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.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 `$GENN_PATH/userproject/xxx_project` and `$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.bat 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 GeNN 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 `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 `test1` and will be created in a sub-directory with the name `test1_output`. More about the DEBUG flag in the debugging section . The parameter `FLOAT` will run the model in float (single precision floating point), using `DOUBLE` would use double precision. The REU↩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 `generate_run` engine that allows to run a Insect olfaction model of the insect mushroom body:

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

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 `generate_run.exe` will generate connectivity matrices for the model `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 `test1` and will be created in a sub-directory with the name `test1_output`. More about the DEBUG flag in the debugging section . The parameter `FLOAT` will run the model in float (single precision floating point), using `DOUBLE` would use double precision. The REU↩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↩_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 `genn-buildmodel.sh` (On Linux or Mac) or `genn-buildmodel.bat` (on Windows). In the example of the MBody1_project this entails writing neuron numbers into `userproject/include/sizes.h`, 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/MBody1.cc`. It then executes the GeNN code generator to generate the complete model simulation code for the MBody1 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.

   ```
   NeuronModels::Poisson::ParamValues poissonParams(
     10.0,        // firing rate
     2.5,        // refratory period
     20.0,       // Vspike
     -60.0);       // Vrest
   ```

   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);
   ```

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/MBody1_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 `copyStateFromDevice()` 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)

```
Izhikevich neuron(s) without any connections
====================================

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
-----

generate_run <0(CPU)/1(GPU)> <n> <DIR> <MODEL>

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.

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
-----

generate_run <0(CPU)/1(GPU)> <nPoisson> <nIzhikevich> <pConn> <gscale> <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]

4.3 Pulse-coupled Izhikevich network

```
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‘ \<synapse population>=‘_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/MBody1_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)> <nAL> <nKC> <nLH> <nDN> <gScale> <DIR> <MODEL>

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/MBody1.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 MBody1_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
  -----

generate_run <0(CPU)/1(GPU)/n(GPU n-2)> <nAL> <nKC> <nLH> <nDN> <gScale> <DIR> <MODEL>

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_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
-----

generate_run <0(CPU)/1(GPU)/n(GPU n-2)> <nAL> <nKC> <nLH> <nDN> <gScale> <DIR> <MODEL>

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_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),
doi:10.1007/s00422-005-0019-7
```

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

4.8 Insect Olfaction Model using delayed synapses

```
Locust olfactory system (Nowotny et al. 2005)
=============================================

A variation of the \ref ex_mbody example using synaptic delays.
In this example, the Kenyon Cell-Decision Neuron synapses are
delayed by (5 * 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
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
-----

generate_run <0(CPU)/1(GPU)/n(GPU n-2)> <nAL> <nKC> <nLH> <nDN> <gScale> <DIR> <MODEL>

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
```

```
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> <protocol> <nPop> <totalT> <outdir>

Mandatory parameters:
GPU/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:
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 -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.
2. 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.

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 postsynaptic 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 implemented 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_CODE/definitions.h`, rather than `myModel_CODE/runner.cc`. Including `runner.cc` will likely result in pages of linking errors at best!*

Developer Side Changes

1. 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`, `showPtxInfo`, `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.
3. 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\<SYNAPSENAME\>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.

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.
5. 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

```
float myPNKC_p[3]= {
0.0,            // 0 - Erev: Reversal potential
-20.0,          // 1 - Epre: Presynaptic threshold potential
1.0             // 2 - tau_S: decay time constant for S [ms]
};
```

   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

```
float *myPNKC_p= NULL;
float postExpPNKC[2]={
  1.0,              // 0 - tau_S: decay time constant for S [ms]
  0.0               // 1 - Erev: Reversal potential
};
```

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

   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 `generate_run` type are now all located together with the models they run instead of in a common `tools` directory. Generally the structure now is that each example project contains the wrapper script `generate_run` and a `model` 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 `model` subdirectory in its own `modelname_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.

9. 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 `neuron_tme`, `synapse_tme` and `learning_tme`. 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 `MBody1.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

1. `allocateSparseArrays()` has been changed to take the number of connections, connN, as an argument rather than expecting it to have been set in the Connetion 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

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

```
model.addSynapsePopulation<WeightUpdateModel, PostsynapticModel>(name,
       mType, delay, preName, postName, weightParamValues, weightVarValues, postsynapticParamValues,
      postsynapticVarValues);
```

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", "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

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 $\tau \frac{dV}{dt} = -V + I_{\text{syn}}$ 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 $\exp\left(\frac{-t}{\tau}\right)$:

```
class LeakyIntegrator2 : public NeuronModels::Base
{
public:
    DECLARE_MODEL(LeakyIntegrator2, 1, 1);

    SET_SIM_CODE("$(V) = $(Isyn) - $(ExpTC)*($(Isyn) - $(V));");

    SET_THRESHOLD_CONDITION_CODE("$(V) >= -50.0");

    SET_PARAM_NAMES({"tau"});

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

    SET_DERIVED_PARAMS({
        {"ExpTC", [](const vector<double> &pars, double dt){ return std::exp(-dt / pars[0]); }}});
};
```

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.

```
networkModel.addNeuronPopulation<LeakyIntegrator>("Neurons", 1,
                                                   LeakyIntegrator::ParamValues(20.0 /*tau*/),
                                                   LeakyIntegrator::VarValues(0.0/*V*/));
```

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_TIME(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 indInG array, with the number of connections being the size of ind. More specifically, `indIng`[i+1]-indIng[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 caled `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`[indIng[n]]=gp[indIng[n]+1]. See tools/gen_syns_sparse_IzhModel used in Izh_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 NNmodel::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_NAMES(), 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:

```
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↩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 μ`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,

```
// definition of tenHHModel
NeuronModels::TraubMiles::ParamValues p(
  7.15,          // 0 - gNa: Na conductance in muS
  50.0,          // 1 - ENa: Na equi potential in mV
  1.43,          // 2 - gK: K conductance in muS
 -95.0,          // 3 - EK: K equi potential in mV
  0.02672,       // 4 - gl: leak conductance in muS
  -63.563,       // 5 - El: leak equi potential in mV
  0.143          // 6 - Cmem: membr. capacity density in nF
);

NeuronModels::TraubMiles::VarValues ini(
  -60.0,         // 0 - membrane potential V
  0.0529324,     // 1 - prob. for Na channel activation m
  0.3176767,     // 2 - prob. for not Na channel blocking h
  0.5961207      // 3 - prob. for K channel activation n
);
```

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,

```
model.addNeuronPopulation<NeuronModels::TraubMiles>("Pop1", 10,
      p, ini);
```

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(
    7.15,          // 0 - gNa: Na conductance in muS
    50.0,          // 1 - ENa: Na equi potential in mV
    1.43,          // 2 - gK: K conductance in muS
   -95.0,          // 3 - EK: K equi potential in mV
    0.02672,       // 4 - gl: leak conductance in muS
    -63.563,       // 5 - El: leak equi potential in mV
    0.143          // 6 - Cmem: membr. capacity density in nF
  );

  NeuronModels::TraubMiles::VarValues ini(
    -60.0,         // 0 - membrane potential V
    0.0529324,     // 1 - prob. for Na channel activation m
    0.3176767,     // 2 - prob. for not Na channel blocking h
    0.5961207      // 3 - prob. for K channel activation n
  );
  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 `stepTimeCPU()` (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();
}
```

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;
}
```

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

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

```
EXECUTABLE      :=tenHHSimulation
SOURCES         :=tenHHSimulation.cc

include $(GENN_PATH)/userproject/include/makefile_common_gnu.mk
```

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();
```

Note

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

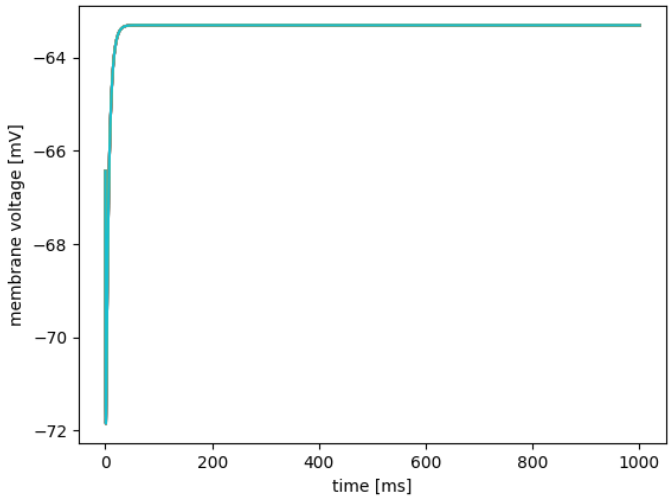
You will also need to add:

```
#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:

```
WeightUpdateModels::StaticPulse::VarValues s_ini(
  0.0 // 0 - g: the synaptic conductance value
);

PostsynapticModels::ExpCond::ParamValues ps_p(
  1.0,      // 0 - tau_S: decay time constant for S [ms]
  -80.0     // 1 - Erev: Reversal potential
);
```

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>(
  "Pop1self", SynapseMatrixType::DENSE_INDIVIDUALG, 10,
  "Pop1", "Pop1",
  {}, s_ini,
  ps_p, {});
```

The addSynapsePopulation parameters are

- WeightUpdateModel: template parameter specifying the type of weight update model (derived from WeightUpdateModels::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::VarValues 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(
    7.15,          // 0 - gNa: Na conductance in muS
    50.0,          // 1 - ENa: Na equi potential in mV
    1.43,          // 2 - gK: K conductance in muS
    -95.0,          // 3 - EK: K equi potential in mV
    0.02672,       // 4 - gl: leak conductance in muS
    -63.563,       // 5 - El: leak equi potential in mV
    0.143         // 6 - Cmem: membr. capacity density in nF
  );

  NeuronModels::TraubMiles::VarValues ini(
    -60.0,         // 0 - membrane potential V
    0.0529324,     // 1 - prob. for Na channel activation m
    0.3176767,     // 2 - prob. for not Na channel blocking h
    0.5961207      // 3 - prob. for K channel activation n
  );
  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(
    1.0,      // 0 - tau_S: decay time constant for S [ms]
    -80.0     // 1 - Erev: Reversal potential
  );

  model.addSynapsePopulation<WeightUpdateModels::StaticPulse
      , 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;
  gPop1self[pre*10+post]= -0.2;
}
pushPop1selfStateToDevice();
```

After memory allocation and initialization `gPop1self` will contain only zeros. We then assign in the loop a non-zero conductivity of 0.2 μS 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
```

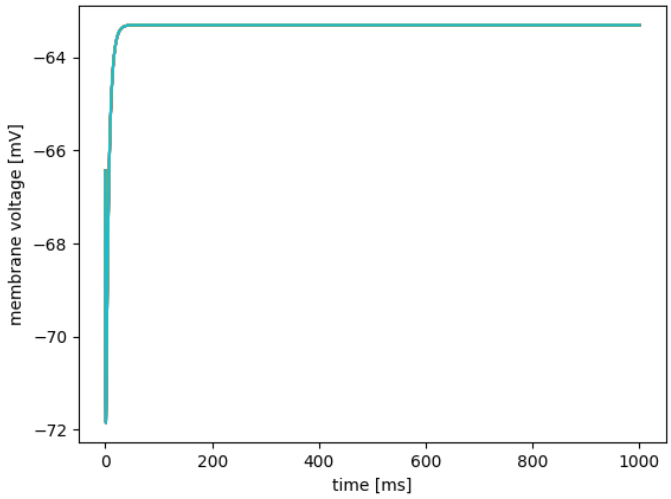
and make it

```
>> 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↩Definition(...)` function:

```
model.addNeuronPopulation<NeuronModels::SpikeSource>("Stim", 1,
       {}, {});
model.addSynapsePopulation<WeightUpdateModels::StaticPulse
      , PostsynapticModels::ExpCond>(
        "StimPop1", SynapseMatrixType::DENSE_INDIVIDUALG,
      NO_DELAY,
        "Stim", "Pop1",
        {}, s_ini,
        ps_p, {});
```

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

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

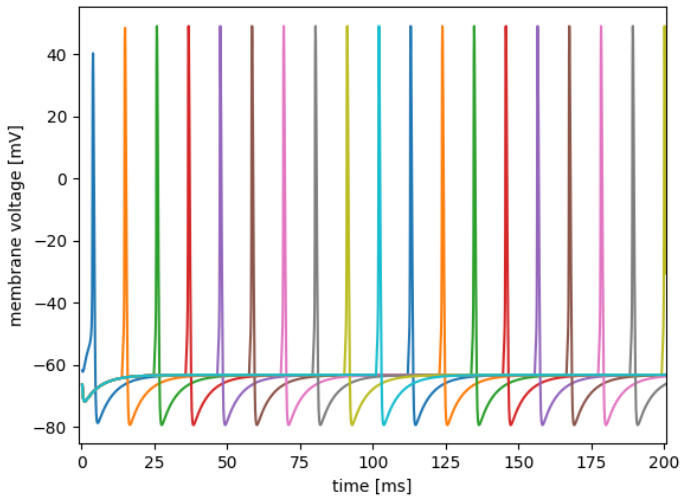
and finally inject a spike in the first timestep

```
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;
      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 `modelDefinition`)

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\<neuron or synapse name\>StateToDevice()` to copy from the host to the GPU. At the end of your simulation, if you want to access the variables you need to copy them back from the device using the `pull\<neuron or synapse name\>StateFromDevice()` function or one of the more fine-grained functions listed above. Alternatively, you can directly use the CUDA memcopy functions. **Copying elements between the GPU and the host memory is very costly in terms of performance and should only be done when needed.**

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 `pullPopStateFrom`↩ `Device()` 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

  ```
  SET_CURRENT_CONVERTER_CODE("$(inSyn)*($(E)-$(V))");
  ```

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

  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 `cuda-gdb` to debug on the GPU. Example projects in the `userproject` 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:

12 Hierarchical Index

12.1 Class Hierarchy

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

13 Class Index

13.1 Class List

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

14 File Index

14.1 File List

Here is a list of all files with brief descriptions:

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 VarNameIterCtx &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 ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void postSynapseDecay (std::string &pdCode, const SynapseGroup *sg, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronThresholdCondition (std::string &thCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronSim (std::string &sCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronSpikeEventCondition (std::string &eCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void neuronReset (std::string &rCode, const NeuronGroup &ng, const VarNameIterCtx &nmVars, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nmExtraGlobalParams, const std::string &ftype)
- void weightUpdateThresholdCondition (std::string &eCode, const SynapseGroup &sg, const DerivedParamNameIterCtx &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 &wuExtraGlobalParams, const string &preIdx, const string &postIdx, const string &devPrefix, const std::string &ftype)
- void 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 weightUpdatePostLearn (std::string &code, const SynapseGroup *sg, const DerivedParamNameIterCtx &wuDerivedParams, const ExtraGlobalParamNameIterCtx &wuExtraGlobalParams, const string &preIdx, const string &postIdx, 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

psCode	the code string to work on

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 & *preIdx,* const string & *postIdx,* const string & *devPrefix,* const std::string & *ftype*)

Parameters

preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	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 & *preIdx,* const string & *postIdx,* const string & *devPrefix,* const std::string & *ftype*)

Parameters

preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	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 & *preIdx,* const string & *postIdx,* const string & *devPrefix,* const std::string & *ftype*)

Parameters

preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	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 & *preIdx,* const string & *postIdx,* const string & *devPrefix,* const std::string & *ftype*)

Parameters

preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	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

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.

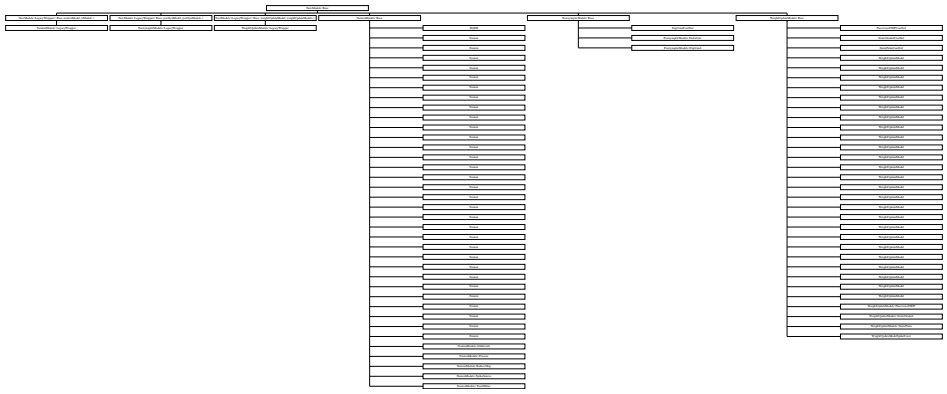
16 Class Documentation

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
- typedef std::vector< std::string > StringVec
- 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::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, WeightUpdateModels::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, PiecewiseSTDPUserDef, MyIzhikevich, MyIzhikevich, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, WeightUpdateModel, 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, WeightUpdateModels::StaticPulse, NeuronModels::RulkovMap, StaticGradedUserDef, StaticPulseUserDef, MyHH, WeightUpdateModelSpikeEvent, MyIzhikevichVariable, PiecewiseSTDPUserDef, WeightUpdateModel, Neuron, and Neuron.

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

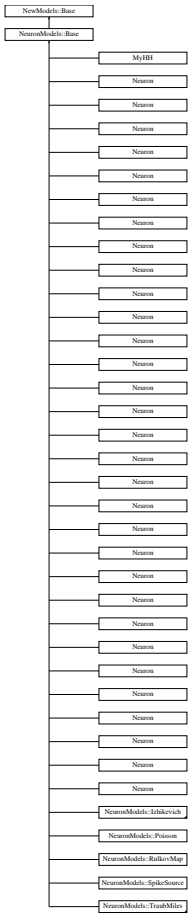
- newModels.h

16.2 NeuronModels::Base Class Reference

Base class for all neuron models.

`#include <newNeuronModels.h>`

Inheritance diagram for NeuronModels::Base:



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::TraubMilesFast, NeuronModels::TraubMiles, NeuronModels::Poisson, NeuronModels::Izhikevich, NeuronModels::RulkovMap, MyHH, MyIzhikevichVariable, MyIzhikevich, MyIzhikevich, Neuron, and Neuron.

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, NeuronModels::Izhikevich, NeuronModels::RulkovMap, MyHH, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, Neuron, and Neuron.

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

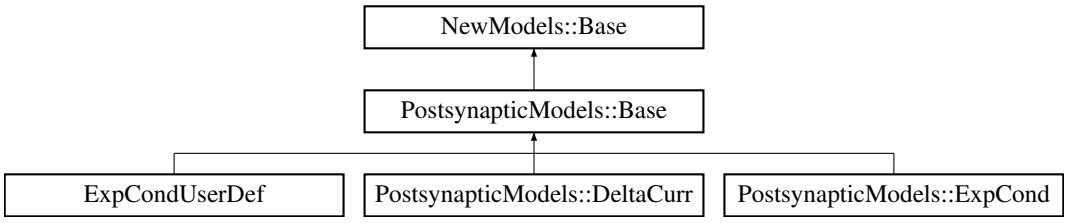
- newNeuronModels.h

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:

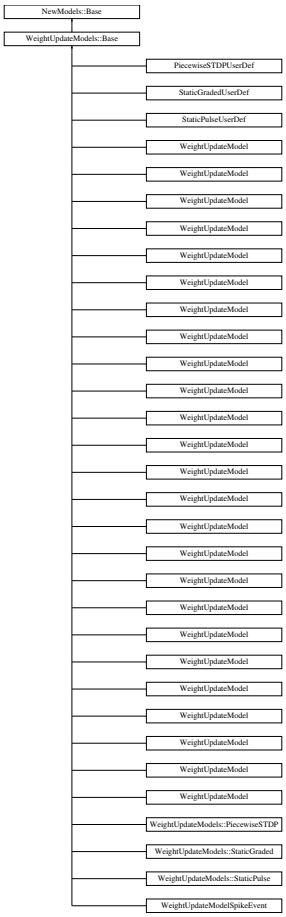
- newPostsynapticModels.h

16.4 WeightUpdateModels::Base Class Reference

Base class for all weight update models.

`#include <newWeightUpdateModels.h>`

Inheritance diagram for WeightUpdateModels::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, 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, 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, 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, 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.

16.4.2.9 virtual std::string WeightUpdateModels::Base::getSynapseDynamicsSuppportCode ( ) const [inline], [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.

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 ()
- ~classIzh ()
- 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 sumPInh

16.5.1 Constructor & Destructor Documentation

16.5.1.1 classlzh::classlzh ( )

16.5.1.2 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 classlzh::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	name of synapse group
gscale	the maximal conductance of generated synapses

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 classlzh::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 classlzh::init (unsigned int *which*)

16.5.2.10 void classlzh::initializeAllVars (unsigned int *which*)

16.5.2.11 void classlzh::output_params (FILE * *f*, FILE * *f2*)

16.5.2.12 void classlzh::output_spikes (FILE * *f*, unsigned int *which*)

16.5.2.13 void classlzh::output_state (FILE * *f*, unsigned int *which*)

16.5.2.14 void classlzh::randomizeVar (scalar * *Var*, scalar *strength*, const NeuronGroup * *neuronGrp*)

16.5.2.15 void classlzh::randomizeVarSq (scalar * *Var*, scalar *strength*, const NeuronGroup * *neuronGrp*)

16.5.2.16 void classlzh::read_input_values (FILE *)

16.5.2.17 void classlzh::read_sparsesyns_par (const char * *synGrpName*, struct SparseProjection *C*, FILE * *f_ind*, FILE * *f_indInG*, FILE * *f_g*, scalar * *g*)

Read sparse connectivity from a file.

Parameters

synGrpName	index of the synapse population to be worked on
C	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
f_g	File handle for a file containing sparse conductivity values
g	array to receive the conductance values

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 classlzh::sumPInh

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

- lzh_sparse_model.h
- lzh_sparse_model.cc

16.6 classol Class Reference

This class cpontains 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 cpontains 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 ( )

16.6.3.28 void classol::getSpikeNumbersFromGPU ( )

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 *which*)

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	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_indInG,* FILE * *f_g,* double * *g*)

Read sparse connectivity from a file.

Parameters

synGrpName	name of the synapse population to be worked on
C	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
C	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_indlnG	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
C	contains the arrays to be initialized from file
f_ind	file pointer for the indices of post-synaptic neurons
f_indlnG	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 *runtime*)

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

f	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::sumDN

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:

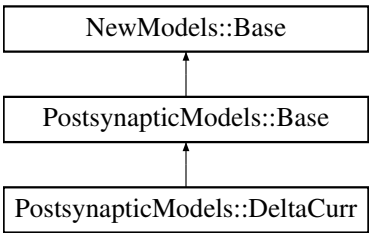
- hr_time.h
- hr_time.cc

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.

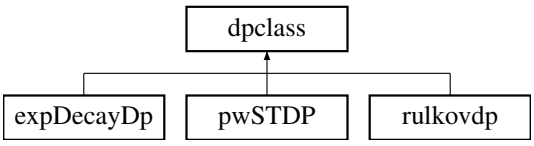
The documentation for this class was generated from the following file:

- 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

16.10.1.1 virtual double dpclass::calculateDerivedParameter (int , vector< double > , double = 0.5) [inline], [virtual]

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:

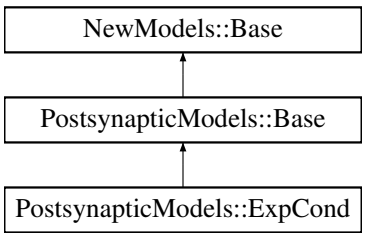
- GA.cc

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.

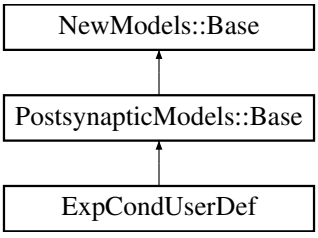
Reimplemented from NewModels::Base.

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:

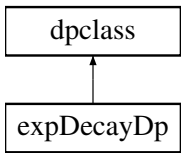
- MBody_userdef.cc

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:

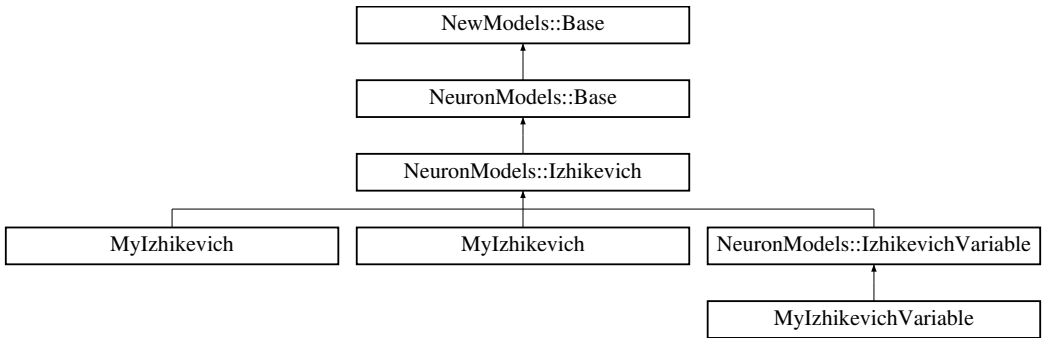
- helper.h

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

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

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:

- V - 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 StringVec NeuronModels::Izhikevich::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

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

16.16.3.3 virtual std::string NeuronModels::Izhikevich::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 MyIzhikevichVariable, MyIzhikevich, and MyIzhikevich.

16.16.3.4 virtual std::string NeuronModels::Izhikevich::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::Izhikevich::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

Reimplemented in NeuronModels::IzhikevichVariable, and MyIzhikevichVariable.

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

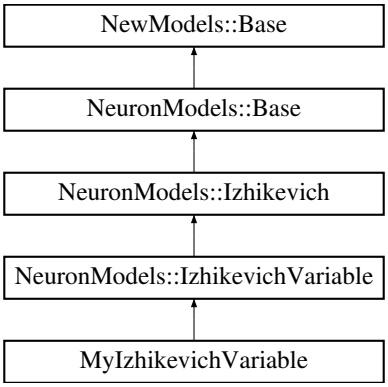
- newNeuronModels.h

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:

- `V` - 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 StringVec NeuronModels::IzhikevichVariable::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

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

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

Reimplemented from NeuronModels::Izhikevich.

Reimplemented in MyIzhikevichVariable.

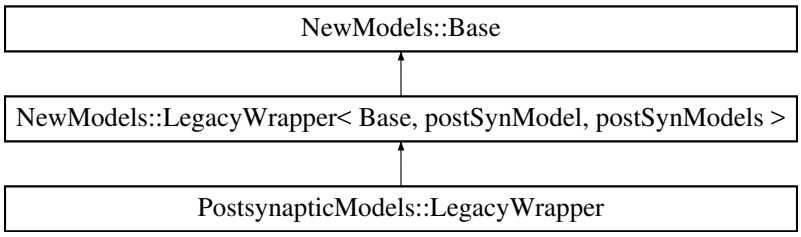
The documentation for this class was generated from the following file:

- 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:

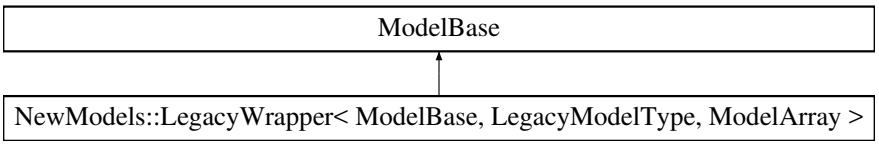
- newPostsynapticModels.h
- newPostsynapticModels.cc

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:

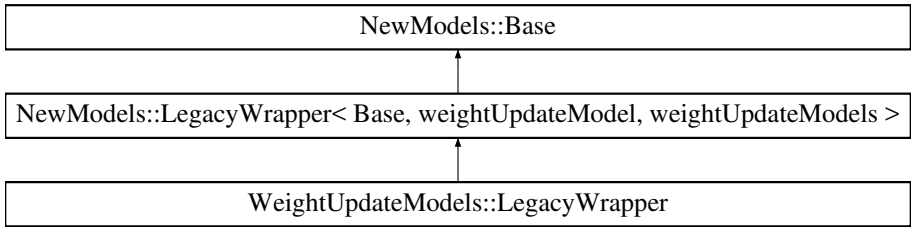
- newModels.h

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:

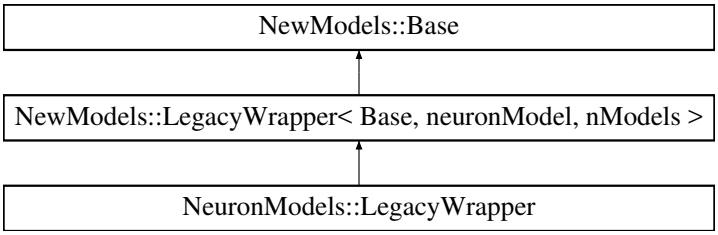
- newWeightUpdateModels.h
- newWeightUpdateModels.cc

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").

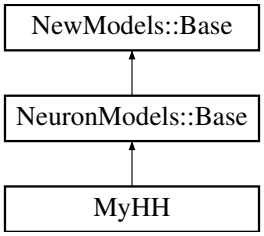
16.21.3.6 bool NeuronModels::LegacyWrapper::isPoisson ( ) const [virtual]

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.

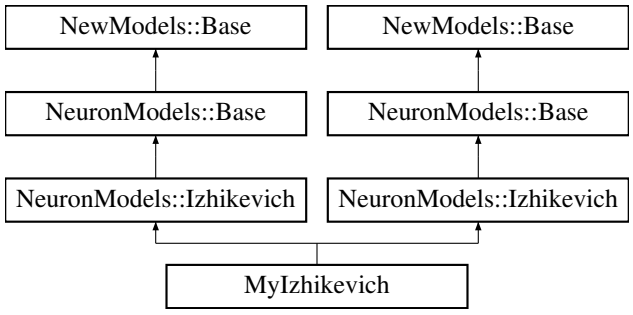
Reimplemented from NewModels::Base.

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

- HHVClamp.cc

16.23 MyIzhikevich Class Reference

Inheritance diagram for MyIzhikevich:



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 MyIzhikevich * getInstance ()
- static const MyIzhikevich * getInstance ()

16.23.1 Member Typedef Documentation

16.23.1.1 typedef NewModels::ValueBase< 5 > MyIzhikevich::ParamValues

16.23.1.2 typedef NewModels::ValueBase< 5 > MyIzhikevich::ParamValues

16.23.1.3 typedef NewModels::ValueBase< 2 > MyIzhikevich::VarValues

16.23.1.4 typedef NewModels::ValueBase< 2 > MyIzhikevich::VarValues

16.23.2 Member Function Documentation

16.23.2.1 static const MyIzhikevich* MyIzhikevich::getInstance ( ) [inline], [static]

16.23.2.2 static const MyIzhikevich* MyIzhikevich::getInstance ( ) [inline], [static]

16.23.2.3 virtual StringVec MyIzhikevich::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

16.23.2.4 virtual StringVec MyIzhikevich::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

16.23.2.5 virtual std::string MyIzhikevich::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 MyIzhikevich::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.

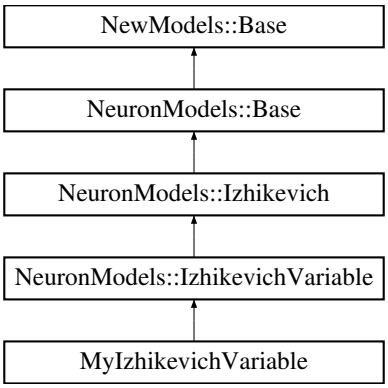
Reimplemented from NeuronModels::Izhikevich.

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

- OneComp.cc
- SynDelay.cc

16.24 MyIzhikevichVariable Class Reference

Inheritance diagram for MyIzhikevichVariable:



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 MyIzhikevichVariable * getInstance ()

16.24.1 Member Typedef Documentation

16.24.1.1 typedef NewModels::ValueBase< 0 > MyIzhikevichVariable::ParamValues

16.24.1.2 typedef NewModels::ValueBase< 7 > MyIzhikevichVariable::VarValues

16.24.2 Member Function Documentation

16.24.2.1 static const MyIzhikevichVariable* MyIzhikevichVariable::getInstance ( ) [inline], [static]

16.24.2.2 virtual std::string MyIzhikevichVariable::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 NameIterCtx< Container > Struct Template Reference

`#include <standardSubstitutions.h>`

Public Types

- typedef PairKeyConstIter< typename Container::const_iterator > NameIter

Public Member Functions

- NameIterCtx (const Container &c)

Public Attributes

- const Container container
- const NameIter nameBegin
- const NameIter nameEnd

16.25.1 Member Typedef Documentation

16.25.1.1 template<typename Container > typedef PairKeyConstIter<typename Container::const_iterator> NameIterCtx< Container >::NameIter

16.25.2 Constructor & Destructor Documentation

16.25.2.1 template<typename Container > NameIterCtx< Container >::NameIterCtx (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

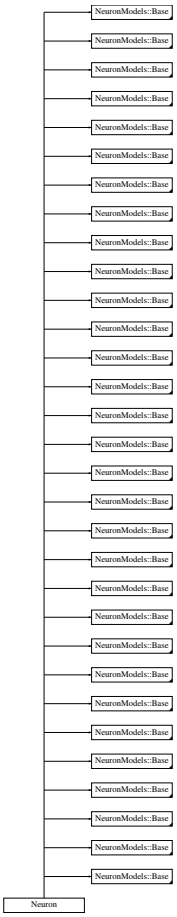
16.25.3.3 template<typename Container > const NameIter NameIterCtx< Container >::nameEnd

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

- standardSubstitutions.h

16.26 Neuron Class Reference

Inheritance diagram for Neuron:



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 ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- static const Neuron * getInstance ()
- 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::ParamValues

16.26.1.3 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.4 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.5 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.6 typedef NewModels::ValueBase< 1 > Neuron::ParamValues

16.26.1.7 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.8 typedef NewModels::ValueBase< 1 > Neuron::ParamValues

16.26.1.9 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.10 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.11 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.12 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.13 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.14 typedef NewModels::ValueBase< 1 > Neuron::ParamValues

16.26.1.15 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.16 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.17 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.18 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.19 typedef NewModels::ValueBase< 1 > Neuron::ParamValues

16.26.1.20 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.21 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.22 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.23 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.24 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.25 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.26 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.27 typedef NewModels::ValueBase< 1 > Neuron::ParamValues

16.26.1.28 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.29 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.30 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.31 typedef NewModels::ValueBase< 0 > Neuron::ParamValues

16.26.1.32 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.33 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.34 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.35 typedef NewModels::ValueBase< 1 > Neuron::VarValues

16.26.1.36 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.37 typedef NewModels::ValueBase< 1 > Neuron::VarValues

16.26.1.38 typedef NewModels::ValueBase< 2 > 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 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.43 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.44 typedef NewModels::ValueBase< 1 > Neuron::VarValues

16.26.1.45 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.46 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.47 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.48 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.49 typedef NewModels::ValueBase< 1 > Neuron::VarValues

16.26.1.50 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.51 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.52 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.53 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.54 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.55 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.56 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.57 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.58 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.59 typedef NewModels::ValueBase< 2 > Neuron::VarValues

16.26.1.60 typedef NewModels::ValueBase< 2 > 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 StringVec Neuron::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.26.2.36 virtual StringVec 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 StringVec Neuron::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.26.2.39 virtual StringVec 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.

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.

16.26.2.86 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.94 virtual StringPairVec 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 StringPairVec 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.104 virtual StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.105 virtual StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.106 virtual StringPairVec Neuron::getVars ( ) 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.109 virtual StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.110 virtual StringPairVec 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.114 virtual StringPairVec Neuron::getVars ( ) 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.117 virtual StringPairVec Neuron::getVars ( ) 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 StringPairVec Neuron::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.26.2.120 virtual StringPairVec Neuron::getVars ( ) 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 StringPairVec Neuron::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:

- 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 addInSyn (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::addInSyn (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::pair<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

16.29.3.1 NNmodel neuronpop::model

16.29.3.2 unsigned int neuronpop::sumIzh1

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.
- template<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 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 string &, const vector< double > &, 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
p	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
p	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
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.

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
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.

16.30.2.7 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 > & *PSVini*, 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.
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.

16.30.2.9 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, 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, 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, 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

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 specifcation 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?

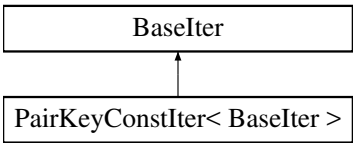
The documentation for this class was generated from the following files:

- modelSpec.h
- src/modelSpec.cc

16.31 PairKeyConstIter< BaseIter > 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 BaseIter > PairKeyConstIter< BaseIter >::PairKeyConstIter ( ) [inline]

16.31.1.2 template<typename BaseIter > PairKeyConstIter< BaseIter >::PairKeyConstIter (BaseIter *iter*) [inline]

16.31.2 Member Function Documentation

16.31.2.1 template<typename BaseIter > const KeyType& PairKeyConstIter< BaseIter >::operator* ( ) const [inline]

16.31.2.2 template<typename BaseIter > const KeyType* PairKeyConstIter< BaseIter >::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:

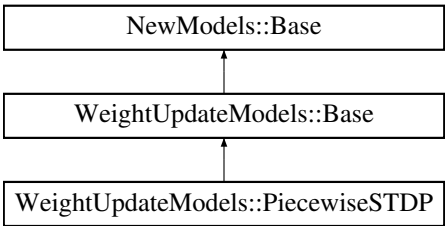
- experiment.cc

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.

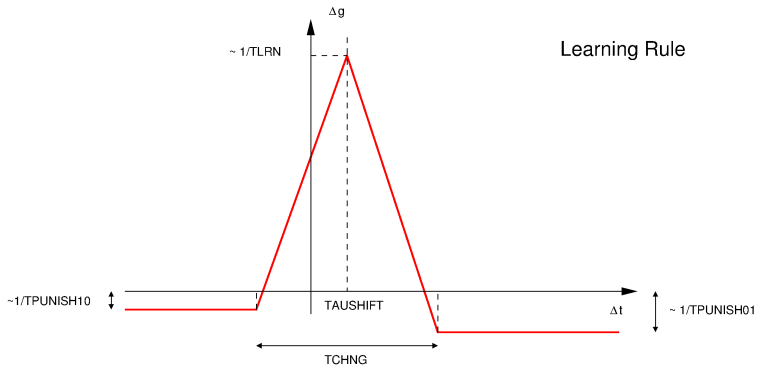
Static Public Member Functions

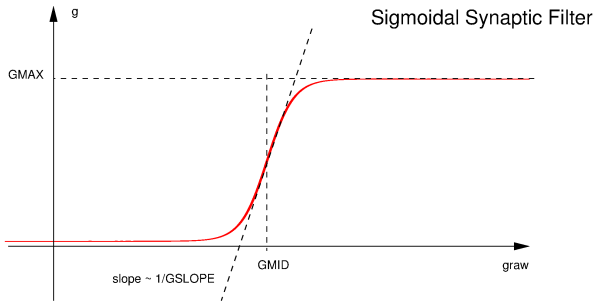
- static const PiecewiseSTDP * getInstance ()

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;
```

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 StringVec WeightUpdateModels::PiecewiseSTDP::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.

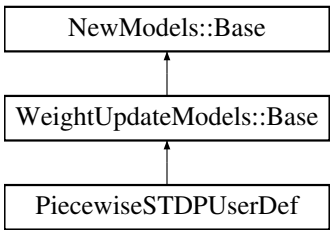
Reimplemented from WeightUpdateModels::Base.

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 StringVec PiecewiseSTDPUserDef::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:

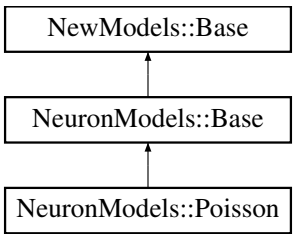
- MBody_userdef.cc

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 StringVec NeuronModels::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::~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:

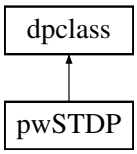
- postSynapseModels.h
- postSynapseModels.cc

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) }
- 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

N

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 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:

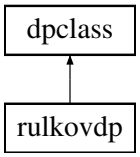
- randomGen.h
- randomGen.cc

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:

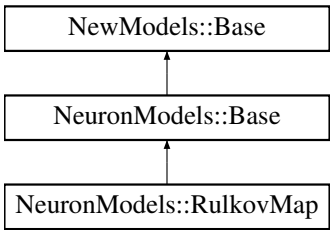
- neuronModels.h

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:

- `V` - the membrane potential
- `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 α= 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 β= 2.64 $\mathrm{M}\Omega$.

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 StringVec 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 StringPairVec NeuronModels::RulkovMap::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 recordingIdx)
- FILE * openRecordingFile (UINT recordingIndex)
- void applyLearningRuleSynapses (float *synapsesPNAN)
- void initialiseInputData ()
- void load_VR_data ()
- void setCorrectClass (UINT recordingIdx)
- UINT getClassCluster (UINT anIdx)
- 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 *recordingIdx*)

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 *anIdx*)

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 ( )

16.44.4.31 bool Schmuker2014_classifier::loadArrayFromTextFile (string *path,* void * *array,* string *delim,* UINT *arrayLen,* 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 *recordingIdx*)

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

16.44.5.1 string Schmuker2014_classifier::cacheDir

16.44.5.2 unsigned int* Schmuker2014_classifier::classLabel

16.44.5.3 bool Schmuker2014_classifier::clearedDownDevice

16.44.5.4 unsigned int* Schmuker2014_classifier::clusterSpikeCountAN

16.44.5.5 UINT Schmuker2014_classifier::correctClass

16.44.5.6 unsigned int Schmuker2014_classifier::countAN

16.44.5.7 unsigned int Schmuker2014_classifier::countPN

16.44.5.8 unsigned int Schmuker2014_classifier::countPNAN

16.44.5.9 unsigned int Schmuker2014_classifier::countRN

16.44.5.10 uint64_t* Schmuker2014_classifier::d_inputRates

16.44.5.11 double Schmuker2014_classifier::d_maxRandomNumber

16.44.5.12 string Schmuker2014_classifier::datasetName

16.44.5.13 unsigned int* Schmuker2014_classifier::individualSpikeCountPN

16.44.5.14 uint64_t* Schmuker2014_classifier::inputRates

16.44.5.15 unsigned int Schmuker2014_classifier::inputRatesSize

16.44.5.16 FILE* Schmuker2014_classifier::log

16.44.5.17 NNmodel Schmuker2014_classifier::model

16.44.5.18 string Schmuker2014_classifier::outputDir

16.44.5.19 unsigned int* Schmuker2014_classifier::overallWinnerSpikeCountAN

16.44.5.20 float Schmuker2014_classifier::param_CONNECTIVITY_AN_AN

16.44.5.21 float Schmuker2014_classifier::param_CONNECTIVITY_PN_AN

16.44.5.22 float Schmuker2014_classifier::param_CONNECTIVITY_PN_PN

16.44.5.23 float Schmuker2014_classifier::param_CONNECTIVITY_RN_PN

16.44.5.24 float Schmuker2014_classifier::param_GLOBAL_WEIGHT_SCALING

16.44.5.25 UINT Schmuker2014_classifier::param_MAX_FIRING_RATE_HZ

16.44.5.26 float Schmuker2014_classifier::param_MAX_WEIGHT_PN_AN

16.44.5.27 UINT Schmuker2014_classifier::param_MIN_FIRING_RATE_HZ

16.44.5.28 float Schmuker2014_classifier::param_MIN_WEIGHT_PN_AN

16.44.5.29 float Schmuker2014_classifier::param_PLASTICITY_INTERVAL_MS

16.44.5.30 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::uniqueRunId

16.44.5.40 float* Schmuker2014_classifier::vrData

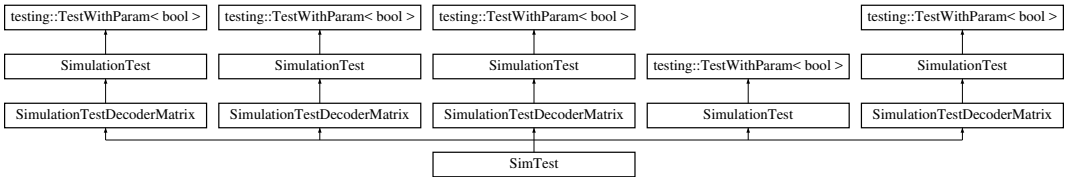
16.44.5.41 int Schmuker2014_classifier::winningClass

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

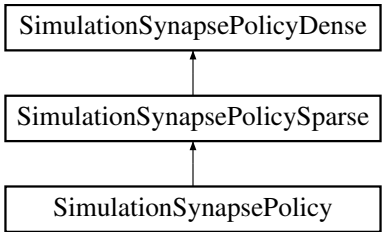
16.47.1.1 void SimulationNeuronPolicyPreVar::Init ( ) [inline]

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

- simulation_neuron_policy_pre_var.h

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]

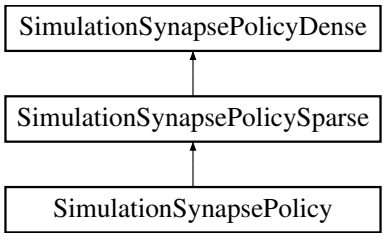
The documentation for this class was generated from the following file:

- 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]

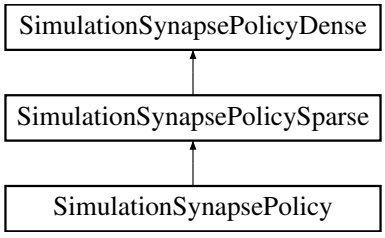
The documentation for this class was generated from the following file:

- 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]

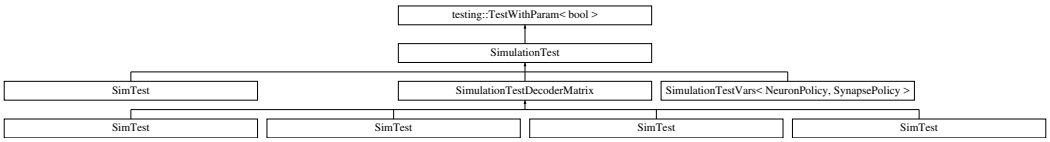
The documentation for this class was generated from the following file:

- 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]

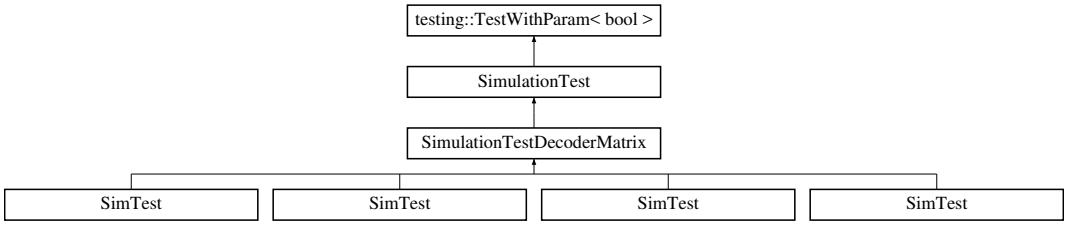
The documentation for this class was generated from the following file:

- 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]

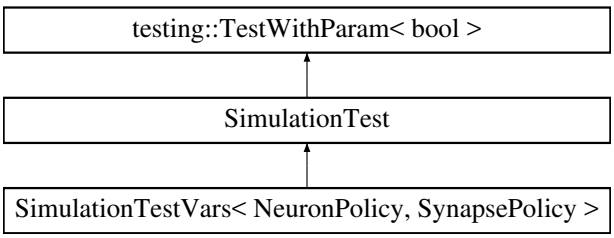
The documentation for this class was generated from the following file:

- 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.

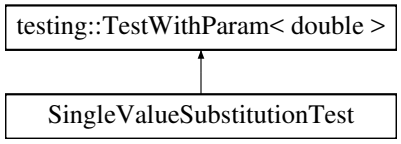
16.54.1.2 template<typename NeuronPolicy , typename SynapsePolicy > template<typename UpdateFn > float SimulationTestVars< NeuronPolicy, SynapsePolicy >::Simulate (UpdateFn *update*) [inline], [protected]

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 * indInG
- 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::indInG

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:

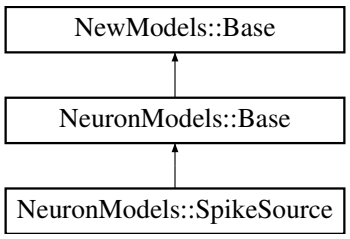
- sparseProjection.h

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:

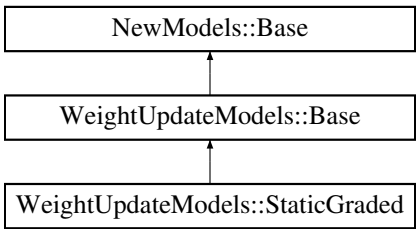
- newNeuronModels.h

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_{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:

```
$(addtoinSyn) = $(g)* tanh(($(V_pre)-($(Epre)))*DT*2/$(Vslope));
$(updatelinsyn);
```

`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 StringVec WeightUpdateModels::StaticGraded::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.

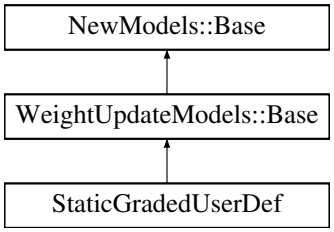
Reimplemented from NewModels::Base.

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 typedef NewModels::ValueBase< 1 > StaticGradedUserDef::VarValues

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.

Reimplemented from WeightUpdateModels::Base.

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 StringVec 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:

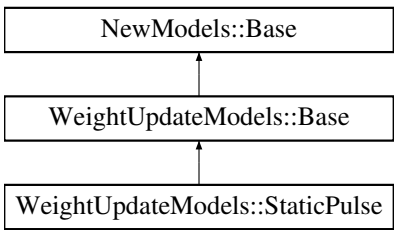
- MBody_userdef.cc

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.

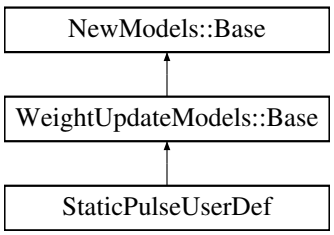
Reimplemented from NewModels::Base.

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 > &psInitVals, NeuronGroup *srcNeuronGroup, NeuronGroup *trgNeuronGroup)
- 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 > & *wuInitVals,* 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::getPaddedKernelIDRange ( ) 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>& SynapseGroup::getPSInitVals ( ) const [inline]

16.64.3.15 const **PostsynapticModels::Base*** SynapseGroup::getPSModel ( ) const [inline]

16.64.3.16 const std::vector<double>& SynapseGroup::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>& SynapseGroup::getWUDerivedParams ( ) 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*)

16.64.3.27 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.

16.64.3.30 bool SynapseGroup::isSpikeEventRequired ( ) const [inline]

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::~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:

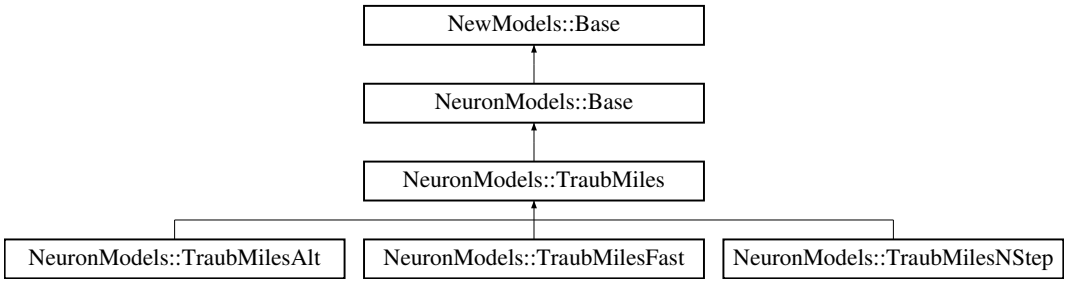
- SynDelaySim.h
- SynDelaySim.cc

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:

$$\begin{aligned} C\frac{dV}{dt} &= -I_{\mathrm{Na}} - I_K - I_{\mathrm{leak}} - I_M - I_{i,DC} - I_{i,\mathrm{syn}} - I_i, \\ I_{\mathrm{Na}}(t) &= g_{\mathrm{Na}} m_i(t)^3 h_i(t)(V_i(t) - E_{\mathrm{Na}}) \\ I_{\mathrm{K}}(t) &= g_{\mathrm{K}} n_i(t)^4 (V_i(t) - E_{\mathrm{K}}) \\ \frac{dy(t)}{dt} &= \alpha_y(V(t))(1 - y(t)) - \beta_y(V(t))y(t), \end{aligned}$$

where $y_i = m, h, n$, and

$$\begin{aligned} \alpha_n &= 0.032(-50 - V)/\big(\exp((-50 - V)/5) - 1\big) \\ \beta_n &= 0.5\exp((-55 - V)/40) \\ \alpha_m &= 0.32(-52 - V)/\big(\exp((-52 - V)/4) - 1\big) \\ \beta_m &= 0.28(25 + V)/\big(\exp((25 + V)/5) - 1\big) \\ \alpha_h &= 0.128\exp((-48 - V)/18) \\ \beta_h &= 4/\big(\exp((-25 - V)/5) + 1\big). \end{aligned}$$

and typical parameters are $C = 0.143$ nF, $g_{\mathrm{leak}} = 0.02672$ μS, $E_{\mathrm{leak}} = -63.563$ mV, $g_{\mathrm{Na}} = 7.15$ μS, $E_{\mathrm{Na}} = 50$ mV, $g_{\mathrm{K}} = 1.43$ μS, $E_{\mathrm{K}} = -95$ 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^2)
- `ENa` - Na equi potential in mV
- `gK` - K conductance in 1/(mOhms $*$ cm^2)
- `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^2

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 `lDT= 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::TraubMilesFast.

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:

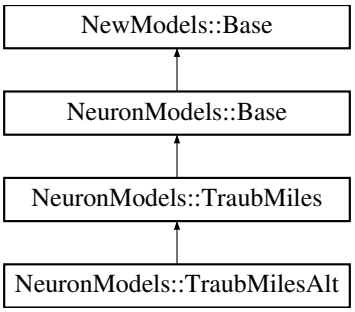
- newNeuronModels.h

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:

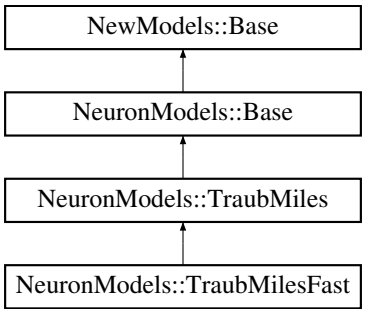
- newNeuronModels.h

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:

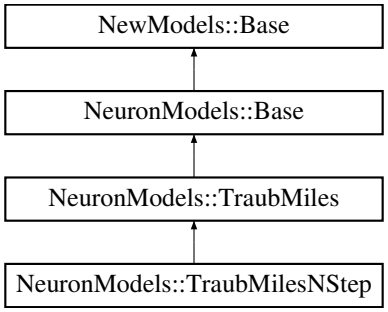
- newNeuronModels.h

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 typedef NewModels::ValueBase< 4 > NeuronModels::TraubMilesNStep::VarValues

16.69.3 Member Function Documentation

16.69.3.1 static const NeuronModels::TraubMilesNStep* NeuronModels::TraubMilesNStep::getInstance ( ) [inline], [static]

16.69.3.2 virtual StringVec NeuronModels::TraubMilesNStep::getParamNames ( ) 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.
- 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> weightUpdateModel::varNames

Names of the variables in the postsynaptic model.

16.72.3.17 vector<string> weightUpdateModel::varTypes

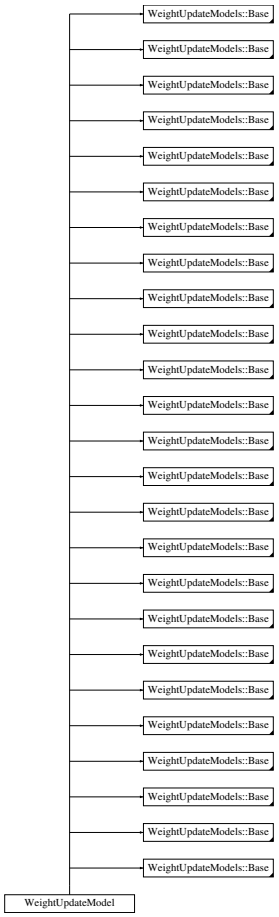
Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

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
- 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< 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< 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 ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- static const WeightUpdateModel $*$ getInstance ()
- 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**

16.73.1.3 typedef **NewModels::ValueBase**$<$ **0** $>$ **WeightUpdateModel::ParamValues**

16.73.1.4 typedef **NewModels::ValueBase**$<$ **0** $>$ **WeightUpdateModel::ParamValues**

16.73.1.5 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**

16.73.1.8 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

16.73.1.40 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.

Reimplemented from WeightUpdateModels::Base.

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.

Reimplemented from WeightUpdateModels::Base.

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; functions 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 StringVec WeightUpdateModel::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.73.2.47 virtual StringVec WeightUpdateModel::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.73.2.48 virtual StringVec WeightUpdateModel::getParamNames ( ) const [inline], [virtual]

Gets names of of (independent) model parameters.

Reimplemented from NewModels::Base.

16.73.2.49 virtual StringVec WeightUpdateModel::getParamNames ( ) 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.

Reimplemented from WeightUpdateModels::Base.

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.

16.73.2.67 virtual std::string WeightUpdateModel::getSynapseDynamicsSuppportCode ( ) const [inline], [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.

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 StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.70 virtual StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.75 virtual StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.76 virtual StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.83 virtual StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.84 virtual StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.85 virtual StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.86 virtual StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

Reimplemented from NewModels::Base.

16.73.2.89 virtual StringPairVec WeightUpdateModel::getVars ( ) 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 StringPairVec WeightUpdateModel::getVars ( ) const [inline], [virtual]

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

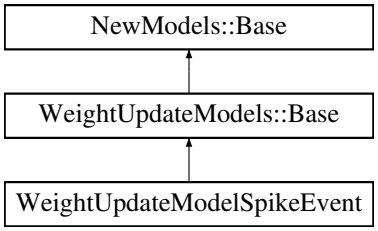
Reimplemented from NewModels::Base.

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

- 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 StringVec WeightUpdateModelSpikeEvent::getParamNames ( ) 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

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; 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

17.19.2.8 #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 Variable Documentation

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

17.22.2.7 #define TOTAL_TME 5000.0

17.22.3 Variable Documentation

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 &preIdx, const string &postIdx, 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 & *preIdx*, const string & *postIdx*, 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
preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	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 >::min(), 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 BaseIter >
  PairKeyConstIter< BaseIter > GetPairKeyConstIter (BaseIter 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 NameIter >
  void name_substitutions (string &code, const string &prefix, NameIter namesBegin, NameIter 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 NameIter >
  void value_substitutions (string &code, NameIter namesBegin, NameIter 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 BaseIter > PairKeyConstIter<BaseIter> GetPairKeyConstIter (BaseIter *iter*) [inline]

17.25.1.4 template<typename NameIter > void name_substitutions (string & *code,* const string & *prefix,* NameIter *namesBegin,* NameIter *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 NameIter > void value_substitutions (string & *code,* NameIter *namesBegin,* NameIter *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 <sstream>
#include <vector>
```

Classes

- class CodeHelper

Macros

- #define SAVEP(X) "(" << X << ")"
- #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 argv 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

17.31 extra_neurons.h File Reference

Functions

- n varNames clear ()
- n varNames push_back ("V")

- 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")

17.31.1.16 n pNames push_back ("max_t_NB")

17.31.2 Variable Documentation

17.31.2.1 n simCode

Initial value:

```
= " \
            $(V) = -1000000; \
            if ($(__regime_val)==1) { \n \
$(V_NB) += (Isyn_NB/$(Cm_NB)+($(VRest_NB)-$(V_NB))/$(TAUm_NB))*DT; \n \
                  if ($(V_NB)>$(VThresh_NB)) { \n \
$(V_NB) = $(VReset_NB); \n \
$(tSpike_NB) = t; \n \
                  $(V) = 100000; \
$(__regime_val) = 2; \n \
} \n \
} \n \
if ($(__regime_val)==2) { \n \
if (t-$(tSpike_NB) > $(tRefrac_NB)) { \n \
$(__regime_val) = 1; \n \
} \n \
} \n \
"
```

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:

```
= " \
  0; \n \
              float Isyn_NB = 0; \n \
               { \n \
             float v_PS = lV_NB; \n \
               float g_in_PS = $(inSyn); \
$(g_PS) = $(g_PS)+g_in_PS; \n \
Isyn_NB += ($(g_PS)*($(E_PS)-v_PS)); \n \
          } \n \
"
```

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 Documentation

17.37.2.1 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) | (0x80000000 >> (i)))

  Set the bit at the specified position i in x to 1.
- #define delB(x, i) x= ((x) & (∼(0x80000000 >> (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) & (0x80000000 >> (i)))

Extract the bit at the specified position i from x.

17.41.2.2 #define delB(*x, i*) x= ((x) & (~(0x80000000 >> (i))))

Set the bit at the specified position i in x to 0.

17.41.2.3 #define setB(*x, i*) x= ((x) | (0x80000000 >> (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 > indInG_ee
- std::vector< unsigned int > ind_ee
- double * garray_ei
- std::vector< double > g_ei
- std::vector< unsigned int > indInG_ei
- std::vector< unsigned int > ind_ei
- double * garray_ie
- std::vector< double > g_ie
- std::vector< unsigned int > indInG_ie
- std::vector< unsigned int > ind_ie
- double * garray_ii
- std::vector< double > g_ii
- std::vector< unsigned int > indInG_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 > & *v*)

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> indInG_ee

17.44.3.17 std::vector<unsigned int> indInG_ei

17.44.3.18 std::vector<unsigned int> indInG_ie

17.44.3.19 std::vector<unsigned int> indInG_ii

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 Documentation

17.53.2.1 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 <MODEL>
#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 <cmath>
#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 (generateCPU.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 CUDA 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 inititate 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↩ 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()", "copyStateToDevice()", "copyStateFromDevice()", "copySpikesFromDevice()", "copySpike↩ NFromDevice()" and "stepTimeGPU()". 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()", "copyStateToDevice()", "copyStateFromDevice()", "copySpikesFromDevice()", "copySpike↩NFromDevice()" and "stepTimeGPU()". 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

17.63 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 generate_random_gpuInput_xorwow< float > (curandState *state, float *result, int sizeof↩Result, float Rstrength, float Rshift, dim3 sGrid, dim3 sThreads)
- template void generate_random_gpuInput_xorwow< 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_gpuInput_xorwow (curandState * *state*, T * *result*, int *sizeofResult*, T *Rstrength*, T *Rshift*)

17.63.1.2 template<typename T > void generate_random_gpuInput_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_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)

Variables

- const int BlkSz = 256

17.64.1 Function Documentation

17.64.1.1 template<typename T > __global__ void generate_random_gpuInput_xorwow (curandState * *state*, T * *result*, int *sizeofResult*, T *Rstrength*, T *Rshift*)

17.64.1.2 template<typename T > void generate_random_gpuInput_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 theDevice

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)
- 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 initI (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 initI (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*(3+2^^8)-1) = 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 lzh_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 lzh_sparse.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "stringUtils.h"
#include <vector>
#include "sizes.h"
```

Classes

- class MyIzhikevichVariable

Functions

- IMPLEMENT_MODEL (MyIzhikevichVariable)
- 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
- MyIzhikevichVariable::VarValues IzhExc_ini (-65.0, 0.0, 0.02, 0.2,-65.0, 8.0, 0.0)
- MyIzhikevichVariable::VarValues IzhInh_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 (MyIzhikevichVariable)

17.74.1.2 void modelDefinition (NNmodel & *model*)

17.74.2 Variable Documentation

17.74.2.1 MyIzhikevichVariable::VarValues IzhExc_ini(-65.0,0.0,0.02,0.2,-65.0,8.0,0.0)

17.74.2.2 MyIzhikevichVariable::VarValues IzhInh_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 meanInpInh = 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 SynIzh_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 lzh_sparse_model.h File Reference

Classes

- class classIzh

17.77 lzh_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 cpontains 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 cpontains 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 cpontains the methods for running the MBody1 example model.

17.89 map_classol.h File Reference

```
#include <stdint.h>
```

Classes

- class classol

  This class cpontains 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 cpontains 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*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

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 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*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

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)

17.92.2.4 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)

17.92.2.5 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"
```

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)

17.93.1 Function Documentation

17.93.1.1 void modelDefinition (NNmodel & *model*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

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 myPOI_ini(-60.0,0,-10.0)

17.93.2.9 NeuronModels::Poisson::ParamValues myPOI_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*)

This function defines the MBody_delayedSyn model, and it is a good example of how networks should be defined.

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)

17.94.3.3 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)

17.94.3.4 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 myPOI_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*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

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 myPOI_ini(-60.0,0,-10.0)

17.95.3.8 NeuronModels::Poisson::ParamValues myPOI_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

Functions

- 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 myPOI_ini(-60.0,0,-10.0)

17.96.4.10 NeuronModels::Poisson::ParamValues myPOI_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"
```

Functions

- 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:

```
= {
    0.0
}
```

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:

```
= {
    0.0
}
```

17.98.2.2 double synapses_ini[1]

Initial value:

```
= {
    1.0
}
```

17.99 model.cc File Reference

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

Functions

- 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:

```
= {
    0.0
}
```

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:

```
= {
    0.0
}
```

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"
```

Functions

- 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:

```
= {
    0.0,
    0.0
}
```

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
}
```

17.105.2.2 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

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,
    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:

```
= {
    0.0,
    0.0
}
```

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,
    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:

```
= {
    0.0
}
```

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:

```
= {
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

17.111.2.2 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

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,
    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:

```
= {
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

17.114.2.2 double neuron_p[1]

Initial value:

```
={
    1.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

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:

```
= {
    0.0
}
```

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:

```
= {
    0.0
}
```

17.119.2.3 double synapses_p[1]

Initial value:

```
= {
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

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:

```
= {
    0.0,
    0.0
}
```

17.123.2.2 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

17.123.2.3 double synapses_p[1]

Initial value:

```
= {
    0.0
}
```

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,
    0.0
}
```

17.124.2.2 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

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

17.125.2.1 double neuron_ini[2]

Initial value:

```
= {
    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:

```
= {
    0.0,
    0.0
}
```

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*)

17.130 model_new.cc File Reference

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

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

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

Classes

- class Neuron
- class WeightUpdateModel

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

variable attaching the name "TRAUBMILES_SAFE"

17.167 newModels.h File Reference

```
#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)

17.169 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_CURRENT_CONVERTER_CODE(CURRENT_CONVERTER_CODE) virtual std::string get↩ CurrentConverterCode() const{ return CURRENT_CONVERTER_CODE; }
- #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; }
- #define SET_NEEDS_POST_SPIKE_TIME(POST_SPIKE_TIME_REQUIRED) virtual bool isPostSpike↩ TimeRequired() const{ return POST_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 MyIzhikevich

Functions

- IMPLEMENT_MODEL (MyIzhikevich)
- void modelDefinition (NNmodel &model)

Variables

- MyIzhikevich::ParamValues exIzh_p (0.02, 0.2,-65, 6, 4.0)
- MyIzhikevich::VarValues exIzh_ini (-65,-20)

17.174.1 Function Documentation

17.174.1.1 IMPLEMENT_MODEL (**MyIzhikevich**)

17.174.1.2 void modelDefinition (**NNmodel** & *model*)

17.174.2 Variable Documentation

17.174.2.1 **MyIzhikevich::VarValues** exIzh_ini(-65,-20)

17.174.2.2 **MyIzhikevich::ParamValues** exIzh_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++)
- 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 PoissonIzh-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 PoissonIzh-model.h File Reference

```
#include <stdint.h>
```

Classes

- class classol

  This class cpontains 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::Izhikevich::ParamValues exIzh_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 myPOI_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 PoissonIzh_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; 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 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 <cstring>
#include <time.h>
#include "Schmuker2014_classifier.h"
#include "Schmuker_2014_classifier_CODE/definitions.h"
#include "stringUtils.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:

```
case I:              \
          allocatesyn##I(10); \
          theC= &Csyn##I;     \
          break;
```

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_TEST_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:

```
for(int i_##__LINE__ = 0; i_##__LINE__ < COUNT; i++)  \
    {                                                 \
        ARRAY_NAME[i_##__LINE__] = VAR_PREFIX##i_##__LINE__; \
    }
```

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 _NInh 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

17.201 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

17.201.1 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 _NIzh 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 * *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)

17.209.2.4 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 * *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 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 SparseProjection 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_gRNPN, 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 SparseProjection 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::neuronOutputInit (std::ostream &os, const NeuronGroup &ng, const std::string &devPrefix)
- 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 NameIterCtx< NewModels::Base::StringPairVec > VarNameIterCtx
- typedef NameIterCtx< NewModels::Base::DerivedParamVec > DerivedParamNameIterCtx
- typedef NameIterCtx< NewModels::Base::StringPairVec > ExtraGlobalParamNameIterCtx

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::weightUpdateThresholdCondition (std::string &eCode, const SynapseGroup &sg, const DerivedParamNameIterCtx &nmDerivedParams, const ExtraGlobalParamNameIterCtx &nm↩ ExtraGlobalParams, const string &preIdx, const string &postIdx, const string &devPrefix, 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 NameIterCtx<NewModels::Base::DerivedParamVec> DerivedParamNameIterCtx

17.214.1.2 typedef NameIterCtx<NewModels::Base::StringPairVec> ExtraGlobalParamNameIterCtx

17.214.1.3 typedef NameIterCtx<NewModels::Base::StringPairVec> VarNameIterCtx

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), SynapseMatrixWeight::INDIVIDUAL = (1 << 4) }

- enum SynapseMatrixType : unsigned int { SynapseMatrixType::SPARSE_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::SPARSE) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL), SynapseMatrixType::SPARSE_INDIVIDUALG = 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::DENSE) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL), SynapseMatrixType::BITMASK_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::BITMASK) | 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 vector<weightUpdateModel> weightUpdateModels

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 MyIzhikevich

Functions

- IMPLEMENT_MODEL (MyIzhikevich)
- void modelDefinition (NNmodel &model)

17.221.1 Function Documentation

17.221.1.1 IMPLEMENT_MODEL (MyIzhikevich)

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 "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)

17.224.1 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 Function Documentation

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 > SimTest

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, SimulationSynapsePolicyDense> 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 > SimTest

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, SimulationSynapsePolicyDense> 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> SimTest

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 > SimTest

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, SimulationSynapsePolicyDense> 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 > SimTest

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, SimulationSynapsePolicySparse> 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 > SimTest

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, SimulationSynapsePolicyDense> 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 > SimTest

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, SimulationSynapsePolicySparse> 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

- typedef SimulationTestVars< SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > SimTest

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, SimulationSynapsePolicyDense> 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 > SimTest

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, SimulationSynapsePolicySparse> 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

- typedef SimulationTestVars< SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > SimTest

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, SimulationSynapsePolicyDense> 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 theSize (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) | (0x80000000 >> (i)))

  Set the bit at the specified position i in x to 1.
- #define delB(x, i) x= ((x) & (~(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) & (0x80000000 >> (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);                                              \
      }                                                                  \
  }
```

17.256.2.5 #define delB(*x, i*) x= ((x) & (~(0x80000000 >> (i))))

Set the bit at the specified position i in x to 0.

17.256.2.6 #define setB(*x, i*) x= ((x) | (0x80000000 >> (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 theSize (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

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