTANTIATE\_TEST\_CASE\_P(prefix, test\_case\_name, generator)

# 17.197 simulation\_test\_decoder\_matrix.h File Reference

```
#include "simulation_test.h"
```

# Classes

class SimulationTestDecoderMatrix

# 17.198 simulation\_test\_vars.h File Reference

```
#include "simulation_test.h"
```

# Classes

class SimulationTestVars< NeuronPolicy, SynapsePolicy >

# Macros

#define ASSIGN\_ARRAY\_VARS(ARRAY\_NAME, VAR\_PREFIX, COUNT)

#### 17.198.1 Macro Definition Documentation

17.198.1.1 #define ASSIGN\_ARRAY\_VARS( ARRAY\_NAME, VAR\_PREFIX, COUNT )

# Value:

# 17.199 sizes.h File Reference

#### Macros

- #define \_NExc 8000
- #define \_NInh 2000
- #define \_NMaxConnP0 845
- #define \_NMaxConnP1 301
- #define \_NMaxConnP2 834
- #define \_NMaxConnP3 237
- #define inputFac 1
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR\_MIN FLT\_MIN
- #define SCALAR\_MAX FLT\_MAX

#### 17.199.1 Macro Definition Documentation

```
17.199.1.1 #define FTYPE GENN_FLOAT
```

17.199.1.2 #define \_NExc 8000

17.199.1.3 #define \_Nlnh 2000

17.199.1.4 #define \_NMaxConnP0 845

17.199.1.5 #define \_NMaxConnP1 301

17.199.1.6 #define \_NMaxConnP2 834

17.199.1.7 #define \_NMaxConnP3 237

17.199.1.8 #define inputFac 1

17.199.1.9 #define scalar float

17.199.1.10 #define SCALAR\_MAX FLT\_MAX

17.199.1.11 #define SCALAR\_MIN FLT\_MIN

# 17.200 sizes.h File Reference

# Macros

- #define \_NAL 100
- #define \_NMB 1000
- #define \_NLHI 20
- #define \_NLB 100
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

17.200.1 Macro Definition Documentation
17.200.1.1 #define \_FTYPE GENN\_FLOAT
17.200.1.2 #define \_NAL 100
17.200.1.3 #define \_NLB 100
17.200.1.4 #define \_NLHI 20
17.200.1.5 #define \_NMB 1000
17.200.1.6 #define scalar float
17.200.1.7 #define SCALAR\_MAX 3.40282e+38f
17.200.1.8 #define SCALAR\_MIN 1.17549e-38f

#### Macros

• #define \_NAL 100

17.201 sizes.h File Reference

- #define \_NMB 1000
- #define NLHI 20
- #define \_NLB 100
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.201.1 Macro Definition Documentation

- 17.201.1.1 #define \_FTYPE GENN\_FLOAT
- 17.201.1.2 #define \_NAL 100
- 17.201.1.3 #define \_NLB 100
- 17.201.1.4 #define \_NLHI 20
- 17.201.1.5 #define \_NMB 1000
- 17.201.1.6 #define scalar float
- 17.201.1.7 #define SCALAR\_MAX 3.40282e+38f
- 17.201.1.8 #define SCALAR\_MIN 1.17549e-38f

# 17.202 sizes.h File Reference

## Macros

- #define \_NAL 100
- #define NMB 1000
- #define \_NLHI 20
- #define \_NLB 100
- #define \_FTYPE GENN\_FLOAT
- #define scalar float

- #define SCALAR\_MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f
- #define gPNKC\_GLOBAL 0.0025

#### 17.202.1 Macro Definition Documentation

- 17.202.1.1 #define \_FTYPE GENN\_FLOAT
- 17.202.1.2 #define \_NAL 100
- 17.202.1.3 #define \_NLB 100
- 17.202.1.4 #define \_NLHI 20
- 17.202.1.5 #define \_NMB 1000
- 17.202.1.6 #define gPNKC\_GLOBAL 0.0025
- 17.202.1.7 #define scalar float
- 17.202.1.8 #define SCALAR\_MAX 3.40282e+38f
- 17.202.1.9 #define SCALAR\_MIN 1.17549e-38f

# 17.203 sizes.h File Reference

#### Macros

- #define \_NAL 1024
- #define \_NMB 5000
- #define \_NLHI 20
- #define \_NLB 10
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.203.1 Macro Definition Documentation

- 17.203.1.1 #define \_FTYPE GENN\_FLOAT
- 17.203.1.2 #define \_NAL 1024
- 17.203.1.3 #define \_NLB 10
- 17.203.1.4 #define \_NLHI 20
- 17.203.1.5 #define \_NMB 5000
- 17.203.1.6 #define scalar float
- 17.203.1.7 #define SCALAR\_MAX 3.40282e+38f
- 17.203.1.8 #define SCALAR\_MIN 1.17549e-38f
- 17.204 sizes.h File Reference

#### **Macros**

- #define \_NAL 1024
- #define \_NMB 5000
- #define NLHI 20
- #define \_NLB 10
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR\_MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.204.1 Macro Definition Documentation

- 17.204.1.1 #define \_FTYPE GENN\_FLOAT
- 17.204.1.2 #define \_NAL 1024
- 17.204.1.3 #define \_NLB 10
- 17.204.1.4 #define \_NLHI 20
- 17.204.1.5 #define \_NMB 5000
- 17.204.1.6 #define scalar float
- 17.204.1.7 #define SCALAR\_MAX 3.40282e+38f
- 17.204.1.8 #define SCALAR\_MIN 1.17549e-38f

# 17.205 sizes.h File Reference

## Macros

- #define \_NAL 100
- #define \_NMB 1000
- #define \_NLHI 20
- #define \_NLB 100
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR\_MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.205.1 Macro Definition Documentation

- 17.205.1.1 #define \_FTYPE GENN\_FLOAT
- 17.205.1.2 #define \_NAL 100
- 17.205.1.3 #define \_NLB 100
- 17.205.1.4 #define \_NLHI 20
- 17.205.1.5 #define \_NMB 1000
- 17.205.1.6 #define scalar float
- 17.205.1.7 #define SCALAR\_MAX 3.40282e+38f

# 17.205.1.8 #define SCALAR\_MIN 1.17549e-38f

#### 17.206 sizes.h File Reference

#### Macros

- #define \_NC1 1
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR\_MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.206.1 Macro Definition Documentation

- 17.206.1.1 #define \_FTYPE GENN\_FLOAT
- 17.206.1.2 #define \_NC1 1
- 17.206.1.3 #define scalar float
- 17.206.1.4 #define SCALAR\_MAX 3.40282e+38f
- 17.206.1.5 #define SCALAR\_MIN 1.17549e-38f

# 17.207 sizes.h File Reference

# Macros

- #define \_NPoisson 100
- #define NIzh 10
- #define \_FTYPE GENN\_FLOAT
- #define scalar float
- #define SCALAR MIN 1.17549e-38f
- #define SCALAR\_MAX 3.40282e+38f

# 17.207.1 Macro Definition Documentation

- 17.207.1.1 #define \_FTYPE GENN\_FLOAT
- 17.207.1.2 #define \_Nlzh 10
- 17.207.1.3 #define \_NPoisson 100
- 17.207.1.4 #define scalar float
- 17.207.1.5 #define SCALAR\_MAX 3.40282e+38f
- 17.207.1.6 #define SCALAR\_MIN 1.17549e-38f

# 17.208 sparseProjection.h File Reference

# Classes

• struct SparseProjection

class (struct) for defining a spars connectivity projection

# 17.209 sparseUtils.cc File Reference

```
#include "sparseUtils.h"
#include "utils.h"
#include <vector>
```

#### Macros

#define SPARSEUTILS CC

#### **Functions**

- void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection \*C)
  - Utility to generate the SPARSE array structure with post-to-pre arrangement from the original pre-to-post arrangement where postsynaptic feedback is necessary (learning etc)
- void createPreIndices (unsigned int preN, unsigned int postN, SparseProjection \*C)
  - Function to create the mapping from the normal index array "ind" to the "reverse" array revInd, i.e. the inverse mapping of remap. This is needed if SynapseDynamics accesses pre-synaptic variables.
- void initializeSparseArray (SparseProjection C, unsigned int \*dInd, unsigned int \*dIndInG, unsigned int preN)

  Function for initializing conductance array indices for sparse matrices on the GPU (by copying the values from the host)
- void initializeSparseArrayRev (SparseProjection C, unsigned int \*dRevInd, unsigned int \*dRevIndInG, unsigned int \*dRemap, unsigned int postN)
  - Function for initializing reversed conductance array indices for sparse matrices on the GPU (by copying the values from the host)
- void initializeSparseArrayPreInd (SparseProjection C, unsigned int \*dPreInd)
  - Function for initializing reversed conductance arrays presynaptic indices for sparse matrices on the GPU (by copying the values from the host)
- 17.209.1 Macro Definition Documentation
- 17.209.1.1 #define SPARSEUTILS\_CC
- 17.209.2 Function Documentation
- 17.209.2.1 void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection \* C)

Utility to generate the SPARSE array structure with post-to-pre arrangement from the original pre-to-post arrangement where postsynaptic feedback is necessary (learning etc)

17.209.2.2 void createPreIndices (unsigned int preN, unsigned int postN, SparseProjection \* C)

Function to create the mapping from the normal index array "ind" to the "reverse" array revInd, i.e. the inverse mapping of remap. This is needed if SynapseDynamics accesses pre-synaptic variables.

17.209.2.3 void initializeSparseArray ( SparseProjection *C*, unsigned int \* *dlnd*, unsigned int \* *dlndlnG*, unsigned int preN

Function for initializing conductance array indices for sparse matrices on the GPU (by copying the values from the host)

17.209.2.4 void initializeSparseArrayPreInd ( SparseProjection C, unsigned int \* dPreInd )

Function for initializing reversed conductance arrays presynaptic indices for sparse matrices on the GPU (by copying the values from the host)

17.209.2.5 void initializeSparseArrayRev ( SparseProjection *C*, unsigned int \* *dRevInd*, unsigned int \* *dRevIndlnG*, unsigned int \* *dRemap*, unsigned int postN )

Function for initializing reversed conductance array indices for sparse matrices on the GPU (by copying the values from the host)

# 17.210 sparseUtils.h File Reference

```
#include "sparseProjection.h"
#include "global.h"
#include <cstdlib>
#include <cstdio>
#include <string>
#include <cmath>
```

#### **Functions**

template < class DATATYPE >
 unsigned int countEntriesAbove (DATATYPE \*Array, int sz, double includeAbove)

Utility to count how many entries above a specified value exist in a float array.

template < class DATATYPE >

```
DATATYPE getG (DATATYPE *wuvar, SparseProjection *sparseStruct, int x, int y)
```

DEPRECATED Utility to get a synapse weight from a SPARSE structure by x,y coordinates NB: as the Sparse—Projection struct doesnt hold the preN size (it should!) it is not possible to check the parameter validity. This fn may therefore crash unless user knows max poss X.

- template < class DATATYPE >
   float getSparseVar (DATATYPE \*wuvar, SparseProjection \*sparseStruct, unsigned int x, unsigned int y)
- template < class DATATYPE >
   void setSparseConnectivityFromDense (DATATYPE \*wuvar, int preN, int postN, DATATYPE \*tmp\_gRNPN,
   SparseProjection \*sparseStruct)

Function for setting the values of SPARSE connectivity matrix.

template<class DATATYPE >
 void createSparseConnectivityFromDense (DATATYPE \*wuvar, int preN, int postN, DATATYPE \*tmp\_gR
 NPN, SparseProjection \*sparseStruct, bool runTest)

Utility to generate the SPARSE connectivity structure from a simple all-to-all array.

• void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection \*C)

Utility to generate the SPARSE array structure with post-to-pre arrangement from the original pre-to-post arrangement where postsynaptic feedback is necessary (learning etc)

void createPreIndices (unsigned int preN, unsigned int postN, SparseProjection \*C)

Function to create the mapping from the normal index array "ind" to the "reverse" array revInd, i.e. the inverse mapping of remap. This is needed if SynapseDynamics accesses pre-synaptic variables.

- void initializeSparseArray (SparseProjection C, unsigned int \*dInd, unsigned int \*dIndInG, unsigned int preN)
   Function for initializing conductance array indices for sparse matrices on the GPU (by copying the values from the host)
- void initializeSparseArrayRev (SparseProjection C, unsigned int \*dRevInd, unsigned int \*dRevIndInG, unsigned int \*dRemap, unsigned int postN)

Function for initializing reversed conductance array indices for sparse matrices on the GPU (by copying the values from the host)

void initializeSparseArrayPreInd (SparseProjection C, unsigned int \*dPreInd)

Function for initializing reversed conductance arrays presynaptic indices for sparse matrices on the GPU (by copying the values from the host)

17.210.1 Function Documentation

17.210.1.1 template < class DATATYPE > unsigned int countEntriesAbove ( DATATYPE \* Array, int sz, double includeAbove )

Utility to count how many entries above a specified value exist in a float array.

17.210.1.2 void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection \* C)

Utility to generate the SPARSE array structure with post-to-pre arrangement from the original pre-to-post arrangement where postsynaptic feedback is necessary (learning etc)

17.210.1.3 void createPreIndices ( unsigned int preN, unsigned int postN, SparseProjection \* C)

Function to create the mapping from the normal index array "ind" to the "reverse" array revInd, i.e. the inverse mapping of remap. This is needed if SynapseDynamics accesses pre-synaptic variables.

17.210.1.4 template < class DATATYPE > void createSparseConnectivityFromDense ( DATATYPE \* wuvar, int preN, int postN, DATATYPE \* tmp\_gRNPN, SparseProjection \* sparseStruct, bool runTest )

Utility to generate the SPARSE connectivity structure from a simple all-to-all array.

17.210.1.5 template < class DATATYPE > DATATYPE getG ( DATATYPE \* wuvar, SparseProjection \* sparseStruct, int x, int y)

DEPRECATED Utility to get a synapse weight from a SPARSE structure by x,y coordinates NB: as the Sparse Projection struct doesnt hold the preN size (it should!) it is not possible to check the parameter validity. This fn may therefore crash unless user knows max poss X.

- 17.210.1.6 template < class DATATYPE > float getSparseVar ( DATATYPE \* wuvar, SparseProjection \* sparseStruct, unsigned int x, unsigned int y )
- 17.210.1.7 void initializeSparseArray ( SparseProjection *C*, unsigned int \* dInd, unsigned int \* dIndInG, unsigned int preN

Function for initializing conductance array indices for sparse matrices on the GPU (by copying the values from the host)

17.210.1.8 void initializeSparseArrayPreInd ( SparseProjection *C*, unsigned int \* *dPreInd* )

Function for initializing reversed conductance arrays presynaptic indices for sparse matrices on the GPU (by copying the values from the host)

17.210.1.9 void initializeSparseArrayRev ( SparseProjection *C*, unsigned int \* *dRevInd*, unsigned int \* *dRevIndInG*, unsigned int \* *dRemap*, unsigned int *postN* )

Function for initializing reversed conductance array indices for sparse matrices on the GPU (by copying the values from the host)

17.210.1.10 template < class DATATYPE > void setSparseConnectivityFromDense ( DATATYPE \* wuvar, int preN, int postN, DATATYPE \* tmp\_gRNPN, SparseProjection \* sparseStruct )

Function for setting the values of SPARSE connectivity matrix.

# 17.211 standardGeneratedSections.cc File Reference

```
#include "standardGeneratedSections.h"
#include "CodeHelper.h"
#include "modelSpec.h"
```

# 17.212 standardGeneratedSections.h File Reference

```
#include <string>
#include "codeGenUtils.h"
#include "newNeuronModels.h"
#include "standardSubstitutions.h"
```

#### **Namespaces**

StandardGeneratedSections

#### **Functions**

- void StandardGeneratedSections::neuronLocalVarInit (std::ostream &os, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const std::string &devPrefix, const std::string &localID)
- void StandardGeneratedSections::neuronLocalVarWrite (std::ostream &os, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const std::string &devPrefix, const std::string &localID)
- void StandardGeneratedSections::neuronSpikeEventTest (std::ostream &os, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &localID, const std::string &ftype)

#### 17.213 standardSubstitutions.cc File Reference

```
#include "standardSubstitutions.h"
#include "CodeHelper.h"
#include "modelSpec.h"
```

# 17.214 standardSubstitutions.h File Reference

```
#include <string>
#include "codeGenUtils.h"
#include "newNeuronModels.h"
```

# Classes

struct NameIterCtx< Container >

#### **Namespaces**

· StandardSubstitutions

# **Typedefs**

- typedef NamelterCtx < NewModels::Base::StringPairVec > VarNamelterCtx
- typedef NamelterCtx < NewModels::Base::DerivedParamVec > DerivedParamNamelterCtx
- $\hbox{\color{red} \bullet typedef Name IterCtx} < New Models :: Base :: StringPairVec > Extra Global Param Name IterCtx \\$

#### **Functions**

- void StandardSubstitutions::postSynapseCurrentConverter (std::string &psCode, const SynapseGroup \*sg, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerived
   — Params, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::postSynapseDecay (std::string &pdCode, const SynapseGroup \*sg, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::neuronThresholdCondition (std::string &thCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParam NameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::neuronSim (std::string &sCode, const NeuronGroup &ng, const VarNameIter Ctx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::neuronSpikeEventCondition (std::string &eCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::neuronReset (std::string &rCode, const NeuronGroup &ng, const VarNameIter
   Ctx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx
   &nmExtraGlobalParams, const std::string &ftype)
- void StandardSubstitutions::weightUpdateSim (std::string &wCode, const SynapseGroup &sg, const Var
   NameIterCtx &wuVars, const DerivedParamNameIterCtx &wuDerivedParams, const ExtraGlobalParam
   NameIterCtx &wuExtraGlobalParams, const string &preIdx, const string &postIdx, const string &devPrefix, const std::string &ftype)
- void StandardSubstitutions::weightUpdateDynamics (std::string &SDcode, const SynapseGroup \*sg, const VarNameIterCtx &wuVars, const DerivedParamNameIterCtx &wuDerivedParams, const string &preIdx, const string &postIdx, const string &devPrefix, const std::string &ftype)
- void StandardSubstitutions::weightUpdatePostLearn (std::string &code, const SynapseGroup \*sg, const DerivedParamNameIterCtx &wuDerivedParams, const ExtraGlobalParamNameIterCtx &wuExtraGlobal← Params, const string &preIdx, const string &postIdx, const string &devPrefix, const std::string &ftype)

```
17.214.1 Typedef Documentation
```

17.214.1.1 typedef NamelterCtx<NewModels::Base::DerivedParamVec> DerivedParamNamelterCtx

17.214.1.2 typedef NamelterCtx < NewModels::Base::StringPairVec > ExtraGlobalParamNamelterCtx

17.214.1.3 typedef NamelterCtx<NewModels::Base::StringPairVec> VarNamelterCtx

# 17.215 stringUtils.h File Reference

```
#include <string>
#include <sstream>
```

#### Macros

#define tS(X) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

#### **Functions**

```
    template < class T > std::string toString (T t) template functions for conversion of various types to C++ strings
    17.215.1 Macro Definition Documentation
    17.215.1.1 #define tS( X ) toString(X)
    Macro providing the abbreviated syntax tS() instead of toString().
    17.215.2 Function Documentation
    17.215.2.1 template < class T > std::string toString ( T t )
```

template functions for conversion of various types to C++ strings

# 17.216 synapseGroup.cc File Reference

```
#include "synapseGroup.h"
#include <algorithm>
#include <cmath>
#include "codeGenUtils.h"
#include "standardSubstitutions.h"
#include "utils.h"
```

# 17.217 synapseGroup.h File Reference

```
#include <map>
#include <set>
#include <string>
#include <vector>
#include "neuronGroup.h"
#include "newPostsynapticModels.h"
#include "newWeightUpdateModels.h"
#include "synapseMatrixType.h"
```

#### Classes

· class SynapseGroup

# 17.218 synapseMatrixType.h File Reference

# **Enumerations**

- enum SynapseMatrixConnectivity:: unsigned int { SynapseMatrixConnectivity::SPARSE = (1 << 0), SynapseMatrixConnectivity::DENSE = (1 << 1), SynapseMatrixConnectivity::BITMASK = (1 << 2)}
  - < Flags defining differnet types of synaptic matrix connectivity
- enum SynapseMatrixWeight:: unsigned int { SynapseMatrixWeight::GLOBAL = (1 << 3), SynapseMatrix
   Weight::INDIVIDUAL = (1 << 4) }</li>

```
    enum SynapseMatrixType : unsigned int {

            SynapseMatrixType::SPARSE_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::SP↔
            ARSE) \ | \ static\_cast < unsigned \ int > (Synapse Matrix Weight:: GLOBAL), \ Synapse Matrix Type:: SPARSE\_IN \leftarrow INCOME. The property of the
            DIVIDUALG = static_cast<unsigned int>(SynapseMatrixConnectivity::SPARSE) | static_cast<unsigned
            int>(SynapseMatrixWeight::INDIVIDUAL), SynapseMatrixType::DENSE_GLOBALG = static_cast<unsigned
            int>(SynapseMatrixConnectivity::DENSE) | static cast<unsigned int>(SynapseMatrixWeight::GLOBAL),
            SynapseMatrixType::DENSE INDIVIDUALG = static cast<unsigned int>(SynapseMatrixConnectivity::D↔
            ENSE) | static cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL),
            SynapseMatrixType::BITMASK GLOBALG = static cast<unsigned int>(SynapseMatrixConnectivity::BIT←
            MASK) | static cast<unsigned int>(SynapseMatrixWeight::GLOBAL) }
Functions

    bool operator& (SynapseMatrixType type, SynapseMatrixConnectivity connType)

    bool operator& (SynapseMatrixType type, SynapseMatrixWeight weightType)

17.218.1 Enumeration Type Documentation
17.218.1.1 enum SynapseMatrixConnectivity: unsigned int [strong]
 < Flags defining differnet types of synaptic matrix connectivity
Enumerator
           SPARSE
           DENSE
           BITMASK
17.218.1.2 enum SynapseMatrixType: unsigned int [strong]
Enumerator
           SPARSE_GLOBALG
           SPARSE INDIVIDUALG
           DENSE_GLOBALG
           DENSE_INDIVIDUALG
           BITMASK_GLOBALG
17.218.1.3 enum SynapseMatrixWeight: unsigned int [strong]
Enumerator
           GLOBAL
           INDIVIDUAL
17.218.2 Function Documentation
17.218.2.1 bool operator& (SynapseMatrixType type, SynapseMatrixConnectivity connType) [inline]
17.218.2.2 bool operator& (SynapseMatrixType type, SynapseMatrixWeight weightType) [inline]
17.219 synapseModels.cc File Reference
 #include "codeGenUtils.h"
 #include "synapseModels.h"
 #include "extra_weightupdates.h"
```

#### **Macros**

• #define SYNAPSEMODELS\_CC

#### **Functions**

void prepareWeightUpdateModels ()

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

# Variables

vector< weightUpdateModel > weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

```
17.219.1 Macro Definition Documentation
```

17.219.1.1 #define SYNAPSEMODELS CC

17.219.2 Function Documentation

17.219.2.1 void prepareWeightUpdateModels ( )

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

17.219.3 Variable Documentation

17.219.3.1 unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

17.219.3.2 unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

17.219.3.3 unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

 $17.219.3.4 \quad vector {<} weight Update Model {>} \ weight Update Models$ 

Global C++ vector containing all weightupdate model descriptions.

# 17.220 synapseModels.h File Reference

```
#include "dpclass.h"
#include <string>
#include <vector>
```

#### Classes

class weightUpdateModel

Class to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

class pwSTDP

TODO This class definition may be code-generated in a future release.

#### **Functions**

void prepareWeightUpdateModels ()

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

#### **Variables**

vector< weightUpdateModel > weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

const unsigned int SYNTYPENO = 4

#### 17.220.1 Function Documentation

17.220.1.1 void prepareWeightUpdateModels ( )

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

17.220.2 Variable Documentation

17.220.2.1 unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

17.220.2.2 unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

17.220.2.3 unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

17.220.2.4 const unsigned int SYNTYPENO = 4

17.220.2.5 vector<weightUpdateModel> weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

# 17.221 SynDelay.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
```

#### Classes

· class Mylzhikevich

#### **Functions**

- IMPLEMENT\_MODEL (Mylzhikevich)
- · void modelDefinition (NNmodel &model)

```
17.221.1 Function Documentation
```

```
17.221.1.1 IMPLEMENT_MODEL ( Mylzhikevich )
```

17.221.1.2 void modelDefinition ( NNmodel & model )

# 17.222 SynDelaySim.cc File Reference

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "hr_time.h"
#include "utils.h"
#include "stringUtils.h"
#include "SynDelaySim.h"
#include "SynDelay_CODE/definitions.h"
```

#### Macros

• #define SYNDELAYSIM\_CU

#### **Functions**

• int main (int argc, char \*argv[])

17.222.1 Macro Definition Documentation

17.222.1.1 #define SYNDELAYSIM\_CU

17.222.2 Function Documentation

17.222.2.1 int main ( int argc, char \* argv[] )

# 17.223 SynDelaySim.h File Reference

## Classes

class SynDelay

# Macros

```
#define TOTAL_TIME 5000.0f
#define REPORT_TIME 1000.0f

17.223.1 Macro Definition Documentation
17.223.1.1 #define REPORT_TIME 1000.0f
17.223.1.2 #define TOTAL_TIME 5000.0f
17.224 test.cc File Reference
```

#include "../../utils/simulation\_test\_decoder\_matrix.h"

# Classes

• class SimTest

#include "gtest/gtest.h"

#include <DEFINITIONS\_HEADER>

# **Functions**

- TEST\_P (SimTest, CorrectDecoding)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.224.1 Function Documentation
```

```
17.224.1.1 TEST_P ( SimTest , CorrectDecoding )
```

- 17.224.1.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.224.2 Variable Documentation
- 17.224.2.1 auto simulatorBackends = ::testing::Values(true, false)

# 17.225 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test.h"
```

# Classes

class SimTest

#### **Functions**

```
• TEST_P (SimTest, CorrectDecoding)
```

```
    WRAPPED_INSTANTIATE_TEST_CASE_P (MODEL_NAME, SimTest, simulatorBackends)
```

#### **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.225.1.1 TEST_P( SimTest, CorrectDecoding )

17.225.1.2 WRAPPED_INSTANTIATE_TEST_CASE_P( MODEL_NAME, SimTest, simulatorBackends )

17.225.2 Variable Documentation

17.225.2.1 auto simulatorBackends = ::testing::Values(true, false)

17.226 test.cc File Reference

#include "gtest/gtest.h"

#include <DEFINITIONS_HEADER>
#include "../../utils/simulation test decoder matrix.h"
```

#### Classes

· class SimTest

# Functions

- TEST\_P (SimTest, CorrectDecoding)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

#### **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.226.1 Function Documentation

17.226.1.1 TEST_P ( SimTest , CorrectDecoding )

17.226.1.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )

17.226.2 Variable Documentation

17.226.2.1 auto simulatorBackends = ::testing::Values(true, false)
```

# 17.227 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_decoder_matrix.h"
```

#### Classes

class SimTest

#### **Functions**

- TEST P (SimTest, CorrectDecoding)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.227.1 Function Documentation
```

```
17.227.1.1 TEST_P ( SimTest , CorrectDecoding )
```

```
17.227.1.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
```

17.227.2 Variable Documentation

17.227.2.1 auto simulatorBackends = ::testing::Values(true, false)

### 17.228 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_decoder_matrix.h"
```

## Classes

class SimTest

### **Functions**

- TEST\_P (SimTest, CorrectDecoding)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.228.1 Function Documentation
```

```
17.228.1.1 TEST_P ( SimTest , CorrectDecoding )
```

17.228.1.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.228.2 Variable Documentation

17.228.2.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.229 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_none.h"
```

### **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyNone > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.229.1 Typedef Documentation
```

17.229.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyNone> SimTest

```
17.229.2 Function Documentation
```

```
17.229.2.1 TEST_P ( SimTest , AcceptableError )
```

17.229.2.2 WRAPPED INSTANTIATE TEST CASE P ( MODEL NAME , SimTest , simulatorBackends )

### 17.229.3 Variable Documentation

17.229.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.230 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

### Classes

class SimulationSynapsePolicy

## **Typedefs**

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicy > SimTest

## **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.230.1 Typedef Documentation
```

17.230.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicy > SimTest

```
17.230.2 Function Documentation
```

```
17.230.2.1 TEST_P ( SimTest , AcceptableError )
```

17.230.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.230.3 Variable Documentation

17.230.3.1 auto simulatorBackends = ::testing::Values(true, false)

# 17.231 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

# **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > Sim
 — Test

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED INSTANTIATE TEST CASE P (MODEL NAME, SimTest, simulatorBackends)

## **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

## 17.231.1 Typedef Documentation

- 17.231.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Dense> SimTest
- 17.231.2 Function Documentation
- 17.231.2.1 TEST\_P ( SimTest , AcceptableError )

```
17.231.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.231.3 Variable Documentation
17.231.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.232 test.cc File Reference
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
Typedefs

    typedef SimulationTestVars
    SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > Sim←

Functions
   • TEST_P (SimTest, AcceptableError)
   • WRAPPED_INSTANTIATE_TEST_CASE_P (MODEL_NAME, SimTest, simulatorBackends)
Variables

    auto simulatorBackends = ::testing::Values(true, false)

17.232.1 Typedef Documentation
17.232.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy ←
         Dense > SimTest
17.232.2 Function Documentation
17.232.2.1 TEST_P ( SimTest , AcceptableError )
17.232.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.232.3 Variable Documentation
17.232.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.233 test.cc File Reference
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## Typedefs

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

## 17.233.1 Typedef Documentation

- 17.233.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense>
- 17.233.2 Function Documentation

```
17.233.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.233.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.233.3 Variable Documentation
- 17.233.3.1 auto simulatorBackends = ::testing::Values(true, false)

### 17.234 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED INSTANTIATE TEST CASE P (MODEL NAME, SimTest, simulatorBackends)

# Variables

• auto simulatorBackends = ::testing::Values(true, false)

# 17.234.1 Typedef Documentation

- 17.234.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.234.2 Function Documentation
- 17.234.2.1 TEST\_P ( SimTest , AcceptableError )

```
17.234.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.234.3 Variable Documentation
17.234.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.235 test.cc File Reference
#include "gtest/gtest.h"
#include <DEFINITIONS HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
Typedefs

    typedef SimulationTestVars
    SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > Sim←

Functions
   • TEST P (SimTest, AcceptableError)

    WRAPPED_INSTANTIATE_TEST_CASE_P (MODEL_NAME, SimTest, simulatorBackends)

Variables

    auto simulatorBackends = ::testing::Values(true, false)

17.235.1 Typedef Documentation
17.235.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
         Dense > SimTest
17.235.2 Function Documentation
17.235.2.1 TEST_P ( SimTest , AcceptableError )
17.235.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.235.3 Variable Documentation
17.235.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.236 test.cc File Reference
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
```

#include "../../utils/simulation\_synapse\_policy\_sparse.h"

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicySparse > Sim
 — Test

### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

#### Variables

auto simulatorBackends = ::testing::Values(true, false)

```
17.236.1 Typedef Documentation
```

```
17.236.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Sparse>SimTest
```

17.236.2 Function Documentation

```
17.236.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.236.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.236.3 Variable Documentation
- 17.236.3.1 auto simulatorBackends = ::testing::Values(true, false)

# 17.237 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

### **Typedefs**

typedef SimulationTestVars< SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > Sim
 — Test

### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.237.1 Typedef Documentation
17.237.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
          Dense> SimTest
17.237.2 Function Documentation
17.237.2.1 TEST_P ( SimTest , AcceptableError )
17.237.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.237.3 Variable Documentation
17.237.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.238 test.cc File Reference
#include "gtest/gtest.h"
#include <DEFINITIONS HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
Typedefs

    typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicySparse > Sim

     Test
Functions
    • TEST_P (SimTest, AcceptableError)

    WRAPPED INSTANTIATE TEST CASE P (MODEL NAME, SimTest, simulatorBackends)

Variables
    • auto simulatorBackends = ::testing::Values(true, false)
17.238.1 Typedef Documentation
17.238.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
          Sparse > SimTest
17.238.2 Function Documentation
17.238.2.1 TEST_P ( SimTest , AcceptableError )
17.238.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
17.238.3 Variable Documentation
17.238.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.239 test.cc File Reference
#include "gtest/gtest.h"
```

```
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

#### **Variables**

• auto simulatorBackends = ::testing::Values(true, false)

```
17.239.1 Typedef Documentation
```

17.239.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Dense> SimTest

17.239.2 Function Documentation

```
17.239.2.1 TEST_P ( SimTest , AcceptableError )
```

17.239.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.239.3 Variable Documentation

17.239.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.240 test.cc File Reference

```
#include <functional>
#include <numeric>
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicySparse > Sim
 — Test

# **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.240.1 Typedef Documentation
```

```
17.240.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Sparse> SimTest
```

```
17.240.2 Function Documentation
```

```
17.240.2.1 TEST_P ( SimTest , AcceptableError )
```

```
17.240.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
```

17.240.3 Variable Documentation

```
17.240.3.1 auto simulatorBackends = ::testing::Values(true, false)
```

### 17.241 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.241.1 Typedef Documentation
```

- 17.241.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.241.2 Function Documentation

```
17.241.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.241.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.241.3 Variable Documentation
- 17.241.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.242 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimTest

#### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## **Variables**

auto simulatorBackends = ::testing::Values(true, false)

```
17.242.1 Typedef Documentation
```

17.242.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimTest

17.242.2 Function Documentation

```
17.242.2.1 TEST_P ( SimTest , AcceptableError )
```

17.242.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.242.3 Variable Documentation

17.242.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.243 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

# Typedefs

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.243.1 Typedef Documentation
```

- 17.243.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.243.2 Function Documentation

```
17.243.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.243.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.243.3 Variable Documentation
- 17.243.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.244 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

# Functions

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.244.1 Typedef Documentation
- 17.244.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.244.2 Function Documentation
- 17.244.2.1 TEST\_P ( SimTest , AcceptableError )
- 17.244.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.244.3 Variable Documentation
- 17.244.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.245 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

# **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimTest

#### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## **Variables**

auto simulatorBackends = ::testing::Values(true, false)

```
17.245.1 Typedef Documentation
```

17.245.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimTest

```
17.245.2 Function Documentation
```

```
17.245.2.1 TEST_P ( SimTest , AcceptableError )
```

```
17.245.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
```

17.245.3 Variable Documentation

17.245.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.246 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

# Typedefs

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.246.1 Typedef Documentation
```

- 17.246.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimTest
- 17.246.2 Function Documentation

```
17.246.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.246.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.246.3 Variable Documentation
- 17.246.3.1 auto simulatorBackends = ::testing::Values(true, false)

### 17.247 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimTest

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.247.1 Typedef Documentation
- 17.247.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimTest
- 17.247.2 Function Documentation
- 17.247.2.1 TEST\_P ( SimTest , AcceptableError )
- 17.247.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.247.3 Variable Documentation
- 17.247.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.248 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

#### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## **Variables**

auto simulatorBackends = ::testing::Values(true, false)

```
17.248.1 Typedef Documentation
```

17.248.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest

```
17.248.2 Function Documentation
```

```
17.248.2.1 TEST_P ( SimTest , AcceptableError )
```

17.248.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.248.3 Variable Documentation

17.248.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.249 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

# Typedefs

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.249.1 Typedef Documentation
```

```
17.249.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimTest
```

```
17.249.2 Function Documentation
```

```
17.249.2.1 TEST_P ( SimTest , AcceptableError )
```

```
17.249.2.2 WRAPPED_INSTANTIATE_TEST_CASE_P ( MODEL_NAME , SimTest , simulatorBackends )
```

```
17.249.3 Variable Documentation
```

```
17.249.3.1 auto simulatorBackends = ::testing::Values(true, false)
```

### 17.250 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.250.1 Typedef Documentation
```

17.250.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest

```
17.250.2 Function Documentation
```

```
17.250.2.1 TEST_P ( SimTest , AcceptableError )
```

17.250.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )

17.250.3 Variable Documentation

17.250.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.251 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

#### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## **Variables**

auto simulatorBackends = ::testing::Values(true, false)

### 17.251.1 Typedef Documentation

- 17.251.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.251.2 Function Documentation
- 17.251.2.1 TEST\_P ( SimTest , AcceptableError )
- 17.251.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.251.3 Variable Documentation
- 17.251.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.252 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

# Typedefs

• typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

### **Functions**

- TEST P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

### **Variables**

```
• auto simulatorBackends = ::testing::Values(true, false)
```

```
17.252.1 Typedef Documentation
```

- 17.252.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.252.2 Function Documentation

```
17.252.2.1 TEST_P ( SimTest , AcceptableError )
```

- 17.252.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.252.3 Variable Documentation
- 17.252.3.1 auto simulatorBackends = ::testing::Values(true, false)

### 17.253 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

## **Typedefs**

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > SimTest

## **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

# Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.253.1 Typedef Documentation
- 17.253.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimTest
- 17.253.2 Function Documentation
- 17.253.2.1 TEST\_P ( SimTest , AcceptableError )
- 17.253.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.253.3 Variable Documentation
- 17.253.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.254 test.cc File Reference

```
#include "gtest/gtest.h"
#include <DEFINITIONS_HEADER>
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

### **Typedefs**

### **Functions**

- TEST\_P (SimTest, AcceptableError)
- WRAPPED\_INSTANTIATE\_TEST\_CASE\_P (MODEL\_NAME, SimTest, simulatorBackends)

## Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.254.1 Typedef Documentation
```

- 17.254.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy

  Dense> SimTest
- 17.254.2 Function Documentation
- 17.254.2.1 TEST\_P ( SimTest , AcceptableError )
- 17.254.2.2 WRAPPED\_INSTANTIATE\_TEST\_CASE\_P ( MODEL\_NAME , SimTest , simulatorBackends )
- 17.254.3 Variable Documentation
- 17.254.3.1 auto simulatorBackends = ::testing::Values(true, false)

## 17.255 utils.cc File Reference

```
#include "utils.h"
#include <fstream>
#include <stdint.h>
```

## **Macros**

#define UTILS\_CC

# **Functions**

- CUresult cudaFuncGetAttributesDriver (cudaFuncAttributes \*attr, CUfunction kern)
   Function for getting the capabilities of a CUDA device via the driver API.
- void gennError (const string &error)

Function called upon the detection of an error. Outputs an error message and then exits.

void writeHeader (ostream &os)

Function to write the comment header denoting file authorship and contact details into the generated code.

unsigned int theSize (const string &type)

Tool for determining the size of variable types on the current architecture.

```
17.255.1 Macro Definition Documentation
```

```
17.255.1.1 #define UTILS CC
```

17.255.2 Function Documentation

17.255.2.1 CUresult cudaFuncGetAttributesDriver ( cudaFuncAttributes \* attr, CUfunction kern )

Function for getting the capabilities of a CUDA device via the driver API.

```
17.255.2.2 void gennError ( const string & error )
```

Function called upon the detection of an error. Outputs an error message and then exits.

```
17.255.2.3 unsigned int the Size (const string & type)
```

Tool for determining the size of variable types on the current architecture.

```
17.255.2.4 void writeHeader (ostream & os)
```

Function to write the comment header denoting file authorship and contact details into the generated code.

# 17.256 utils.h File Reference

This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models.

```
#include <iostream>
#include <string>
#include <cuda.h>
#include <cuda_runtime.h>
```

### Macros

• #define \_UTILS\_H\_

macro for avoiding multiple inclusion during compilation

• #define CHECK CU ERRORS(call) call

Macros for catching errors returned by the CUDA driver and runtime APIs.

- #define CHECK\_CUDA\_ERRORS(call)
- #define B(x, i) ((x) & (0x80000000 >> (i)))

Bit tool macros.

• #define setB(x, i) x = ((x) | (0x800000000 >> (i)))

Set the bit at the specified position i in x to 1.

• #define delB(x, i) x= ((x) & ( $\sim$ (0x80000000 >> (i))))

Set the bit at the specified position i in x to 0.

#define USE(expr) do { (void)(expr); } while (0)

Miscellaneous macros.

#### **Functions**

CUresult cudaFuncGetAttributesDriver (cudaFuncAttributes \*attr, CUfunction kern)

Function for getting the capabilities of a CUDA device via the driver API.

void gennError (const string &error)

Function called upon the detection of an error. Outputs an error message and then exits.

unsigned int theSize (const string &type)

Tool for determining the size of variable types on the current architecture.

void writeHeader (ostream &os)

Function to write the comment header denoting file authorship and contact details into the generated code.

### 17.256.1 Detailed Description

This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models.

### 17.256.2 Macro Definition Documentation

```
17.256.2.1 #define _UTILS_H_
```

macro for avoiding multiple inclusion during compilation

```
17.256.2.2 #define B(x, i) ((x) & (0x800000000 >> (i)))
```

Bit tool macros.

Extract the bit at the specified position i from x

```
17.256.2.3 #define CHECK_CU_ERRORS( call ) call
```

Macros for catching errors returned by the CUDA driver and runtime APIs.

```
17.256.2.4 #define CHECK_CUDA_ERRORS( call )
```

# Value:

```
{
  cudaError_t error = call;
  if (error != cudaSuccess)
  {
    cerr << __FILE__ << ": " << __LINE__;
    cerr << ": cuda runtime error " << error << ": ";
    cerr << cudaGetErrorString(error) << endl;
    exit(EXIT_FAILURE);
  }
}</pre>
```

Set the bit at the specified position i in x to 0.

```
17.256.2.6 #define setB( x, i) x=((x) | (0x800000000 >> (i)))
```

Set the bit at the specified position i in x to 1.

```
17.256.2.7 #define USE( expr ) do { (void)(expr); } while (0)
```

Miscellaneous macros.

Silence 'unused parameter' warnings

### 17.256.3 Function Documentation

17.256.3.1 CUresult cudaFuncGetAttributesDriver ( cudaFuncAttributes \* attr, CUfunction kern )

Function for getting the capabilities of a CUDA device via the driver API.

```
17.256.3.2 void gennError ( const string & error )
```

Function called upon the detection of an error. Outputs an error message and then exits.

```
17.256.3.3 unsigned int the Size (const string & type)
```

Tool for determining the size of variable types on the current architecture.

```
17.256.3.4 void writeHeader (ostream & os)
```

Function to write the comment header denoting file authorship and contact details into the generated code.

# 17.257 VClampGA.cc File Reference

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU.

```
#include "VClampGA.h"
```

## **Functions**

• int main (int argc, char \*argv[])

This function is the entry point for running the project.

# 17.257.1 Detailed Description

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU.

## 17.257.2 Function Documentation

```
17.257.2.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the project.

# 17.258 VClampGA.h File Reference

Header file containing global variables and macros used in running the HHVClamp/VClampGA model.

```
#include <cassert>
#include <cuda_runtime.h>
#include "hr_time.h"
#include "stringUtils.h"
#include "utils.h"
#include "HHVClamp.cc"
#include "HHVClamp_CODE/definitions.h"
#include "randomGen.h"
#include "gauss.h"
#include "helper.h"
#include "GA.cc"
```

## Macros

#define RAND(Y, X) Y = Y \* 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767

### **Variables**

- · randomGen R
- · randomGauss RG
- CStopWatch timer

## 17.258.1 Detailed Description

Header file containing global variables and macros used in running the HHVClamp/VClampGA model.

```
17.258.2 Macro Definition Documentation
```

```
17.258.2.1 #define RAND( Y, X) Y = Y * 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767
```

17.258.3 Variable Documentation

17.258.3.1 randomGen R

17.258.3.2 randomGauss RG

17.258.3.3 CStopWatch timer

384 REFERENCES

# References

[1] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. 8, 9, 10, 11, 51, 65, 98, 100, 329

- [2] T. Nowotny. Parallel implementation of a spiking neuronal network model of unsupervised olfactory learning on NVidia CUDA. In P. Sobrevilla, editor, *IEEE World Congress on Computational Intelligence*, pages 3238–3245, Barcelona, 2010. IEEE. 29
- [3] Thomas Nowotny, Ramón Huerta, Henry DI Abarbanel, and Mikhail I Rabinovich. Self-organization in the olfactory system: one shot odor recognition in insects. *Biological cybernetics*, 93(6):436–446, 2005. 12, 14, 15, 16, 164
- [4] Nikolai F Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65(4):041922, 2002. 164
- [5] R. D. Traub and R. Miles. *Neural Networks of the Hippocampus*. Cambridge University Press, New York, 1991. 12, 14, 15, 16, 17, 32, 190

# Index

_FTYPE	_NMB
HHVClampParameters.h, 265	MBody1_project/model/sizes.h, 345
Izh_sparse_project/model/sizes.h, 344	MBody_delayedSyn_project/model/sizes.h, 345
MBody1_project/model/sizes.h, 345	MBody_individualID_project/model/sizes.h, 346
MBody_delayedSyn_project/model/sizes.h, 345	MBody_map_project/model/sizes.h, 346
MBody_individualID_project/model/sizes.h, 346	MBody_map_robot1_project/model/sizes.h, 347
MBody_map_project/model/sizes.h, 346	MBody_userdef_project/model/sizes.h, 347
MBody_map_robot1_project/model/sizes.h, 347	_NMaxConnP0
MBody_userdef_project/model/sizes.h, 347	lzh_sparse_project/model/sizes.h, 344
OneComp_project/model/sizes.h, 348	_NMaxConnP1
PoissonIzh_project/model/sizes.h, 348	lzh_sparse_project/model/sizes.h, 344
_IZH_SPARSE_MODEL_CC_	_NMaxConnP2
Izh_sparse_model.cc, 268	lzh_sparse_project/model/sizes.h, 344
_MAP_CLASSOL_CC_	_NMaxConnP3
MBody1_project/model/map_classol.cc, 270	lzh_sparse_project/model/sizes.h, 344
MBody_delayedSyn_project/model/map_classol.←	_NPoisson
cc, 270	PoissonIzh_project/model/sizes.h, 348
MBody_individualID_project/model/map_classol. ←	_ONECOMP_MODEL_CC_
cc, 271	OneComp_model.cc, 334
MBody_map_project/model/map_classol.cc, 271	_POISSONIZHMODEL_CC_
MBody_map_robot1_project/model/map_classol. ←	PoissonIzh-model.cc, 336
cc, 271	_SCHMUKER2014_CLASSIFIER_
MBody_userdef_project/model/map_classol.cc,	Schmuker2014_classifier.cc, 341
272	_UTILS_H_
_MODELSPEC_H_	utils.h, 381
modelSpec.h, 322	ISAAC_HPP
_NAL	isaac.cc, 267
MBody1_project/model/sizes.h, 345	$\sim$ NNmodel
MBody_delayedSyn_project/model/sizes.h, 345	NNmodel, 140
MBody_individualID_project/model/sizes.h, 346	~QTIsaac
MBody_map_project/model/sizes.h, 346	QTIsaac, 159
MBody_map_robot1_project/model/sizes.h, 347	~Schmuker2014_classifier
MBody_userdef_project/model/sizes.h, 347	Schmuker2014_classifier, 168
_NC1	~SynDelay
OneComp_project/model/sizes.h, 348	SynDelay, 188
_NExc	~classIzh
Izh_sparse_project/model/sizes.h, 344	classIzh, 77
_Ninh	~classol
Izh_sparse_project/model/sizes.h, 344	classol, 83
_Nizh	~neuronModel
PoissonIzh_project/model/sizes.h, 348	neuronModel, 135
_NLB	~neuronpop
MBody1_project/model/sizes.h, 345 MBody_delayedSyn_project/model/sizes.h, 345	neuronpop, 136 ~postSynModel
MBody_individualID_project/model/sizes.h, 346	postSynModel, 157
MBody map project/model/sizes.h, 346	~randctx
MBody_map_robot1_project/model/sizes.h, 347	QTIsaac::randctx, 160
MBody_map_robot1_project/model/sizes.h, 347  MBody_userdef_project/model/sizes.h, 347	~randomGauss
NLHI	randomGauss, 161
MBody1_project/model/sizes.h, 345	~randomGen
MBody_delayedSyn_project/model/sizes.h, 345	randomGen, 162
MBody_individualID_project/model/sizes.h, 346	~stdRG
MBody_map_project/model/sizes.h, 346	stdRG, 185
MBody_map_robot1_project/model/sizes.h, 347	~weightUpdateModel
MBody_userdef_project/model/sizes.h, 347	weightUpdateModel, 198
· ,	· · · · · · · · · · · · · · · · · · ·

00_MainPage.dox, 215	BITMASK_GLOBALG
01_Installation.dox, 215	synapseMatrixType.h, 355
02_Quickstart.dox, 215	BITMASK
03_Examples.dox, 215	synapseMatrixType.h, 355
09_ReleaseNotes.dox, 215	baserates
10_UserManual.dox, 215	classol, 90
11_Tutorial.dox, 215	baseV
12_Tutorial.dox, 215	inputSpec, 97
13_UserGuide.dox, 215	BlkSz
14 Credits.dox, 215	GeNNHelperKrnls.h, 258
	byte
ALLTOALL	QTIsaac, 159
modelSpec.h, 323	
ASSIGN ARRAY VARS	CACHE_DIR
simulation_test_vars.h, 343	experiment.h, 230
AUTODEVICE	CHECK_CU_ERRORS
modelSpec.h, 322	utils.h, <mark>381</mark>
activateDirectInput	CHECK_CUDA_ERRORS
NNmodel, 140	utils.h, 381
addExtraGlobalNeuronParams	CLUST SIZE AN
	Model_Schmuker_2014_classifier.cc, 319
SynapseGroup, 187	CLUST SIZE PN
addExtraGlobalParams	
NeuronGroup, 132	Model_Schmuker_2014_classifier.cc, 319
addExtraGlobalSynapseParams	CLUST_SIZE_RN
SynapseGroup, 187	Model_Schmuker_2014_classifier.cc, 319
addInSyn	CONNECTIVITY_AN_AN
NeuronGroup, 132	experiment.h, 230
addInputRate	CONNECTIVITY_PN_AN
Schmuker2014_classifier, 168	experiment.h, 230
addNeuronPopulation	CONNECTIVITY_PN_PN
NNmodel, 140	experiment.h, 230
addOutSyn	CONNECTIVITY_RN_PN
NeuronGroup, 132	experiment.h, 231
addSpkEventCondition	CPU
NeuronGroup, 132	modelSpec.h, 322
addSynapsePopulation	CStopWatch, 91
NNmodel, 141–143	CStopWatch, 92
•	getElapsedTime, 92
allocate_device_mem_input	startTimer, 92
classIzh, 77	stopTimer, 92
classol, 83	cacheDir
allocate_device_mem_patterns	
classIzh, 77	Schmuker2014_classifier, 170
classol, 83	calcKernelSizes
allocateHostAndDeviceMemory	SynapseGroup, 187
Schmuker2014_classifier, 168	calcNeurons
applyInputToClassifier	GENN_FLAGS, 63
experiment.cc, 229	calcSizes
applyLearningRuleSynapses	NeuronGroup, 132
Schmuker2014 classifier, 168	calcSynapseDynamics
asGoodAsZero	GENN_FLAGS, 63
GENN_PREFERENCES, 64	calcSynapses
autoChooseDevice	GENN FLAGS, 63
GENN_PREFERENCES, 64	calculateCurrentWindowWinner
autoRefractory	Schmuker2014 classifier, 168
GENN PREFERENCES, 64	calculateDerivedParameter
GLIVIN_I TILI ENERVOES, 04	dpclass, 93
В	expDecayDp, 97
gen_pnkc_syns_indivID.cc, 239	pwSTDP, 158
utils.h, 381	rulkovdp, 163
ulli3.11, 301	ταικοναρ, του

calculateOverallWinner	allocate_device_mem_patterns, 83
Schmuker2014_classifier, 168	baserates, 90
calculateVrResponse	classol, 83
Schmuker2014_classifier, 168	d_baserates, 90
calculateWinner	d_pattern, 90
Schmuker2014_classifier, 168	free_device_mem, 83, 84
CB	generate_baserates, 84
CodeHelper.h, 227	get_kcdnsyns, 84
Cexp	getSpikeNumbersFromGPU, 84
helper.h, 263	getSpikesFromGPU, 84, 85
checkContents	init, 85
Schmuker2014_classifier, 168	model, 90
checkNumDelaySlots	offset, 90
NeuronGroup, 132	output_spikes, 85
checkUnreplacedVariables	output_state, 85, 86
codeGenUtils.h, 226	outputDN_spikes, 86
lib/src/codeGenUtils.cc, 224	p_pattern, 90
chooseDevice	pattern, 90
generateALL.cc, 249	read_PNIzh1syns, 87
generateALL.h, 250	read_input_patterns, 86
classIzh, 76	read_kcdnsyns, 86, 87
∼classlzh, 77	read_pnkcsyns, 87
allocate_device_mem_input, 77	read_pnlhisyns, 87
allocate_device_mem_patterns, 77	read_sparsesyns_par, 87, 88
classIzh, 77	run, 88
copy_device_mem_input, 77	runCPU, 88, 89
create_input_values, 77	runGPU, 89
d_input1, 78	size_g, 90
d_input2, 78	sum_spikes, 89
free_device_mem, 77	sumDN, 90
gen_alltoall_syns, 77	sumlzh1, 90
getSpikeNumbersFromGPU, 77	sumKC, 91
getSpikesFromGPU, 77	sumLHI, 91
init, 78	sumPN, 91
initializeAllVars, 78	theRates, 91
input1, 78	write_kcdnsyns, 89
input2, 78	write_pnkcsyns, 90
model, 78	write_pnlhisyns, 90
output_params, 78	classol_sim.cc, 215–217
output_spikes, 78	classol_sim.h, 217, 219–223
output_state, 78	clear
randomizeVar, 78	extra_neurons.h, 232
randomizeVarSq, 78	extra_postsynapses.h, 233
read_input_values, 78	clearDownDevice
read_sparsesyns_par, 78	Schmuker2014_classifier, 168
run, 78	clearedDownDevice
setInput, 78	Schmuker2014_classifier, 170
sum_spikes, 78	closeBrace
sumPExc, 78	CodeHelper, 91
sumPlnh, 78	clusterSpikeCountAN
write_input_to_file, 78	Schmuker2014_classifier, 170
classLabel	codeGenUtils.cc, 224, 225
Schmuker2014_classifier, 170	codeGenUtils.h, 225
classifier	checkUnreplacedVariables, 226
experiment.cc, 229	ensureFtype, 226
classol, 79	GetPairKeyConstIter, 226
∼classol, 83	name_substitutions, 226
allocate_device_mem_input, 83	substitute, 226

value_substitutions, 226	Schmuker2014_classifier, 168
CodeHelper, 91	cudaFuncGetAttributesDriver
closeBrace, 91	utils.cc, 380
CodeHelper, 91	utils.h, 382
endl, 91	
openBrace, 91	D_MAX_RANDOM_NUM
setVerbose, 91	experiment.h, 231
CodeHelper.h, 227	d_baserates
CB, 227	classol, 90
ENDL, 227	d_input1
hlp, 227	classIzh, 78
OB, 227	d_input2
SAVEP, 227	classIzh, 78
command_line_processing.h, 227	d_inputRates
extract_bool_value, 228	Schmuker2014_classifier, 170
extract_option, 228	d_maxRandomNumber
extract string value, 228	Schmuker2014_classifier, 170
toLower, 228	d_pattern
toUpper, 228	classol, 90
compareErrTupel	DATASET_NAME
GA.cc, 234	experiment.h, 231
connN	DBG_SIZE
SparseProjection, 178	lzh_sparse_sim.h, 269
container	MBody_delayedSyn_project/model/classol_sim.h,
NamelterCtx, 111	219
convertToRateCode	MBody_individualID_project/model/classol_sim.h,
Schmuker2014_classifier, 168	220
copy_device_mem_input	OneComp_sim.h, 335
classIzh, 77	DECLARE_MODEL
copy_var	newModels.h, 328
helper.h, 263	DENSE_GLOBALG
correctClass	synapseMatrixType.h, 355
Schmuker2014_classifier, 170	DENSE_INDIVIDUALG
countAN	synapseMatrixType.h, 355 DENSE
Schmuker2014_classifier, 170	modelSpec.h, 323
countEntriesAbove	synapseMatrixType.h, 355
sparseUtils.h, 351	data_type
countPNAN	Schmuker2014_classifier, 168
Schmuker2014_classifier, 170	data type double
countPN	Schmuker2014_classifier, 168
Schmuker2014 classifier, 170	data_type_float
countRN	Schmuker2014_classifier, 168
Schmuker2014_classifier, 170	data_type_int
cpu_only	Schmuker2014 classifier, 168
parse options.h, 335	data_type_uint
create_input_values	Schmuker2014_classifier, 168
classIzh, 77	datasetName
createDirectory	Schmuker2014_classifier, 170
experiment.cc, 229	dbgMode
createPosttoPreArray	parse_options.h, 335
sparseUtils.cc, 349	debugCode
sparseUtils.h, 351	GENN PREFERENCES, 64
createPreIndices	decode_matrix_globalg_bitmask/model.cc
sparseUtils.cc, 349	modelDefinition, 283
sparseUtils.h, 351	neuron_ini, 284
createSparseConnectivityFromDense	synapses_ini, 284
sparseUtils.h, 351	decode_matrix_globalg_bitmask/model_new.cc
createWTAConnectivity	IMPLEMENT_MODEL, 306
•	<del>-</del>

modelDefinition, 306	DerivedParamVec
decode_matrix_globalg_bitmask/test.cc	NewModels::Base, 70
simulatorBackends, 359	deviceCount
TEST_P, 359	global.cc, 259
WRAPPED_INSTANTIATE_TEST_CASE_P, 359	global.h, 261
decode_matrix_globalg_dense/model.cc	deviceProp
modelDefinition, 284	global.cc, 259
neuron_ini, 284	global.h, 261
synapses_ini, 284	directoryExists
decode_matrix_globalg_dense/model_new.cc	experiment.cc, 229
IMPLEMENT_MODEL, 306	divi
modelDefinition, 306	experiment.h, 231
decode_matrix_globalg_dense/test.cc	dpNames
simulatorBackends, 360	neuronModel, 135 postSynModel, 157
TEST_P, 360	weightUpdateModel, 198
WRAPPED_INSTANTIATE_TEST_CASE_P, 360	dpclass, 93
decode_matrix_globalg_sparse/model.cc	calculateDerivedParameter, 93
modelDefinition, 285	dpclass.h, 228
neuron_ini, 285 synapses_ini, 285	dps
decode_matrix_globalg_sparse/model_new.cc	neuronModel, 135
IMPLEMENT_MODEL, 306	postSynModel, 157
modelDefinition, 306	weightUpdateModel, 198
decode matrix globalg sparse/test.cc	DT
simulatorBackends, 360	Model_Schmuker_2014_classifier.cc, 319
TEST_P, 360	FIZ
WRAPPED_INSTANTIATE_TEST_CASE_P, 360	EKexp
decode_matrix_individualg_dense/model.cc	helper.h, 263 ENDL
modelDefinition, 285	
neuron_ini, 285	CodeHelper.h, 227 ENaexp
synapses_ini, 285	helper.h, 263
decode_matrix_individualg_dense/model_new.cc	EXITSYN
IMPLEMENT_MODEL, 307	modelSpec.h, 322
modelDefinition, 307	EXPDECAY
decode_matrix_individualg_dense/test.cc	postSynapseModels.cc, 338
simulatorBackends, 361	postSynapseModels.h, 339
TEST_P, 361	Elexp
WRAPPED_INSTANTIATE_TEST_CASE_P, 361	helper.h, 263
decode_matrix_individualg_sparse/model.cc	endl
modelDefinition, 286	CodeHelper, 91
neuron_ini, 286	ensureFtype
synapses_ini, 286	codeGenUtils.h, 226
decode_matrix_individualg_sparse/model_new.cc	lib/src/codeGenUtils.cc, 224
IMPLEMENT_MODEL, 307 modelDefinition, 307	err
decode_matrix_individualg_sparse/test.cc	errTupel, 94
simulatorBackends, 361	errTupel, 94
TEST_P, 361	err, 94 id, 94
WRAPPED_INSTANTIATE_TEST_CASE_P, 361	evntThreshold
defaultDevice	weightUpdateModel, 198
GENN_PREFERENCES, 64	exlzh_ini
delB	OneComp.cc, 333
gen_pnkc_syns_indivID.cc, 239	PoissonIzh.cc, 336
utils.h, 381	exlzh_p
DerivedParamFunc	OneComp.cc, 333
NewModels::Base, 70	PoissonIzh.cc, 336
DerivedParamNameIterCtx	ExpCondUserDef, 95
standardSubstitutions.h, 353	getCurrentConverterCode, 96

matDaggy Cada 00	IMPLEMENT MODEL 200
getDecayCode, 96	IMPLEMENT_MODEL, 308
getDerivedParams, 96	modelDefinition, 308
getInstance, 96	extra_global_params_in_sim_code/test.cc
getParamNames, 96	SimTest, 362
ParamValues, 96	simulatorBackends, 362
VarValues, 96	TEST_P, 362
expDecayDp, 96	WRAPPED_INSTANTIATE_TEST_CASE_P, 362
calculateDerivedParameter, 97	extra_global_params_in_sim_code_event_sparse_
experiment.cc, 228	inv/model.cc
applyInputToClassifier, 229	modelDefinition, 287
classifier, 229	neuron_ini, 287
createDirectory, 229	synapses_ini, 287
directoryExists, 229	extra_global_params_in_sim_code_event_sparse_
getAverage, 229	inv/model_new.cc
getStdDev, 229	IMPLEMENT_MODEL, 308
getUniqueRunId, 229	modelDefinition, 308
main, 229	extra_global_params_in_sim_code_event_sparse_
outputRunParameters, 229	inv/test.cc
Parameter, 229	SimTest, 363
printTextFile, 229	simulatorBackends, 363
RAND, 229	TEST_P, 363
S_ISDIR, 229	WRAPPED_INSTANTIATE_TEST_CASE_P, 363
setDefaultParamValues, 229	extra_global_post_param_in_sim_code/model.cc
vectorContains, 229	modelDefinition, 288
experiment.h, 230	neuron_ini, 288
CACHE_DIR, 230	synapses_ini, 288
CONNECTIVITY AN AN, 230	extra_global_post_param_in_sim_code/model_new.cc
CONNECTIVITY PN AN, 230	IMPLEMENT_MODEL, 308
CONNECTIVITY_PN_PN, 230	modelDefinition, 308
CONNECTIVITY_RN_PN, 231	extra_global_post_param_in_sim_code/test.cc
D_MAX_RANDOM_NUM, 231	SimTest, 363
DATASET_NAME, 231	simulatorBackends, 364
divi, 231	TEST_P, 363
FLAG_RUN_ON_CPU, 231	WRAPPED_INSTANTIATE_TEST_CASE_P, 363
GLOBAL_WEIGHT_SCALING, 231	extra_global_pre_param_in_sim_code/model.cc
MAX_FIRING_RATE_HZ, 231	modelDefinition, 288
MAX_WEIGHT_PN_AN, 231	neuron_ini, 288
MIN_FIRING_RATE_HZ, 231	synapses_ini, 289
MIN_WEIGHT_PN_AN, 231	extra_global_pre_param_in_sim_code/model_new.cc
N FOLDING, 231	IMPLEMENT_MODEL, 309
OUTPUT_DIR, 231	modelDefinition, 309
PLASTICITY_INTERVAL_MS, 231	extra_global_pre_param_in_sim_code/test.cc
RECORDING_TIME_MS, 231	SimTest, 364
RECORDINGS DIR, 231	simulatorBackends, 364
REPEAT_LEARNING_SET, 231	TEST_P, 364
SPIKING_ACTIVITY_THRESHOLD_HZ, 231	WRAPPED INSTANTIATE TEST CASE P, 364
TOTAL RECORDINGS, 231	extra_neurons.h, 231
<del>-</del>	
timer, 231	clear, 232
UINT, 231	push_back, 232
VR_DATA_FILENAME, 231	simCode, 232
WEIGHT_DELTA_PN_AN, 231	extra_postsynapses.h, 233
WEIGHT_RN_PN, 231	clear, 233
WEIGHT_WTA_AN_AN, 231	postSynDecay, 233
WEIGHT_WTA_PN_PN, 231	postSyntoCurrent, 233
extra_global_params_in_sim_code/model.cc	push_back, 233
modelDefinition, 287	extra_weightupdates.h, 234
neuron_ini, 287	extraGlobalNeuronKernelParameterTypes
extra_global_params_in_sim_code/model_new.cc	neuronModel, 135

extraGlobalNeuronKernelParameters	GENN_PREFERENCES, 63
neuronModel, 135	asGoodAsZero, 64
ExtraGlobalParamNameIterCtx	autoChooseDevice, 64
standardSubstitutions.h, 353	autoRefractory, 64
extraGlobalSynapseKernelParameterTypes	debugCode, 64
weightUpdateModel, 198	defaultDevice, 64
extraGlobalSynapseKernelParameters	learningBlockSize, 64
weightUpdateModel, 198	neuronBlockSize, 64
extract_bool_value	optimiseBlockSize, 64
command_line_processing.h, 228	optimizeCode, 64
extract_option	showPtxInfo, 65
command_line_processing.h, 228	synapseBlockSize, 65
extract_string_value	synapseDynamicsBlockSize, 65
command_line_processing.h, 228	userCxxFlagsGNU, 65
FLAG BUN ON OBU	userCxxFlagsWIN, 65
FLAG_RUN_ON_CPU	userNvccFlags, 65
experiment.h, 231	gKexp
finalize	helper.h, 263
NNmodel, 144	GLOBAL_CC
findMaxMinSampleDistances	global.cc, 259
Schmuker2014_classifier, 168	GLOBAL_WEIGHT_SCALING
findNeuronGroup	experiment.h, 231
NNmodel, 144	GLOBALG
findSynapseGroup	modelSpec.h, 323
NNmodel, 144	GLOBAL
fixsynapse	synapseMatrixType.h, 355
parse_options.h, 335	gNaexp
FloatType	helper.h, 263
modelSpec.h, 322	GOLDEN_RATIO
for	isaac.cc, 267
parse_options.h, 335	gPNKC_GLOBAL
free_device_mem	MBody_individuaIID_project/model/sizes.h, 346
classIzh, 77	GPU
classol, 83, 84	modelSpec.h, 322
ftype parse options.h, 335	garray
parse_options.rr, 333	gen_syns_sparse_izhModel.cc, 241
g_ee	garray_ee
gen_syns_sparse_izhModel.cc, 241	gen_syns_sparse_izhModel.cc, 241
g_ei	garray_ei
gen_syns_sparse_izhModel.cc, 241	gen_syns_sparse_izhModel.cc, 241
g_ie	garray_ie
gen_syns_sparse_izhModel.cc, 241	gen_syns_sparse_izhModel.cc, 241
g_ii	garray_ii
gen_syns_sparse_izhModel.cc, 241	gen_syns_sparse_izhModel.cc, 241
GA.cc, 234	gauss.cc, 234
compareErrTupel, 234	GAUSS_CC, 235
procreatePop, 234	gauss.h, 235
GAUSS_CC	GAUSS_H, 235
gauss.cc, 235	GeNNHelperKrnls.cu, 257
GAUSS H	generate_random_gpuInput_xorwow, 257
gauss.h, 235	<pre>generate_random_gpuInput_xorwow&lt; double &gt;,</pre>
GENN_FLAGS, 63	258
calcNeurons, 63	generate_random_gpuInput_xorwow< float >, 258
calcSynapseDynamics, 63	setup_kernel, 258
calcSynapses, 63	xorwow_setup, 258
learnSynapsesPost, 63	GeNNHelperKrnls.h, 258
GENN_LONG_DOUBLE	BlkSz, 258
modelSpec.h, 323	<pre>generate_random_gpuInput_xorwow, 258</pre>

setup_kernel, 258	R, 242
xorwow_setup, 258	Rind, 242
GeNNReady	genMakefile
modelSpec.h, 323	generateRunner.cc, 255
src/modelSpec.cc, 321	generateRunner.h, 256
gen_alltoall_syns	genNeuronFunction
classIzh, 77	generateCPU.cc, 252
gen_input_structured.cc, 235	generateCPU.h, 252
main, 236	genNeuronKernel
R, 236	generateKernels.cc, 253
gen_kcdn_syns.cc, 236	generateKernels.h, 254
main, 236	genRunner
R, 236	generateRunner.cc, 255
RG, 236	generateRunner.h, 256
gen_kcdn_syns_fixto10K.cc, 237	genRunnerGPU
main, 237	generateRunner.cc, 255
R, 237	generateRunner.h, 257
R2, 237	genSynapseFunction
RG, 237	generateCPU.cc, 252 generateCPU.h, 253
gen_pnkc_syns.cc, 237	
main, 238 R, 238	genSynapseKernel
RG, 238	generateKernels.cc, 253 generateKernels.h, 254
	generate_baserates
gen_pnkc_syns_indivID.cc, 238 B, 239	classol, 84
delB, 239	generate_inputrates_dataset
main, 239	Schmuker2014_classifier, 168
R, 239	generate_model_runner
RG, 239	generateALL.cc, 250
setB, 239	generateALL.h, 251
gen_pnlhi_syns.cc, 239	generate_or_load_inputrates_dataset
main, 239	Schmuker2014 classifier, 168
gen_syns_sparse.cc, 239	generate_random_gpuInput_xorwow
main, 240	GeNNHelperKrnls.cu, 257
R, 240	GeNNHelperKrnls.h, 258
RG, 240	generate_random_gpuInput_xorwow< double >
gen_syns_sparse_izhModel.cc, 240	GeNNHelperKrnls.cu, 258
g_ee, 241	generate_random_gpulnput_xorwow< float >
g_ei, 241	GeNNHelperKrnls.cu, 258
g_ie, 241	generate_run.cc, 242-248
g_ii, 241	generateALL.cc, 249
garray, 241	chooseDevice, 249
garray_ee, 241	generate_model_runner, 250
garray_ei, 241	hlp, 250
garray_ie, 241	main, 250
garray_ii, 241	generateALL.h, 250
gsyn, 241	chooseDevice, 250
ind, 241	generate_model_runner, 251
ind_ee, 242	generateCPU.cc, 251
ind_ei, 242	genNeuronFunction, 252
ind_ie, 242	genSynapseFunction, 252
ind_ii, 242	generateCPU.h, 252
indlnG_ee, 242	genNeuronFunction, 252
indlnG_ei, 242	genSynapseFunction, 253
indlnG_ie, 242	generateKernels.cc, 253
indlnG_ii, 242	genNeuronKernel, 253
main, 241	genSynapseKernel, 253
printVector, 241	generateKernels.h, 254

genNeuronKernel, 254	getEventThresholdConditionCode
genSynapseKernel, 254	StaticGradedUserDef, 181
generateRunner.cc, 255	WeightUpdateModel, 206, 207
genMakefile, 255	WeightUpdateModelSpikeEvent, 214
genRunner, 255	WeightUpdateModels::Base, 75
genRunnerGPU, 255	WeightUpdateModels::LegacyWrapper, 104
generateRunner.h, 256	WeightUpdateModels::StaticGraded, 180
genMakefile, 256	getExtraGlobalParams
genRunner, 256	MyHH, 107
genRunnerGPU, 257	Neuron, 120
generateSimulatedTimeSeriesData	NeuronModels::Base, 71
Schmuker2014 classifier, 168	
<del>-</del>	NeuronModels::LegacyWrapper, 106 NeuronModels::Poisson, 155
gennError utils.cc, 380	WeightUpdateModel, 207
utils.h, 382	WeightUpdateModels::Base, 75
get_kcdnsyns	WeightUpdateModels::LegacyWrapper, 104
classol, 84	getIDRange
getAverage	NeuronGroup, 132
experiment.cc, 229	getInSyn
getClassCluster	NeuronGroup, 132
Schmuker2014_classifier, 168	getInitVals
getClusterIndex	NeuronGroup, 132
Schmuker2014_classifier, 168	getInstance
GetCode	ExpCondUserDef, 96
SingleValueSubstitutionTest, 177	MyHH, 107
getCurrentConverterCode	Mylzhikevich, 109
ExpCondUserDef, 96	MylzhikevichVariable, 110
PostsynapticModels::Base, 73	Neuron, 120, 121
PostsynapticModels::DeltaCurr, 93	NeuronModels::Izhikevich, 99
PostsynapticModels::ExpCond, 95	NeuronModels::IzhikevichVariable, 101
PostsynapticModels::LegacyWrapper, 101	NeuronModels::Poisson, 155
getDecayCode	NeuronModels::RulkovMap, 165
ExpCondUserDef, 96	NeuronModels::SpikeSource, 179
PostsynapticModels::Base, 73	NeuronModels::TraubMiles, 191
PostsynapticModels::ExpCond, 95	NeuronModels::TraubMilesAlt, 192
PostsynapticModels::LegacyWrapper, 102	NeuronModels::TraubMilesFast, 193
getDelaySteps	NeuronModels::TraubMilesNStep, 194
SynapseGroup, 187	PiecewiseSTDPUserDef, 153
getDerivedParams	PostsynapticModels::DeltaCurr, 93
ExpCondUserDef, 96	PostsynapticModels::ExpCond, 95
NeuronGroup, 132	StaticGradedUserDef, 182
NeuronModels::RulkovMap, 165	StaticPulseUserDef, 184
NewModels::Base, 70	WeightUpdateModel, 207, 208
NewModels::LegacyWrapper, 103	WeightUpdateModelSpikeEvent, 214
PiecewiseSTDPUserDef, 153	WeightUpdateModels::PiecewiseSTDP, 151
PostsynapticModels::ExpCond, 95	WeightUpdateModels::StaticGraded, 180
WeightUpdateModels::PiecewiseSTDP, 151	WeightUpdateModels::StaticPulse, 183
getDT	getLearnPostCode
NNmodel, 144	PiecewiseSTDPUserDef, 153
getElapsedTime	WeightUpdateModel, 208
CStopWatch, 92	WeightUpdateModels::Base, 75
getEventCode	WeightUpdateModels::LegacyWrapper, 104
StaticGradedUserDef, 181	WeightUpdateModels::PiecewiseSTDP, 151
WeightUpdateModel, 206	getLearnPostSupportCode
WeightUpdateModelSpikeEvent, 214	WeightUpdateModel, 208
WeightUpdateModels::Base, 75	WeightUpdateModels::Base, 75
WeightUpdateModels::LegacyWrapper, 104	WeightUpdateModels::LegacyWrapper, 104
WeightUpdateModels::StaticGraded, 180	getManhattanDistance

Schmuker2014_classifier, 168	PiecewiseSTDPUserDef, 153
getMatrixType	PostsynapticModels::DeltaCurr, 93
SynapseGroup, 187	PostsynapticModels::ExpCond, 95
getMaxConnections	StaticGradedUserDef, 182
SynapseGroup, 187	WeightUpdateModel, 209
getName	WeightUpdateModelSpikeEvent, 214
NNmodel, 144	WeightUpdateModels::PiecewiseSTDP, 151
NeuronGroup, 132	WeightUpdateModels::StaticGraded, 180
SynapseGroup, 187	getParams
getNeuronGridSize	NeuronGroup, 132
NNmodel, 144	getPrecision
getNeuronGroups	NNmodel, 144
NNmodel, 144	getQueueOffset
getNeuronKernelParameters	NeuronGroup, 132
NNmodel, 144	getRNType
getNeuronModel	NNmodel, 145
NeuronGroup, 132	getRand0to1
getNumDelaySlots	Schmuker2014 classifier, 168
NeuronGroup, 132	getRecordingFilename
getNumNeurons	Schmuker2014 classifier, 169
NNmodel, 144	getResetCode
NeuronGroup, 132	NeuronModels::Base, 72
getOffsetPost	NeuronModels::LegacyWrapper, 106
SynapseGroup, 187	getResetKernel
getOffsetPre	NNmodel, 144
SynapseGroup, 187	getSampleDistance
getOutSyn	Schmuker2014_classifier, 169
NeuronGroup, 132	getSeed
getPSDerivedParams	NNmodel, 145
SynapseGroup, 187	getSimCode
getPSInitVals	MyHH, 108
SynapseGroup, 187	Mylzhikevich, 109
getPSModel	MylzhikevichVariable, 110
SynapseGroup, 187	Neuron, 121–125
getPSParams	NeuronModels::Base, 72
SynapseGroup, 187	NeuronModels::Izhikevich, 99
getPaddedDynKernelSize	NeuronModels::LegacyWrapper, 106
SynapseGroup, 187	NeuronModels::Poisson, 156
getPaddedIDRange	NeuronModels::RulkovMap, 165
NeuronGroup, 132	NeuronModels::TraubMiles, 191
getPaddedKernelIDRange	NeuronModels::TraubMilesAlt, 192
SynapseGroup, 187	NeuronModels::TraubMilesFast, 193
getPaddedPostLearnKernelSize	NeuronModels::TraubMilesNStep, 195
SynapseGroup, 187	PiecewiseSTDPUserDef, 153
GetPairKeyConstIter	StaticPulseUserDef, 184
codeGenUtils.h, 226	WeightUpdateModel, 209, 210
getParamNames	WeightUpdateModels::Base, 75
ExpCondUserDef, 96	WeightUpdateModels::LegacyWrapper, 105
Mylzhikevich, 109	WeightUpdateModels::PiecewiseSTDP, 152
Neuron, 121	WeightUpdateModels::StaticPulse, 183
NeuronModels::Izhikevich, 99	getSimLearnPostKernelParameters
NeuronModels::IzhikevichVariable, 101	NNmodel, 145
NeuronModels::Poisson, 156	getSimSupportCode
NeuronModels::RulkovMap, 165	WeightUpdateModel, 210
NeuronModels::TraubMiles, 191	WeightUpdateModels::Base, 75
NeuronModels::TraubMilesNStep, 195	WeightUpdateModels::LegacyWrapper, 105
NewModels::Base, 70	getSpanType
NewModels::LegacyWrapper, 103	SynapseGroup, 187

getSparseVar	NeuronModels::SpikeSource, 179
sparseUtils.h, 351	NeuronModels::TraubMiles, 191
getSpikeEventCondition	getTrgNeuronGroup
NeuronGroup, 132	SynapseGroup, 187
getSpikeNumbersFromGPU	getUniqueRunId
classIzh, 77	experiment.cc, 229
classol, 84	getValues
neuronpop, 136	NewModels::ValueBase, 195
Schmuker2014 classifier, 169	NewModels::ValueBase< 0 >, 196
getSpikesFromGPU	getVars
classIzh, 77	MyHH, 108
classol, 84, 85	MylzhikevichVariable, 110
neuronpop, 137	Neuron, 128–131
Schmuker2014_classifier, 169	NeuronModels::Izhikevich, 99
getSrcNeuronGroup	NeuronModels::IzhikevichVariable, 101
SynapseGroup, 187	NeuronModels::Poisson, 156
getStdDev	NeuronModels::RulkovMap, 165
experiment.cc, 229	NeuronModels::TraubMiles, 191
getSupportCode	NewModels::Base, 70
Neuron, 125	NewModels::LegacyWrapper, 103
NeuronModels::Base, 72	PiecewiseSTDPUserDef, 153
NeuronModels::LegacyWrapper, 106	StaticGradedUserDef, 182
PostsynapticModels::Base, 73	StaticPulseUserDef, 184
PostsynapticModels::LegacyWrapper, 102	WeightUpdateModel, 211–213
getSynapseDynamicsCode	WeightUpdateModelSpikeEvent, 214
WeightUpdateModel, 210, 211	WeightUpdateModels::PiecewiseSTDP, 152
WeightUpdateModels::Base, 76	WeightUpdateModels::StaticGraded, 180
WeightUpdateModels::LegacyWrapper, 105	WeightUpdateModels::StaticPulse, 183
getSynapseDynamicsGridSize	getWUDerivedParams
NNmodel, 145	SynapseGroup, 187
getSynapseDynamicsGroups	getWUInitVals
NNmodel, 145	SynapseGroup, 187
getSynapseDynamicsKernelParameters	getWUModel
NNmodel, 145	SynapseGroup, 187
getSynapseDynamicsSuppportCode	getWUParams
WeightUpdateModel, 211	SynapseGroup, 187
WeightUpdateModels::Base, 76	getG
WeightUpdateModels::LegacyWrapper, 105	sparseUtils.h, 351
getSynapseGroups	glexp
NNmodel, 145	helper.h, 263
getSynapseKernelGridSize	global.cc, 258
NNmodel, 145	deviceCount, 259
getSynapseKernelParameters	deviceProp, 259
NNmodel, 145	GLOBAL_CC, 259
getSynapsePostLearnGridSize	hostCount, 259
NNmodel, 145	learnBlkSz, 259
getSynapsePostLearnGroups	neuronBlkSz, 259
NNmodel, 145	synDynBlkSz, 259
GetTheW	synapseBlkSz, 259
SimulationSynapsePolicyDense, 174	theDevice, 259
getThresholdConditionCode	global.h, 260
MyHH, 108	deviceCount, 261
Neuron, 126–128	deviceProp, 261
NeuronModels::Base, 72	hostCount, 261
NeuronModels::Izhikevich, 99	learnBlkSz, 261
NeuronModels::LegacyWrapper, 106	neuronBlkSz, 261
NeuronModels::Poisson, 156	synDynBlkSz, 261
NeuronModels::RulkovMap, 165	synapseBlkSz, 261

theDevice, 261 gpKCDN MBody_userdef.cc, 283 gpPNKC	generateALL.cc, 250 hostCount global.cc, 259 global.h, 261
MBody_userdef.cc, 283	hr_time.cc, 265
gsyn	HR_TIMER, 265
gen_syns_sparse_izhModel.cc, 241	hr_time.h, 265
HHVClamp.cc, 264 IMPLEMENT_MODEL, 264 modelDefinition, 264	IMPLEMENT_MODEL decode_matrix_globalg_bitmask/model_new.cc,
myHH_ini, 264	306
HHVClampParameters.h, 265 _FTYPE, 265	decode_matrix_globalg_dense/model_new.cc, 306 decode_matrix_globalg_sparse/model_new.cc,
NPOP, 265	306
SCALAR_MAX, 265	decode_matrix_individualg_dense/model_new.cc,
SCALAR_MIN, 265	307
scalar, 265	decode_matrix_individualg_sparse/model_new.cc,
TOTALT, 265	307
HHVclampGA_project/generate_run.cc main, 242	extra_global_params_in_sim_code/model_new.cc, 308
HR_TIMER	extra_global_params_in_sim_code_event_ <i>←</i>
hr_time.cc, 265	sparse_inv/model_new.cc, 308
helper.h, 262	extra_global_post_param_in_sim_code/model_←
Cexp, 263	new.cc, 308
copy_var, 263	extra_global_pre_param_in_sim_code/model_←
EKexp, 263	new.cc, 309
ENaexp, 263	HHVClamp.cc, 264
Elexp, 263	Izh_sparse.cc, 268
gKexp, 263	MBody_userdef.cc, 282
gNaexp, 263	Model_Schmuker_2014_classifier.cc, 320
glexp, 263	neuron_support_code_sim/model_new.cc, 309
hexp, 263	neuron_support_code_threshold/model_new.cc,
initexpHH, 263	310
initl, 263	newModels.h, 328
limit, 263	newNeuronModels.cc, 328, 329
mexp, 263	newPostsynapticModels.cc, 330
nexp, 263	newWeightUpdateModels.cc, 331
operator<<, 263	OneComp.cc, 333
runexpHH, 263	post_vars_in_post_learn/model_new.cc, 310
sigENa, 264	post_vars_in_post_learn_sparse/model_new.cc,
sigEK, 263	311
sigEl, 263	post_vars_in_sim_code/model_new.cc, 311
sigGNa, 264	post_vars_in_sim_code_sparse/model_new.cc,
sigGK, 264	312
sigGl, 264	post_vars_in_synapse_dynamics/model_new.cc,
sigC, 263	312
single_var_init_fullrange, 263	post_vars_in_synapse_dynamics_sparse/model⊷
single_var_reinit, 263	_new.cc, 312
truevar_init, 263	pre_vars_in_post_learn/model_new.cc, 313
truevar_initexpHH, 263	pre_vars_in_post_learn_sparse/model_new.cc,
var_init_fullrange, 263	313
var_reinit, 263	pre_vars_in_sim_code/model_new.cc, 314
Vexp, 264	pre_vars_in_sim_code_event/model_new.cc, 314
write_para, 263	pre_vars_in_sim_code_event_sparse/model_↔
hexp	new.cc, 315
helper.h, 263	pre_vars_in_sim_code_event_sparse_inv/model←
hlp	_new.cc, 315
CodeHelper.h, 227	pre_vars_in_sim_code_sparse/model_new.cc, 316

	pre_vars_in_synapse_dynamics/model_new.cc,	indlr	nG_ie
	316		gen_syns_sparse_izhModel.cc, 242
	pre_vars_in_synapse_dynamics_sparse/model_	indlr	nG_ii
	new.cc, 316		gen_syns_sparse_izhModel.cc, 242
	SynDelay.cc, 358	indlr	nG
	synapse_support_code_event_sim_code/model←		SparseProjection, 178
	_new.cc, 317	indiv	vidualSpikeCountPN
	synapse_support_code_event_threshold/model_		Schmuker2014_classifier, 170
	new.cc, 317	Init	
	synapse_support_code_post_learn/model_new.cc,		SimTest, 171, 172
	318		SimulationNeuronPolicyPrePostVar, 172
	synapse_support_code_sim_code/model_new.cc,		SimulationNeuronPolicyPreVar, 172
	318		SimulationSynapsePolicy, 173
	synapse_support_code_synapse_dynamics/model↔		SimulationSynapsePolicyDense, 174
	_new.cc, 319		SimulationSynapsePolicyNone, 174
INIDI	VIDUALID		SimulationSynapsePolicyNone, 174 SimulationSynapsePolicySparse, 175
	modelSpec.h, 323		SimulationTest, 175
	VIDUALG		SimulationTestVars, 176
		::4	Simulation restvars, 176
	modelSpec.h, 323 VIDUAL	init	alacalah 70
			classIzh, 78
	synapseMatrixType.h, 355		classol, 85
	BSYN		neuronpop, 137
	modelSpec.h, 322	InitD	DerivedParams
_	_SPARSE		NeuronGroup, 132
	simulation_test.h, 343		SynapseGroup, 187
	TANTIATE_TEST_CASE_P	initG	GeNN
	tests/unit/codeGenUtils.cc, 225		modelSpec.h, 323
ISAA	AC_INT		src/modelSpec.cc, 320
	isaac.cc, 267	inite	xpHH
IZHII	KEVICH_PS		helper.h, 263
	postSynapseModels.cc, 338	initl	
	postSynapseModels.h, 339		helper.h, 263
IZHII	KEVICH_V	initia	aliseInputData
	neuronModels.cc, 325		Schmuker2014_classifier, 169
	neuronModels.h, 326	initia	aliseWeights_DENSE_PN_AN
IZHII	KEVICH		Schmuker2014_classifier, 169
	neuronModels.cc, 325	initia	aliseWeights_SPARSE_RN_PN
	neuronModels.h, 326		Schmuker2014_classifier, 169
id		initia	aliseWeights_WTA_AN_AN
	errTupel, 94		Schmuker2014_classifier, 169
if		initia	aliseWeights_WTA_PN_PN
	parse_options.h, 335		Schmuker2014_classifier, 169
ind		initia	alizeAllVars
	gen_syns_sparse_izhModel.cc, 241		classIzh, 78
	QTIsaac, 159	initia	alizeSparseArray
	SparseProjection, 178		sparseUtils.cc, 349
ind 6	•		sparseUtils.h, 351
_	gen_syns_sparse_izhModel.cc, 242	initia	alizeSparseArrayPreInd
ind_e			sparseUtils.cc, 349
	gen_syns_sparse_izhModel.cc, 242		sparseUtils.h, 351
ind_i		initia	alizeSparseArrayRev
	gen_syns_sparse_izhModel.cc, 242		sparseUtils.cc, 349
ind i			sparseUtils.h, 351
	gen_syns_sparse_izhModel.cc, 242	inpu	•
indln	ig_ee	ıı ıpu	classIzh, 78
	gen_syns_sparse_izhModel.cc, 242	inpu	
		ıııpu	
iiiuliî	G_ei	Inn	classIzh, 78
	gen_syns_sparse_izhModel.cc, 242	mpu	tBaseRate

MBody1_project/model/classol_sim.h, 218 MBody_delayedSyn_project/model/classol_sim.h,	isSpikeZeroCopyEnabled NeuronGroup, 133
219	isSynapseGroupDynamicsRequired
MBody_individualID_project/model/classol_sim.h,	NNmodel, 146
221	isSynapseGroupPostLearningRequired
MBody_map_project/model/classol_sim.h, 221	NNmodel, 146
MBody_map_robot1_project/model/classol_sim.h,	isTimingEnabled
MPady upped f project/model/classed sim h 222	NNmodel, 146
MBody_userdef_project/model/classol_sim.h, 223 Poissonlzh_sim.h, 337	isTrueSpikeRequired NeuronGroup, 133
inputFac	SynapseGroup, 188
Izh_sparse_project/model/sizes.h, 344	isVarQueueRequired
inputRates	NeuronGroup, 133
Schmuker2014_classifier, 170	isVarZeroCopyEnabled
inputRatesSize	NeuronGroup, 133
Schmuker2014_classifier, 170	isWUVarZeroCopyEnabled
inputSpec, 97	SynapseGroup, 188
baseV, 97	isZeroCopyEnabled
N, 97	NeuronGroup, 133
st, 97	SynapseGroup, 188
t, 97	isaac
V, 97	QTIsaac, 159
isDelayRequired	isaac.cc, 266
NeuronGroup, 133	ISAAC_HPP, 267
isEventThresholdReTestRequired	GOLDEN_RATIO, 267
SynapseGroup, 187	ISAAC_INT, 267
isFinalized	lzh_sim_sparse.cc, 267
NNmodel, 145	main, 267
isPSAtomicAddRequired	Izh_sparse.cc, 267
SynapseGroup, 188	IMPLEMENT_MODEL, 268
isPSVarZeroCopyEnabled	IzhExc_ini, 268
SynapseGroup, 188	IzhInh_ini, 268
isParamRequiredBySpikeEventCondition	meanInpExc, 268
NeuronGroup, 133	meanInpInh, 268
isPoisson	modelDefinition, 268
NeuronModels::LegacyWrapper, 106	neuronPSize, 268
NeuronModels::Poisson, 156	neuronVSize, 268
isPostSpikeTimeRequired PiecewiseSTDPUserDef, 153	SynIzh_ini, 268
WeightUpdateModels::Base, 76	synapsePSize, 268 lzh_sparse_model.cc, 268
WeightUpdateModels::LegacyWrapper, 105	_IZH_SPARSE_MODEL_CC_, 268
WeightUpdateModels::PiecewiseSTDP, 152	R, 268
isPreSpikeTimeRequired	RG, 268
PiecewiseSTDPUserDef, 154	Izh_sparse_model.h, 269
WeightUpdateModels::Base, 76	lzh_sparse_project/generate_run.cc
WeightUpdateModels::LegacyWrapper, 105	main, 243
WeightUpdateModels::PiecewiseSTDP, 152	openFileGetMax, 243
isQueueRequired	lzh_sparse_project/model/sizes.h
NeuronGroup, 133	FTYPE, 344
isSpikeEventRequired	_NExc, 344
NeuronGroup, 133	_NInh, 344
SynapseGroup, 188	_NMaxConnP0, 344
isSpikeEventZeroCopyEnabled	_NMaxConnP1, 344
NeuronGroup, 133	_NMaxConnP2, 344
isSpikeTimeRequired	_NMaxConnP3, 344
NeuronGroup, 133	inputFac, 344
isSpikeTimeZeroCopyEnabled	SCALAR_MAX, 344
NeuronGroup, 133	SCALAR_MIN, 344

1 044	145
scalar, 344	MBody1_project/generate_run.cc
Izh_sparse_sim.h, 269	main, 244
DBG_SIZE, 269	MBody1_project/model/MBody1.cc
T_REPORT_TME, 269	modelDefinition, 275
TOTAL_TME, 269	myDNDN_ini, 275
timer, 269	myDNDN_p, 275
IzhExc_ini	myKCDN_ini, 275
lzh_sparse.cc, 268	myKCDN_p, 275
IzhInh ini	-
<del>-</del>	myLHIKC_ini, 275
Izh_sparse.cc, 268	myLHIKC_p, 275
L EADNIA OVALADOE	myPNKC_ini, 275
LEARN1SYNAPSE	myPNLHI_ini, 275
synapseModels.cc, 356	myPOI_ini, 275
synapseModels.h, 357	myPOI_p, 275
LEARNING	postExpDNDN, 275
modelSpec.h, 322	postExpKCDN, 275
learnBlkSz	postExpLHIKC, 275
global.cc, 259	•
global.h, 261	postExpPNKC, 275
learnSynapsesPost	postExpPNLHI, 275
GENN_FLAGS, 63	stdTM_ini, 275
	stdTM_p, 275
learningBlockSize	MBody1_project/model/classol_sim.cc
GENN_PREFERENCES, 64	main, 215
LegacyWrapper	MBody1_project/model/classol_sim.h
NeuronModels::LegacyWrapper, 106	InputBaseRate, 218
NewModels::LegacyWrapper, 103	MYRAND, 218
PostsynapticModels::LegacyWrapper, 101	
WeightUpdateModels::LegacyWrapper, 104	PAT_TIME, 218
lib/src/codeGenUtils.cc	PATFTIME, 218
checkUnreplacedVariables, 224	PATTERNNO, 218
ensureFtype, 224	patFireTime, 218
neuron_substitutions_in_synaptic_code, 224	patSetTime, 218
	SYN_OUT_TME, 218
REGEX_OPERATIONAL, 224	T_REPORT_TME, 218
substitute, 225	TOTAL TME, 218
limit	timer, 218
helper.h, 263	MBody1_project/model/map_classol.cc
load_VR_data	_MAP_CLASSOL_CC_, 270
Schmuker2014_classifier, 169	
loadArrayFromTextFile	MBody1_project/model/sizes.h
Schmuker2014_classifier, 169	_FTYPE, 345
loadClassLabels	_NAL, 345
Schmuker2014 classifier, 169	_NLB, 345
log	_NLHI, 345
Schmuker2014 classifier, 170	_NMB, 345
Odimulcizo14_diaddinoi, 170	SCALAR MAX, 345
m_LegacyTypeIndex	SCALAR MIN, 345
NewModels::LegacyWrapper, 103	scalar, 345
MAPNEURON	MBody_delayedSyn.cc, 278
	modelDefinition, 279
neuronModels.cc, 325	
neuronModels.h, 326	myDNDN_ini, 279
MAX_FIRING_RATE_HZ	myDNDN_p, 279
experiment.h, 231	myKCDN_ini, 279
MAX_WEIGHT_PN_AN	myKCDN_p, 279
experiment.h, 231	myLHIKC_ini, 279
MAXNRN	myLHIKC_p, 279
neuronModels.h, 326	myPNKC_ini, 279
MAXPOSTSYN	myPNLHI_ini, 279
postSynapseModels.h, 339	myPOI ini, 279
• • •	<del>-</del> -
MBody1.cc, 274, 276, 277	myPOI_p, 279

	MD - de lie divide - UD annois - t/o de l/ele - ed eine le
postExpDNDN, 279	MBody_individualID_project/model/classol_sim.h
postExpKCDN, 279	DBG_SIZE, 220
postExpLHIKC, 279	InputBaseRate, 221
postExpPNKC, 279	MYRAND, 220
postExpPNLHI, 279	PAT_TIME, 220
stdTM_ini, 279	PATFTIME, 220
stdTM_p, 280	PATTERNNO, 220
MBody_delayedSyn_project/generate_run.cc	patFireTime, 221
main, 245	patSetTime, 221
MBody_delayedSyn_project/model/classol_sim.cc	SYN_OUT_TME, 220
main, 216	T_REPORT_TME, 220
MBody_delayedSyn_project/model/classol_sim.h	TOTAL_TME, 220
DBG_SIZE, 219	timer, 221
InputBaseRate, 219	MBody_individualID_project/model/map_classol.co
MYRAND, 219	_MAP_CLASSOL_CC_, 271
PAT_TIME, 219	MBody_individualID_project/model/sizes.h
PATFTIME, 219	FTYPE, 346
PATTERNNO, 219	NAL, 346
patFireTime, 219	_NLB, 346
patSetTime, 220	_NLHI, 346
SYN OUT TME, 219	NMB, 346
/	<del>-</del> · · ·
T_REPORT_TME, 219	gPNKC_GLOBAL, 346
TOTAL_TME, 219	SCALAR_MAX, 346
timer, 220	SCALAR_MIN, 346
MBody_delayedSyn_project/model/map_classol.cc	scalar, 346
_MAP_CLASSOL_CC_, 270	MBody_map_project/generate_run.cc
MBody_delayedSyn_project/model/sizes.h	main, 246
_FTYPE, 345	MBody_map_project/model/MBody1.cc
_NAL, 345	modelDefinition, 276
_NLB, 345	myDNDN_ini, 276
_NLHI, 345	myDNDN_p, 276
_NMB, 345	myKCDN_ini, 276
SCALAR_MAX, 345	myKCDN_p, 276
SCALAR_MIN, 345	myLHIKC_ini, 276
scalar, 345	myLHIKC_p, 276
MBody individualID.cc, 280	myPNKC ini, 276
modelDefinition, 280	myPNLHI ini, 276
myDNDN_ini, 280	myPOI_ini, 277
myDNDN p, 281	myPOI p, 277
myKCDN ini, 281	postExpDNDN, 277
myKCDN_p, 281	postExpKCDN, 277
myLHIKC_ini, 281	postExpLHIKC, 277
myLHIKC p, 281	postExpPNKC, 277
myPNKC_ini, 281	postExpPNLHI, 277
myPNLHI_ini, 281	stdRMP_ini, 277
myPOI ini, 281	stdRMP_p, 277
• -	
myPOI_p, 281	MBody_map_project/model/classol_sim.cc
postExpDNDN, 281	main, 217
postExpKCDN, 281	MBody_map_project/model/classol_sim.h
postExpLHIKC, 281	InputBaseRate, 221
postExpPNKC, 281	MYRAND, 221
postExpPNLHI, 281	PAT_TIME, 221
stdTM_ini, 281	PATFTIME, 221
stdTM_p, 281	PATTERNNO, 221
MBody_individualID_project/generate_run.cc	patFireTime, 222
main, 245	patSetTime, 222
MBody_individualID_project/model/classol_sim.cc	SYN_OUT_TME, 221
main, 216	T_REPORT_TME, 221

TOTAL_TME, 221	scalar, 347
timer, 222	MBody_userdef.cc, 281
MBody_map_project/model/map_classol.cc	gpKCDN, 283
_MAP_CLASSOL_CC_, 271	gpPNKC, 283
MBody_map_project/model/sizes.h	IMPLEMENT_MODEL, 282
_FTYPE, 346	modelDefinition, 282
_NAL, 346	myDNDN_ini, 282
NLB, 346	myDNDN p, 283
NLHI, 346	myKCDN_ini, 283
_NMB, 346	myKCDN_p, 283
SCALAR MAX, 346	myLHIKC_ini, 283
<del>-</del> · · · ·	· · · · · · · · · · · · · · · · · · ·
SCALAR_MIN, 346	myLHIKC_p, 283
scalar, 346	myPNKC_ini, 283
MBody_map_robot1_project/generate_run.cc	myPNLHI_ini, 283
main, 246	myPOI_ini, 283
MBody_map_robot1_project/model/MBody1.cc	myPOI_p, 283
modelDefinition, 277	postExpDNDN, 283
myDNDN_ini, 278	postExpKCDN, 283
myDNDN_p, 278	postExpLHIKC, 283
myKCDN_ini, 278	postExpPNKC, 283
myKCDN_p, 278	postExpPNLHI, 283
myLHIKC_ini, 277	stdTM_ini, 283
myLHIKC_p, 278	stdTM_p, 283
myPNKC_ini, 278	TIMING, 282
myPNLHI_ini, 278	MBody_userdef_project/generate_run.cc
myPOI_ini, 278	main, 247
myPOI_p, 278	MBody_userdef_project/model/classol_sim.cc
postExpDNDN, 278	main, 217
postExpKCDN, 278	MBody_userdef_project/model/classol_sim.h
postExpLHIKC, 278	InputBaseRate, 223
postExpPNKC, 278	MYRAND, 223
postExpPNLHI, 278	PAT_TIME, 223
stdRMP_ini, 278	PATFTIME, 223
stdRMP_p, 278	PATTERNNO, 223
MBody_map_robot1_project/model/classol_sim.cc	patFireTime, 224
main, 217	patSetTime, 224
MBody_map_robot1_project/model/classol_sim.h	SYN OUT TME, 223
InputBaseRate, 222	T REPORT TME, 223
MYRAND, 222	TOTAL_TME, 223
PAT_TIME, 222	timer, 224
PATTERNIA 222	MBody_userdef_project/model/map_classol.cc
PATTERNNO, 222	_MAP_CLASSOL_CC_, 272
patFireTime, 222	MBody_userdef_project/model/sizes.h
patSetTime, 222	_FTYPE, 347
SYN_OUT_TME, 222	_NAL, 347
T_REPORT_TME, 222	_NLB, 347
TOTAL_TME, 222	_NLHI, 347
timer, 223	_NMB, 347
MBody_map_robot1_project/model/map_classol.cc	SCALAR_MAX, 347
_MAP_CLASSOL_CC_, 271	SCALAR_MIN, 347
MBody_map_robot1_project/model/sizes.h	scalar, 347
_FTYPE, 347	MIN_FIRING_RATE_HZ
_NAL, 347	experiment.h, 231
_NLB, 347	MIN_WEIGHT_PN_AN
_NLHI, 347	experiment.h, 231
_NMB, 347	MODELSPEC_CC
SCALAR_MAX, 347	src/modelSpec.cc, 320
SCALAR_MIN, 347	MYRAND

MBody1_project/model/classol_sim.h, 218 MBody_delayedSyn_project/model/classol_sim.h, 219	classol, 90 neuronpop, 137 Schmuker2014_classifier, 170
MBody_individualID_project/model/classol_sim.h,	model.cc, 283-305
220	Model_Schmuker_2014_classifier.cc, 319
MBody_map_project/model/classol_sim.h, 221	CLUST_SIZE_AN, 319
MBody_map_robot1_project/model/classol_sim.h,	CLUST_SIZE_PN, 319
222	CLUST_SIZE_RN, 319
MBody_userdef_project/model/classol_sim.h, 223	DT, 319
PoissonIzh_sim.h, 337	IMPLEMENT_MODEL, 320
main	modelDefinition, 320
experiment.cc, 229	NETWORK_SCALE, 319
gen_input_structured.cc, 236	NUM CLASSES, 320
gen_kcdn_syns.cc, 236	NUM FEATURES, 320
gen_kcdn_syns_fixto10K.cc, 237	NUM_VR, 320
gen_pnkc_syns.cc, 238	SYNAPSE TAU ANAN, 320
gen_pnkc_syns_indivID.cc, 239	SYNAPSE_TAU_PNAN, 320
gen_pnlhi_syns.cc, 239	SYNAPSE TAU PNPN, 320
gen_syns_sparse.cc, 240	SYNAPSE_TAU_RNPN, 320
gen_syns_sparse_izhModel.cc, 241	
· - · - · -	model_new.cc, 305–318
generateALL.cc, 250	modelDefinition
HHVclampGA_project/generate_run.cc, 242	decode_matrix_globalg_bitmask/model.cc, 283
Izh_sim_sparse.cc, 267	decode_matrix_globalg_bitmask/model_new.cc,
Izh_sparse_project/generate_run.cc, 243	306
MBody1_project/generate_run.cc, 244	decode_matrix_globalg_dense/model.cc, 284
MBody1_project/model/classol_sim.cc, 215	decode_matrix_globalg_dense/model_new.cc, 306
MBody_delayedSyn_project/generate_run.cc, 245	decode_matrix_globalg_sparse/model.cc, 285
MBody_delayedSyn_project/model/classol_sim.cc,	decode_matrix_globalg_sparse/model_new.cc,
216	306
MBody_individualID_project/generate_run.cc, 245	decode_matrix_individualg_dense/model.cc, 285
MBody_individualID_project/model/classol_sim.cc, 216	decode_matrix_individualg_dense/model_new.cc, 307
MBody_map_project/generate_run.cc, 246	decode_matrix_individualg_sparse/model.cc, 286
MBody_map_project/model/classol_sim.cc, 217	decode_matrix_individualg_sparse/model_new.cc,
MBody_map_robot1_project/generate_run.cc, 246	307
MBody_map_robot1_project/model/classol_sim.	extra_global_params_in_sim_code/model.cc, 287
cc, 217	extra_global_params_in_sim_code/model_new.cc,
MBody_userdef_project/generate_run.cc, 247	308
MBody_userdef_project/model/classol_sim.cc, 217	extra_global_params_in_sim_code_event_←
make_input_pats.cc, 269	sparse_inv/model.cc, 287
OneComp_project/generate_run.cc, 248	extra_global_params_in_sim_code_event_
OneComp_sim.cc, 334	sparse_inv/model_new.cc, 308
PoissonIzh_project/generate_run.cc, 249	extra_global_post_param_in_sim_code/model.cc,
PoissonIzh_sim.cc, 337	288
SynDelaySim.cc, 358	extra_global_post_param_in_sim_code/model_
VClampGA.cc, 382	new.cc, 308
make_input_pats.cc, 269	extra_global_pre_param_in_sim_code/model.cc,
main, 269	288
map_classol.cc, 270-272	extra_global_pre_param_in_sim_code/model_←
map_classol.h, 272-274	new.cc, 309
meanInpExc	HHVClamp.cc, 264
lzh_sparse.cc, 268	lzh_sparse.cc, 268
meanInpInh	MBody1_project/model/MBody1.cc, 275
lzh_sparse.cc, 268	MBody_delayedSyn.cc, 279
mexp	MBody_individualID.cc, 280
helper.h, 263	MBody_map_project/model/MBody1.cc, 276
model	MBody_map_robot1_project/model/MBody1.cc,
classIzh, 78	277

MBody_userdef.cc, 282	cc, 303
Model_Schmuker_2014_classifier.cc, 320	synapse_support_code_event_threshold/model_
neuron_support_code_sim/model.cc, 289	new.cc, 317
neuron_support_code_sim/model_new.cc, 309	synapse_support_code_post_learn/model.cc, 303
neuron_support_code_threshold/model.cc, 290	synapse_support_code_post_learn/model_new.cc,
neuron_support_code_threshold/model_new.cc,	318
310	synapse_support_code_sim_code/model.cc, 304
OneComp.cc, 333	synapse_support_code_sim_code/model_new.cc,
PoissonIzh.cc, 336	318
post_vars_in_post_learn/model.cc, 290	synapse_support_code_synapse_dynamics/model.
post_vars_in_post_learn/model_new.cc, 310	cc, 305
post_vars_in_post_learn_sparse/model.cc, 291	synapse_support_code_synapse_dynamics/model
post_vars_in_post_learn_sparse/model_new.cc,	_new.cc, 319
311	modelSpec.cc, 320
post_vars_in_sim_code/model.cc, 292	modelSpec.h, 321
post_vars_in_sim_code/model_new.cc, 311	_MODELSPEC_H_, 322
post_vars_in_sim_code_sparse/model.cc, 293	ALLTOALL, 323
post_vars_in_sim_code_sparse/model_new.cc,	AUTODEVICE, 322
312	CPU, 322
post_vars_in_synapse_dynamics/model.cc, 294	DENSE, 323
post_vars_in_synapse_dynamics/model_new.cc,	EXITSYN, 322
312	FloatType, 322
post_vars_in_synapse_dynamics_sparse/model.	GENN_LONG_DOUBLE, 323
cc, 294	GLOBALG, 323
post_vars_in_synapse_dynamics_sparse/model ←	GPU, 322
_new.cc, 312	GeNNReady, 323
pre_vars_in_post_learn/model.cc, 295	INDIVIDUALID, 323
pre_vars_in_post_learn/model_new.cc, 313	INDIVIDUALG, 323
pre_vars_in_post_learn_sparse/model.cc, 296	INHIBSYN, 322
pre_vars_in_post_learn_sparse/model_new.cc,	initGeNN, 323
313	LEARNING, 322
pre_vars_in_sim_code/model.cc, 297	NO_DELAY, 322
pre_vars_in_sim_code/model_new.cc, 314	NOLEARNING, 322
pre_vars_in_sim_code_event/model.cc, 297	SPARSE, 323
pre_vars_in_sim_code_event/model_new.cc, 314	SynapseConnType, 323
pre_vars_in_sim_code_event_sparse/model.cc,	SynapseGType, 323
298	myDNDN_ini
pre_vars_in_sim_code_event_sparse/model_←	MBody1_project/model/MBody1.cc, 275
new.cc, 315	MBody_delayedSyn.cc, 279
pre_vars_in_sim_code_event_sparse_inv/model.←	MBody_individualID.cc, 280
cc, 299	MBody_map_project/model/MBody1.cc, 276
pre_vars_in_sim_code_event_sparse_inv/model↔	MBody_map_robot1_project/model/MBody1.cc,
_new.cc, 315	278
pre_vars_in_sim_code_sparse/model.cc, 300	MBody_userdef.cc, 282
pre_vars_in_sim_code_sparse/model_new.cc, 316	myDNDN_p
pre_vars_in_synapse_dynamics/model.cc, 300	MBody1_project/model/MBody1.cc, 275
pre_vars_in_synapse_dynamics/model_new.cc,	MBody_delayedSyn.cc, 279
316	MBody_individualID.cc, 281
pre_vars_in_synapse_dynamics_sparse/model.cc,	MBody_map_project/model/MBody1.cc, 276
301	MBody_map_robot1_project/model/MBody1.cc,
pre_vars_in_synapse_dynamics_sparse/model_	278
new.cc, 316	MBody_userdef.cc, 283
SynDelay.cc, 358	myHH_ini
synapse_support_code_event_sim_code/model.	HHVClamp.cc, 264
cc, 302	MyHH, 107
synapse_support_code_event_sim_code/model	getExtraGlobalParams, 107
_new.cc, 317	getInstance, 107
synapse_support_code_event_threshold/model.	getSimCode, 108

·T	MD 1 11 10 070
getThresholdConditionCode, 108	MBody_delayedSyn.cc, 279
getVars, 108	MBody_individualID.cc, 281
ParamValues, 107	MBody_map_project/model/MBody1.cc, 276
VarValues, 107	MBody_map_robot1_project/model/MBody1.cc,
Mylzhikevich, 108	278
getInstance, 109	MBody_userdef.cc, 283
getParamNames, 109	myPOI_ini
-	MBody1_project/model/MBody1.cc, 275
getSimCode, 109	
ParamValues, 109	MBody_delayedSyn.cc, 279
VarValues, 109	MBody_individualID.cc, 281
MylzhikevichVariable, 110	MBody_map_project/model/MBody1.cc, 277
getInstance, 110	MBody_map_robot1_project/model/MBody1.cc,
getSimCode, 110	278
getVars, 110	MBody_userdef.cc, 283
ParamValues, 110	PoissonIzh.cc, 336
	myPOI_p
VarValues, 110	MBody1 project/model/MBody1.cc, 275
myKCDN_ini	• - •
MBody1_project/model/MBody1.cc, 275	MBody_delayedSyn.cc, 279
MBody_delayedSyn.cc, 279	MBody_individualID.cc, 281
MBody_individualID.cc, 281	MBody_map_project/model/MBody1.cc, 277
MBody_map_project/model/MBody1.cc, 276	MBody_map_robot1_project/model/MBody1.cc,
MBody_map_robot1_project/model/MBody1.cc,	278
278	MBody_userdef.cc, 283
MBody userdef.cc, 283	PoissonIzh.cc, 336
· · · · · · · · · · · · · · · · · · ·	mySyn_ini
myKCDN_p	PoissonIzh.cc, 336
MBody1_project/model/MBody1.cc, 275	1 01330111211.00, 000
MBody_delayedSyn.cc, 279	N
MBody_individualID.cc, 281	inputSpec, 97
MBody_map_project/model/MBody1.cc, 276	QTIsaac, 159
MBody_map_robot1_project/model/MBody1.cc,	
278	n
MBody_userdef.cc, 283	randomGauss, 161
•	randomGen, 162
myLHIKC_ini	stdRG, 185
MBody1_project/model/MBody1.cc, 275	N_FOLDING
MBody_delayedSyn.cc, 279	experiment.h, 231
MBody_individualID.cc, 281	NETWORK_SCALE
MBody_map_project/model/MBody1.cc, 276	Model_Schmuker_2014_classifier.cc, 319
MBody_map_robot1_project/model/MBody1.cc,	NEURONMODELS CC
277	<del>-</del>
MBody_userdef.cc, 283	neuronModels.cc, 324
myLHIKC p	NGRADSYNAPSE
•	synapseModels.cc, 356
MBody1_project/model/MBody1.cc, 275	synapseModels.h, 357
MBody_delayedSyn.cc, 279	nModels
MBody_individualID.cc, 281	neuronModels.cc, 325
MBody_map_project/model/MBody1.cc, 276	neuronModels.h, 327
MBody_map_robot1_project/model/MBody1.cc,	NNmodel, 137
278	$\sim$ NNmodel, 140
MBody_userdef.cc, 283	activateDirectInput, 140
myPNKC_ini	addNeuronPopulation, 140
MBody1_project/model/MBody1.cc, 275	•
	addSynapsePopulation, 141–143
MBody_delayedSyn.cc, 279	finalize, 144
MBody_individualID.cc, 281	findNeuronGroup, 144
MBody_map_project/model/MBody1.cc, 276	findSynapseGroup, 144
MBody_map_robot1_project/model/MBody1.cc,	getDT, 144
278	getName, 144
MBody_userdef.cc, 283	getNeuronGridSize, 144
myPNLHI_ini	getNeuronGroups, 144
MBody1_project/model/MBody1.cc, 275	getNeuronKernelParameters, 144
mbody i_project/industribudy 1.66, 270	gonvouronnomen arameters, 144

getNumNeurons, 144	Namelter
getPrecision, 144	NamelterCtx, 111
getRNType, 145	NamelterCtx
getResetKernel, 144	container, 111
getSeed, 145	nameBegin, 111
getSimLearnPostKernelParameters, 145	nameEnd, 111
getSynapseDynamicsGridSize, 145	Namelter, 111
getSynapseDynamicsGroups, 145	NamelterCtx, 111
getSynapseDynamicsKernelParameters, 145	NamelterCtx Container >, 111
getSynapseGroups, 145	
getSynapseKernelGridSize, 145	needPostSt
getSynapseKernelParameters, 145	weightUpdateModel, 198
getSynapsePostLearnGridSize, 145	needPreSt
getSynapsePostLearnGroups, 145	weightUpdateModel, 198
isFinalized, 145	Neuron, 111
isSynapseGroupDynamicsRequired, 146	getExtraGlobalParams, 120
isSynapseGroupPostLearningRequired, 146	getInstance, 120, 121
	getParamNames, 121
isTimingEnabled, 146	getSimCode, 121-125
NNmodel, 140	getSupportCode, 125
scalarExpr, 146	getThresholdConditionCode, 126-128
setConstInp, 146	getVars, 128-131
setDT, 146	ParamValues, 118
setGPUDevice, 146	VarValues, 119
setMaxConn, 146	neuron ini
setName, 146	decode_matrix_globalg_bitmask/model.cc, 284
setNeuronClusterIndex, 146	decode_matrix_globalg_dense/model.cc, 284
setPopulationSums, 146	decode_matrix_globalg_sparse/model.cc, 285
setPrecision, 147	decode_matrix_individualg_dense/model.cc, 285
setRNType, 147	decode_matrix_individualg_sparse/model.cc, 286
setSeed, 147	extra_global_params_in_sim_code/model.cc, 287
setSpanTypeToPre, 147	extra_global_params_in_sim_code_event_ ↔
setSynapseClusterIndex, 147	sparse_inv/model.cc, 287
setSynapseG, 147	extra_global_post_param_in_sim_code/model.cc,
setTiming, 147	288
zeroCopyInUse, 148	extra global pre param in sim code/model.cc,
NO_DELAY	288
modelSpec.h, 322	neuron_support_code_sim/model.cc, 289
NOLEARNING	neuron_support_code_threshold/model.cc, 290
modelSpec.h, 322	post vars in post learn/model.cc, 290
NPOP	post_vars_in_post_learn_sparse/model.cc, 291
HHVClampParameters.h, 265	post_vars_in_sim_code/model.cc, 292
NSYNAPSE	post_vars_in_sim_code sparse/model.cc, 293
synapseModels.cc, 356	
synapseModels.h, 357	post_vars_in_synapse_dynamics/model.cc, 294
NUM_CLASSES	post_vars_in_synapse_dynamics_sparse/model.
Model_Schmuker_2014_classifier.cc, 320	cc, 294
NUM_FEATURES	pre_vars_in_post_learn/model.cc, 295
Model_Schmuker_2014_classifier.cc, 320	pre_vars_in_post_learn_sparse/model.cc, 296
NUM_VR	pre_vars_in_sim_code/model.cc, 297
Model_Schmuker_2014_classifier.cc, 320	pre_vars_in_sim_code_event/model.cc, 297
name	pre_vars_in_sim_code_event_sparse/model.cc,
Parameter, 149	298
name_substitutions	pre_vars_in_sim_code_event_sparse_inv/model.
codeGenUtils.h, 226	cc, 299
nameBegin	pre_vars_in_sim_code_sparse/model.cc, 300
NamelterCtx, 111	pre_vars_in_synapse_dynamics/model.cc, 300
nameEnd	pre_vars_in_synapse_dynamics_sparse/model.cc,
NamelterCtx, 111	301

synapse_support_code_event_sim_code/model. ←	getDerivedParams, 132
cc, 302	getIDRange, 132
synapse_support_code_event_threshold/model. ← cc, 303	getInSyn, 132 getInitVals, 132
synapse_support_code_post_learn/model.cc, 303	getName, 132
synapse_support_code_sim_code/model.cc, 304	getNeuronModel, 132
synapse_support_code_synapse_dynamics/model.cc	•
cc, 305	getNumNeurons, 132
neuron_p	getOutSyn, 132
post_vars_in_post_learn/model.cc, 290	getPaddedIDRange, 132
post_vars_in_post_learn_sparse/model.cc, 291	getParams, 132
pre_vars_in_post_learn/model.cc, 295	getQueueOffset, 132
pre_vars_in_post_learn_sparse/model.cc, 296	getSpikeEventCondition, 132
synapse_support_code_post_learn/model.cc, 304	initDerivedParams, 132
neuron_p2	isDelayRequired, 133
post_vars_in_post_learn/model.cc, 291	isParamRequiredBySpikeEventCondition, 133
post_vars_in_post_learn_sparse/model.cc, 292	isQueueRequired, 133
pre_vars_in_post_learn/model.cc, 295	isSpikeEventRequired, 133
	·
pre_vars_in_post_learn_sparse/model.cc, 296	isSpikeEventZeroCopyEnabled, 133
synapse_support_code_post_learn/model.cc, 304	isSpikeTimeRequired, 133
neuron_substitutions_in_synaptic_code lib/src/codeGenUtils.cc, 224	isSpikeTimeZeroCopyEnabled, 133
	isSpikeZeroCopyEnabled, 133
neuron_support_code_sim/model.cc	isTrueSpikeRequired, 133
modelDefinition, 289	isVarQueueRequired, 133
neuron_ini, 289	isVarZeroCopyEnabled, 133
synapses_ini, 289	isZeroCopyEnabled, 133
neuron_support_code_sim/model_new.cc	NeuronGroup, 132
IMPLEMENT_MODEL, 309	setClusterIndex, 133
modelDefinition, 309	setSpikeEventRequired, 133
neuron_support_code_sim/test.cc	setSpikeEventZeroCopyEnabled, 133
SimTest, 365	setSpikeTimeRequired, 133
simulatorBackends, 365	setSpikeTimeZeroCopyEnabled, 133
TEST_P, 365	setSpikeZeroCopyEnabled, 133
WRAPPED_INSTANTIATE_TEST_CASE_P, 365	setTrueSpikeRequired, 133
neuron_support_code_threshold/model.cc	setVarZeroCopyEnabled, 133
modelDefinition, 290	updateVarQueues, 133
neuron_ini, 290	neuronGroup.cc, 323
synapses_ini, 290	neuronGroup.h, 323
neuron_support_code_threshold/model_new.cc	neuronLocalVarInit
IMPLEMENT_MODEL, 310	StandardGeneratedSections, 66
modelDefinition, 310	neuronLocalVarWrite
neuron_support_code_threshold/test.cc	StandardGeneratedSections, 66
SimTest, 365	neuronModel, 134
simulatorBackends, 366	~neuronModel, 135
TEST_P, 365	dpNames, 135
WRAPPED_INSTANTIATE_TEST_CASE_P, 365	dps, 135
neuronBlkSz	extraGlobalNeuronKernelParameterTypes, 135
global.cc, 259	extraGlobalNeuronKernelParameters, 135
global.h, 261	neuronModel, 135
neuronBlockSize	pNames, 135
GENN_PREFERENCES, 64	resetCode, 135
NeuronGroup, 131	simCode, 135
addExtraGlobalParams, 132	supportCode, 135
addlnSyn, 132	thresholdConditionCode, 135
addOutSyn, 132	tmpVarNames, 136
addSpkEventCondition, 132	tmpVarTypes, 136
calcSizes, 132	varNames, 136
checkNumDelaySlots, 132	varTypes, 136

NeuronModels, 65	getExtraGlobalParams, 155
neuronModels.cc, 324	getInstance, 155
IZHIKEVICH_V, 325	getParamNames, 156
IZHIKEVICH, 325	getSimCode, 156
MAPNEURON, 325	•
	getThresholdConditionCode, 156
NEURONMODELS_CC, 324	getVars, 156
nModels, 325	isPoisson, 156
POISSONNEURON, 325	ParamValues, 155
prepareStandardModels, 324	VarValues, 155
SPIKESOURCE, 325	NeuronModels::RulkovMap, 163
TRAUBMILES_ALTERNATIVE, 325	getDerivedParams, 165
TRAUBMILES_FAST, 325	getInstance, 165
TRAUBMILES_PSTEP, 325	getParamNames, 165
TRAUBMILES_SAFE, 325	getSimCode, 165
TRAUBMILES, 325	getThresholdConditionCode, 165
neuronModels.h, 325	getVars, 165
IZHIKEVICH_V, 326	ParamValues, 165
IZHIKEVICH, 326	VarValues, 165
MAPNEURON, 326	NeuronModels::SpikeSource, 178
MAXNRN, 326	getInstance, 179
nModels, 327	getThresholdConditionCode, 179
POISSONNEURON, 327	ParamValues, 179
prepareStandardModels, 326	VarValues, 179
SPIKESOURCE, 327	NeuronModels::TraubMiles, 189
TRAUBMILES_ALTERNATIVE, 327	getInstance, 191
TRAUBMILES_FAST, 327	getParamNames, 191
TRAUBMILES_PSTEP, 327	getSimCode, 191
TRAUBMILES_SAFE, 327	getThresholdConditionCode, 191
TRAUBMILES, 327	getVars, 191
NeuronModels::Base, 70	ParamValues, 191
getExtraGlobalParams, 71	VarValues, 191
getResetCode, 72	NeuronModels::TraubMilesAlt, 191
getSimCode, 72	getInstance, 192
getSupportCode, 72	getSimCode, 192
getThresholdConditionCode, 72	ParamValues, 192
NeuronModels::Izhikevich, 98	VarValues, 192
getInstance, 99	NeuronModels::TraubMilesFast, 193
getParamNames, 99	getInstance, 193
getSimCode, 99	getSimCode, 193
getThresholdConditionCode, 99	ParamValues, 193
getVars, 99	VarValues, 193
	NeuronModels::TraubMilesNStep, 194
ParamValues, 99 VarValues, 99	• •
	getInstance, 194
NeuronModels::IzhikevichVariable, 100	getParamNames, 195
getInstance, 101	getSimCode, 195
getParamNames, 101	ParamValues, 194
getVars, 101	VarValues, 194
ParamValues, 101	neuronOutputInit
VarValues, 101	StandardGeneratedSections, 66
NeuronModels::LegacyWrapper, 105	neuronPSize
getExtraGlobalParams, 106	Izh_sparse.cc, 268
getResetCode, 106	neuronReset
getSimCode, 106	StandardSubstitutions, 67
getSupportCode, 106	neuronSim
getThresholdConditionCode, 106	StandardSubstitutions, 67
isPoisson, 106	neuronSpikeEventCondition
LegacyWrapper, 106	StandardSubstitutions, 67
NeuronModels::Poisson, 154	neuronSpikeEventTest

04	IMPLEMENT MODEL 000
StandardGeneratedSections, 66	IMPLEMENT_MODEL, 330
neuronThresholdCondition	newPostsynapticModels.h, 330
StandardSubstitutions, 67	SET_CURRENT_CONVERTER_CODE, 331
neuronVSize	SET_DECAY_CODE, 331
lzh_sparse.cc, 268	SET_SUPPORT_CODE, 331
neuronpop, 136	newWeightUpdateModels.cc, 331
$\sim$ neuronpop, 136	IMPLEMENT_MODEL, 331
getSpikeNumbersFromGPU, 136	newWeightUpdateModels.h, 331
getSpikesFromGPU, 137	SET_EVENT_CODE, 332
init, 137	SET_EVENT_THRESHOLD_CONDITION_CODE,
model, 137	332
neuronpop, 136	SET_EXTRA_GLOBAL_PARAMS, 332
output_spikes, 137	SET_LEARN_POST_CODE, 332
output_state, 137	SET_LEARN_POST_SUPPORT_CODE, 333
run, 137	SET_NEEDS_POST_SPIKE_TIME, 333
sum_spikes, 137	SET_NEEDS_PRE_SPIKE_TIME, 333
sumIzh1, 137	SET_SIM_CODE, 333
NewModels, 66	SET_SIM_SUPPORT_CODE, 333
newModels.h, 327	SET_SYNAPSE_DYNAMICS_CODE, 333
DECLARE_MODEL, 328	SET SYNAPSE DYNAMICS SUPPORT CODE,
IMPLEMENT MODEL, 328	333
SET_DERIVED_PARAMS, 328	nexp
SET PARAM NAMES, 328	helper.h, 263
	nlong
SET_VARS, 328	stdRG, 185
NewModels::Base, 69	,
DerivedParamFunc, 70	OUTPUT_DIR
DerivedParamVec, 70	experiment.h, 231
getDerivedParams, 70	OB
getParamNames, 70	CodeHelper.h, 227
getVars, 70	offset
StringPairVec, 70	classol, 90
StringVec, 70	OneComp.cc, 333
NewModels::LegacyWrapper	exlzh_ini, 333
getDerivedParams, 103	exlzh_p, 333
getParamNames, 103	IMPLEMENT_MODEL, 333
getVars, 103	modelDefinition, 333
LegacyWrapper, 103	OneComp_model.cc, 334
m_LegacyTypeIndex, 103	_ONECOMP_MODEL_CC_, 334
zipStringVectors, 103	OneComp_model.h, 334
NewModels::LegacyWrapper< ModelBase, Legacy↔	OneComp_project/generate_run.cc
ModelType, ModelArray >, 102	main, 248
NewModels::ValueBase	OneComp_project/model/sizes.h
getValues, 195	_FTYPE, 348
operator[], 195	_NC1, 348
ValueBase, 195	SCALAR_MAX, 348
NewModels::ValueBase< 0 >, 196	SCALAR_MIN, 348
getValues, 196	scalar, 348
ValueBase, 196	OneComp_sim.cc, 334
NewModels::ValueBase< NumValues >, 195	main, 334
newNeuronModels.cc, 328	OneComp_sim.h, 334
IMPLEMENT_MODEL, 328, 329	DBG_SIZE, 335
newNeuronModels.h, 329	T_REPORT_TME, 335
SET_EXTRA_GLOBAL_PARAMS, 330	TOTAL_TME, 335
SET_RESET_CODE, 330	timer, 335
SET_SIM_CODE, 330	openBrace
SET_SUPPORT_CODE, 330	CodeHelper, 91
SET_THRESHOLD_CONDITION_CODE, 330	openFileGetMax
newPostsynapticModels.cc, 330	<pre>lzh_sparse_project/generate_run.cc, 243</pre>

openRecordingFile Schmuker2014_classifier, 169	MBody_map_robot1_project/model/classol_sim.h,
operator<<	MBody_userdef_project/model/classol_sim.h, 223
helper.h, 263	PATTERNNO
operator*	MBody1_project/model/classol_sim.h, 218
·	MBody_delayedSyn_project/model/classol_sim.h,
PairKeyConstlter, 148	219
operator->	
PairKeyConstIter, 148	MBody_individualID_project/model/classol_sim.h, 220
operator&	MBody_map_project/model/classol_sim.h, 221
synapseMatrixType.h, 355	MBody map robot1 project/model/classol sim.h,
operator[]	222
NewModels::ValueBase, 195	
optimiseBlockSize	MBody_userdef_project/model/classol_sim.h, 223 PLASTICITY_INTERVAL_MS
GENN_PREFERENCES, 64	
optimizeCode	experiment.h, 231
GENN_PREFERENCES, 64	pNames
option	neuronModel, 135
parse_options.h, 335	postSynModel, 157
output_params	weightUpdateModel, 198
classIzh, 78	POISSONNEURON
output_spikes	neuronModels.cc, 325
classIzh, 78	neuronModels.h, 327
classol, 85	POSTSYNAPSEMODELS_CC
neuronpop, 137	postSynapseModels.cc, 338
output_state	POSTSYNAPTIC
classIzh, 78	SynapseGroup, 186
classol, 85, 86	PRESYNAPTIC
neuronpop, 137	SynapseGroup, 186
outputDN_spikes	PairKeyConstIter
classol, 86	operator*, 148
outputDir	operator->, 148
Schmuker2014_classifier, 170	PairKeyConstIter, 148
outputRunParameters	PairKeyConstIter< BaseIter >, 148
experiment.cc, 229	param_CONNECTIVITY_AN_AN
outputSpikes	Schmuker2014_classifier, 170
Schmuker2014_classifier, 169	param_CONNECTIVITY_PN_AN
overallWinnerSpikeCountAN	Schmuker2014_classifier, 170
Schmuker2014_classifier, 170	param_CONNECTIVITY_PN_PN
	Schmuker2014_classifier, 170
p_pattern	param_CONNECTIVITY_RN_PN
classol, 90	Schmuker2014_classifier, 170
PAT_TIME	param_GLOBAL_WEIGHT_SCALING
MBody1_project/model/classol_sim.h, 218	Schmuker2014_classifier, 170
MBody_delayedSyn_project/model/classol_sim.h,	param_MAX_FIRING_RATE_HZ
219	Schmuker2014_classifier, 170
MBody_individualID_project/model/classol_sim.h,	param_MAX_WEIGHT_PN_AN
220	Schmuker2014_classifier, 170
MBody_map_project/model/classol_sim.h, 221	param_MIN_FIRING_RATE_HZ
MBody_map_robot1_project/model/classol_sim.h,	Schmuker2014_classifier, 170
222	param_MIN_WEIGHT_PN_AN
MBody_userdef_project/model/classol_sim.h, 223	Schmuker2014_classifier, 170
PATFTIME	param_PLASTICITY_INTERVAL_MS
MBody1_project/model/classol_sim.h, 218	Schmuker2014_classifier, 170
MBody_delayedSyn_project/model/classol_sim.h,	param_SPIKING_ACTIVITY_THRESHOLD_HZ
219	Schmuker2014_classifier, 170
MBody_individualID_project/model/classol_sim.h,	param_WEIGHT_DELTA_PN_AN
220	Schmuker2014_classifier, 170
MBody_map_project/model/classol_sim.h, 221	param_WEIGHT_RN_PN

Schmuker2014_classifier, 171	MBody_map_project/model/classol_sim.h, 222
param_WEIGHT_WTA_AN_AN	MBody_map_robot1_project/model/classol_sim.h
Schmuker2014_classifier, 171	222
param_WEIGHT_WTA_PN_PN	MBody_userdef_project/model/classol_sim.h, 224
Schmuker2014_classifier, 171	pattern
ParamValues	classol, 90
ExpCondUserDef, 96	PiecewiseSTDPUserDef, 152
MyHH, 107	getDerivedParams, 153
Mylzhikevich, 109	getInstance, 153
MylzhikevichVariable, 110	getLearnPostCode, 153
Neuron, 118	getParamNames, 153
NeuronModels::Izhikevich, 99	getSimCode, 153
NeuronModels::IzhikevichVariable, 101	getVars, 153
NeuronModels::Poisson, 155	isPostSpikeTimeRequired, 153
NeuronModels::RulkovMap, 165	isPreSpikeTimeRequired, 154
NeuronModels::SpikeSource, 179	ParamValues, 153
NeuronModels::TraubMiles, 191	VarValues, 153
NeuronModels::TraubMilesAlt, 192	plasticWeights
NeuronModels::TraubMilesFast, 193	Schmuker2014_classifier, 171
NeuronModels::TraubMilesNStep, 194	PoissonIzh-model.cc, 335
• •	_POISSONIZHMODEL_CC_, 336
PiecewiseSTDPUserDef, 153	
PostsynapticModels::DeltaCurr, 93	PoissonIzh-model.h, 336
PostsynapticModels::ExpCond, 95	PoissonIzh.cc, 336
StaticGradedUserDef, 181	exlzh_ini, 336
StaticPulseUserDef, 184	exlzh_p, 336
WeightUpdateModel, 204, 205	modelDefinition, 336
WeightUpdateModelSpikeEvent, 214	myPOI_ini, 336
WeightUpdateModels::PiecewiseSTDP, 151	myPOI_p, 336
WeightUpdateModels::StaticGraded, 180	mySyn_ini, 336
WeightUpdateModels::StaticPulse, 183	PoissonIzh_project/generate_run.cc
Parameter, 148	main, 249
experiment.cc, 229	PoissonIzh_project/model/sizes.h
name, 149	_FTYPE, 348
value, 149	_Nlzh, 348
parse_options.h, 335	_NPoisson, 348
cpu_only, 335	SCALAR_MAX, 348
dbgMode, 335	SCALAR_MIN, 348
fixsynapse, 335	scalar, 348
for, 335	PoissonIzh_sim.cc, 337
ftype, 335	main, 337
if, 335	PoissonIzh_sim.h, 337
option, 335	InputBaseRate, 337
patFireTime	MYRAND, 337
MBody1_project/model/classol_sim.h, 218	SYN_OUT_TME, 337
MBody delayedSyn project/model/classol sim.h,	T_REPORT_TME, 337
219	TOTAL_TME, 337
MBody_individualID_project/model/classol_sim.h,	timer, 337
221	populateDeviceMemory
	• •
MBody_map_project/model/classol_sim.h, 222	Schmuker2014_classifier, 169
MBody_map_robot1_project/model/classol_sim.h,	post_vars_in_post_learn/model.cc
222	modelDefinition, 290
MBody_userdef_project/model/classol_sim.h, 224	neuron_ini, 290
patSetTime	neuron_p, 290
MBody1_project/model/classol_sim.h, 218	neuron_p2, 291
MBody_delayedSyn_project/model/classol_sim.h,	synapses_ini, 291
220	post_vars_in_post_learn/model_new.cc
MBody_individualID_project/model/classol_sim.h,	IMPLEMENT_MODEL, 310
221	modelDefinition, 310

most very in most leave/test so	avenues ini 004
post_vars_in_post_learn/test.cc	synapses_ini, 294
SimTest, 366	post_vars_in_synapse_dynamics_sparse/model_new.
simulatorBackends, 366	CC
TEST_P, 366	IMPLEMENT_MODEL, 312
WRAPPED_INSTANTIATE_TEST_CASE_P, 366	modelDefinition, 312
post_vars_in_post_learn_sparse/model.cc	post_vars_in_synapse_dynamics_sparse/test.cc
modelDefinition, 291	SimTest, 370
neuron_ini, 291	simulatorBackends, 370
neuron_p, 291	TEST_P, 370
neuron_p2, 292	WRAPPED_INSTANTIATE_TEST_CASE_P, 370
synapses ini, 292	postExpDNDN
post_vars_in_post_learn_sparse/model_new.cc	MBody1_project/model/MBody1.cc, 275
IMPLEMENT_MODEL, 311	MBody_delayedSyn.cc, 279
modelDefinition, 311	MBody_individualID.cc, 281
post_vars_in_post_learn_sparse/test.cc	MBody_map_project/model/MBody1.cc, 277
SimTest, 367	MBody_map_robot1_project/model/MBody1.cc,
simulatorBackends, 367	278
TEST_P, 367	MBody_userdef.cc, 283
WRAPPED_INSTANTIATE_TEST_CASE_P, 367	postExpKCDN
post_vars_in_sim_code/model.cc	MBody1_project/model/MBody1.cc, 275
modelDefinition, 292	MBody_delayedSyn.cc, 279
neuron_ini, 292	MBody_individualID.cc, 281
synapses_ini, 292	MBody_map_project/model/MBody1.cc, 277
post_vars_in_sim_code/model_new.cc	MBody_map_robot1_project/model/MBody1.cc,
IMPLEMENT_MODEL, 311	278
modelDefinition, 311	MBody_userdef.cc, 283
post_vars_in_sim_code/test.cc	postExpLHIKC
SimTest, 368	MBody1_project/model/MBody1.cc, 275
simulatorBackends, 368	MBody_delayedSyn.cc, 279
TEST_P, 368	MBody_individualID.cc, 281
WRAPPED_INSTANTIATE_TEST_CASE_P, 368	MBody_map_project/model/MBody1.cc, 277
post_vars_in_sim_code_sparse/model.cc	MBody_map_robot1_project/model/MBody1.cc,
	278
modelDefinition, 293	
neuron_ini, 293	MBody_userdef.cc, 283
synapses_ini, 293	postExpPNKC
post_vars_in_sim_code_sparse/model_new.cc	MBody1_project/model/MBody1.cc, 275
IMPLEMENT_MODEL, 312	MBody_delayedSyn.cc, 279
modelDefinition, 312	MBody_individualID.cc, 281
post_vars_in_sim_code_sparse/test.cc	MBody_map_project/model/MBody1.cc, 277
SimTest, 368	MBody_map_robot1_project/model/MBody1.cc,
simulatorBackends, 368	278
TEST_P, 368	MBody_userdef.cc, 283
WRAPPED_INSTANTIATE_TEST_CASE_P, 368	postExpPNLHI
post_vars_in_synapse_dynamics/model.cc	MBody1_project/model/MBody1.cc, 275
modelDefinition, 294	MBody_delayedSyn.cc, 279
neuron_ini, 294	MBody_individualID.cc, 281
synapses_ini, 294	MBody_map_project/model/MBody1.cc, 277
post_vars_in_synapse_dynamics/model_new.cc	MBody_map_robot1_project/model/MBody1.cc,
IMPLEMENT_MODEL, 312	278
modelDefinition, 312	MBody_userdef.cc, 283
	•
post_vars_in_synapse_dynamics/test.cc	postSynDecay
SimTest, 369	extra_postsynapses.h, 233
simulatorBackends, 369	postSynModel, 157
TEST_P, 369	postSynModel, 156
WRAPPED_INSTANTIATE_TEST_CASE_P, 369	~postSynModel, 157
post_vars_in_synapse_dynamics_sparse/model.cc	dpNames, 157
modelDefinition, 294	dps, 157
neuron_ini, 294	pNames, 157

postSynapseModels.c., 338 postSynapseModels.h, 339 postSynapseDeurentConverter StandardSubstitutions, 67 postSynapseDecay StandardSubstitutions, 68 postSynapseModels.c., 338 IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynapseModels, 338 preparePostSynModels, 338 postSynapseModels, 338 postSynapseModels, 338 postSynapseModels, 339 postSynthodels, 339 IZHIKEVICH_PS, 338 IZHIKEVICH_PS, 33	postSynDecay, 157	pre_vars_in_post_learn/model_new.cc
supportCode, 157 varNames, 158 varTypes, 158 postSymApseModels.cc, 338 postSynapseModels.c, 339 postSynapseModels.h, 339 postSynapseModels.h, 339 postSynapseModels.h, 339 postSynapseModels.c, 338 postSynapseModels.c, 338 postSynapseModels.c, 338 postSynapseModels.co, 338 postSynapseModels.co, 338 postSynapseModels.co, 338 postSynapseModels.de postSynapseModels.co, 338 postSynapseModels.de postSynapseSynapseSynapses.de postSynapseSynapseSynapses.de postSynapseModels.de postSynapseModels.de postSynapseModels.de postSynapseModels.de postSynapseModels.de postSynapseSynapseSynapses postSynapseSynapseSynapses postSynapseSynapses postSynapseSynapseSynapses postSynapseSynapseSynapses postSynapseSynapseSynapses postSynapseSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapses postSynapseSynapseSynapses postSy	postSynModel, 157	IMPLEMENT_MODEL, 313
varNames, 158 varTypes, 158 postSynModels postSynapseModels.cc, 338 postSynapseCurrentConverter StandardSubstitutions, 67 postSynapseDecay StandardSubstitutions, 68 postSynapseModels, 338 EXPDECAY, 338 IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynModels, 338 postSynModels, 338 postSynModels, 338 postSynModels, 338 postSynModels, 338 postSynModels, 339 postSynModels, 339 postSymModels, 339 postSymmodel, 157 PostsynapitoModels, 266 PostsymapitoModels, 266 PostsymapitoModels, 266 PostsymapitoModels, 266 PostsymapitoModels, 268 PostsymapitoModels, 269 getCurrentConverterCode, 33 getInstance, 39 getParamNames, 95 paramValues, 95 ParamValues, 95 ParamValues, 95 ParamValues, 95 ParamWalues, 96 ParamWalues, 96 ParamWalues, 96 ParamWalues, 96 ParamWalues, 97 ParamWalues, 98 ParamW	postSyntoCurrent, 157	modelDefinition, 313
varTypes, 158 postSynnapseModels.cc, 338 postSynapseModels.h, 339 postSynapseModels.h, 339 postSynapseModels.h, 339 postSynapseModels.c, 338 postSynapseModels.cd, 338 postSynapseModels.dd, 339 postSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapseSynapse	supportCode, 157	pre_vars_in_post_learn/test.cc
postSynapseModels.cc, 338     postSynapseModels.ch, 339     postSynapseCurrentConverter     StandardSubstitutions, 68     postSynapseModels.co, 338     EXPDECAY, 338     IZHIKEVICH_PS, 338     PoSTSYNAPSEMODELS_CC, 338     postSynModels, 338     postSynModels, 338     postSynModels, 338     postSynModels, 339     postSynapticModels, 339     postSynapticModels	varNames, 158	SimTest, 370
postSynapseModels.c, 338 postSynapseModels.h, 339 postSynapseModels.h, 339 postSynapseModels.c, 338 postSynapseModels.c, 338 EXPDECAY, 338 IZHIKEVICH_PS, 338 postSynapseModels.c, 338 preparePostSynModels, 338 preparePostSynModels, 338 postSynapseModels.h, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 postSymModels, 340 postSymModels, 366 PostsynapitoModels, 370 postSymModels, 370 postSymportCode, 73 postSymportCode, 73 postSymportCode, 73 getDecayCode, 73 getDecayCode, 73 postSymapitoModels: ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 75 paramValues, 95 ParamValues, 95 ParamValues, 95 PostsynapitoModels: ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 70 getDecayCode, 70 getCurrentConverterCode, 101 getDecayCode, 102 getSupportCode, 102 getSuppo	varTypes, 158	simulatorBackends, 370
postSynapseModels.h. 339 postSynapseDecay StandardSubstitutions, 67 postSynapseDecay StandardSubstitutions, 68 postSynapseModels.cc, 338 EXPDECAY, 338 EXPDECAY, 338 POSTSYNAPSEMODELS CC, 338 postSynapseModels, 338 preparePostSynModels, 338 preparePostSynModels, 338 preparePostSynModels, 339 postSynapseModels, 339 postSynModels, 36 PostsynapticModels, 66 PostsynapticModels: DeltaCurr, 92 getCurrentConverterCode, 73 getDecayCode, 75 getDecayCode, 95 getDerwedParams, 95 getDerwedParams, 95 getDerwedParams, 95 getDecayCode, 102 getSupportCode, 102 get	postSynModels	TEST_P, 370
postSynapseCurrentConverter StandardSubstitutions, 67 postSynapseDecay StandardSubstitutions, 68 postSynapseModels. C, 338 EXPDECAY, 338 IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynapseModels. S, 338 preparePostSynModels, 338 preparePostSynModels, 338 postSynapseModels. S, 339 postSynapseModels. S, 339 postSynapseModels. S, 339 postSynModels, 366 PostsynapticModels, 366 PostsynapticModels, 367 PostsynapticModels, 368 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getIbecayCode, 73 getIbecayCode, 73 getIbecayCode, 95 getIDecayCode, 95 getIDecayCode, 95 getIDecayCode, 95 getIDecayCode, 95 getIDecayCode, 102 LegacyWrapper, 101 getCurrentConverterCode, 101 getEdeayCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn_sparse/model_new.cc IMPLEMENT_MODEL, 313 modelDefinition, 315 pre_vars_in_post_learn_sparse/model_new.cc IMPLEMENT_MODEL, 313 modelDefinition, 314 pre_vars_in_sim_code/model.cc modelDefinition, 317 pre_vars_in_sim_code/model.cc modelDefinition, 297 revars_in_sim_code/model.cc modelDefinition, 317 pre_vars_in_sim_code/model.cc modelDefinition, 297 revars_in_sim_code/event/model.cc modelDefinition, 314 pre_vars_in_sim_code/event/model.cc modelDefinition, 297 revars_in_sim_code/event/model.cc modelDefinition, 317 pre_vars_in_sim_code/event/model.cc modelDefinition, 319 preparePostSynModels. pre_vars_in_sim_code/event/model.cc modelDefinition, 317 pre_vars_in_sim_code/event/model.cc modelDefinition, 319 pre_vars_in_sim_code/event/model.cc modelDefinition, 299 preparePostSynModels. pre_vars_in_sim_code_event/model_cc modelDefinition, 319 pre_vars_in_sim_code_even	postSynapseModels.cc, 338	WRAPPED_INSTANTIATE_TEST_CASE_P, 370
StandardSubstitutions, 67 postSynapseDecay StandardSubstitutions, 68 postSynapseModels.co, 338 EXPDECAY, 338 EXPDECAY, 338 EXPDECAY, 338 PostSynAPSEMODELS_CC, 338 postSynModels, 338 preparePostSynModels, 338 preparePostSynModels, 338 postSynModels, 339 postSynModels, 66 PostsynapseModels.66 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getBoeayCode, 73 getBoeayCode, 73 postSynapicModels: ExpCond, 94 getCurrentConverterCode, 93 getInstance, 93 ParamValues, 93 ParamValues, 93 ParamValues, 93 ParamValues, 95 petPostsynapticModels: ExpCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 getInstance,	postSynapseModels.h, 339	pre_vars_in_post_learn_sparse/model.cc
postSynapseDecay StandardSubstitutions, 68 postSynapseModels.cc, 338 EXPDECAY, 338 POSTSYNAPSEMODELS_CC, 338 postSynModels, 338 postSynapseModels, 338 postSynapseModels, 338 EXPDECAY, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 postSynModels, 339 postSynmpicModels: Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getDecayCode, 73 getSupportCode, 73 getDerawlamse, 33 PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getTaramNames, 33 Port synapses_ini, 296 port vars in jost_learn sparse/test.cc SimTest, 371 IEST_P, 371 WRAPPED_INSTANTIATE_TEST_CASE_P, 371 pre_vars in sim_code/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars in ism_code/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars in ism_code/event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 317 pre_vars in ism_code/event/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 314 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 neuron_ini, 297 synapses_ini, 296 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 314 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 319 pre_vars in_ism_code_event/model_new.cc IMPLEMENT_MODEL,	postSynapseCurrentConverter	modelDefinition, 296
StandardSubstitutions, 68 EXPDECAY, 338 EXPDECAY, 338 IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynyModels, 338 preparePostSynModels, 338 EXPDECAY, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 postSynmodels, 339 postSynmodels, 339 postSynmodels, 339 postSynmodels, 339 postSynmodels, 339 postSyntoCurent extra_postsynapses.h, 233 postSynmodels, 537 PostsynapitcModels: Base, 72 getCurrentConverterCode, 73 getBeayCode, 73 postSynmodels: Sase, 72 getCurrentConverterCode, 73 getBeayCode, 73 getBeayCode, 73 PostsynapitcModels: Eacur, 92 getCurrentConverterCode, 33 getParamNames, 93 ParamValues, 93 PostsynapitcModels: ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getDecayCode, 95 getDecayCode, 95 getDerivedParams, 95 getBeayCode, 95 getDecayCode, 102 LegacyWrapper, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 102 LegacyWrapper, 101 pre_vars in_post_learn sparse/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 neuron_ini, 297 synapses_ini, 297 pre_vars_in_sim_code_event/model.cc modelDefinition, 297 neuron_ini, 297 synapses_ini, 297 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 neuron_ini, 297 synapses_ini, 298 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 revars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 synapses_ini, 297 vynapses_ini, 297 vynapses_ini, 297 vynapses_ini, 297 synapses_ini, 297 vynapses_ini, 297 vyna	StandardSubstitutions, 67	neuron_ini, 296
postSynapseModels.cc, 338     EXPDECAY, 338     POSTSYNAPSEMODELS_CC, 338     postSynModels, 338     postSynapseModels, 338     postSynapseModels, 338     EXPDECAY, 339     postSynapseModels, 338     EXPDECAY, 339     JZHIKEVICH_PS, 339     MAXPOSTSYN, 339     postSynModels, 339     postSynapseModels, 167     PostsynapticModels, 66     Poversing fin, 296     modelDefinition, 314     Trest_p, 371     WRAPPED_INSTANTIATE_TEST_CASE_P, 371     modelDefinition, 297     synapses_ini, 297     synapses_ini, 297     synapses_ini, 297     synapses_ini, 298     getOurentConverterCode, 93     getInstance, 93     quelCurrentConverterCode, 93     getParamNames, 93     PostsynapticModels::ExpCond, 94     getCurrentConverterCode, 95     getDearwAlues, 93     PostsynapticModels::ExpCond, 94     getCurrentConverterCode, 95     getDerivedParams, 95     getDerivedParams, 95     getDerivedParams, 95     getDerivedParams, 95     getDerivedParams, 95     getParamNames, 95     ParamValues, 95     VarValues, 95     VarValues, 95     VarValues, 95     VarValues, 95     PostsynapticModels::LegacyWrapper, 101     getCurrentConverterCode, 102     LegacyWrapper, 101     pre_vars_in_post_learn/model.cc     modelDefinition, 298     revars_in_post_learn/model.cc     modelDefinition, 298     revars_in_sim_code_event/model_new.cc     IMPLEMENT_MODEL, 314     modelDefinition, 297     revars_in_sim_code_event/model_new.cc     IMPLEMENT_MODEL, 314     modelDefinition, 298     revars_in_sim_code_event/model_new.cc     IMPLEMENT_MODEL, 314     modelDefinition, 298     revars_in_sim_code_event_sparse/model.cc     modelDefinition, 298     revars_in_sim_code_event_sparse/model_new.cc     IMPLEMENT_MODEL, 315     modelDefinition, 298     revars_in_sim_code_event_sparse/model_new.cc	postSynapseDecay	neuron_p, 296
EXPDECAY, 338 IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynModels, 338 preparePostSynModels, 338 postSynapseModels, 338 EXPDECAY, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 postSynModels, 339 postSynmbodel, 157 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getDecayCode, 73 getDecayCode, 73 getBaramNames, 93 ParamValues, 93 VarValues, 93 PostSynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 getDerivedParams, 95 getDerivedParams, 95 getDerivedParams, 95 getDerivedParams, 95 ParamValues, 95 PostsynapticModels::EapcCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 getDerivedParams, 95 petDerivedParams, 95 petDecayCode, 95 getDerivedParams, 95 petDecayCode, 95 getDerivedParams, 95 petDecayCode, 96 getParamNames, 95 ParamValues, 95 PostsynapticModels::LepacyWrapper, 101 getCurrentConverterCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn_sparse/redel_new.cc IMPLEMENT_MODEL, 313 modelDefinition, 313 pre_vars_in_sim_code/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 319 preparePostSynModels_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 397 resurs_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 397 resurs_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 319 pre_vars_in_sim_code_event_sparse/model_cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 297 vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 315 pre_vars_in_ism_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 315 pre_vars_in_ism_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 316 modelDefinition, 315 pre_vars_in_ism_code_event_sparse/lest.cc SimTest, 372 SimUstorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE	StandardSubstitutions, 68	neuron_p2, 296
IZHIKEVICH_PS, 338 POSTSYNAPSEMODELS_CC, 338 postSynModels, 338 presparePostSynModels, 338 postSynapseModels, 338 postSynapseModels, 338 postSynapseModels, 338 postSynapseModels, 339 postSynModels, 66 PostsynapticModels::Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getDecayCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn_sparse/test.cc SimTest, 372 simulatorBackends, 371 TEST_P, 371 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 Pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 297 neuron_ini, 297 pre_vars_in_isim_code/event/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 313 pre_vars_in_ism_code/model.cc modelDefinition, 297 neuron_ini, 297 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 313 pre_vars_in_sim_code/event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 313 pre_vars_in_sim_code_event_sparse/lest.cc SimTest, 372 simulatorBackends, 371 TEST_P, 371 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 Varialues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 313 pre_vars_in_post_learn_post_learn_post_learn_sparse/lest.cc SimTest, 371 simulatorBackends, 371 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 VarValues, 93 Pre_vars_in_sim_code_event/model_cc modelDefinition, 297 neu	postSynapseModels.cc, 338	synapses_ini, 296
POSTSYNAPSEMODELS_CC, 338     postSynModels, 338     preparePostSynModels, 338     postSynapseModels, 1, 338     EXPDECAY, 339     IZHIKEVICH_PS, 339     MAXPOSTSYN, 339     postSynModels, 369     postSynModels, 369     postSynModels, 379     pre_vars in, sim_code/event/model.cc     modelDefinition, 297     pre_vars in, sim_code/event/model.cc     modelDefinition, 314     pre_vars in, sim_code_event/model.cc     modelDefinition, 314     pre_vars in, sim_code_event/model.cc     modelDefinition, 314		
postSynModels, 338 postSynapseModels, 338 postSynapseModels, 338 EXPDECAY, 339 IZHIKEVICH PS, 339 IZHIKEVICH PS, 339 postSynModels, 366 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getDeacyCode, 73 PostsynapticModels: DeltaCurr, 92 getCurrentConverterCode, 93 getParamNames, 93 ParamValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 g		
preparePostSynModels, 338 postSynapseModels.h, 338 EXPDECAY, 339 IZHIKEVICH_PS, 339 IZHIXEVICH_PS, 319 IZHIX		
postSynapseModels.h, 338    EXPDECAY, 339    IZHIKEVICH_PS, 339    MAXPOSTSYN, 339    postSynModels, 339    postSynModel, 157    postSynapses.h, 233    postSynapses.h, 233    postSynapses.h, 237    postSynapticModels:Base, 72    getCurrentConverterCode, 73    getSupportCode, 73    getSupportCode, 73    getSuportSynapticModels:DeltaCurr, 92    getCurrentConverterCode, 93    getParamNames, 93    ParamValues, 93    PostsynapticModels:ExpCond, 94    getCurrentConverterCode, 95    getDecayCode, 95    getParamNames, 95    yarValues, 95    VarValues, 95    PostsynapticModels:LegacyWrapper, 101    getCurrentConverterCode, 101    getCurrentConverterCode, 102    getSupportCode, 102    synapses ini, 297    wra in sim code event sparse/model.cc    modelDefinition, 298    neuron_ini, 298    synapses_ini, 297    synapses_prode    imutatorBackends, 372    TEST_P, 372    WRAPPED_INSTANTIATE_TEST_CASE_P,	•	
EXPDECAY, 339 IZHIKEVICH_PS, 339 IZHIKEVICH_PS, 339 IZHIKEVICH_PS, 339 MAXPOSTSYN, 339 postSynModels, 339 postSynModels, 339 postSynModels, 339 postSynModels, 339 postSynModels, 339 postSynModels, 339 postSynModel, 157 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getDecayCode, 73 PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getDecayCode, 95 getDecayCode, 95 getDecayCode, 95 getDecayCode, 95 getDecayCode, 102 getSupportCode, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 102 getSupportCode, 103 getDecayCode, 104 getCurrentConverterCode, 105 getDecayCode, 106 getDecayCode, 107 getCurrentConverterCode,	• •	•
IZHIKEVICH_PS, 339  MAXPOSTSYN, 339  postSynModels, 339  preparePostSynModels, 339  postSyntoCurrent     extra_postsynapses.h, 233     postSynModel, 157  PostsynapticModels, 66  PostsynapticModels::Base, 72     getCurrentConverterCode, 73     getSupportCode, 73  getSupportCode, 73  getParamNames, 93  ParamValues, 93  VarValues, 95  PostsynapticModels::ExpCond, 94     getCurrentConverterCode, 95     getDerivedParams, 95     getDerivedParams, 95     paramValues, 95  VarValues, 95  PostsynapticModels::LegacyWrapper, 101     getCurrentConverterCode, 102     getSupportCode, 102     getSupportCode, 102     getSupportCode, 102     getSupportCode, 102     getDecayCode, 102     getSupportCode, 102     getDecayCode, 102     getSupportCode, 102     ge		
MAXPOSTSYN, 339 postSynModels, 339 preparePostSynModels, 339 postSynModels, 537  PostSynapticModels, 66 PostSynapticModels:Base, 72 getCurrentCorverterCode, 73 getDecayCode, 73 getDecayCode, 73 postSynapticModels:DeltaCurr, 92 getCurrentCorverterCode, 93 getInstance, 93 getParamNames, 93 ParamValues, 93 PostSynapticModels:ExpCond, 94 getCurrentCorverterCode, 95 getDecayCode, 95 getDerivedParams, 95 getDerivedParams, 95 getDerivedParams, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 ParamValues, 95 PostsynapticModels:LegacyWrapper, 101 getCurrentCorverterCode, 101 getDecayCode, 102 getSupportCode, 102	•	
postSynModels, 339 preparePostSynModels, 339 postSyntoCurrent extra_postsynapses.h, 233 postSynModel, 157 PostsynapticModels, 66 PostsynapticModels: Base, 72 getCurrentConverterCode, 73 getBecayCode, 73 getSupportCode, 73 PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getInstance, 93 ParamValues, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getDerivedParams, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getDecayCode, 102 getSupportCode, 102		
preparePostSynModels, 339 postSyntoCurrent extra_postsynapses.h, 233 postSynModel, 157 PostsynapticModels, 66 PostsynapticModels::Base, 72 getCurrentConverterCode, 73 getBecayCode, 73 getSupportCode, 73 getSupportCode, 73 getInstance, 93 getParamNames, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDerivedParams, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 getSupportCode, 102 getDecayCode, 102 getSupportCode, 102 getDecayCode, 102 getSupportCode, 102 getDecayCode, 102 getDecayCode, 102 getSupportCode, 102 getDecayCode, 102 getDecayCode, 102 getDecayCode, 102 getDecayCode, 102 getSupportCode, 102 getSupportCode		
postSyntoCurrent extra_postsynapses.h, 233 postSynModel, 157 PostsynapticModels, 66 PostsynapticModels::Base, 72 getDecayCode, 73 getSupportCode, 93 getParamNames, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getDerivedParams, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 PostsynapticModels::DegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 getSup	•	
extra_postsynapses.h, 233 postSynModel, 157  PostsynapticModels, 66 PostsynapticModels:Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getParamNames, 93 paramValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getIDecayCode, 95 getIDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 legacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p2, 295 pre_vars_in_sim_code/event_sparse/test.cc lMPLEMENT_MODEL, 314 modelDefinition, 297 neuron_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 314 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	• •	
postSynModel, 157 PostsynapticModels, 66 PostsynapticModels:Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 getParamNames, 93 PostsynapticModels:ExpCond, 94 getCurrentConverterCode, 95 getInstance, 95 getInstance, 95 ParamValues, 95 VarValues, 98 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 316 pre_vars_in_sim_code_event_sparse/res		
PostsynapticModels, 66 PostsynapticModels:Base, 72 getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 getSupportCode, 73 TEST_P, 372 PostsynapticModels::DeltaCurr, 92 getInstance, 93 getInstance, 93 ParamValues, 93 ParamValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getIbecayCode, 95 getIbratance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getCurrentConverterCode, 101 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p, 295 neuron_p2, 295  modelDefinition, 314 pre_vars_in_sim_code_event/model_new.cc SimTest, 372 pre_vars_in_sim_code_event/test.cc SimTest, 372 synapses_p, 298 pre_vars_in_sim_code_event/test.cc SimTest, 372 SimUatorBackends, 372 pre_vars_in_sim_code_event/trodel_new.cc modelDefinition, 314 pre_vars_in_sim_code_event/model_new.cc modelDefinition, 298 neuron_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		• – – – – –
PostsynapticModels::Base, 72 getCurrentConverterCode, 73 getSupportCode, 73 getSupportCode, 73 PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getParamNames, 93 ParamValues, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getDerardParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 ge	•	
getCurrentConverterCode, 73 getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 TEST_P, 372  PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getInstance, 95 getInstance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getDecayCode, 102 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p, 295 neuron_p2, 295  SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 Pre_vars_in_sim_code_event_sparse/model_cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_p, 288 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	• •	
getDecayCode, 73 getSupportCode, 73 getSupportCode, 73 PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295  simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WrapportCode_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	• •	
getSupportCode, 73  PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getInstance, 93 getParamNames, 93 ParamValues, 93  PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getInstance, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 paramValues, 95 VarValues, 95  PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getCurrentConverterCode, 101 getDecayCode, 102 getSupportCode, 102 getSupportC	_	
PostsynapticModels::DeltaCurr, 92 getCurrentConverterCode, 93 getInstance, 93 getParamNames, 93 ParamValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getParamNames, 95 ParamValues, 95 YarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 ParamValues, 95 VarValues, 95 ParamValues, 95 VarValues, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 297 synapses_ini, 297 synapses_ini, 297 synapses_in, 297 synapses_in, 298 pre_vars_in_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 WRAPPED_INSTANTIATE_TE	-	
getCurrentConverterCode, 93 getInstance, 93 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 YarValues, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 YarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 Test_p, 372 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PeasumVary 101 getDecayCode, 102 getSupportCode, 103 getDecayCode, 104 getDecayCode, 95 getParamNames, 95 getParamNam	÷	
getInstance, 93 getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getDecayCode, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 YarValues, 95 getParamNames, 95 getParamNames, 95 getParamNames, 95 YarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getSupportCode, 102 getSuportCode, 102 getSuportC	• •	
getParamNames, 93 ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getInstance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 modelDefinition, 298 pre_vars_in_sim_code_event_vest.cc SimTest, 372 SimulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 Pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	_	
ParamValues, 93 VarValues, 93 PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getDerivedParams, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 101 gre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p, 295 neuron_p2, 295  synapses_ini, 297 synapses_ini, 297 synapses_ini, 297 synapses_ini, 298 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 synapses_ini, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
VarValues, 93  PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getInstance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 modelDefinition, 298 pre_vars_in_bost_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295  synapses_p, 298 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
PostsynapticModels::ExpCond, 94 getCurrentConverterCode, 95 getDecayCode, 95 getDerivedParams, 95 getParamNames, 95 paramValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 getSupportCode, 102 modelDefinition, 298 pre_vars_in_sim_code_event/model_new.cc IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		· ·
getCurrentConverterCode, 95 getDecayCode, 95 getDecayCode, 95 getDerivedParams, 95 getInstance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getDecayCode, 102 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p, 295 neuron_p2, 295  IMPLEMENT_MODEL, 314 modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 SimTest, 372 SimTest, 372 SimTest, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 Pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_ini, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		• • —·
getDecayCode, 95 getDerivedParams, 95 getInstance, 95 getParamNames, 95 paramValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 getSupportCode, 102 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 pre_vars_in_sim_code_event_sparse/model_new.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295  modelDefinition, 314 pre_vars_in_sim_code_event/test.cc SimTest, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 synapses_ini, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
getDerivedParams, 95 getInstance, 95 getParamNames, 95 getParamValues, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295  pre_vars_in_sim_code_event_sparse/model_new.cc SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
getInstance, 95 getParamNames, 95 ParamValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_post_learn/model.cc modelDefinition, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_p, 295 neuron_p2, 295  SimTest, 372 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 neuron_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	- · · · · · · · · · · · · · · · · · · ·	
getParamNames, 95 ParamValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295 simulatorBackends, 372 TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	_	
ParamValues, 95 VarValues, 95 VarValues, 95 PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295  TEST_P, 372 WRAPPED_INSTANTIATE_TEST_CASE_P, 372 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
VarValues, 95  PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getSupportCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295  WRAPPED_INSTANTIATE_TEST_CASE_P, 372  WRAPPED_INSTANTIATE_TEST_CASE_P, 372  pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	<u> </u>	
PostsynapticModels::LegacyWrapper, 101 getCurrentConverterCode, 101 getDecayCode, 102 getSupportCode, 102 LegacyWrapper, 101 pre_vars_in_sim_code_event_sparse/model.cc modelDefinition, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295  pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
getCurrentConverterCode, 101  getDecayCode, 102  getSupportCode, 102  LegacyWrapper, 101  pre_vars_in_post_learn/model.cc  modelDefinition, 295  neuron_ini, 295  neuron_p2, 295  modelDefinition, 298  synapses_ini, 298  synapses_p, 298  pre_vars_in_sim_code_event_sparse/model_new.cc  IMPLEMENT_MODEL, 315  modelDefinition, 315  pre_vars_in_sim_code_event_sparse/test.cc  SimTest, 373	•	
getDecayCode, 102 getSupportCode, 102 LegacyWrapper, 101 synapses_ini, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295 neuron_p2, 295 neuron_ini, 298 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		. – – – – – .
getSupportCode, 102 LegacyWrapper, 101 synapses_ini, 298 synapses_p, 298 pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295 synapses_ini, 298 synapses_p, 298 pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373		
LegacyWrapper, 101 synapses_p, 298  pre_vars_in_post_learn/model.cc pre_vars_in_sim_code_event_sparse/model_new.cc  modelDefinition, 295 IMPLEMENT_MODEL, 315  neuron_ini, 295 modelDefinition, 315  neuron_p, 295 pre_vars_in_sim_code_event_sparse/test.cc  neuron_p2, 295 SimTest, 373	<del>-</del>	
pre_vars_in_post_learn/model.cc modelDefinition, 295 neuron_ini, 295 neuron_p2, 295  pre_vars_in_sim_code_event_sparse/model_new.cc IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	÷	• • —
modelDefinition, 295 neuron_ini, 295 neuron_p, 295 neuron_p2, 295  IMPLEMENT_MODEL, 315 modelDefinition, 315 pre_vars_in_sim_code_event_sparse/test.cc SimTest, 373	pre_vars_in_post_learn/model.cc	• • —
neuron_ini, 295 modelDefinition, 315 neuron_p, 295 pre_vars_in_sim_code_event_sparse/test.cc neuron_p2, 295 SimTest, 373		
neuron_p, 295 pre_vars_in_sim_code_event_sparse/test.cc neuron_p2, 295 SimTest, 373		
<del></del>		
synapses_ini, 295 simulatorBackends, 373	neuron_p2, 295	SimTest, 373
	synapses_ini, 295	simulatorBackends, 373

TEST_P, 373	neuronModels.cc, 324
WRAPPED_INSTANTIATE_TEST_CASE_P, 373	neuronModels.h, 326
pre_vars_in_sim_code_event_sparse_inv/model.cc	prepareWeightUpdateModels
modelDefinition, 299	synapseModels.cc, 356
neuron_ini, 299	synapseModels.h, 357
synapses_ini, 299	printSeparator
synapses_p, 299	Schmuker2014_classifier, 169
pre_vars_in_sim_code_event_sparse_inv/model_	printTextFile
new.cc	experiment.cc, 229
IMPLEMENT_MODEL, 315	printVector
modelDefinition, 315	gen_syns_sparse_izhModel.cc, 241
pre_vars_in_sim_code_event_sparse_inv/test.cc	procreatePop
SimTest, 374	GA.cc, 234
simulatorBackends, 374	push_back
TEST_P, 374	extra_neurons.h, 232
WRAPPED_INSTANTIATE_TEST_CASE_P, 374	extra_postsynapses.h, 233
pre_vars_in_sim_code_sparse/model.cc	pwSTDP, 158
modelDefinition, 300	calculateDerivedParameter, 158
neuron_ini, 300	
synapses_ini, 300	QTIsaac
pre_vars_in_sim_code_sparse/model_new.cc	∼QTlsaac, 159
IMPLEMENT_MODEL, 316	byte, 159
modelDefinition, 316	ind, 159
pre_vars_in_sim_code_sparse/test.cc	isaac, 159
SimTest, 374	N, 159
simulatorBackends, 374	QTIsaac, 159
TEST_P, 374	rand, 159
WRAPPED_INSTANTIATE_TEST_CASE_P, 374	randinit, 159
pre_vars_in_synapse_dynamics/model.cc	rngstep, 159
modelDefinition, 300	shuffle, 160
neuron_ini, 300	srand, 160
synapses_ini, 300	QTIsaac $<$ ALPHA, T $>$ , 159
pre_vars_in_synapse_dynamics/model_new.cc	QTIsaac< ALPHA, T >::randctx, 160
IMPLEMENT MODEL, 316	QTIsaac::randctx
<del>-</del> · · · ·	$\sim$ randctx, 160
modelDefinition, 316	randa, 160
pre_vars_in_synapse_dynamics/test.cc	randb, 160
SimTest, 375	randc, 160
simulatorBackends, 375	randont, 160
TEST_P, 375	randctx, 160
WRAPPED_INSTANTIATE_TEST_CASE_P, 375	randmem, 160
pre_vars_in_synapse_dynamics_sparse/model.cc	randrsl, 160
modelDefinition, 301	
neuron_ini, 301	R
synapses_ini, 301	gen_input_structured.cc, 236
pre_vars_in_synapse_dynamics_sparse/model_new.cc	gen_kcdn_syns.cc, 236
IMPLEMENT_MODEL, 316	gen_kcdn_syns_fixto10K.cc, 237
modelDefinition, 316	gen_pnkc_syns.cc, 238
pre_vars_in_synapse_dynamics_sparse/test.cc	gen_pnkc_syns_indivID.cc, 239
SimTest, 376	gen_syns_sparse.cc, 240
simulatorBackends, 376	gen_syns_sparse_izhModel.cc, 242
TEST_P, 376	lzh_sparse_model.cc, 268
WRAPPED_INSTANTIATE_TEST_CASE_P, 376	VClampGA.h, 383
preInd	R2
SparseProjection, 178	gen_kcdn_syns_fixto10K.cc, 237
preparePostSynModels	RANDOMGEN_CC
postSynapseModels.cc, 338	randomGen.cc, 340
postSynapseModels.h, 339	RANDOMGEN_H
prepareStandardModels	randomGen.h, 340

RAND	classol, 86, 87
experiment.cc, 229	read_pnkcsyns
VClampGA.h, 383	classol, 87
RECORDING_TIME_MS	read_pnlhisyns
experiment.h, 231	classol, 87
RECORDINGS_DIR	read_sparsesyns_par
experiment.h, 231	classIzh, 78
REGEX_OPERATIONAL	classol, 87, 88
lib/src/codeGenUtils.cc, 224	recordingsDir
REPEAT_LEARNING_SET	Schmuker2014_classifier, 171
experiment.h, 231	remap
REPORT_TIME	SparseProjection, 178
SynDelaySim.h, 359	resetClusterSpikeCountAN
rand	Schmuker2014_classifier, 169 resetCode
QTIsaac, 159	
randa	neuronModel, 135 resetDevice
QTIsaac::randctx, 160	Schmuker2014_classifier, 169
randb	resetIndividualSpikeCountPN
QTIsaac::randctx, 160	Schmuker2014 classifier, 169
randc	resetOverallWinner
QTIsaac::randctx, 160	Schmuker2014_classifier, 169
randont	revInd
QTIsaac::randctx, 160	SparseProjection, 178
randetx	revIndInG
QTIsaac::randctx, 160 randinit	SparseProjection, 178
	RG
QTIsaac, 159 randmem	gen_kcdn_syns.cc, 236
QTIsaac::randetx, 160	gen_kcdn_syns_fixto10K.cc, 237
randomEventOccurred	gen_pnkc_syns.cc, 238
Schmuker2014_classifier, 169	gen_pnkc_syns_indivID.cc, 239
randomGauss, 161	gen_syns_sparse.cc, 240
~randomGauss, 161	lzh_sparse_model.cc, 268
n, 161	VClampGA.h, 383
randomGauss, 161	Rind
srand, 161	gen_syns_sparse_izhModel.cc, 242
randomGen, 162	rngstep
∼randomGen, 162	QTIsaac, 159
n, 162	rulkovdp, 163
randomGen, 162	calculateDerivedParameter, 163
srand, 162	run
randomGen.cc, 339	classIzh, 78
RANDOMGEN_CC, 340	classol, 88
randomGen.h, 340	neuronpop, 137
RANDOMGEN_H, 340	Schmuker2014_classifier, 169
randomizeVar	SynDelay, 189 runCPU
classIzh, 78	
randomizeVarSq	classol, 88, 89 runGPU
classIzh, 78	classol, 89
randrsl	runexpHH
QTIsaac::randctx, 160	helper.h, 263
read_PNIzh1syns	, 200
classol, 87	S_ISDIR
read_input_patterns	experiment.cc, 229
classol, 86	SAVEP
read_input_values	CodeHelper.h, 227
classIzh, 78	SCALAR_MAX
read_kcdnsyns	HHVClampParameters.h, 265

Izh_sparse_project/model/sizes.h, 344	newNeuronModels.h, 330
MBody1_project/model/sizes.h, 345	SET_VARS
MBody_delayedSyn_project/model/sizes.h, 345	newModels.h, 328
MBody_individualID_project/model/sizes.h, 346	SETUP_THE_C
MBody_map_project/model/sizes.h, 346	simulation_synapse_policy_sparse.h, 342
MBody_map_robot1_project/model/sizes.h, 347	SPARSE_GLOBALG
MBody_userdef_project/model/sizes.h, 347	synapseMatrixType.h, 355
OneComp_project/model/sizes.h, 348	SPARSE_INDIVIDUALG
PoissonIzh_project/model/sizes.h, 348	synapseMatrixType.h, 355
SCALAR_MIN	SPARSEUTILS_CC
HHVClampParameters.h, 265	sparseUtils.cc, 349
lzh_sparse_project/model/sizes.h, 344	SPARSE
MBody1_project/model/sizes.h, 345	modelSpec.h, 323
MBody_delayedSyn_project/model/sizes.h, 345	synapseMatrixType.h, 355
MBody_individualID_project/model/sizes.h, 346	SPIKESOURCE
MBody_map_project/model/sizes.h, 346	neuronModels.cc, 325
MBody_map_robot1_project/model/sizes.h, 347	neuronModels.h, 327
MBody_userdef_project/model/sizes.h, 347	SPIKING_ACTIVITY_THRESHOLD_HZ
OneComp_project/model/sizes.h, 348	experiment.h, 231
PoissonIzh_project/model/sizes.h, 348	SYN_OUT_TME
SET_CURRENT_CONVERTER_CODE	MBody1_project/model/classol_sim.h, 218
newPostsynapticModels.h, 331	MBody_delayedSyn_project/model/classol_sim.h
SET_DECAY_CODE	219
newPostsynapticModels.h, 331	MBody_individualID_project/model/classol_sim.h
SET_DERIVED_PARAMS	220
newModels.h, 328	MBody_map_project/model/classol_sim.h, 221
SET_EVENT_CODE	MBody_map_robot1_project/model/classol_sim.h
newWeightUpdateModels.h, 332	222
SET_EVENT_THRESHOLD_CONDITION_CODE	MBody_userdef_project/model/classol_sim.h, 223
newWeightUpdateModels.h, 332	PoissonIzh_sim.h, 337
SET_EXTRA_GLOBAL_PARAMS	SYNAPSE_TAU_ANAN
newNeuronModels.h, 330	Model_Schmuker_2014_classifier.cc, 320
newWeightUpdateModels.h, 332	SYNAPSE_TAU_PNAN
SET_LEARN_POST_CODE	Model_Schmuker_2014_classifier.cc, 320
newWeightUpdateModels.h, 332	SYNAPSE_TAU_PNPN
SET_LEARN_POST_SUPPORT_CODE	Model_Schmuker_2014_classifier.cc, 320
newWeightUpdateModels.h, 333	SYNAPSE_TAU_RNPN
SET_NEEDS_POST_SPIKE_TIME	Model_Schmuker_2014_classifier.cc, 320
newWeightUpdateModels.h, 333	SYNAPSEMODELS_CC
SET_NEEDS_PRE_SPIKE_TIME	synapseModels.cc, 356
newWeightUpdateModels.h, 333	SYNDELAYSIM_CU
SET_PARAM_NAMES	SynDelaySim.cc, 358
newModels.h, 328	SYNTYPENO
SET_RESET_CODE	synapseModels.h, 357
newNeuronModels.h, 330	sampleDistance
SET_SIM_CODE	Schmuker2014_classifier, 171
newNeuronModels.h, 330	scalar
newWeightUpdateModels.h, 333	HHVClampParameters.h, 265
SET_SIM_SUPPORT_CODE	Izh_sparse_project/model/sizes.h, 344
newWeightUpdateModels.h, 333	MBody1_project/model/sizes.h, 345
SET_SUPPORT_CODE	MBody_delayedSyn_project/model/sizes.h, 345
newNeuronModels.h, 330	MBody_individualID_project/model/sizes.h, 346
newPostsynapticModels.h, 331	MBody_map_project/model/sizes.h, 346
SET_SYNAPSE_DYNAMICS_CODE	MBody_map_robot1_project/model/sizes.h, 347
newWeightUpdateModels.h, 333	MBody_userdef_project/model/sizes.h, 347
SET_SYNAPSE_DYNAMICS_SUPPORT_CODE	OneComp_project/model/sizes.h, 348
newWeightUpdateModels.h, 333	PoissonIzh_project/model/sizes.h, 348
SET_THRESHOLD_CONDITION_CODE	scalarExpr

NNmodel, 146	outputSpikes, 169
Schmuker2014_classifier, 165	overallWinnerSpikeCountAN, 170
~Schmuker2014_classifier, 168	param_CONNECTIVITY_AN_AN, 170
addInputRate, 168	param_CONNECTIVITY_PN_AN, 170
allocateHostAndDeviceMemory, 168	param_CONNECTIVITY_PN_PN, 170
applyLearningRuleSynapses, 168	param_CONNECTIVITY_RN_PN, 170
cacheDir, 170	param_GLOBAL_WEIGHT_SCALING, 170
calculateCurrentWindowWinner, 168	param_MAX_FIRING_RATE_HZ, 170
calculateOverallWinner, 168	param_MAX_WEIGHT_PN_AN, 170
calculateVrResponse, 168	param_MIN_FIRING_RATE_HZ, 170
calculateWinner, 168	param_MIN_WEIGHT_PN_AN, 170
checkContents, 168	param_PLASTICITY_INTERVAL_MS, 170
classLabel, 170	param_SPIKING_ACTIVITY_THRESHOLD_HZ,
clearDownDevice, 168	170
clearedDownDevice, 170	param_WEIGHT_DELTA_PN_AN, 170
clusterSpikeCountAN, 170	param_WEIGHT_RN_PN, 171
convertToRateCode, 168	param_WEIGHT_WTA_AN_AN, 171
correctClass, 170	param_WEIGHT_WTA_PN_PN, 171
countAN, 170	plasticWeights, 171
countPNAN, 170	populateDeviceMemory, 169
countPN, 170	printSeparator, 169
countRN, 170	randomEventOccurred, 169
createWTAConnectivity, 168	recordingsDir, 171
d_inputRates, 170	resetClusterSpikeCountAN, 169
d_maxRandomNumber, 170	resetDevice, 169
data_type, 168	resetIndividualSpikeCountPN, 169
data_type_double, 168	resetOverallWinner, 169
data_type_float, 168	run, 169
data_type_int, 168	sampleDistance, 171
data_type_uint, 168	Schmuker2014_classifier, 168
datasetName, 170	setCorrectClass, 169
findMaxMinSampleDistances, 168	setMaxMinSampleDistances, 169
generate_inputrates_dataset, 168	startLog, 169
generate_or_load_inputrates_dataset, 168	timestepsPerRecording, 171
generateSimulatedTimeSeriesData, 168	uniqueRunId, 171
getClassCluster, 168	update_input_data_on_device, 169
getClusterIndex, 168	updateClusterSpikeCountAN, 169
getManhattanDistance, 168	updateIndividualSpikeCountPN, 169
getRand0to1, 168	updateWeights_PN_AN_on_device, 169
getRecordingFilename, 169	updateWeights_PN_AN, 169
getSampleDistance, 169	vrData, 171
getSpikeNumbersFromGPU, 169	winningClass, 171
getSpikesFromGPU, 169	Schmuker2014_classifier.cc, 341
individualSpikeCountPN, 170	_SCHMUKER2014_CLASSIFIER_, 341
initialiseInputData, 169	Schmuker2014_classifier.h, 341
initialiseWeights_DENSE_PN_AN, 169	setClusterIndex
initialiseWeights SPARSE RN PN, 169	NeuronGroup, 133
initialiseWeights WTA AN AN, 169	SynapseGroup, 188
initialiseWeights_WTA_PN_PN, 169	setConstInp
inputRates, 170	NNmodel, 146
inputRatesSize, 170	setCorrectClass
load_VR_data, 169	Schmuker2014_classifier, 169
loadArrayFromTextFile, 169	setDefaultParamValues
loadClassLabels, 169	experiment.cc, 229
log, 170	setDT
model, 170	NNmodel, 146
openRecordingFile, 169	setEventThresholdReTestRequired
outputDir, 170	SynapseGroup, 188

setGPUDevice	CodeHelper, 91
NNmodel, 146	setWUVarZeroCopyEnabled
setInput	SynapseGroup, 188
classIzh, 78	setB
setMaxConn	gen_pnkc_syns_indivID.cc, 239
NNmodel, 146	utils.h, 381
setMaxConnections	setup_kernel
SynapseGroup, 188	GeNNHelperKrnls.cu, 258
setMaxMinSampleDistances	GeNNHelperKrnls.h, 258
Schmuker2014_classifier, 169	showPtxInfo
setName	GENN_PREFERENCES, 65
NNmodel, 146	shuffle
setNeuronClusterIndex	QTIsaac, 160
NNmodel, 146	sigENa
setPSVarZeroCopyEnabled	helper.h, 264
SynapseGroup, 188	sigEK
setPopulationSums	helper.h, 263
NNmodel, 146	sigEl
setPrecision	helper.h, 263
NNmodel, 147	sigGNa
setRNType	helper.h, 264
NNmodel, 147	sigGK
setSeed	helper.h, 264
NNmodel, 147	sigGl
setSpanType	helper.h, 264
SynapseGroup, 188	sigC
setSpanTypeToPre	helper.h, 263
NNmodel, 147	simCode
setSparseConnectivityFromDense	extra_neurons.h, 232
sparseUtils.h, 351	neuronModel, 135
setSpikeEventRequired	weightUpdateModel, 198
NeuronGroup, 133	simCode_supportCode
SynapseGroup, 188	weightUpdateModel, 198
setSpikeEventZeroCopyEnabled	simCodeEvnt
NeuronGroup, 133	weightUpdateModel, 198
setSpikeTimeRequired	simLearnPost
NeuronGroup, 133	weightUpdateModel, 198
setSpikeTimeZeroCopyEnabled	simLearnPost supportCode
NeuronGroup, 133	weightUpdateModel, 199
setSpikeZeroCopyEnabled	SimTest, 171
NeuronGroup, 133	extra_global_params_in_sim_code/test.cc, 362
setSynapseClusterIndex	extra_global_params_in_sim_code_event_←
NNmodel, 147	sparse_inv/test.cc, 363
setSynapseG	extra_global_post_param_in_sim_code/test.cc,
NNmodel, 147	363
SetTheW	extra global pre param in sim code/test.cc, 364
SimulationSynapsePolicyDense, 174	Init, 171, 172
setTiming	neuron_support_code_sim/test.cc, 365
NNmodel, 147	neuron_support_code_threshold/test.cc, 365
setTrueSpikeRequired	post_vars_in_post_learn/test.cc, 366
NeuronGroup, 133	post_vars_in_post_learn_sparse/test.cc, 367
SynapseGroup, 188	post_vars_in_post_learn_sparse/lest.cc, 367 post_vars_in_sim_code/test.cc, 368
· · · · · · · · · · · · · · · · · · ·	
SetUp SimulationTest 175	post_vars_in_sim_code_sparse/test.cc, 368
SimulationTest, 175	post_vars_in_synapse_dynamics_test.cc, 369
SingleValueSubstitutionTest, 177	post_vars_in_synapse_dynamics_sparse/test.cc,
setVarZeroCopyEnabled	370
NeuronGroup, 133	pre_vars_in_post_learn/test.cc, 370
setVerbose	pre_vars_in_post_learn_sparse/test.cc, 371

pre_vars_in_sim_code/test.cc, 372	Init, 175
pre_vars_in_sim_code_event/test.cc, 372	SetUp, 175
pre_vars_in_sim_code_event_sparse/test.cc, 373	StepGeNN, 175
pre_vars_in_sim_code_event_sparse_inv/test.cc,	TearDown, 175
374	SimulationTestDecoderMatrix, 176
pre_vars_in_sim_code_sparse/test.cc, 374	Simulate, 176
pre_vars_in_synapse_dynamics/test.cc, 375	SimulationTestVars
pre_vars_in_synapse_dynamics_sparse/test.cc,	Init, 176
376	Simulate, 176
Simulate, 172	SimulationTestVars< NeuronPolicy, SynapsePolicy >,
synapse_support_code_event_sim_code/test.cc,	176
376	simulatorBackends
synapse_support_code_event_threshold/test.cc, 377	decode_matrix_globalg_bitmask/test.cc, 359 decode_matrix_globalg_dense/test.cc, 360
synapse_support_code_post_learn/test.cc, 378	decode_matrix_globalg_sparse/test.cc, 360
synapse_support_code_sim_code/test.cc, 378	decode_matrix_individualg_dense/test.cc, 361
synapse_support_code_synapse_dynamics/test.←	decode_matrix_individualg_sparse/test.cc, 361
cc, 379	extra_global_params_in_sim_code/test.cc, 362
Simulate	extra_global_params_in_sim_code_event_~
SimTest, 172	sparse_inv/test.cc, 363
SimulationSynapsePolicy, 173	extra_global_post_param_in_sim_code/test.cc,
SimulationSynapsePolicyDense, 174	364
SimulationSynapsePolicyNone, 174	extra_global_pre_param_in_sim_code/test.cc, 364
SimulationSynapsePolicySparse, 175	neuron_support_code_sim/test.cc, 365
SimulationTestDecoderMatrix, 176	neuron_support_code_threshold/test.cc, 366
SimulationTestVars, 176	post_vars_in_post_learn/test.cc, 366
simulation_neuron_policy_pre_post_var.h, 341	post_vars_in_post_learn_sparse/test.cc, 367
simulation_neuron_policy_pre_var.h, 342	post_vars_in_sim_code/test.cc, 368
simulation_synapse_policy_dense.h, 342	post_vars_in_sim_code_sparse/test.cc, 368
simulation_synapse_policy_none.h, 342	post_vars_in_synapse_dynamics/test.cc, 369
simulation_synapse_policy_sparse.h, 342 SETUP_THE_C, 342	post_vars_in_synapse_dynamics_sparse/test.cc, 370
simulation_test.h, 343	
INIT_SPARSE, 343	pre_vars_in_post_learn/test.cc, 370 pre_vars_in_post_learn_sparse/test.cc, 371
TOKENPASTE, 343	pre vars in sim code/test.cc, 372
WRAPPED_INSTANTIATE_TEST_CASE_P, 343	pre_vars_in_sim_code_event/test.cc, 372 pre_vars_in_sim_code_event/test.cc, 372
simulation test decoder matrix.h, 343	pre_vars_in_sim_code_event_sparse/test.cc, 373
simulation_test_vars.h, 343	pre_vars_in_sim_code_event_sparse_inv/test.cc,
ASSIGN_ARRAY_VARS, 343	374
SimulationNeuronPolicyPrePostVar, 172	pre_vars_in_sim_code_sparse/test.cc, 374
Init, 172	pre_vars_in_synapse_dynamics/test.cc, 375
SimulationNeuronPolicyPreVar, 172	pre_vars_in_synapse_dynamics_sparse/test.cc,
Init, 172	376
SimulationSynapsePolicy, 173	synapse_support_code_event_sim_code/test.cc,
Init, 173	376
Simulate, 173	synapse_support_code_event_threshold/test.cc,
SimulationSynapsePolicyDense, 173	377
GetTheW, 174	synapse_support_code_post_learn/test.cc, 378
Init, 174	synapse_support_code_sim_code/test.cc, 378
SetTheW, 174	synapse_support_code_synapse_dynamics/test.
Simulate, 174	cc, 379
SimulationSynapsePolicyNone, 174	single_var_init_fullrange
Init, 174	helper.h, 263
Simulate, 174	single_var_reinit
SimulationSynapsePolicySparse, 174	helper.h, 263
Init, 175	SingleValueSubstitutionTest, 177
Simulate, 175	GetCode, 177
SimulationTest, 175	SetUp, 177

cizo a	weightUpdateSim, 68
size_g	•
classol, 90 sizes.h, 344–348	weightUpdateThresholdCondition, 68
•	standardSubstitutions.cc, 352
SpanType	standardSubstitutions.h, 352
SynapseGroup, 186	DerivedParamNameIterCtx, 353
SparseProjection, 177	ExtraGlobalParamNameIterCtx, 353
connN, 178	VarNamelterCtx, 353
ind, 178	start
indlnG, 178	stopWatch, 185
preInd, 178	startLog
remap, 178	Schmuker2014_classifier, 169
revInd, 178	startTimer
revIndInG, 178	CStopWatch, 92
sparseProjection.h, 348	StaticGradedUserDef, 181
sparseUtils.cc, 349	getEventCode, 181
createPosttoPreArray, 349	getEventThresholdConditionCode, 181
createPreIndices, 349	getInstance, 182
initializeSparseArray, 349	getParamNames, 182
initializeSparseArrayPreInd, 349	getVars, 182
initializeSparseArrayRev, 349	ParamValues, 181
SPARSEUTILS_CC, 349	VarValues, 181
sparseUtils.h, 350	StaticPulseUserDef, 183
countEntriesAbove, 351	getInstance, 184
createPosttoPreArray, 351	getSimCode, 184
createPreIndices, 351	getVars, 184
createSparseConnectivityFromDense, 351	ParamValues, 184
getSparseVar, 351	VarValues, 184
getG, 351	stdRMP ini
initializeSparseArray, 351	MBody_map_project/model/MBody1.cc, 277
initializeSparseArrayPreInd, 351	MBody map robot1 project/model/MBody1.cc,
initializeSparseArrayPreInd, 351 initializeSparseArrayRev, 351	MBody_map_robot1_project/model/MBody1.cc, 278
initializeSparseArrayRev, 351	278
	278 stdRMP_p
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160	278 stdRMP_p
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278 stdRG, 184
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278 stdRG, 184 ~stdRG, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278 stdRG, 184 ~stdRG, 185 n, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278 stdRG, 184 ~stdRG, 185 n, 185 nlong, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320	278 stdRMP_p MBody_map_project/model/MBody1.cc, 277 MBody_map_robot1_project/model/MBody1.cc, 278 stdRG, 184 ~stdRG, 185 n, 185 nlong, 185 stdRG, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st	278  stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdTM_ini
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdRG, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronOutputInit, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronOutputInit, 66 neuronOutputInit, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody1_project/model/MBody1.cc, 275  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronOutputInit, 66 neuronOutputInit, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_individualID.cc, 281  MBody_individualID.cc, 281  MBody_userdef.cc, 283
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66 neuronReset, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections, 66 neuronReset, 67 neuronSim, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections, 66 neuronReset, 67 neuronSpikeEventCondition, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175  stop
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronLocalVarWrite, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66 neuronReset, 67 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdTM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175  stop  stopWatch, 185
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronDutputInit, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67 neuronThresholdCondition, 67 postSynapseCurrentConverter, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175  stop  stopWatch, 185  stopTimer
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronOutputInit, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66 neuronReset, 67 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67 neuronThresholdCondition, 67 postSynapseCurrentConverter, 67 postSynapseDecay, 68	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175  stop  stopWatch, 185  stopTimer  CStopWatch, 92
initializeSparseArrayRev, 351 setSparseConnectivityFromDense, 351 srand QTIsaac, 160 randomGauss, 161 randomGen, 162 src/modelSpec.cc GeNNReady, 321 initGeNN, 320 MODELSPEC_CC, 320 st inputSpec, 97 StandardGeneratedSections, 66 neuronLocalVarInit, 66 neuronDutputInit, 66 neuronOutputInit, 66 neuronSpikeEventTest, 66 standardGeneratedSections.cc, 351 standardGeneratedSections.h, 352 StandardSubstitutions, 66 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67 neuronSpikeEventCondition, 67 neuronThresholdCondition, 67 postSynapseCurrentConverter, 67	stdRMP_p  MBody_map_project/model/MBody1.cc, 277  MBody_map_robot1_project/model/MBody1.cc, 278  stdRG, 184  ~stdRG, 185  n, 185  nlong, 185  stdRG, 185  stdRG, 185  stdRM_ini  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 279  MBody_individualID.cc, 281  MBody_userdef.cc, 283  stdTM_p  MBody1_project/model/MBody1.cc, 275  MBody_delayedSyn.cc, 280  MBody_individualID.cc, 281  MBody_userdef.cc, 283  StepGeNN  SimulationTest, 175  stop  stopWatch, 185  stopTimer

stop, 185	IMPLEMENT_MODEL, 317
StringPairVec	modelDefinition, 317
NewModels::Base, 70	synapse_support_code_event_sim_code/test.cc
stringUtils.h, 353	SimTest, 376
toString, 354	simulatorBackends, 376
tS, 354	TEST_P, 376
StringVec	WRAPPED_INSTANTIATE_TEST_CASE_P, 376
NewModels::Base, 70	synapse_support_code_event_threshold/model.cc
substitute	modelDefinition, 303
codeGenUtils.h, 226	neuron_ini, 303
lib/src/codeGenUtils.cc, 225	synapses_ini, 303
sum_spikes	synapses_p, 303
classIzh, 78	synapse_support_code_event_threshold/model_new.cc
classol, 89	IMPLEMENT_MODEL, 317
neuronpop, 137	modelDefinition, 317
sumDN	synapse_support_code_event_threshold/test.cc
classol, 90	SimTest, 377
sumIzh1	simulatorBackends, 377
classol, 90	TEST_P, 377
neuronpop, 137	WRAPPED_INSTANTIATE_TEST_CASE_P, 377
sumKC	synapse support code post learn/model.cc
classol, 91	modelDefinition, 303
sumLHI	neuron_ini, 303
classol, 91	neuron_p, 304
sumPExc	neuron_p2, 304
classIzh, 78	synapses_ini, 304
sumPInh	synapse_support_code_post_learn/model_new.cc
classIzh, 78	IMPLEMENT_MODEL, 318
sumPN	modelDefinition, 318
classol, 91	synapse_support_code_post_learn/test.cc
supportCode	SimTest, 378
neuronModel, 135	simulatorBackends, 378
postSynModel, 157	TEST_P, 378
SynDelay, 188	WRAPPED_INSTANTIATE_TEST_CASE_P, 378
~SynDelay, 188	synapse_support_code_sim_code/model.cc
run, 189	modelDefinition, 304
SynDelay, 188	neuron ini, 304
SynDelay.cc, 358	synapses_ini, 304
IMPLEMENT_MODEL, 358	synapse_support_code_sim_code/model_new.cc
modelDefinition, 358	IMPLEMENT_MODEL, 318
SynDelaySim.cc, 358	modelDefinition, 318
main, 358	synapse_support_code_sim_code/test.cc
SYNDELAYSIM_CU, 358	SimTest, 378
SynDelaySim.h, 358	simulatorBackends, 378
REPORT TIME, 359	TEST_P, 378
TOTAL_TIME, 359	WRAPPED_INSTANTIATE_TEST_CASE_P, 378
synDynBlkSz	synapse_support_code_synapse_dynamics/model.cc
global.cc, 259	modelDefinition, 305
global.h, 261	neuron_ini, 305
SynIzh_ini	synapses_ini, 305
Izh_sparse.cc, 268	synapse_support_code_synapse_dynamics/model_
synapse_support_code_event_sim_code/model.cc	new.cc
modelDefinition, 302	IMPLEMENT_MODEL, 319
neuron_ini, 302	modelDefinition, 319
synapses_ini, 302	synapse_support_code_synapse_dynamics/test.cc
synapses_p, 302	SimTest, 379
synapse_support_code_event_sim_code/model_new.	simulatorBackends, 379
CC CC	TEST_P, 379
•••	0 1 _ 1 , 0 1 0

WRAPPED_INSTANTIATE_TEST_CASE_P, 379	SpanType, 186
synapseBlkSz	SynapseGroup, 187
global.cc, 259	synapseGroup.cc, 354
global.h, 261	synapseGroup.h, 354
synapseBlockSize	SynapseMatrixConnectivity
GENN PREFERENCES, 65	synapseMatrixType.h, 355
SynapseConnType	SynapseMatrixType
modelSpec.h, 323	synapseMatrixType.h, 355
synapseDynamics	synapseMatrixType.h, 354
weightUpdateModel, 199	BITMASK_GLOBALG, 355
synapseDynamics_supportCode	BITMASK, 355
weightUpdateModel, 199	DENSE_GLOBALG, 355
synapseDynamicsBlockSize	DENSE INDIVIDUALG, 355
GENN_PREFERENCES, 65	DENSE, 355
SynapseGType	GLOBAL, 355
modelSpec.h, 323	INDIVIDUAL, 355
SynapseGroup, 185	operator&, 355
addExtraGlobalNeuronParams, 187	SPARSE_GLOBALG, 355
addExtraGlobalSynapseParams, 187	SPARSE_INDIVIDUALG, 355
calcKernelSizes, 187	SPARSE, 355
getDelaySteps, 187	SynapseMatrixConnectivity, 355
getMatrixType, 187	SynapseMatrixType, 355
getMaxConnections, 187	SynapseMatrixWeight, 355
getName, 187	SynapseMatrixWeight SynapseMatrixWeight
getOffsetPost, 187	synapseMatrixType.h, 355
getOffsetPre, 187	synapseModels.cc, 355
-	LEARN1SYNAPSE, 356
getPSDerivedParams, 187	
getPSInitVals, 187	NGRADSYNAPSE, 356
getPSModel, 187	NSYNAPSE, 356
getPSParams, 187	prepareWeightUpdateModels, 356
getPaddedDynKernelSize, 187	SYNAPSEMODELS_CC, 356
getPaddedKernelIDRange, 187	weightUpdateModels, 356
getPaddedPostLearnKernelSize, 187	synapseModels.h, 356
getSpanType, 187	LEARN1SYNAPSE, 357
getSrcNeuronGroup, 187	NGRADSYNAPSE, 357
getTrgNeuronGroup, 187	NSYNAPSE, 357
getWUDerivedParams, 187	prepareWeightUpdateModels, 357
getWUInitVals, 187	SYNTYPENO, 357
getWUModel, 187	weightUpdateModels, 357
getWUParams, 187	synapsePSize
initDerivedParams, 187	Izh_sparse.cc, 268
isEventThresholdReTestRequired, 187	synapses_ini
isPSAtomicAddRequired, 188	decode_matrix_globalg_bitmask/model.cc, 284
isPSVarZeroCopyEnabled, 188	decode_matrix_globalg_dense/model.cc, 284
isSpikeEventRequired, 188	decode_matrix_globalg_sparse/model.cc, 285
isTrueSpikeRequired, 188	decode_matrix_individualg_dense/model.cc, 285
isWUVarZeroCopyEnabled, 188	decode_matrix_individualg_sparse/model.cc, 286
isZeroCopyEnabled, 188	extra_global_params_in_sim_code_event_←
POSTSYNAPTIC, 186	sparse_inv/model.cc, 287
PRESYNAPTIC, 186	extra_global_post_param_in_sim_code/model.cc,
setClusterIndex, 188	288
setEventThresholdReTestRequired, 188	extra_global_pre_param_in_sim_code/model.cc,
setMaxConnections, 188	289
setPSVarZeroCopyEnabled, 188	neuron_support_code_sim/model.cc, 289
setSpanType, 188	neuron_support_code_threshold/model.cc, 290
setSpikeEventRequired, 188	post_vars_in_post_learn/model.cc, 291
setTrueSpikeRequired, 188	post_vars_in_post_learn_sparse/model.cc, 292
setWUVarZeroCopyEnabled, 188	post_vars_in_sim_code/model.cc, 292

р	ost_vars_in_sim_code_sparse/model.cc, 293	extra_global_post_param_in_sim_code/test.cc,
р	ost_vars_in_synapse_dynamics/model.cc, 294	363
р	ost_vars_in_synapse_dynamics_sparse/model. ←	extra_global_pre_param_in_sim_code/test.cc, 364
	cc, 294	neuron_support_code_sim/test.cc, 365
р	re_vars_in_post_learn/model.cc, 295	neuron_support_code_threshold/test.cc, 365
р	re_vars_in_post_learn_sparse/model.cc, 296	post_vars_in_post_learn/test.cc, 366
р	re_vars_in_sim_code/model.cc, 297	post_vars_in_post_learn_sparse/test.cc, 367
р	re_vars_in_sim_code_event/model.cc, 297	post_vars_in_sim_code/test.cc, 368
р	re_vars_in_sim_code_event_sparse/model.cc,	post_vars_in_sim_code_sparse/test.cc, 368
	298	post_vars_in_synapse_dynamics/test.cc, 369
р	re_vars_in_sim_code_event_sparse_inv/model. ← cc, 299	post_vars_in_synapse_dynamics_sparse/test.cc,
р	re_vars_in_sim_code_sparse/model.cc, 300	pre_vars_in_post_learn/test.cc, 370
	re_vars_in_synapse_dynamics/model.cc, 300	pre_vars_in_post_learn_sparse/test.cc, 371
	re_vars_in_synapse_dynamics_sparse/model.cc,	pre_vars_in_sim_code/test.cc, 372
	301	pre_vars_in_sim_code_event/test.cc, 372
S	ynapse_support_code_event_sim_code/model.↔	pre_vars_in_sim_code_event_sparse/test.cc, 373
	cc, 302	pre_vars_in_sim_code_event_sparse_inv/test.cc,
S	ynapse_support_code_event_threshold/model. ← cc, 303	374
9	ynapse_support_code_post_learn/model.cc, 304	pre_vars_in_sim_code_sparse/test.cc, 374
	ynapse_support_code_sim_code/model.cc, 304	pre_vars_in_synapse_dynamics/test.cc, 375
	ynapse_support_code_synapse_dynamics/model.	pre_vars_in_synapse_dynamics_sparse/test.cc, 376
	cc, 305	synapse_support_code_event_sim_code/test.cc,
synap		376
	re_vars_in_sim_code_event/model.cc, 298	synapse_support_code_event_threshold/test.cc,
р	re_vars_in_sim_code_event_sparse/model.cc,	377
	298	synapse_support_code_post_learn/test.cc, 378
р	re_vars_in_sim_code_event_sparse_inv/model.  ———————————————————————————————————	synapse_support_code_sim_code/test.cc, 378
	cc, 299	synapse_support_code_synapse_dynamics/test.
S	ynapse_support_code_event_sim_code/model.←	cc, 379
	cc, 302	tests/unit/codeGenUtils.cc, 225
S	ynapse_support_code_event_threshold/model.↔	TIMING
	cc, 303	MBody_userdef.cc, 282
		TOKENPASTE
t :		simulation test.h, 343
	nputSpec, 97	TOTAL RECORDINGS
_	OTTI_TIME	experiment.h, 231
	zh_sparse_sim.h, 269	TOTAL TIME
	/Body1_project/model/classol_sim.h, 218	SynDelaySim.h, 359
IV	MBody_delayedSyn_project/model/classol_sim.h,	TOTAL TME
	210	Izh sparse sim.h, 269
IV	MBody_individualID_project/model/classol_sim.h,	MBody1 project/model/classol sim.h, 218
	220	MBody_delayedSyn_project/model/classol_sim.h,
	MBody_map_project/model/classol_sim.h, 221	
IV	/IBody_map_robot1_project/model/classol_sim.h,	219
	222	MBody_individualID_project/model/classol_sim.h,
	MBody_userdef_project/model/classol_sim.h, 223	220
	OneComp_sim.h, 335	MBody_map_project/model/classol_sim.h, 221
	oissonIzh_sim.h, 337	MBody_map_robot1_project/model/classol_sim.h,
TEST_	_P	222
d	ecode_matrix_globalg_bitmask/test.cc, 359	MBody_userdef_project/model/classol_sim.h, 223
d	ecode_matrix_globalg_dense/test.cc, 360	OneComp_sim.h, 335
	ecode_matrix_globalg_sparse/test.cc, 360	PoissonIzh_sim.h, 337
	ecode_matrix_individualg_dense/test.cc, 361	TOTALT
	ecode_matrix_individualg_sparse/test.cc, 361	HHVClampParameters.h, 265
	xtra_global_params_in_sim_code/test.cc, 362	TRAUBMILES_ALTERNATIVE
	xtra_global_params_in_sim_code_event_	neuronModels.cc, 325
G	sparse_inv/test.cc, 363	neuronModels.h, 327
	oparso_mv/tost.66, 000	Houroniviouoloan, UL1

TRALIDAM FO. FACT	halaan ha 000
TRAUBMILES_FAST	helper.h, 263
neuronModels.cc, 325	tS
neuronModels.h, 327	stringUtils.h, 354
TRAUBMILES_PSTEP	LUNET
neuronModels.cc, 325	UINT
neuronModels.h, 327	experiment.h, 231
TRAUBMILES_SAFE	USE
neuronModels.cc, 325	utils.h, 381
neuronModels.h, 327	UTILS_CC
TRAUBMILES	utils.cc, 380
neuronModels.cc, 325	uniqueRunId
neuronModels.h, 327	Schmuker2014_classifier, 171
TearDown	update_input_data_on_device
SimulationTest, 175	Schmuker2014_classifier, 169
test.cc, 359–379	updateClusterSpikeCountAN
tests/unit/codeGenUtils.cc	Schmuker2014_classifier, 169
	updateIndividualSpikeCountPN
INSTANTIATE_TEST_CASE_P, 225	Schmuker2014_classifier, 169
TEST_P, 225	updateVarQueues
theDevice	NeuronGroup, 133
global.cc, 259	updateWeights_PN_AN_on_device
global.h, 261	Schmuker2014_classifier, 169
theRates	updateWeights_PN_AN
classol, 91	• – –
theSize	Schmuker2014_classifier, 169
utils.cc, 380	userCxxFlagsGNU
utils.h, 382	GENN_PREFERENCES, 65
thresholdConditionCode	userCxxFlagsWIN
neuronModel, 135	GENN_PREFERENCES, 65
timer	userNvccFlags
experiment.h, 231	GENN_PREFERENCES, 65
Izh_sparse_sim.h, 269	utils.cc, 379
MBody1_project/model/classol_sim.h, 218	cudaFuncGetAttributesDriver, 380
	gennError, 380
MBody_delayedSyn_project/model/classol_sim.h,	theSize, 380
ADD a divide a UD president/president/alaccellaire la	UTILS_CC, 380
MBody_individualID_project/model/classol_sim.h,	writeHeader, 380
221	utils.h, 380
MBody_map_project/model/classol_sim.h, 222	_UTILS_H_, 381
MBody_map_robot1_project/model/classol_sim.h,	B, 381
223	CHECK CU ERRORS, 381
MBody_userdef_project/model/classol_sim.h, 224	CHECK_CUDA_ERRORS, 381
OneComp_sim.h, 335	cudaFuncGetAttributesDriver, 382
PoissonIzh_sim.h, 337	delB, 381
VClampGA.h, 383	gennError, 382
timestepsPerRecording	setB, 381
Schmuker2014_classifier, 171	theSize, 382
tmpVarNames	USE, 381
neuronModel, 136	writeHeader, 382
tmpVarTypes	willer leader, 302
neuronModel, 136	V
toLower	inputSpec, 97
command_line_processing.h, 228	
toString	VClampGA.cc, 382
_	main, 382
stringUtils.h, 354	VClampGA.h, 382
toUpper	R, 383
command_line_processing.h, 228	RAND, 383
truevar_init	RG, 383
helper.h, 263	timer, 383
truevar_initexpHH	VR_DATA_FILENAME

experiment.h, 231	experiment.h, 231
value	WEIGHT_WTA_PN_PN
Parameter, 149	experiment.h, 231
value_substitutions	WRAPPED_INSTANTIATE_TEST_CASE_P
codeGenUtils.h, 226	decode_matrix_globalg_bitmask/test.cc, 359
ValueBase	decode_matrix_globalg_dense/test.cc, 360
NewModels::ValueBase, 195	decode_matrix_globalg_sparse/test.cc, 360
NewModels::ValueBase< 0 >, 196	decode_matrix_individualg_dense/test.cc, 361
var_init_fullrange	decode_matrix_individualg_sparse/test.cc, 361
helper.h, 263	extra_global_params_in_sim_code/test.cc, 362
var_reinit	extra_global_params_in_sim_code_event_←
helper.h, 263	sparse_inv/test.cc, 363
VarNameIterCtx	extra_global_post_param_in_sim_code/test.cc,
standardSubstitutions.h, 353	363
varNames	extra_global_pre_param_in_sim_code/test.cc, 364
neuronModel, 136	neuron_support_code_sim/test.cc, 365
postSynModel, 158	neuron_support_code_threshold/test.cc, 365
weightUpdateModel, 199	post_vars_in_post_learn/test.cc, 366
varTypes	post_vars_in_post_learn_sparse/test.cc, 367
neuronModel, 136	post_vars_in_sim_code/test.cc, 368
postSynModel, 158	post_vars_in_sim_code_sparse/test.cc, 368
weightUpdateModel, 199	post_vars_in_synapse_dynamics/test.cc, 369
VarValues	post_vars_in_synapse_dynamics_sparse/test.cc,
ExpCondUserDef, 96	370
MyHH, 107	pre_vars_in_post_learn/test.cc, 370
Mylzhikevich, 109	pre_vars_in_post_learn_sparse/test.cc, 371
MylzhikevichVariable, 110	pre_vars_in_sim_code/test.cc, 372
Neuron, 119	pre_vars_in_sim_code_event/test.cc, 372
NeuronModels::Izhikevich, 99	pre_vars_in_sim_code_event_sparse/test.cc, 373
NeuronModels::IzhikevichVariable, 101	pre_vars_in_sim_code_event_sparse_inv/test.cc,
NeuronModels::Poisson, 155	374
NeuronModels::RulkovMap, 165	pre_vars_in_sim_code_sparse/test.cc, 374
NeuronModels::SpikeSource, 179	pre_vars_in_synapse_dynamics/test.cc, 375
NeuronModels::TraubMiles, 191	pre_vars_in_synapse_dynamics_sparse/test.cc,
NeuronModels::TraubMilesAlt, 192	376
NeuronModels::TraubMilesFast, 193	simulation_test.h, 343
NeuronModels::TraubMilesNStep, 194	synapse_support_code_event_sim_code/test.cc,
PiecewiseSTDPUserDef, 153	376
PostsynapticModels::DeltaCurr, 93	synapse_support_code_event_threshold/test.cc,
PostsynapticModels::ExpCond, 95	377
StaticGradedUserDef, 181	synapse_support_code_post_learn/test.cc, 378
StaticPulseUserDef, 184	synapse_support_code_sim_code/test.cc, 378
Weight IndeteModel, 205, 206	synapse_support_code_synapse_dynamics/test. ←
Weight IndeteModelSpikeEvent, 214	cc, 379
Weight IndeteModels::PiecewiseSTDP, 151	weightUpdateDynamics
Weight IndeteModels::StaticGraded, 180	StandardSubstitutions, 68
WeightUpdateModels::StaticPulse, 183 vectorContains	WeightUpdateModel, 199
	getEventCode, 206
experiment.cc, 229	getEventThresholdConditionCode, 206, 207
Vexp helper.h, 264	getExtraGlobalParams, 207
vrData	getInstance, 207, 208
Schmuker2014_classifier, 171	getLearnPostCode, 208
Schilluker2014_classifier, 171	getLearnPostSupportCode, 208
WEIGHT_DELTA_PN_AN	getParamNames, 209
experiment.h, 231	getSimCode, 209, 210
WEIGHT_RN_PN	getSimSupportCode, 210
experiment.h, 231	getSynapseDynamicsCode, 210, 211
WEIGHT_WTA_AN_AN	getSynapseDynamicsSuppportCode, 211

getVars, 211-213	isPreSpikeTimeRequired, 105
ParamValues, 204, 205	LegacyWrapper, 104
VarValues, 205, 206	WeightUpdateModels::PiecewiseSTDP, 149
weightUpdateModel, 196	getDerivedParams, 151
~weightUpdateModel, 198	getInstance, 151
dpNames, 198	getLearnPostCode, 151
dps, 198	getParamNames, 151
evntThreshold, 198	getSimCode, 152
extraGlobalSynapseKernelParameterTypes, 198	getVars, 152
extraGlobalSynapseKernelParameters, 198	isPostSpikeTimeRequired, 152
	isPreSpikeTimeRequired, 152
needPostSt, 198	ParamValues, 151
needPreSt, 198	VarValues, 151
pNames, 198	WeightUpdateModels::StaticGraded, 179
simCode, 198	
simCode_supportCode, 198	getEventCode, 180
simCodeEvnt, 198	getEventThresholdConditionCode, 180
simLearnPost, 198	getInstance, 180
simLearnPost_supportCode, 199	getParamNames, 180
synapseDynamics, 199	getVars, 180
synapseDynamics_supportCode, 199	ParamValues, 180
varNames, 199	VarValues, 180
varTypes, 199	WeightUpdateModels::StaticPulse, 182
weightUpdateModel, 198	getInstance, 183
WeightUpdateModelSpikeEvent, 213	getSimCode, 183
getEventCode, 214	getVars, 183
getEventThresholdConditionCode, 214	ParamValues, 183
getInstance, 214	VarValues, 183
getParamNames, 214	weightUpdatePostLearn
getVars, 214	StandardSubstitutions, 68
ParamValues, 214	weightUpdateSim
VarValues, 214	StandardSubstitutions, 68
WeightUpdateModels, 68	weightUpdateThresholdCondition
weightUpdateModels	StandardSubstitutions, 68
synapseModels.cc, 356	winningClass
synapseModels.h, 357	Schmuker2014_classifier, 171
WeightUpdateModels::Base, 73	write_input_to_file
getEventCode, 75	classIzh, 78
getEventThresholdConditionCode, 75	write_kcdnsyns
getExtraGlobalParams, 75	classol, 89
getLearnPostCode, 75	write_para
getLearnPostSupportCode, 75	helper.h, 263
getSimCode, 75	write_pnkcsyns
getSimSupportCode, 75	classol, 90
getSynapseDynamicsCode, 76	write_pnlhisyns
	classol, 90
getSynapseDynamicsSuppportCode, 76	writeHeader
isPostSpikeTimeRequired, 76	utils.cc, 380
isPreSpikeTimeRequired, 76	utils.h, 382
WeightUpdateModels::LegacyWrapper, 103	
getEventCode, 104	xorwow_setup
getEventThresholdConditionCode, 104	GeNNHelperKrnls.cu, 258
getExtraGlobalParams, 104	GeNNHelperKrnls.h, 258
getLearnPostCode, 104	2
getLearnPostSupportCode, 104	zeroCopyInUse
getSimCode, 105	NNmodel, 148
getSimSupportCode, 105	zipStringVectors
getSynapseDynamicsCode, 105	NewModels::LegacyWrapper, 103
getSynapseDynamicsSuppportCode, 105	
isPostSpikeTimeRequired, 105	