1 GeNN Documentation 1

1 GeNN Documentation

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

GeNN is currently developed and maintained by

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

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

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

Note

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

Next

2 Installation

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

2.1 Downloading a release

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

2.2 Obtaining a Git snapshot

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

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

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

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

```
git checkout development
```

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

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

2.3 Installing GeNN

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

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

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

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

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

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

in your login script, or in windows,

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

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

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

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

```
\verb"setx CUDA\_PATH C:\path\to\cuda"
```

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

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

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

common[win|gnu].mk.

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

If you are using GeNN in Windows, the Visual Studio development environment must be set up within every instance of the CMD.EXE command window used. One can open an instance of CMD.EXE with the development environment already set up by navigating to Start - All Programs - visual studio - tools - visual studio native command prompt. You may wish to create a shortcut for this tool on the desktop, for convenience. Note that all C++ tools should have been installed during the Visual Studio install process for this to work. Alternatively one can use the make. 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 Ge-NN on the background of this philosophy, it comes with a number of complete examples containing both the model description code that is used by GeNN for code generation and the "user side code" to run the generated model and safe the results. Running these complete examples should be achievable in a few minutes. The necessary steps are described below.

3.1 Running an Example Model in Unix

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

make

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

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

Possible options:

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

To compile $generate_run.cc$, navigate to the $userproject/MBody1_project$ directory and type

make

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

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

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

The tool generate_run will generate connectivity matrices for the model $\mathtt{MBody1}$ and store them into files, compile and run the model on an automatically chosen GPU, using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument 1 with the device number of the desired GPU plus 2, e.g., 2 for GPU 0. All input and output files will be prefixed with $\mathtt{test1}$ and will be created in a sub-directory with the name $\mathtt{test1_output}$. More about the DEBUG flag in the debugging section . The parameter \mathtt{FLOAT} will run the model in float (single precision floating point), using \mathtt{DOUBLE} would use double precision. The REU \leftarrow SE parameter regulates whether previously generated files for connectivity and input should be reused (1) or files should be generated anew (0).

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

3.2 Running an Example Model in Windows

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

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

Then type

```
nmake /f WINmakefile
```

to compile a number of tools that are used by the example projects to generate connectivity and inputs to model networks. Then navigate to the userproject/MBody1_project directory.

```
cd ..\MBody1_project
```

By typing

```
nmake /f WINmakefile
```

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

To invoke generate_run.exe type, e.g.,

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

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

The tool <code>generate_run.exe</code> will generate connectivity matrices for the model <code>MBody1</code> and store them into files, compile and run the model on an automatically chosen GPU, using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument 1 with the device number of the desired GPU plus 2, e.g., 2 for GPU 0. All input and output files will be prefixed with <code>test1</code> and will be created in a sub-directory with the name <code>test1_output</code>. More about the DEBUG flag in the <code>debugging section</code>. The parameter <code>FLOAT</code> will run the model in float (single precision floating point), using <code>DOUBLE</code> would use double precision. The REU \leftarrow SE parameter regulates whether previously generated files for connectivity and input should be reused (1) or files should be generated anew (0).

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

3.3 How to use GeNN for New Projects

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

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

- 1. Either using tools (programs) to generate connectivity and input matrix files, which are then loaded into the user's simulation code at runtime, or generating these matrices directly inside the user's simulation code.
- 2. Building the source code of a model simulation using <code>genn-buildmodel.sh</code> (On Linux or Mac) or <code>genn-buildmodel.bat</code> (on Windows). In the example of the MBody1_project this entails writing neuron numbers into <code>userproject/include/sizes.h</code>, and executing

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

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

3. Compiling the generated code, found in model/MBody1_CODE/, by calling:

```
make clean all
```

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

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

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

Note

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

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

3.4 Defining a New Model in GeNN

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

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

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

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

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

Note

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

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

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

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

Note

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

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

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

4 Examples 7

Note

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

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

- 3. The programmer defines their own "user-side" modeling code similar to the code in userproject/M← Body1_project/model/map_classol.* and userproject/MBody1_project/model/classol← sim.*. In this code,
 - a) They define the connectivity matrices between neuron groups. (In the MBody1 example those are read from files). Refer to the User Manual for the required format of connectivity matrices for dense or sparse connectivities.
 - b) They define input patterns (e.g. for Poisson neurons like in the MBody1 example) or individual initial values for neuron and / or synapse variables.

Note

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

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

Note

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

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

Previous | Top | Next

4 Examples

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

4.1 Single compartment Izhikevich neuron(s)

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

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

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' \model oppulation>='_ee'). These name conventions
are not part of the core GeNN definitions and it is the privilege (or burden)
of the user to find their own in their own versions of 'generate_run'.
Another example of an invocation would be:
generate_run.exe 0 10000 1000 1 outdir Izh_sparse 1.0 FTYPE=DOUBLE DEBUG=0 CPU_ONLY=1
for Windows users, or:
./generate_run 0 10000 1000 1 outdir Izh_sparse 1.0 FTYPE=DOUBLE DEBUG=0 CPU_ONLY=1
for Linux, Mac and other UNIX users.
Izhikevich neuron model: [1]
```

4.4 Izhikevich network with delayed synapses

Izhikevich network with delayed synapses

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

```
COMPILE (WINDOWS)
```

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

```
then compile the project by typing:

nmake /f WINmakefile

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

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

then compile the project by typing:

make

USAGE
-----

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

Izhikevich neuron model: [1]
```

4.5 Insect olfaction model

```
Locust olfactory system (Nowotny et al. 2005)
This project implements the insect olfaction model by Nowotny et
al. that demonstrates self-organized clustering of odours in a
simulation of the insect antennal lobe and mushroom body. As provided
the model works with conductance based Hodgkin-Huxley neurons and
several different synapse types, conductance based (but pulse-coupled)
excitatory synapses, graded inhibitory synapses and synapses with a
simplified STDP rule. This example project contains a helper executable called "generate_run", which also
prepares additional synapse connectivity and input pattern data, before compiling and
executing the model.
To compile it, navigate to genn/userproject/MBody1_project and type:
nmake /f WINmakefile
for Windows users, or:
make
for Linux, Mac and other UNIX users.
USAGE
\verb|generate_run| < 0 (CPU) / 1 (GPU) / n (GPU n-2) > < nAL > < nKC > < nLH > < nDN > < gScale > < DIR > < MODEL > < nLH > < nCD > < nLH > < nDN > < gScale > < DIR > < MODEL > < nLH > < nDN > < nDN > < nLH > < nDN 
Mandatory parameters:
CPU/GPU: Choose whether to run the simulation on CPU ('0'), auto GPU ('1'), or GPU (n-2) ('n').
```

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.

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

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

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

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

nLH: Number of lateral horn interneurons, implementing gain control

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

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

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

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

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

This examples recapitulates the exact same model as MBodyl_project,

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

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

4.7 Insect Olfaction Model using INDIVIDUALID connectivity scheme

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

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

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

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

4.8 Insect Olfaction Model using delayed synapses

doi:10.1007/s00422-005-0019-7

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

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

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

4.9 Voltage clamp simulation to estimate Hodgkin-Huxley parameters

Genetic algorithm for tracking parameters in a HH model cell

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

5 Release Notes 17

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

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

Previous | Top | Next

5 Release Notes

Release Notes for GeNN v2.2.3

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

User Side Changes

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

Bug fixes:

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

Release Notes for GeNN v2.2.2

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

User Side Changes

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

Bug fixes:

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

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.

5 Release Notes 19

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 \leftarrow 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).

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

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

Note

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

- 4. The new convenience macros spikeCount_XX and spike_XX where "XX" is the name of the neuron group are now also available for events: spikeEventCount_XX and spikeEvent_XX. They access the values for the current time step even if there are synaptic delays and spikes events are stored in circular queues.
- 5. The old buildmodel.[sh|bat] scripts have been superseded by new genn-buildmodel.[sh|bat] scripts. These scripts accept UNIX style option switches, allow both relative and absolute model file paths, and allow the user to specify the directory in which all output files are placed (-o <path>). Debug (-d), CPU-only (-c) and show help (-h) are also defined.

6. We have introduced a CPU-only "-c" genn-buildmodel switch, which, if it's defined, will generate a GeNN version that is completely independent from CUDA and hence can be used on computers without CUDA installation or CUDA enabled hardware. Obviously, this then can also only run on CPU. CPU only mode can either be switched on by defining CPU_ONLY in the model description file or by passing appropriate parameters during the build, in particular

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

- 7. The new genn-buildmodel "-o" switch allows the user to specify the output directory for all generated files the default is the current directory. For example, a user project could be in '/home/genn_project', whilst the GeNN directory could be '/usr/local/genn'. The GeNN directory is kept clean, unless the user decides to build the sample projects inside of it without copying them elsewhere. This allows the deployment of GeNN to a read-only directory, like '/usr/local' or 'C:\Program Files'. It also allows multiple users i.e. on a compute cluster to use GeNN simultaneously, without overwriting each other's code-generation files, etcetera.
- 8. The ARM architecture is now supported e.g. the NVIDIA Jetson development platform.
- 9. The NVIDIA CUDA SM 5* (Maxwell) architecture is now supported.
- 10. An error is now thrown when the user tries to use double precision floating-point numbers on devices with architecture older than SM 13, since these devices do not support double precision.
- 11. All GeNN helper functions and classes, such as toString() and NNmodel, are defined in the header files at genn/lib/include/, for example stringUtils.h and modelSpec.h, which should be individually included before the functions and classes may be used. The functions and classes are actually implementated in the static library genn\lib\lib\genn.lib\(Windows)\) or genn/lib/lib/libgenn.a (Mac, Linux), which must be linked into the final executable if any GeNN functions or classes are used.
- 12. In the modelDefinition() file, only the header file modelSpec.h should be included i.e. not the source file modelSpec.cc. This is because the declaration and definition of NNmodel, and associated functions, has been separated into modelSpec.h and modelSpec.cc, respectively. This is to enable NNmodel code to be precompiled separately. Henceforth, only the header file modelSpec.h should be included in model definition files!
- 13. In the modelDefinition () file, DT is now preferrably defined using model.setDT (<val>);, rather than #define DT <val>, in order to prevent problems with DT macro redefinition. For backward-compatibility reasons, the old #define DT <val> method may still be used, however users are advised to adopt the new method.
- 14. In preparation for multi-GPU support in GeNN, we have separated out the compilation of generated code from user-side code. This will eventually allow us to optimise and compile different parts of the model with different CUDA flags, depending on the CUDA device chosen to execute that particular part of the model. As such, we have had to use a header file definitions.h as the generated code interface, rather than the runner.cc file. In practice, this means that user-side code should include myModel_COD← E/definitions.h, rather than myModel_CODE/runner.cc. Including runner.cc will likely result in pages of linking errors at best!

Developer Side Changes

- Blocksize optimization and device choice now obtain the ptxas information on memory usage from a CUDA driver API call rather than from parsing ptxas output of the nvcc compiler. This adds robustness to any change in the syntax of the compiler output.
- 2. The information about device choice is now stored in variables in the namespace GENN_PREFERENCES. This includes chooseDevice, optimiseBlockSize, optimizeCode, debugCode, showPtxcInfo, 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.

5 Release Notes 21

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\<S\U004\SYNAPSENAME\>StateFromDevice and push\<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.

5 Release Notes 23

User Side Changes

1. Users are now required to call initGeNN () in the model definition function before adding any populations to the neuronal network model.

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

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

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

Note

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

- 6. Advanced users can now define variables as type scalar when introducing new neuron or synapse types. This will at the code generation stage be translated to the model's floating point type (ftype), float or double. This works for defining variables as well as in all code snippets. Users can also use the expressions SCALAR_MAX and SCALAR_MIN for FLT_MIN, FLT_MAX, DBL_MIN and DBL_MAX, respectively. Corresponding definitions of scalar, SCALAR_MIN and SCALAR_MAX are also available for user-side code whenever the code-generated file runner.cc has been included.
- 7. The example projects have been re-organized so that wrapper scripts of the <code>generate_run</code> type are now all located together with the models they run instead of in a common <code>tools</code> directory. Generally the structure now is that each example project contains the wrapper script <code>generate_run</code> and a <code>model</code> subdirectory which contains the model description file and the user side code complete with Makefiles for Unix and Windows operating systems. The generated code will be deposited in the <code>model</code> subdirectory in its <code>own modelname_CODE</code> folder. Simulation results will always be deposited in a new sub-folder of the main project directory.
- 8. The addSynapsePopulation (...) function has now more mandatory parameters relating to the introduction of separate weightupdate models (pre-synaptic models) and postynaptic models. The correct syntax for the addSynapsePopulation (...) can be found with detailed explanations in teh User Manual.
- We have introduced a simple performance profiling method that users can employ to get an overview over the differential use of time by different kernels. To enable the timers in GeNN generated code, one needs to declare

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

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

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

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

```
model.setTiming(TRUE);
```

in 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 <code>OUTNAME_output/OUTNAME.</code>
<code>timingprofile.</code>

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

- Introduction
- · Defining a network model
- · Neuron models
- · Synapse models

6.2 Introduction 25

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 <u>Defining neuron populations</u>). 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(name, num, type, para, ini);
```

where the arguments are:

- const string name: Unique name of the neuron population
- unsigned int num: number of neurons in the population
- unsigned int type: Type of the neurons, refers to either a standard type (see Neuron models) or userdefined type; this is an integer that indicates the position in the list of all neuron models where the model in question is stored.
- vector<double> para: Parameters of this neuron type

vector<double> ini: 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(name, sType, sConn, gType, delay, postSyn, preName, postName
   , sIni, sParam, postSynIni, postSynParam);
```

where the arguments are

- const string name: The name of the synapse population
- unsigned int sType: The type of synapse to be added. See Built-in Models below for the available predefined synapse types.
- unsigned int sConn: The type of synaptic connectivity. the options currently are "ALLTOALL", "DE⇔ NSE", "SPARSE" (see Connectivity types).
- unsigned int gType: The way how the synaptic conductivity g will be defined. Options are "IN

 DIVIDUALG", "GLOBALG", "INDIVIDUALID" (see LEARN1SYNAPSE (Learning Synapse with a Primitive
 Piece-wise Linear Rule)).
- unsigned int delay: Synaptic delay (in multiples of the simulation time step DT).
- unsigned int postSyn: Postsynaptic integration method. See Postsynaptic integration methods for predefined types.
- const string preName: Name of the (existing!) pre-synaptic neuron population.
- const string postName: Name of the (existing!) post-synaptic neuron population.
- vector<double> sIni: A vector of doubles containing initial values for the (pre-) synaptic variables.
- vector<double> sParam: A vector of double precision that contains parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored. For pre-defined synapse types the required parameters and their meaning are listed in NSYNAPSE (No Learning) below.
- vector<double> psIni: A vector of double precision numbers containing initial values for the postsynaptic model variables.
- vector<double> psPara: A vector of double precision numbers containing parameters fo the postsnaptic model.

Note

If the synapse conductance definition type is "GLOBALG" then the global value of the synapse conductances is taken from the initial value provided in sINI. (The function setSynapseG() from earlier versions of GeNN has been deprecated).

Synaptic updates can occur per "true" spike (i.e at one point per spike, e.g. after a threshold was crossed) or for all "spike type events" (e.g. all points above a given threshold). This is defined within each given synapse type.

Previous | Top | Next

6.4 Neuron models 27

6.4 Neuron models

There is a number of predefined models which can be chosen in the addNeuronGroup(...) function by their unique cardinal number, starting from 0. For convenience, C variables with readable names are predefined

- 0: MAPNEURON
- 1: POISSONNEURON
- 2: TRAUBMILES FAST
- 3: TRAUBMILES ALTERNATIVE
- 4: TRAUBMILES SAFE
- 5: TRAUBMILES_PSTEP
- 6: IZHIKEVICH
- 7: IZHIKEVICH V
- 8: SPIKESOURCE

Note

Ist is best practice to not depend on the unique cardinal numbers but use predefined names. While it is not intended that the numbers will change the unique names are guaranteed to work in all future versions of GeNN.

6.4.1 MAPNEURON (Map Neurons)

The MAPNEURON type is a map based neuron model based on [4] but in the 1-dimensional map form used in [3]:

$$V(t + \Delta t) = \begin{cases} V_{\text{spike}} \left(\frac{\alpha V_{\text{spike}}}{V_{\text{spike}} - V(t)\beta I_{\text{syn}}} + y \right) & V(t) \leq 0 \\ V_{\text{spike}} \left(\alpha + y \right) & V(t) \leq V_{\text{spike}} \left(\alpha + y \right) & V(t \leq V_{\text{spike}}) \end{cases}$$

$$-V_{\text{spike}} \quad \text{otherwise}$$

Note

The MAPNEURON type only works as intended for the single time step size of DT= 0.5.

The MAPNEURON type has 2 variables:

- preV the membrane potential at the previous time step

and it has 4 parameters:

- Vspike determines the amplitude of spikes, typically -60mV
- alpha determines the shape of the iteration function, typically 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 M Ω .

Note

The initial values array for the MAPNEURON type needs two entries for V and Vpre and the parameter array needs four entries for Vspike, alpha, y and beta, in that order.

6.4.2 POISSONNEURON (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 POISSONNEURON 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.

6.4.3 TRAUBMILES FAST (Hodgkin-Huxley neurons with Traub & Miles algorithm)

This conductance based model has been taken from [5] and can be described by the equations:

$$C\frac{dV}{dt} = -I_{Na} - I_{K} - I_{leak} - I_{M} - I_{i,DC} - I_{i,syn} - I_{i},$$

$$I_{Na}(t) = g_{Na}m_{i}(t)^{3}h_{i}(t)(V_{i}(t) - E_{Na})$$

$$I_{K}(t) = g_{K}n_{i}(t)^{4}(V_{i}(t) - E_{K})$$

$$\frac{dy(t)}{dt} = \alpha_{y}(V(t))(1 - y(t)) - \beta_{y}(V(t))y(t),$$

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

$$\alpha_n = 0.032(-50-V)/(\exp((-50-V)/5)-1)$$

 $\beta_n = 0.5 \exp((-55-V)/40)$
 $\alpha_m = 0.32(-52-V)/(\exp((-52-V)/4)-1)$

6.4 Neuron models 29

$$\beta_m = 0.28(25+V)/(\exp((25+V)/5)-1)$$
 $\alpha_h = 0.128 \exp((-48-V)/18)$
 $\beta_h = 4/(\exp((-25-V)/5)+1).$

and typical parameters are C=0.143 nF, $g_{\rm leak}=0.02672~\mu$ S, $E_{\rm leak}=-63.563$ mV, $g_{\rm Na}=7.15~\mu$ S, $E_{\rm Na}=50$ mV, $g_{\rm K}=1.43~\mu$ S, $E_{\rm K}=-95$ mV.

It has 4 variables:

- V membrane potential E
- m probability for Na channel activation m
- h probability for not Na channel blocking h
- n probability for K channel activation n

and 7 parameters:

- gNa Na conductance in 1/(mOhms * cm²)
- ENa Na equi potential in mV
- gK K conductance in 1/(mOhms * cm²)
- EK K equi potential in mV
- ql Leak conductance in 1/(mOhms * cm²)
- El Leak equi potential in mV
- Cmem Membrane capacity density in muF/cm²

Note

Internally, the ordinary differential equations defining the model are integrated with a linear Euler algorithm and GeNN integrates 25 internal time steps for each neuron for each network time step. I.e., if the network is simulated at DT= 0.1 ms, then the neurons are integrated with a linear Euler algorithm with 1DT= 0.004 ms.

Other variants of the same model are TRAUBMILES_ALTERNATIVE, TRAUBMILES_SAFE and TRAUBMILES_ \leftarrow PSTEP. The former two are adressing the problem of singularities in the original Traub & Miles model [5]. At V= -52 mV, -25 mV, and -50 mV, the equations for α have denominators with value 0. Mathematically this is not a problem because the nominator is 0 as well and the left and right limits towards these singular points coincide. Numerically, however this does lead to nan (not-a-number) results through division by 0. The TRAUBMILES_ALTERNATIVE model adds SCALAR_MIN to the denominators at all times which typically is completely effect-free because it is truncated from teh mantissa, except when the denominator is very close to 0, in which case it avoids the singular value.

TRAUBMILES_SAFE takes a much more direct approach in which at the singular points, the correct value calculated offline with l'Hopital's rule in substituted. This is implemented with "if" statements.

Finally, the TRAUBMILES_PSTEP model allows users to control the number of internal loops, or sub-timesteps, that are used. This is enabled by making the number of time steps an explicit parameter of the model.

6.4.4 IZHIKEVICH (Izhikevich neurons with fixed parameters)

This is the Izhikevich model with fixed parameters [1]. It is usually described as

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

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

Variables are:

- ∨ Membrane potential
- U Membrane recovery variable

Parameters are:

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

6.4.5 IZHIKEVICH_V (Izhikevich neurons with variable parameters)

This is the same model as IZHIKEVICH (Izhikevich neurons with fixed parameters) 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.

6.4.6 SPIKESOURCE (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.

6.4.7 Defining your own neuron type

In order to define a new neuron type for use in a GeNN application, it is necessary to populate an object of class neuronModel and append it to the global vector nModels. This can be done conveniently within the model Definition Definition function just before the model is needed. The neuronModel class has several data members that make up the full description of the neuron model:

• simCode of type string: This needs to be assigned a C++ (stl) string that contains the code for executing the integration of the model for one time step. Within 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 . Example:

6.4 Neuron models 31

```
neuronModel model;
model.simCode=String("$(V)+= (-$(a)$(V)+$(Isyn))*DT;");
```

would implement a leaky itegrator $\frac{dV}{dt} = -aV + I_{\text{syn}}$.

thresholdConditionCode of type vector<string> (if applicable): Condition for true spike detection.

• supportCode of type string: This allows to define a code snippet that contains supporting code that will be utilized in the otehr code snippets. Typically, these are functions that are needed in the simCode or thresholdConditionCode. 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 neuronpopulation whose neurons define it. An example of a supportCode definition would be

```
\label{eq:n.supportCode} n.supportCode= tS(" __host__ __device__ mysin(float x) {\n\ return } sin(x); \n\ }");
```

varNames of type vector<string>: This vector needs to be filled with the names of variables that
make up the neuron state. The variables defined here as NAME can then be used in the syntax in the code
string.

Example:

```
model.varNames.push_back(String("V"));
```

would add the variable V as needed by the code string in the example above.

varTypes of type vector<string>: This vector needs to be filled with the variable type (e.g. "float",
 "double", etc) for the variables defined in varNames. Types and variables are matched to each other by
 position in the respective vectors, i.e. the 0th entry of varNames will have the type stored in the 0th entry of
 varTypes and so on.

Example:

```
model.varTypes.push_back(String("float"));
```

would designate the variable V to be of type float.

Note

pNames of type vector<string>: This vector will contain the names of parameters relevant to the
model. If defined as NAME here, they can then be referenced as in the code string. The length of this vector
determines the expected number of parameters in the initialisation of the neuron model. Parameters are
assumed to be always of type double.

```
model.pNames.push_back(String("a"));
```

stores the parameter a needed in the code example above.

dpNames of type vector<string>: Names of "dependent parameters". 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. This vector contains the
names of such dependent parameters.

For example, if in the example above the original model had been $\frac{dV}{dt} = -g/CV + I_{\rm syn}$. Then one could define the code string and parameters as

```
model.simCode=String("$(V)+= (-$(a)$(V)+$(Isyn))*DT;");
model.varNames.push_back(String("V"));
model.varTypes.push_back(String("float"));
model.pNames.push_back(String("g"));
model.pNames.push_back(String("C"));
model.dpNames.push_back(String("C"));
```

- dps of type dpclass*: The dependent parameter class, i.e. an implementation of dpclass which would return
 the value for dependent parameters when queried for them. E.g. in the example above it would return
 a when queried for dependent parameter 0 through dpclass::calculateDerivedParameter().
 Examples how this is done can be found in the pre-defined classes, e.g. expDecayDp, pwSTDP, rulkovdp
 etc.
- extraGlobalNeuronKernelParameters of type vector<string>: On occasion, the neurons in a population share the same parameter. This could, for example, be a global reward signal. Such situations are implemented in GeNN with extraGlobalNeuronKernelParameters. This vector takes the names of such parameters. FOr each name, a variable will be created, with the name of the neuron population name appended, that can take a single value per population of the type defined in the extra—GlobalNeuronKernelParameterTypes vector. This variable is available to all neurons in the population. It can also be used in synaptic code snippets; in this case it needs to be addressed with a _pre or _post postfix. For example, if teh pre-synaptic neuron population is of a neuron type that defines:

```
n.extraGlobalNeuronKernelParameters.push_back(tS("R"));
n.extraGlobalNeuronKernelParameterTypes.push_back(tS("float"));
```

then, a synapse population could have simulation code like

```
s.simCode= tS("$(x) = $(x) + $(R_pre);");
```

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

• extraGlobalNeuronKernelParameterTypes of type vector<string>: These are the types of the extraGlobalNeuronKernelParameters. Types are matched to names by their position in the vector.

Once the completed neuronModel object is appended to the nModels vector,

```
nModels.push_back(model);
```

it can be used in network descriptions by referring to its cardinal number in the nModels vector. I.e., if the model is added as the 4th entry, it would be model "3" (counting starts at 0 in usual C convention). The information of the cardinal number of a new model can be obtained by referring to nModels.size() right before appending the new model, i.e. a typical use case would be.

```
int myModel= nModels.size();
nModels.push_back(model);
```

. Then one can use the model as

```
\verb|networkModel.addNeuronPopulation(..., myModel, ...);|\\
```

6.4.8 Explicit current input to neurons (NOW REMOVED)

In earlier versions of GeNN External input to a neuron group could be activated by calling the activateDirect Input function. This was now removed in favour of defining a new neuron model where the direct input can be a parameter (constant over time and homogeneous across the population), a variable (changing in time and non-homogeneous across the population), or an extraGlobalNeuronKernelParameter (changing in time but homogeneous across the population). How this can be done is illustrated for example in the lzh_sparse example project.

Previous | Top | Next

6.5 Synapse models

A synapse model is a weightUpdateModel object which consists of variables, parameters, and string objects which will be substituted in the code. The three strings that will be substituted in the code for synapse update are:

• simCode: Simulation code that is used when a true spike is detected. The update is performed only once, one time step after threshold condition detection, which is defined by thresholdConditionCode in the neuron model of the corresponding presynaptic neuron population.

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 provided as the value of the by the \$(addtoinsyn) snippet. For example

```
^{"}$(addtoinsyn) = $(inc);"
```

where "inc" is a parameter of the weightupdate model, would define a constant increment of the synaptic input of a post-synaptic neuron for each pre-synaptic spike.

When a spike is detected in the presynaptic neuron population, the simCode is executed. Within the code snippet the \$(addtoinSyn) update just discussed should be be followed by the \$(updatelinsyn) keyword 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 NSYNAPSE (No Learning) for a simple synapse update model and LEARN1SYNAPSE (Learning Synapse with a Primitive Piece-wise Linear Rule) for a more complicated model that uses STDP.

• evntThreshold: Code that defines the condition for a synaptic event. This typically involves the presynaptic variables, e.g. the membrane potential:

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

Whenever this expression evaluates to true, the code in simCodeEvnt is executed. For an example, see NGRADSYNAPSE (Graded Synapse).

- simCodeEvnt: Simulation code that is used for spike like events, where updates are done for all instances in which the event condition defined by evntThreshold is met. evntThreshold is also be provided as a code string. For an example, see NGRADSYNAPSE (Graded Synapse).
- simLearnPost: Simulation code which is used in the learnSynapsesPost kernel/function, which performs updates to synapses that are triggered by post-synaptic spikes. This is rather unusual other than in learning rules like e.g. STDP. For an example that uses simLearnPost, see LEARN1SYNAPSE (Learning Synapse with a Primitive Piece-wise Linear Rule).
- synapseDynamics: Simulation code that applies for every time step, i.e.is unlike the others not gated with a condition. This can be used where synapses have internal variables and dynamics that are described in continuous time, e.g. by ODEs. Using this mechanism is typically computationally veruy costly because of the large number of synapses in a typical network.
- extraGlobalSynapseKernelParameters of type vector<string>: On occasion, the synapses in a synapse population share a global parameter. This could, for example, be a global reward signal. This is supported in GeNN with extraGlobalSynapseKernelParameters. The user defines the names of such parameters and pushes them into this vector. GeNN creates variables of this name, with the name of the synapse population appended, that can take a single value per population of the type defined in the extraGlobalSynapseKernelParameterTypes vector. This variable is then available to all synapses in the population.

Note

No implicit or explicit copy of extraGlobalSynapseKernelParameters is necessary as they are communicated as kernel parameters.

• extraGlobalSynapseKernelParameterTypes of type vector<string>: These are the types of the extraGlobalSynapseKernelParameters. Types are matched to names by their position in the vector.

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

6.5.1 Built-in Models

Currently 3 predefined synapse models are available:

- NSYNAPSE
- NGRADSYNAPSE
- LEARN1SYNAPSE

These are defined in lib.include/utils.h. The MBody_userdef example also includes a modified version of these models as user-defined models.

6.5.2 NSYNAPSE (No Learning)

If this model is selected, 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.

simCode is:

```
" $(addtoinSyn) = $(g);\n\
$(updatelinsyn);\n"
```

6.5.3 NGRADSYNAPSE (Graded 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

simCodeEvnt is:

```
" (addtoinSyn) = (g) * tanh((((v_pre) - ((Epre))) *DT*2/((Vslope)); \n' (updatelinsyn); \n"
```

evntThresholdis:

```
" $(V_pre) > $(Epre)"
```

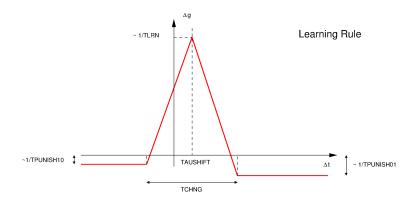
Note

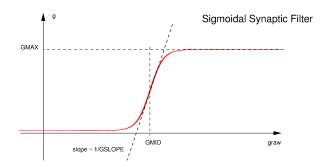
The pre-synaptic variables are referenced with the suffix _pre in synapse related code such as an evntermeshold. Users can also access post-synaptic neuron variables using the suffix _post.

6.5 Synapse models 35

6.5.4 LEARN1SYNAPSE (Learning Synapse with a Primitive Piece-wise Linear Rule)

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse, defined as a piecewise function:





The STDP curve is applied to the raw synaptic conductance gRaw, which is then filtered through the sugmoidal filter displayed above to obtain the value of g.

Note

The STDP curve implies that unpaired pre- and post-synaptic spikes incur a negative increment in gRaw (and hence in g).

The time of the last spike in each neuron, "sTXX", where XX is the name of a neuron population is (somewhat arbitrarily) initialised to -10.0 ms. If neurons never spike, these spike times are used.

It is the raw synaptic conductance gRaw that is subject to the STDP rule. The resulting synaptic conductance is a sigmoid filter of gRaw. This implies that g is initialised but not gRaw, the synapse will revert to the value that corresponds to gRaw.

An example how to use this synapse correctly is given in map_classol.cc (MBody1 userproject):

```
for (int i = 0; i < model.neuronN[1]*model.neuronN[3]; i++) {
    if (gKCDN[i] < 2.0*SCALAR_MIN) {
        cnt++;
        fprintf(stdout, "Too low conductance value %e detected and set to 2*SCALAR_MIN= %e, at index %d
    \n", gKCDN[i], 2*SCALAR_MIN, i);
        gKCDN[i] = 2.0*SCALAR_MIN; //to avoid log(0)/0 below
    }
    scalar tmp = gKCDN[i] / myKCDN_p[5]*2.0;
    gRawKCDN[i] = 0.5 * log( tmp / (2.0 - tmp)) /myKCDN_p[7] +
    myKCDN_p[6];
}
cerr << "Total number of low value corrections: " << cnt << endl;</pre>
```

Note

One cannot set values of g fully to 0, as this leads to gRaw = -infinity and this is not support. I.e., 'g' needs to be some nominal value > 0 (but can be extremely small so that it acts like it's 0).

The model has 2 variables:

- g: conductance of scalar type
- gRaw: raw conductance of scalar type

Parameters are (compare to the figure above):

- Epre: Presynaptic threshold potential
- 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
- qMid: 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

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

6.5.5 Defining a new synapse model

If users want to define their own models, they can add a new weightUpdateModel that includes the variables, parameters, and update codes as desired, and then push this object in the weightUpdateModels vector. The model can be used by referring to its index in the weightUpdateModels vector when adding a new population by with a call to addSynapsePopulation.

6.5.6 Conductance definition methods

The available options work as follows:

- INDIVIDUALG: When this option is chosen in the addSynapsePopulation command, GeNN reserves an array of size n_pre x n_post float for individual conductance values for each combination of pre and postsynaptic neuron. The actual values of the conductances are passed at runtime from the user side code, using the pushXXXXXToDevice function, where XXXX is the name of the synapse population.
- GLOBALG: When this option is chosen, the value of the variables of the synapse model (including its conductance) is taken to be the initial value provided for the synapse model's variables. This option can only be sensibly combined with connectivity type ALLTOALL.
- INDIVIDUALID: When this option is chosen, GeNN expects to use the same maximal conductance for all existing synaptic connections but which synapses exist will be defined at runtime from the user side code, provided as a binary array (see Insect olfaction model).

6.5.7 Connectivity types

Available options are DENSE and SPARSE. Various tools are provided under userproject/tools for creating different connectivity schemes.

Different strategies are used by GeNN for different combinations of connecticity types and Conductance definition methods, as explained in the table below:

	ALLTOALL	DENSE	SPARSE
GLOBALG	Fixed values for all synapse	Fixed values for all synapse	Fixed values for some
	members	members	synapse members (using
			sparse indexing)
INDIVIDUALG	Variable values for all	Variable values for all	Variable values for some
	synapse members	synapse members	synapse members
INDIVIDUALID	Fixed values for some	Fixed values for some	N/A
	synapse members (using a	synapse members (using a	
	binary array). Technically	binary array)	
	possible but not meaningful.		

If INDIVIDUALG is used with ALLTOALL or DENSE connectivity (these are equivalent in this case), synapse variables are stored in an array of size npre * npost .

If the connectivity is of SPARSE type, connectivity indices are stored in a struct named SparseProjection in order to minimize the memory requirements. The struct SparseProjection 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 following arrays: 2: unsigned int ind, of size connN: Indices of corresponding postsynaptic neurons concatenated for each presynaptic neuron. 3: unsigned int *indInG, of size model.neuronN[model.synapseSource[synInd]]+1: 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[n+1]-indIng[n] would give the number of postsynaptic connections for neuron n.

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]
```

A synapse variable of a sparsely connected synapse will be kept in an array using this conductance for indexing. For example, a variable called g will be kept in an array such as: $g=[g_pre0-post1 g_pre0-post2 g_pre1-post0 g_pre1-post2]$ If there are no connections for a presynaptic neuron, then g[indlng[n]=gp[indlng[n]+1].

See tools/gen syns_sparse_lzhModel used in lzh_sparse project to see a working example.

6.5.8 Postsynaptic integration methods

The postSynModel 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.

A postSynModel object consists of variables, parameters, derived parameters and two strings that define the code for current generation and continuous dynamics.

• string postSynDecay: This code defines the continuous time dynamics of the summed presynaptic inputs at the postsynaptic neuron. This usually consists of some kind of decay function.

• string postSyntoCurrent: This code defines how the synaptic inputs lead to an input input current (lsyn) to the postsynaptic neuron.

There are currently 2 built-in postsynaptic integration methods:

EXPDECAY: Exponential decay. Decay time constant and reversal potential parameters are needed for this postsynaptic mechanism.

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).

IZHIKEVICH_PS: Empty postsynaptic rule to be used with Izhikevich neurons.

Previous | Top | Next

7 Tutorial 1

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

7.1 The Model Definition

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

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

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

Note

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

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

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

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

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

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

7.1 The Model Definition 39

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 \form#10S. 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 addNeuronPopulation and 'addSynapse \leftarrow Populationmember functions of the NNmodel object. The arguments to a call toaddNeuronPopulations are

- string name: the name of the population
- int N: The number of neurons in the population
- int type: The type of neurons in the population
- double *p: An array of parameter values for teh neurons in the population
- double *ini: An array of initial values for neuron variables

We first create the parameter and initial variable arrays,

```
// definition of tenHHModel
double p[7] = {
                   // 0 - gNa: Na conductance in muS
  7.15.
                   // 1 - ENa: Na equi potential in mV
// 2 - gK: K conductance in muS
  50.0.
  1.43,
                   // 3 - EK: K equi potential in mV
 0.02672,
                   // 4 - gl: leak conductance in muS
                   // 5 - El: leak equi potential in mV
  -63.563,
                   // 6 - Cmem: membr. capacity density in nF
 0.143
double ini[4]= {
                   // 0 - membrane potential V
  0.0529324.
                   // 1 - prob. for Na channel activation m // 2 - prob. for not Na channel blocking h
  0.3176767,
                   // 3 - prob. for K channel activation n
 0.5961207
```

Note

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

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

```
model.addNeuronPopulation("Pop1", 10, TRAUBMILES, p, ini);
```

Note

TRAUBMILES is a variable defined in the GeNN model specification that contains the index number of the pre-defined Traub & Miles model [5].

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
  model.setDT(0.1);
  model.setName("tenHHModel");
  double p[7] = {
    7.15,
                    // 0 - gNa: Na conductance in muS
                   // 1 - ENa: Na equi potential in mV
    50.0,
    1.43,
                   // 2 - gK: K conductance in muS
                   // 3 - EK: K equi potential in mV
                   // 4 - gl: leak conductance in muS
// 5 - El: leak equi potential in mV
    0.02672,
    -63.563,
                   // 6 - Cmem: membr. capacity density in nF
    0.143
  double ini[4]= {
    -60.0,
                   // 0 - membrane potential V
    0.0529324,
                   // 1 - prob. for Na channel activation m
                   // 2 - prob. for not Na channel blocking h
    0.3176767,
                   // 3 - prob. for K channel activation n
    0.5961207
  model.addNeuronPopulation("Pop1", 10, TRAUBMILES, p, ini);
  model.finalize();
```

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

7.2 User Code

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

```
// tenHHModel simulation code
#include "tenHHModel.cc"
#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 execute the integration of the neuron equations provided by GeNN. To do so, we add after initialize();

```
stepTimeGPU(1000.0);
```

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

```
pullPop1fromDevice();
for (int i= 0; i < 10; i++) {
   cout << VPop1[i] << " ";
   cout << mPop1[i] << " ";
   cout << hPop1[i] << " ";
   cout << nPop1[i] << endl;
}</pre>
```

7.3 Makefile 41

pullPoplfromDevice() copies all relevant state variables of the Popl 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 VPopl, mPopl, hPopl, nPopl.

Note

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

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

```
// tenHHModel simulation code
#include "tenHHModel.cc"
#include "tenHHModel_CODE/definitions.h"
int main()
{
   allocateMem();
   initialize();
   stepTimeGPU(1000.0);
   pullPopIfromDevice();
   for (int i= 0; i < 10; i++) {
      cout << VPopI[i] << " ";
      cout << mPopI[i] << " ";
      cout << nPopI[i] << " ";
      cout << nPopI[i] << " cout << nPopI[i] << " ";
      cout << nPopI[i] << " ";
      cout << nPopI[i] << " ";
      cout << nPopI[i] << endI;
   }
   return 0;</pre>
```

7.3 Makefile

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

This defines that the final executable of this simulation is named tenHHSimulation and the simulation code is given in the file tenHHSimulation.cu 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 
-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
```

This completes this tutorial. You have created a GeNN model and simulated it successfully!

7.5 Adding External Input

In the example we have created so far, the neurons are not connected and do not receive input. As the TRAUB—MILES model is silent in such conditions, the ten neurons simply will simply rest at their resting potential. To make things more interesting, let us add a constant input to all neurons, add to the end of the modelDefinition function

```
model.activateDirectInput("Pop1", CONSTINP);
model.setConstInp("Pop1", 0.1);
```

This will add a constant input of 0.1 nA to all ten neurons. When run with this addition you should observe the output

```
-63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695 -63.1468 0.0211871 0.987233 0.0423695
```

This is still 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 best save them into a file. This can be done in many ways but one sensible way of doing this is to replace the line

```
stepTimeGPU(1000.0);
```

in tenHHSimulation.cu to something like this:

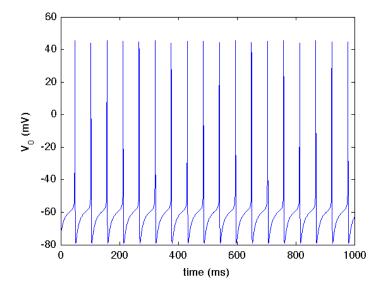
```
ofstream os("tenHH_output.V.dat");
double t= 0.0;
for (int i= 0; i < 5000; i++) {
    stepTimeGPU(0.2);
    pullPop1fromDevice();
    os << t << " ";
    for (int j= 0; j < 10; j++) {
       os << VPop1[j] << " ";
    }
    os << endl;
    t+= 0.2;
}
os.close();</pre>
```

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 column two (voltage of neuron 0), you should observe dynamics like this:

8 Tutorial 2 43



The completed files from this tutorial can be found in userproject/tenHH_project.

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.cu tenHHRingSimulation.cu
```

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 NSYNAPSE weightupdate model and EXPDECAY post-synaptic model. They need intial variables and parameters as follows:

If an array is not needed we set it to the NULL pointer. Here there are for example no synaptic parameters and no initial values for the post-synaptic mechanism. We can then add a synapse population at the end of the model \leftarrow Definition(...) function,

The addSynapsePopulation parameters are

- const char *name: The name of the synapse population, here "Pop1self"
- int sType: The type of synapse to be added, we here use the predefined typse NSYNAPSE. See Built-in Models for all available predefined synapse types.
- int sConn: The type of synaptic connectivity, here DENSE which means we will provide a full connectivity mtrix later.
- int gType: The way how the synaptic conductivity g will be defined. With GLOBALG we indicate that all conductance are of the same conductance value, which will be the value given in sPara.
- int delay: NO_DELAY means that there wil be no delays for synaptic signal propagation.
- int postSyn: Postsynaptic integration method, we are here using the standard model of an exponential decay of synaptic excitation.
- char *preName: Name of the pre-synaptic neuron population, here the Pop1 population.
- char *postName: Name of the post-synaptic neuron population, here also Pop1.
- double *sIni: A C-array of doubles containing initial values for the synaptic variables.
- double *sParam: A C-array of double precision that contains parameter values (common to all synapses of the population)
- double *psIni: A C-array of double precision numbers containing initial values for the post-synaptic model variables
- double *psPara: A C-array of double precision numbers containing parameters to the post-snaptic model.

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 definintion file tenHHModel.cc
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
  // definition of tenHHModel
  initGeNN();
  model.setDT(0.1);
  model.setName("tenHHRingModel");
  double p[7] = {
                     // 0 - qNa: Na conductance in muS
                    // 1 - ENa: Na equi potential in mV
// 2 - gK: K conductance in muS
    50.0,
    1.43.
                   // 3 - EK: K equi potential in mV
// 4 - gl: leak conductance in muS
   -95.0.
    0.02672,
                    // 5 - El: leak equi potential in mV
    -63.563,
                   // 6 - Cmem: membr. capacity density in nF
  double ini[4]= {
    -60.0.
                    // 0 - membrane potential V
    0.0529324,
                    // 1 - prob. for Na channel activation m
                     // 2 - prob. for not Na channel blocking h
    0.3176767,
```

8.2 Defining the Detailed Synaptic Connections

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

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

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

```
allocateMem();
initialize();
// define the connectivity
int pre, post;
for (int i= 0; i < 10; i++) {
   pre= i;
   post= (i+1)%10;
   gPoplself[pre*10+post]= 0.01;
}
pushPoplselftoDevice();</pre>
```

After memory allocation and initialization gPop1self will contain only zeros. We then assign in the loop a non-zero conductivity of 0.01 μ S to all synapses from neuron i to i+1 (and 9 to 0 to close the ring).

After adjusting the GNUmakefile to read

```
EXECUTABLE :=tenHHRingSimulation
SOURCES :=tenHHRingSimulation.cu
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 exectable tenHHRingSimulation, which can be executed,

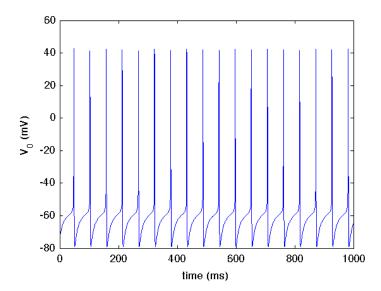
```
>> ./tenHHRingSimulation
```

which should again result in

```
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
```

```
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
```

If we plot the content of columns one and two of tenHHexample.V.dat it looks very similar as in Tutorial 1



This is because the inhibitory synapses we created were all triggered at the same time so that they act during a post-synaptic spike which makes their effect all but invisible.

8.3 Setting Heterogeneous Initial Conditions

If we define different initial conditions for each of the ten neurons, i.e. add after ${\tt initialize}$ (),

```
// set initial variables
for (int i= 0; i < 10; i++) {
   VPop1[i] = -60.0+i;
}
pushPopltoDevice();</pre>
```

then we observe different final values for each neuron,

```
-57.3412 0.06223 0.981374 0.104417

-53.3189 0.442962 0.0664687 0.764513

-73.1413 0.00253709 0.927251 0.0277236

-67.1179 0.00927304 0.986692 0.0206106

-63.5878 0.01938 0.991071 0.0387962

-62.2114 0.0255295 0.990933 0.0504799

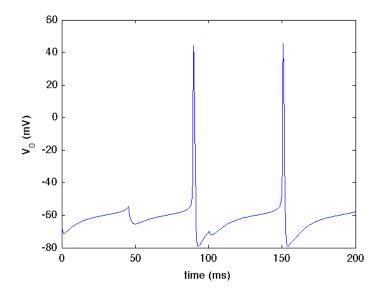
-61.404 0.0298902 0.990949 0.0586459

-60.6691 0.034405 0.990225 0.0668015

-59.8977 0.0397467 0.988701 0.0758159

-58.9727 0.0470178 0.98615 0.0866963
```

and zooming in on the first 200 ms, the voltage of the first neuron now looks like this



The complete codes for this tutorial are in userproject\tenHHRing_project.

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)

• allocateMem()

• stepTimeGPU()

• freeDeviceMem()

• freeMem()

Neuron and synapse variables (if not initialising to the homogeneous initial value provided during model Definition)

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

```
deviceMemAllocate()
initialize()
initializeAllSparseArrays()
convertProbabilityToRandomNumberThreshold()
convertRateToRandomNumberThreshold()
copyStateToDevice()
push\<neuron or synapse name\>StatetoDevice()
pull\<neuron or synapse name\>StatefromDevice()
copyStateFromDevice()
copySpikeNFromDevice()
copySpikesFromDevice()
stepTimeCPU()
```

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

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 core units such as neuronModel, weightUpdateModel or postSynModel objects. The name of a variable is defined when the model type is introduced, i.e. with a statement such as

```
neuronModel model;
model.varNames.push_back(String"x"));
model.varTypes.push_back(String("double"));
...
int myModel= nModels.size();
nModels.push_back(model);
```

This declares that whenever the defined model type of cardinal number myModel is used, there will be a variable of core name x. varType can be of scalar type (see Floating point precision). The full GeNN name of this variable is obtained by directly concatenating the core name with the name of the neuron population in which the model type has been used, i.e. after a definition

```
networkModel.addNeuronPopulation("EN", n, myModel, ...);
```

there will be a variable $x \in \mathbb{N}$ of type double* 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, n in the example above. 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.

```
pullENStatefromDevice();
```

would copy the C array xEN from device memory into host memory (and any other variables that the neuron type of the population EN 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 pullenstatefrom Device() will look like

```
unsigned int size;
size = sizeof(double) * nEN;
cudaMemcpy(xEN, d_xEN, size, cudaMemcpyDeviceToHost);
```

where nEN is an integer containing the population size of the EN neuron population.

The same convention as for neuron variables applies for the variables of synapse populations, both for those originating from weightupdate models and from post-synaptic models, e.g. the variables in type ${\tt NSYNAPSE}$ contain the variable ${\tt g}$ of type float. Then, after

```
networkModel.addSynapsePopulation("ENIN", NSYNAPSE, ...);
```

there will be a global variable of type float* with the name gENIN that is pre-allocated to the right size. There will also be a matching device pointer with the name d gENIN.

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 Connectivity types

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

Since GeNN 2.0, there are no more 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 weightUpdateModel for updating synaptic input. The way it is modified is defined in weightUpdateModel.simCode or weightUpdateModel.simCodeEvnt, 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 postSynModel. For example, the standard EXPDECAY postsynaptic model defines

```
ps.postSyntoCurrent= String("$(inSyn)*($(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 postSyntoCurrent code is assigned to Isyn and can then be used in neuron simCode like so:

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

• sT: As a neuron variable, this is the last spike time in a neuron and is automatically generated for pre and postsynaptic neuron groups of a synapse group i that follows a spike based learning rule (indicated by usesPostLearning[i]= TRUE for the ith synapse population).

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

9.4 Debugging suggestions

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

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

Note

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

Previous | Top | Next

10 Credits

GeNN was created by Thomas Nowotny.

Izhikevich model and sparse connectivity by Esin Yavuz.

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

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

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

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

Previous | Top

11 Namespace Index

11.1 Namespace List

Here is a list of all namespaces with brief descriptions:

(GENN_FLAGS	??
(GENN_PREFERENCES	??
12	Hierarchical Index	
12.1	Class Hierarchy	
This	s inheritance list is sorted roughly, but not completely, alphabetically:	
c	classIzh	64
c	classol	66
(CodeHelper	76
(CStopWatch	77
c	dpclass	77
	expDecayDp	78
	pwSTDP	99
	pwSTDP_userdef	100
	rulkovdp	105
•	errTupel	78
i	inputSpec	79
r	neuronModel	79
r	neuronpop	82
1	NNmodel	83
F	Parameter	97
ķ	postSynModel	97
(${\sf QTIsaac} {<\hspace{1mm}} {\sf ALPHA}, {\sf T} {>\hspace{1mm}}$	101
(QTIsaac < 8, unsigned long >	101
(QTIsaac< ALPHA, T >::randctx	102
r	randomGauss	103
r	randomGen	104
9	Schmuker2014_classifier	105
5	SimulationNeuronPolicyPrePostVar	111
5	SimulationNeuronPolicyPreVar	111
5	SimulationSynapsePolicyDense	112
	SimulationSynapsePolicySparse	113

13 Class Index 53

SimulationSynapsePolicy	112
SimulationSynapsePolicyNone	113
SparseProjection	115
stdRG	116
stopWatch	117
SynDelay TestWithParam	117
SimulationTest	114
${\bf Simulation Test Vars} < {\bf Neuron Policy, Synapse Policy} >$	115
weightUpdateModel	118
13 Class Index	
13.1 Class List	
Here are the classes, structs, unions and interfaces with brief descriptions:	
classizh	64
classol This class cpontains the methods for running the MBody1 example model	66
CodeHelper	76
CStopWatch	77
dpclass	77
errTupel	78
expDecayDp Class defining the dependent parameter for exponential decay	78
inputSpec	79
neuronModel Class for specifying a neuron model	79
neuronpop	82
NNmodel	83
Parameter	97
postSynModel Class to hold the information that defines a post-synaptic model (a model of how affect post-synaptic neuron variables, classically in the form of a synaptic currer allows to define an equation for the dynamics that can be applied to the summer input variable "insyn"	nt). It also
pwSTDP TODO This class definition may be code-generated in a future release	99

p	owSTDP_userdef TODO This class definition may be code-generated in a future release	100
G	QTIsaac< ALPHA, T >	101
C	QTIsaac< ALPHA, T >::randctx	102
ra	andomGauss Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution	103
ra	andomGen Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers	104
rı	ulkovdp Class defining the dependent parameters of the Rulkov map neuron	105
S	Schmuker2014_classifier This class cpontains the methods for running the Schmuker_2014_classifier example model	105
S	SimulationNeuronPolicyPrePostVar	111
S	SimulationNeuronPolicyPreVar	111
S	SimulationSynapsePolicy	112
S	SimulationSynapsePolicyDense	112
S	SimulationSynapsePolicyNone	113
S	SimulationSynapsePolicySparse	113
S	SimulationTest	114
S	SimulationTestVars< NeuronPolicy, SynapsePolicy >	115
S	SparseProjection Class (struct) for defining a spars connectivity projection	115
s	etdRG	116
s	stopWatch	117
S	SynDelay	117
W	veightUpdateModel 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	118
14	File Index	
14.1	File List	
Here	e is a list of all files with brief descriptions:	
N	//Body1_project/model/classol_sim.cc Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a	

part of the complete example of simulating the MBody1 mushroom body model

121

MBody_delayedSyn_project/model/classol_sim.cc Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody_delayedSyn mushroom body model	122
part of the complete example of simulating the MBody_delayedSyll musilionin body model	122
MBody_individualID_project/model/classol_sim.cc Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model	122
MBody_userdef_project/model/classol_sim.cc Main entry point for the classol (CLASSification in OLfaction) model simulation. Provided as a part of the complete example of simulating the MBody1 mushroom body model	123
MBody1_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody1 model	123
MBody_delayedSyn_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody_ delayedSyn model	124
MBody_individualID_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody_← individualID model	125
MBody_userdef_project/model/classol_sim.h Header file containing global variables and macros used in running the classol / MBody1 model	126
CodeHelper.h	127
command_line_processing.h This file contains some tools for parsing the argv array which contains the command line options	128
dpclass.h	129
experiment.cc	129
experiment.h	130
extra_neurons.h	132
extra_postsynapses.h	133
extra_weightupdates.h	134
GA.cc	134
gauss.cc Contains the implementation of the Gaussian random number generator class randomGauss	135
gauss.h Random number generator for Gaussian random variable with mean 0 and standard deviation 1	135
gen_input_structured.cc This file is part of a tool chain for running the classol/MBody1 example model	136
gen_kcdn_syns.cc This file is part of a tool chain for running the classol/MBody1 example model	136
gen_kcdn_syns_fixto10K.cc	137
gen_pnkc_syns.cc This file is part of a tool chain for running the classol/MBody1 example model	138

This file is part of a tool chain for running the classol/MBody1 example model	138
gen_pnlhi_syns.cc This file is part of a tool chain for running the classol/MBody1 example model	140
gen_syns_sparse.cc This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity	140
gen_syns_sparse_izhModel.cc This file is part of a tool chain for running the Izhikevich network model	141
HHVclampGA_project/generate_run.cc This file is used to run the HHVclampGA model with a single command line	143
Izh_sparse_project/generate_run.cc This file is part of a tool chain for running the classIzh/Izh_sparse example model	143
MBody1_project/generate_run.cc This file is part of a tool chain for running the classol/MBody1 example model	144
MBody_delayedSyn_project/generate_run.cc This file is part of a tool chain for running the classol/MBody_delayedSyn example model	145
MBody_individualID_project/generate_run.cc This file is part of a tool chain for running the classol/MBody_individualID example model	145
MBody_userdef_project/generate_run.cc This file is part of a tool chain for running the classol/MBody_userdef example model	146
OneComp_project/generate_run.cc This file is part of a tool chain for running the classol/MBody1 example model	147
PoissonIzh_project/generate_run.cc This file is part of a tool chain for running the classol/MBody1 example model	147
generateALL.cc Main file combining the code for code generation. Part of the code generation section	148
generateALL.h	150
generateCPU.cc Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section	151
generateCPU.h Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section	152
generateKernels.cc Contains functions that generate code for CUDA kernels. Part of the code generation section	153
generateKernels.h Contains functions that generate code for CUDA kernels. Part of the code generation section	155
generateRunner.cc Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section	156

generateRunner.h Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section	158
GeNNHelperKrnls.cu	159
GeNNHelperKrnls.h	160
global.cc	160
global.h Global header file containing a few global variables. Part of the code generation section	162
helper.h	164
HHVClamp.cc This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques	166
HHVClampParameters.h	167
hr_time.cc This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock	167
hr_time.h This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock	168
isaac.cc Header file and implementation of the ISAAC random number generator	168
lzh_sim_sparse.cc	169
lzh_sparse.cc	169
lzh_sparse_model.cc	17 1
lzh_sparse_model.h	171
lzh_sparse_sim.h	171
MBody1_project/model/map_classol.cc Implementation of the classol class	172
MBody_delayedSyn_project/model/map_classol.cc Implementation of the classol class	173
MBody_individualID_project/model/map_classol.cc Implementation of the classol class	173
MBody_userdef_project/model/map_classol.cc Implementation of the classol class	173
MBody1_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	174

MBody_delayedSyn_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	174
MBody_individualID_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	175
MBody_userdef_project/model/map_classol.h Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body	175
HHVclampGA_project/model/MBody1.cc This file contains the model definition of the mushroom body "MBody1" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol—sim)	176
MBody1_project/model/MBody1.cc This file contains the model definition of the mushroom body "MBody1" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol← _sim)	179
MBody_delayedSyn.cc This file contains the model definition of the mushroom body "MBody_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim)	183
MBody_individualID.cc This file contains the model definition of the mushroom body "MBody_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC	187
MBody_userdef.cc This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol_sim)	190
extra_global_params_in_sim_code/model.cc	194
extra_global_params_in_sim_code_event_spare_inv/model.cc	195
extra_global_post_param_in_sim_code/model.cc	195
extra_global_pre_param_in_sim_code/model.cc	196
neuron_support_code_sim/model.cc	197
neuron_support_code_threshold/model.cc	197
post_vars_in_post_learn/model.cc	198
post_vars_in_post_learn_sparse/model.cc	199
post_vars_in_sim_code/model.cc	200
post_vars_in_sim_code_sparse/model.cc	201
nost vars in synapse dynamics/model.cc	201

post_vars_in_synapse_dynamics_sparse/model.cc	202
pre_vars_in_post_learn/model.cc	203
pre_vars_in_post_learn_sparse/model.cc	203
pre_vars_in_sim_code/model.cc	204
pre_vars_in_sim_code_event/model.cc	205
pre_vars_in_sim_code_event_sparse/model.cc	206
pre_vars_in_sim_code_event_sparse_inv/model.cc	207
pre_vars_in_sim_code_sparse/model.cc	207
pre_vars_in_synapse_dynamics/model.cc	208
pre_vars_in_synapse_dynamics_sparse/model.cc	209
synapse_support_code_event_sim_code/model.cc	210
synapse_support_code_event_threshold/model.cc	211
synapse_support_code_post_learn/model.cc	212
synapse_support_code_sim_code/model.cc	212
synapse_support_code_synapse_dynamics/model.cc	213
Model_Schmuker_2014_classifier.cc	214
include/modelSpec.cc	215
src/modelSpec.cc	215
modelSpec.h Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections	215
neuronModels.cc	218
neuronModels.h	220
OneComp.cc	222
OneComp_model.cc	223
OneComp_model.h	223
OneComp_sim.cc	223
OneComp_sim.h	223
parse_options.h	224
PoissonIzh-model.cc	225
PoissonIzh-model.h	225
PoissonIzh.cc	225

Poissonlzh_sim.cc	227
Poissonlzh_sim.h	227
postSynapseModels.cc	228
postSynapseModels.h	229
randomGen.cc Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand()	230
randomGen.h Header file containing the class definition for a uniform random generator based on the ISAAC random number generator	230
Schmuker2014_classifier.cc	231
Schmuker2014_classifier.h Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014"	231
simulation_neuron_policy_pre_post_var.h	232
simulation_neuron_policy_pre_var.h	232
simulation_synapse_policy_dense.h	232
simulation_synapse_policy_none.h	232
simulation_synapse_policy_sparse.h	232
simulation_test.h	233
simulation_test_vars.h	233
lzh_sparse_project/model/sizes.h	233
MBody1_project/model/sizes.h	234
MBody_delayedSyn_project/model/sizes.h	235
MBody_individualID_project/model/sizes.h	235
MBody_userdef_project/model/sizes.h	236
OneComp_project/model/sizes.h	236
Poissonlzh_project/model/sizes.h	237
sparseProjection.h	237
sparseUtils.cc	237
sparseUtils.h	238
stringUtils.cc	240
stringUtils.h	242

synapseModels.cc	244
synapseModels.h	245
SynDelay.cc	246
SynDelaySim.cc	246
SynDelaySim.h	247
extra_global_params_in_sim_code/test.cc	247
extra_global_params_in_sim_code_event_spare_inv/test.cc	248
extra_global_post_param_in_sim_code/test.cc	249
extra_global_pre_param_in_sim_code/test.cc	250
neuron_support_code_sim/test.cc	250
neuron_support_code_threshold/test.cc	251
post_vars_in_post_learn/test.cc	252
post_vars_in_post_learn_sparse/test.cc	252
post_vars_in_sim_code/test.cc	253
post_vars_in_sim_code_sparse/test.cc	254
post_vars_in_synapse_dynamics/test.cc	25 4
post_vars_in_synapse_dynamics_sparse/test.cc	255
pre_vars_in_post_learn/test.cc	256
pre_vars_in_post_learn_sparse/test.cc	257
pre_vars_in_sim_code/test.cc	257
pre_vars_in_sim_code_event/test.cc	258
pre_vars_in_sim_code_event_sparse/test.cc	259
pre_vars_in_sim_code_event_sparse_inv/test.cc	259
pre_vars_in_sim_code_sparse/test.cc	260
pre_vars_in_synapse_dynamics/test.cc	261
pre_vars_in_synapse_dynamics_sparse/test.cc	261
synapse_support_code_event_sim_code/test.cc	262
synapse_support_code_event_threshold/test.cc	263
synapse_support_code_post_learn/test.cc	26 4
synapse_support_code_sim_code/test.cc	26 4
synapse_support_code_synapse_dynamics/test.cc	265
utils.cc	266

utils.h

This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models

267

VClampGA.cc

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU

268

VClampGA.h

Header file containing global variables and macros used in running the HHVClamp/VClampGA model

269

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)

16 Class Documentation

16.1 classIzh Class Reference

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

Public Member Functions

- classIzh ()
- ∼classlzh ()
- void init (unsigned int)
- void allocate_device_mem_patterns ()
- void allocate_device_mem_input ()
- void copy_device_mem_input ()
- void read_sparsesyns_par (int, struct SparseProjection, FILE *, FILE *, FILE *, scalar *)

Read sparse connectivity from a file.

void gen_alltoall_syns (scalar *, unsigned int, unsigned int, 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, unsigned int)
- void randomizeVarSq (scalar *, scalar, unsigned int)
- void initializeAllVars (unsigned int)

Public Attributes

- NNmodel model
- scalar * input1
- scalar * input2
- scalar * d input1
- scalar * d_input2
- · unsigned int sumPExc
- · unsigned int sumPInh

16.1.1 Constructor & Destructor Documentation

```
16.1.1.1 classlzh::classlzh ( )

16.1.1.2 classlzh::~classlzh ( )

16.1.2 Member Function Documentation

16.1.2.1 void classlzh::allocate_device_mem_input ( )

16.1.2.2 void classlzh::allocate_device_mem_patterns ( )

16.1.2.3 void classlzh::copy_device_mem_input ( )

16.1.2.4 void classlzh::create_input_values ( )

16.1.2.5 void classlzh::free_device_mem ( )
```

16.1.2.6 void classIzh::gen_alltoall_syns (scalar * g, unsigned int nPre, unsigned int nPost, scalar gscale)

Generate random conductivity values for an all to all network.

Parameters

g	the resulting synaptic conductances
nPre	number of pre-synaptic neurons
nPost	number of post-synaptic neurons
gscale	the maximal conductance of generated synapses

16.1.2.7 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.

```
16.1.2.8 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.

```
16.1.2.9 void classlzh::init ( unsigned int which )

16.1.2.10 void classlzh::initializeAllVars ( unsigned int which )

16.1.2.11 void classlzh::output_params ( FILE * f, FILE * f2 )
```

16.1.2.12 void classIzh::output_spikes (FILE * f, unsigned int which)

```
16.1.2.13 void classlzh::output_state ( FILE * f, unsigned int which )
16.1.2.14 void classlzh::randomizeVar ( scalar * Var, scalar strength, unsigned int neuronGrp )
16.1.2.15 void classlzh::randomizeVarSq ( scalar * Var, scalar strength, unsigned int neuronGrp )
16.1.2.16 void classlzh::read_input_values ( FILE * )
16.1.2.17 void classlzh::read_sparsesyns_par ( int synInd, struct SparseProjection C, FILE * f_ind, FILE * f_indlnG, FILE * f_g, scalar * g )
```

Read sparse connectivity from a file.

Parameters

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

```
16.1.2.18 void classlzh::run ( double runtime, unsigned int which )

16.1.2.19 void classlzh::setInput ( unsigned int which )

16.1.2.20 void classlzh::sum_spikes ( )

16.1.2.21 void classlzh::write_input_to_file ( FILE * f )

16.1.3 Member Data Documentation

16.1.3.1 scalar* classlzh::d_input1

16.1.3.2 scalar* classlzh::d_input2

16.1.3.3 scalar* classlzh::input1

16.1.3.4 scalar * classlzh::input2

16.1.3.5 NNmodel classlzh::model

16.1.3.6 unsigned int classlzh::sumPExc

16.1.3.7 unsigned int classlzh::sumPInh
```

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

- lzh_sparse_model.h
- Izh_sparse_model.cc

16.2 classol Class Reference

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

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

Public Member Functions

- · classol ()
- \sim 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 (int, 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 (int, 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 (int, 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 *)

    template<class DATATYPE >

  void read_sparsesyns_par (DATATYPE *, int, 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 (int, 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

16.2.1 Detailed Description

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

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

16.2.2 Constructor & Destructor Documentation

```
16.2.2.1 classol::classol ( )
16.2.2.2 classol::∼classol ( )
```

Destructor for olfaction model.

```
16.2.2.3 classol::classol()
16.2.2.4 classol:: ∼ classol ( )
16.2.2.5 classol::classol()
16.2.2.6 classol::\simclassol ( )
16.2.2.7 classol::classol()
16.2.2.8 classol::∼classol ( )
16.2.2.9 classol::classol()
16.2.2.10 classol::\simclassol ( )
16.2.3 Member Function Documentation
16.2.3.1 void classol::allocate_device_mem_input ( )
16.2.3.2 void classol::allocate_device_mem_patterns ( )
16.2.3.3 void classol::allocate_device_mem_patterns ( )
16.2.3.4 void classol::allocate_device_mem_patterns ( )
16.2.3.5 void classol::allocate_device_mem_patterns ( )
Method for allocating memory on the GPU device to hold the input patterns.
16.2.3.6 void classol::free_device_mem ( )
16.2.3.7 void classol::free_device_mem ( )
16.2.3.8 void classol::free_device_mem ( )
16.2.3.9 void classol::free_device_mem ( )
16.2.3.10 void classol::free_device_mem ( )
Methods for unallocating the memory for input patterns on the GPU device.
16.2.3.11 void classol::generate_baserates ( )
16.2.3.12 void classol::generate_baserates ( )
16.2.3.13 void classol::generate_baserates ( )
Method for calculating the baseline rates of the Poisson input neurons.
16.2.3.14 void classol::generate_baserates ( )
16.2.3.15 void classol::generate_baserates ( )
16.2.3.16 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.
16.2.3.17 void classol::get_kcdnsyns ( )
```

```
16.2.3.18 void classol::get_kcdnsyns()

16.2.3.19 void classol::get_kcdnsyns()

16.2.3.20 void classol::getSpikeNumbersFromGPU()

16.2.3.21 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.

```
16.2.3.22 void classol::getSpikeNumbersFromGPU ( )

16.2.3.23 void classol::getSpikeNumbersFromGPU ( )

16.2.3.24 void classol::getSpikeNumbersFromGPU ( )

16.2.3.25 void classol::getSpikesFromGPU ( )

16.2.3.26 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.

```
16.2.3.27 void classol::getSpikesFromGPU()
16.2.3.28 void classol::getSpikesFromGPU()
16.2.3.29 void classol::getSpikesFromGPU()
16.2.3.30 void classol::init ( unsigned int )
16.2.3.31 void classol::init ( unsigned int )
16.2.3.32 void classol::init ( unsigned int )
16.2.3.33 void classol::init ( unsigned int )
16.2.3.34 void classol::init ( unsigned int which )
```

Method for initialising variables.

Parameters

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

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

16.2.3.36 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

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

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

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

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

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

16.2.3.42 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

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

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

16.2.3.45 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

```
16.2.3.46 void classol::read_input_patterns ( FILE * )

16.2.3.47 void classol::read_input_patterns ( FILE * )

16.2.3.48 void classol::read_input_patterns ( FILE * )

16.2.3.49 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

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

16.2.3.51 void classol::read_kcdnsyns ( FILE * )

16.2.3.52 void classol::read_kcdnsyns ( FILE * )

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

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

```
16.2.3.55 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

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

16.2.3.57 void classol::read_pnkcsyns ( FILE * )

16.2.3.58 void classol::read_pnlhisyns ( FILE * )

16.2.3.59 void classol::read_pnlhisyns ( FILE * )

16.2.3.60 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

```
16.2.3.61 void classol::read_pnlhisyns ( FILE * )
```

16.2.3.62 void classol::read_sparsesyns_par (int synlnd, struct SparseProjection C, FILE * f_i nd, FILE * f_i ndlnG, FILE * f_i nd ouble * g_i)

Read sparse connectivity from a file.

Parameters

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

```
16.2.3.63 void classol::read_sparsesyns_par ( int , struct SparseProjection , scalar * , FILE * , FILE * , FILE * )
```

16.2.3.64 void classol::read_sparsesyns_par (int synInd, struct SparseProjection C, scalar * g, FILE * f_ind, FILE * f_indInG, FILE * f_g)

Read sparse connectivity from a file.

Parameters

synInd	index of the synapse population to be worked on
С	contains the arrays to be initialized from file
g	array to receive the conductance values
f_ind	file pointer for the indices of post-synaptic neurons
f_indInG	file pointer for the summed post-synaptic neurons numbers

Parameters

f_g	File handle for a file containing sparse connectivity values
-----	--

```
16.2.3.65 void classol::read_sparsesyns_par(int, struct SparseProjection, scalar*, FILE*, FILE*, FILE*)
```

16.2.3.66 template < class DATATYPE > void classol::read_sparsesyns_par (DATATYPE * wuvar, int synInd, struct SparseProjection C, FILE * f_ind, FILE * f_indInG, FILE * f_g)

Read sparse connectivity from a file.

Parameters

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

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

16.2.3.68 void classol::runCPU (scalar)

16.2.3.69 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

runtimo	Duration of time to run the model for
rununne	Duration of time to run the model for

```
16.2.3.70 void classol::runCPU ( scalar )
```

16.2.3.71 void classol::runCPU (scalar)

16.2.3.72 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
---------	---------------------------------------

```
16.2.3.73 void classol::runGPU ( scalar )
```

16.2.3.74 void classol::runGPU (scalar)

16.2.3.75 void classol::runGPU (scalar)

16.2.3.76 void classol::sum_spikes ()

```
16.2.3.77 void classol::sum_spikes ( )
Method for summing up spike numbers.
16.2.3.78 void classol::sum_spikes ( )
16.2.3.79 void classol::sum_spikes ( )
16.2.3.80 void classol::sum_spikes ( )
16.2.3.81 void classol::write_kcdnsyns ( FILE * f )
Method to write the connectivity between KCs and DNs (detector neurons) to a file.
Parameters
     File handle for a file to write KC to DN (detectore neuron) conductivity values to
16.2.3.82 void classol::write_kcdnsyns ( FILE * )
16.2.3.83 void classol::write_kcdnsyns ( FILE * )
16.2.3.84 void classol::write_kcdnsyns ( FILE * )
16.2.3.85 void classol::write_pnkcsyns ( FILE * )
16.2.3.86 void classol::write_pnkcsyns ( FILE * )
16.2.3.87 void classol::write_pnkcsyns ( FILE * f )
Method for writing the conenctivity between PNs and KCs back into file.
Parameters
     File handle for a file to write PN to KC conductivity values to
16.2.3.88 void classol::write_pnkcsyns ( FILE * )
16.2.3.89 void classol::write_pnlhisyns ( FILE * )
16.2.3.90 void classol::write_pnlhisyns ( FILE * f )
Method for writing the connectivity between PNs and LHIs to a file.
Parameters
     File handle for a file to write PN to LHI conductivity values to
16.2.3.91 void classol::write_pnlhisyns ( FILE * )
16.2.3.92 void classol::write_pnlhisyns ( FILE * )
16.2.4 Member Data Documentation
16.2.4.1 uint64_t * classol::baserates
```

16.2.4.2 uint64_t * classol::d_baserates

```
16.2.4.3 uint64_t * classol::d_pattern
16.2.4.4 NNmodel classol::model
16.2.4.5 unsigned int classol::offset
16.2.4.6 scalar * classol::p_pattern
16.2.4.7 uint64_t * classol::pattern
16.2.4.8 unsigned int classol::size_g
16.2.4.9 unsigned int classol::sumDN
16.2.4.10 unsigned int classol::sumIzh1
16.2.4.11 unsigned int classol::sumKC
16.2.4.12 unsigned int classol::sumLHI
16.2.4.13 unsigned int classol::sumPN
16.2.4.14 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

16.3 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

16.3.1 Constructor & Destructor Documentation

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

16.3.2 Member Function Documentation

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

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

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

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

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

· CodeHelper.h

16.4 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.

16.4.1 Constructor & Destructor Documentation

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

16.4.2 Member Function Documentation

16.4.2.1 double CStopWatch::getElapsedTime ()

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

```
16.4.2.2 void CStopWatch::startTimer ( )
```

This method starts the timer.

16.4.2.3 void CStopWatch::stopTimer ()

This method stops the timer.

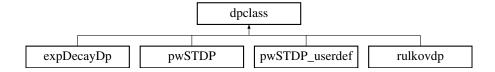
The documentation for this class was generated from the following files:

- hr_time.h
- hr_time.cc

16.5 dpclass Class Reference

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

Inheritance diagram for dpclass:



Public Member Functions

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

16.5.1 Member Function Documentation

16.5.1.1 virtual double dpclass::calculateDerivedParameter (int *index*, vector< double > pars, double dt = 0.5) [inline], [virtual]

Reimplemented in pwSTDP_userdef, rulkovdp, pwSTDP, and expDecayDp.

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

· dpclass.h

16.6 errTupel Struct Reference

Public Attributes

- · unsigned int id
- · double err

16.6.1 Member Data Documentation

16.6.1.1 double errTupel::err

16.6.1.2 unsigned int errTupel::id

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

• GA.cc

16.7 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)
- double expDecay (vector< double > pars, double dt)

16.7.1 Detailed Description

Class defining the dependent parameter for exponential decay.

```
16.7.2 Member Function Documentation
```

```
16.7.2.1 double expDecayDp::calculateDerivedParameter ( int index, vector< double > pars, double dt = 1 . 0 ) [inline], [virtual]
```

Reimplemented from dpclass.

```
16.7.2.2 double expDecayDp::expDecay ( vector< double > pars, double dt ) [inline]
```

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

• postSynapseModels.h

16.8 inputSpec Struct Reference

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

Public Attributes

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

16.8.1 Member Data Documentation

- 16.8.1.1 double inputSpec::baseV
- 16.8.1.2 int inputSpec::N
- 16.8.1.3 vector<double> inputSpec::st
- 16.8.1.4 double inputSpec::t
- $16.8.1.5 \quad vector{<} double{>} inputSpec::V$

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

· helper.h

16.9 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.

bool needPreSt

Whether presynaptic spike times are needed or not.

bool needPostSt

Whether postsynaptic spike times are needed or not.

16.9.1 Detailed Description

class for specifying a neuron model.

16.9.2 Constructor & Destructor Documentation

16.9.2.1 neuronModel::neuronModel()

Constructor for neuronModel objects.

16.9.2.2 neuronModel::∼neuronModel ()

Destructor for neuronModel objects.

16.9.3 Member Data Documentation

16.9.3.1 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.

16.9.3.2 dpclass* neuronModel::dps

Derived parameters.

16.9.3.3 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.

16.9.3.4 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.

16.9.3.5 bool neuronModel::needPostSt

Whether postsynaptic spike times are needed or not.

16.9.3.6 bool neuronModel::needPreSt

Whether presynaptic spike times are needed or not.

16.9.3.7 vector<string> neuronModel::pNames

Names of (independent) parameters of the model.

16.9.3.8 string neuronModel::resetCode

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

16.9.3.9 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.

16.9.3.10 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.

16.9.3.11 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.

```
16.9.3.12 vector<string> neuronModel::tmpVarNames

never used

16.9.3.13 vector<string> neuronModel::tmpVarTypes

never used

16.9.3.14 vector<string> neuronModel::varNames

Names of the variables in the neuron model.

16.9.3.15 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

16.10 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

16.10.1 Constructor & Destructor Documentation

```
16.10.1.1 neuronpop::neuronpop()
16.10.1.2 neuronpop::∼neuronpop()
16.10.2 Member Function Documentation
16.10.2.1 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.

```
16.10.2.2 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.

```
16.10.2.3 void neuronpop::init ( unsigned int which )

16.10.2.4 void neuronpop::output_spikes ( FILE * f, unsigned int which )

16.10.2.5 void neuronpop::output_state ( FILE * f, unsigned int which )

16.10.2.6 void neuronpop::run ( float runtime, unsigned int which )

16.10.2.7 void neuronpop::sum_spikes ( )

16.10.3 Member Data Documentation

16.10.3.1 NNmodel neuronpop::model
```

16.10.3.2 unsigned int neuronpop::sumlzh1

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

- OneComp_model.h
- OneComp_model.cc

16.11 NNmodel Class Reference

```
#include <modelSpec.h>
```

Public Member Functions

- NNmodel ()
- ∼NNmodel ()
- void setName (const 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 checkSizes (unsigned int *, unsigned int *, unsigned int *)
- void setGPUDevice (int)

Method to choose the GPU to be used for the model. If "AUTODEVICE' (-1), GeNN will choose the device based on a heuristic rule.

- string scalarExpr (const double) const
- 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().

• void 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.

• void 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.

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.

unsigned int findNeuronGrp (const string &) const

Find the the ID number of a neuron group by its name.

 void 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.

• void addSynapsePopulation (const string &, unsigned int, SynapseConnType, SynapseGType, unsigned int, unsigned int, const string &, const double *, const double *, const double *)

Overloaded version without initial variables for synapses.

• void 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.

void addSynapsePopulation (const string &, unsigned int, SynapseConnType, SynapseGType, unsigned int, unsigned int, const string &, const vector< double > &)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

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.

- void initLearnGrps ()
- unsigned int findSynapseGrp (const string &) const

This function is a tool to find the numeric ID of a synapse population based on the name of the synapse population.

Public Attributes

· string name

Name of the neuronal newtwork model.

string ftype

Type of floating point variables (float, double, ...; default: float)

string RNtype

Underlying type for random number generation (default: long)

· double dt

The integration time step of the model.

int final

Flag for whether the model has been finalized.

unsigned int needSt

Whether last spike times are needed at all in this network model (related to STDP)

· unsigned int needSynapseDelay

Whether delayed synapse conductance is required in the network.

- · bool timing
- · unsigned int seed
- unsigned int resetKernel

The identity of the kernel in which the spike counters will be reset.

unsigned int neuronGrpN

Number of neuron groups.

vector< string > neuronName

Names of neuron groups.

vector< unsigned int > neuronN

Number of neurons in group.

vector< unsigned int > sumNeuronN

Summed neuron numbers.

vector< unsigned int > padSumNeuronN

Padded summed neuron numbers.

- vector< unsigned int > neuronPostSyn
- vector< unsigned int > neuronType

Postsynaptic methods to the neuron.

vector< vector< double > > neuronPara

Parameters of neurons.

vector< vector< double > > dnp

Derived neuron parameters.

vector< vector< double >> neuronIni

Initial values of neurons.

vector< vector< unsigned int > > inSyn

The ids of the incoming synapse groups.

vector< vector< unsigned int > > outSyn

The ids of the outgoing synapse groups.

vector< bool > neuronNeedSt

Whether last spike time needs to be saved for a group.

vector< bool > neuronNeedTrueSpk

Whether spike-like events from a group are required.

vector< bool > neuronNeedSpkEvnt

Whether spike-like events from a group are required.

vector< vector< bool >> neuronVarNeedQueue

Whether a neuron variable needs queueing for syn code.

vector< set< pair< string, string >>> neuronSpkEvntCondition

Will contain the spike event condition code when spike events are used.

vector< unsigned int > neuronDelaySlots

The number of slots needed in the synapse delay queues of a neuron group.

vector< int > neuronHostID

The ID of the cluster node which the neuron groups are computed on.

vector< int > neuronDeviceID

The ID of the CUDA device which the neuron groups are computed on.

unsigned int synapseGrpN

Number of synapse groups.

vector< string > synapseName

Names of synapse groups.

vector< unsigned int > maxConn

Padded summed maximum number of connections for a neuron in the neuron groups.

- vector< unsigned int > padSumSynapseKrnl
- vector< unsigned int > synapseType

Types of synapses.

vector < SynapseConnType > synapseConnType

Connectivity type of synapses.

vector < SynapseGType > synapseGType

Type of specification method for synaptic conductance.

vector< unsigned int > synapseSpanType

Execution order of synapses in the kernel. It determines whether synapses are executed in parallel for every postsynaptic neuron (0, default), or for every presynaptic neuron (1).

vector< unsigned int > synapseSource

Presynaptic neuron groups.

vector< unsigned int > synapseTarget

Postsynaptic neuron groups.

vector< unsigned int > synapseInSynNo

IDs of the target neurons' incoming synapse variables for each synapse group.

vector< unsigned int > synapseOutSynNo

The target neurons' outgoing synapse for each synapse group.

vector< bool > synapseUsesTrueSpikes

Defines if synapse update is done after detection of real spikes (only one point after threshold)

vector< bool > synapseUsesSpikeEvents

Defines if synapse update is done after detection of spike events (every point above threshold)

vector< bool > synapseUsesPostLearning

Defines if anything is done in case of postsynaptic neuron spiking before presynaptic neuron (punishment in STDP etc.)

vector< bool > synapseUsesSynapseDynamics

Defines if there is any continuos synapse dynamics defined.

vector< bool > needEvntThresholdReTest

Defines whether the Evnt Threshold needs to be retested in the synapse kernel due to multiple non-identical events in the pre-synaptic neuron population.

vector< vector< double >> synapsePara

parameters of synapses

vector< vector< double > > synapselni

Initial values of synapse variables.

• vector< vector< double $>> dsp_w$

Derived synapse parameters (weightUpdateModel only)

vector< unsigned int > postSynapseType

Types of post-synaptic model.

vector< vector< double >> postSynapsePara

parameters of postsynapses

vector< vector< double >> postSynIni

Initial values of postsynaptic variables.

vector< vector< double > > dpsp

Derived postsynapse parameters.

unsigned int IrnGroups

Number of synapse groups with learning.

vector< unsigned int > padSumLearnN

Padded summed neuron numbers of learn group source populations.

vector< unsigned int > IrnSynGrp

Enumeration of the IDs of synapse groups that learn.

vector< unsigned int > synapseDelay

Global synaptic conductance delay for the group (in time steps)

unsigned int synDynGroups

Number of synapse groups that define continuous synapse dynamics.

vector< unsigned int > synDynGrp

Enumeration of the IDs of synapse groups that have synapse Dynamics.

vector< unsigned int > padSumSynDynN

Padded summed neuron numbers of synapse dynamics group source populations.

vector< int > synapseHostID

The ID of the cluster node which the synapse groups are computed on.

vector< int > synapseDeviceID

The ID of the CUDA device which the synapse groups are computed on.

- vector< string > neuronKernelParameters
- vector< string > neuronKernelParameterTypes
- vector< string > synapseKernelParameters
- vector< string > synapseKernelParameterTypes
- vector< string > simLearnPostKernelParameters
- vector< string > simLearnPostKernelParameterTypes
- vector< string > synapseDynamicsKernelParameters
- vector< string > synapseDynamicsKernelParameterTypes

16.11.1 Constructor & Destructor Documentation

```
16.11.1.1 NNmodel::NNmodel ( )
```

16.11.1.2 NNmodel::∼NNmodel ()

16.11.2 Member Function Documentation

16.11.2.1 void NNmodel::activateDirectInput (const string & name, 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

name	Name of the neuron population
type	Type of input: 1 if common input, 2 if custom input from file, 3 if custom input as a rule

16.11.2.2 void 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

	T
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
D	Parameters of this neuron type

Parameters

ini	Initial values for variables of this neuron type
-----	--

16.11.2.3 void NNmodel::addNeuronPopulation (const string & *name*, unsigned int *nNo*, unsigned int *type*, const vector< double > & p, const vector< double > & *ini*)

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

This function adds a neuron population to a neuronal network models, assigning the name, the number of neurons in the group, the neuron type, parameters and initial values. The latter two defined as STL vectors of double.

Parameters

name	The name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
р	Parameters of this neuron type
ini	Initial values for variables of this neuron type

16.11.2.4 void NNmodel::addSynapsePopulation (const string & name, unsigned int syntype, SynapseConnType conntype, SynapseGType gtype, const string & src, const string & target, const double * params)

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
target	Name of the (existing!) post-synaptic neuron population
params	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.

16.11.2.5 void 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

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)
conntype	The type of synaptic connectivity

Parameters

gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.11.2.6 void 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 * ps)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population

This function adds a synapse population to a neuronal network model, assigning the name, the synapse type, the connectivity type, the type of conductance specification, the source and destination neuron populations, and the synaptic parameters.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
synini	A C-type array of doubles that contains the initial values for synapse variables (common to all synapses of the population) which will be used for the defined synapses.
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

16.11.2.7 void 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.
р	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses.

```
16.11.2.8 void NNmodel::checkSizes ( unsigned int * , unsigned int * , unsigned int * )
```

16.11.2.9 void NNmodel::finalize ()

Declare that the model specification is finalised in modelDefinition().

16.11.2.10 unsigned int NNmodel::findNeuronGrp (const string & nName) const

Find the ID number of a neuron group by its name.

This function is a tool to find the numeric ID of a neuron population based on the name of the neuron population.

Parameters

nName	Name of the neuron population
-------	-------------------------------

16.11.2.11 unsigned int NNmodel::findSynapseGrp (const string & sName) const

This function is a tool to find the numeric ID of a synapse population based on the name of the synapse population.

Parameters

sName	Name of the synapse population
-------	--------------------------------

16.11.2.12 void NNmodel::initLearnGrps ()

16.11.2.13 string NNmodel::scalarExpr (const double val) const

16.11.2.14 void NNmodel::setConstInp (const string & sName, double globalInp0)

This function has been deprecated in GeNN 2.2.

This function sets a global input value to the specified neuron group.

16.11.2.15 void NNmodel::setDT (double newDT)

Set the integration step size of the model.

This function sets the integration time step DT of the model.

16.11.2.16 void NNmodel::setGPUDevice (int device)

Method to choose the GPU to be used for the model. If "AUTODEVICE' (-1), GeNN will choose the device based on a heuristic rule.

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.

16.11.2.17 void NNmodel::setMaxConn (const string & sname, unsigned int maxConnP)

This function defines the maximum number of connections for a neuron in the population.

16.11.2.18 void NNmodel::setName (const string inname)

Method to set the neuronal network model name.

16.11.2.19 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

16.11.2.20 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.

16.11.2.21 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.

16.11.2.22 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

16.11.2.23 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
-------	--

16.11.2.24 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

16.11.2.25 void NNmodel::setSynapseG (const string & sName, double g)

This function has been depreciated as of GeNN 2.2.

This functions sets the global value of the maximal synaptic conductance for a synapse population that was idfentified as conductance specification method "GLOBALG".

16.11.2.26 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.

16.11.3 Member Data Documentation

 $16.11.3.1 \quad vector < vector < double >> NNmodel::dnp$

Derived neuron parameters.

16.11.3.2 vector<vector<double> > NNmodel::dpsp

Derived postsynapse parameters.

16.11.3.3 vector<vector<double>> NNmodel::dsp_w

Derived synapse parameters (weightUpdateModel only)

16.11.3.4 double NNmodel::dt

The integration time step of the model.

16.11.3.5 int NNmodel::final

Flag for whether the model has been finalized.

16.11.3.6 string NNmodel::ftype

Type of floating point variables (float, double, ...; default: float)

16.11.3.7 vector<vector<unsigned int>> NNmodel::inSyn

The ids of the incoming synapse groups.

16.11.3.8 unsigned int NNmodel::IrnGroups

Number of synapse groups with learning.

16.11.3.9 vector<unsigned int> NNmodel::IrnSynGrp

Enumeration of the IDs of synapse groups that learn.

16.11.3.10 vector<unsigned int> NNmodel::maxConn

Padded summed maximum number of connections for a neuron in the neuron groups.

16.11.3.11 string NNmodel::name

Name of the neuronal newtwork model.

16.11.3.12 vector
bool> NNmodel::needEvntThresholdReTest

Defines whether the Evnt Threshold needs to be retested in the synapse kernel due to multiple non-identical events in the pre-synaptic neuron population.

16.11.3.13 unsigned int NNmodel::needSt

Whether last spike times are needed at all in this network model (related to STDP)

16.11.3.14 unsigned int NNmodel::needSynapseDelay

Whether delayed synapse conductance is required in the network.

 $16.11.3.15 \quad \text{vector}{<} \text{unsigned int}{>} \, \text{NNmodel}{::} \text{neuronDelaySlots}$

The number of slots needed in the synapse delay queues of a neuron group.

16.11.3.16 vector<int> NNmodel::neuronDeviceID

The ID of the CUDA device which the neuron groups are computed on.

16.11.3.17 unsigned int NNmodel::neuronGrpN

Number of neuron groups.

16.11.3.18 vector<int> NNmodel::neuronHostID

The ID of the cluster node which the neuron groups are computed on.

 $16.11.3.19 \quad vector {<} vector {<} double {>} > NNmodel::neuronIni$

Initial values of neurons.

16.11.3.20 vector<string> NNmodel::neuronKernelParameters

16.11.3.21 vector<string> NNmodel::neuronKernelParameterTypes

16.11.3.22 vector<unsigned int> NNmodel::neuronN

Number of neurons in group.

16.11.3.23 vector<string> NNmodel::neuronName

Names of neuron groups.

16.11.3.24 vector
bool> NNmodel::neuronNeedSpkEvnt

Whether spike-like events from a group are required.

16.11.3.25 vector
bool> NNmodel::neuronNeedSt

Whether last spike time needs to be saved for a group.

16.11.3.26 vector
bool> NNmodel::neuronNeedTrueSpk

Whether spike-like events from a group are required.

16.11.3.27 vector<vector<double> > NNmodel::neuronPara

Parameters of neurons.

16.11.3.28 vector<unsigned int> NNmodel::neuronPostSyn

16.11.3.29 vector<set<pair<string>>> NNmodel::neuronSpkEvntCondition

Will contain the spike event condition code when spike events are used.

16.11.3.30 vector<unsigned int> NNmodel::neuronType

Postsynaptic methods to the neuron.

Types of neurons

16.11.3.31 vector<vector
bool> > NNmodel::neuronVarNeedQueue

Whether a neuron variable needs queueing for syn code.

16.11.3.32 vector<vector<unsigned int>> NNmodel::outSyn

The ids of the outgoing synapse groups.

16.11.3.33 vector<unsigned int> NNmodel::padSumLearnN

Padded summed neuron numbers of learn group source populations.

16.11.3.34 vector < unsigned int > NNmodel::padSumNeuronN

Padded summed neuron numbers.

16.11.3.35 vector<unsigned int> NNmodel::padSumSynapseKrnl

16.11.3.36 vector<unsigned int> NNmodel::padSumSynDynN

Padded summed neuron numbers of synapse dynamics group source populations.

16.11.3.37 vector<vector<double> > NNmodel::postSynapsePara

parameters of postsynapses

16.11.3.38 vector<unsigned int> NNmodel::postSynapseType

Types of post-synaptic model.

16.11.3.39 vector<vector<double>> NNmodel::postSynIni

Initial values of postsynaptic variables.

16.11.3.40 unsigned int NNmodel::resetKernel

The identity of the kernel in which the spike counters will be reset.

16.11.3.41 string NNmodel::RNtype

Underlying type for random number generation (default: long)

16.11.3.42 unsigned int NNmodel::seed

16.11.3.43 vector<string> NNmodel::simLearnPostKernelParameters

16.11.3.44 vector < string > NNmodel::simLearnPostKernelParameterTypes

16.11.3.45 vector<unsigned int> NNmodel::sumNeuronN

Summed neuron numbers.

16.11.3.46 vector < SynapseConnType > NNmodel::synapseConnType

Connectivity type of synapses.

16.11.3.47 vector<unsigned int> NNmodel::synapseDelay

Global synaptic conductance delay for the group (in time steps)

16.11.3.48 vector<int> NNmodel::synapseDeviceID

The ID of the CUDA device which the synapse groups are computed on.

16.11.3.49 vector<string> NNmodel::synapseDynamicsKernelParameters

16.11.3.50 vector<string> NNmodel::synapseDynamicsKernelParameterTypes

16.11.3.51 unsigned int NNmodel::synapseGrpN

Number of synapse groups.

16.11.3.52 vector < Synapse GType > NNmodel::synapse GType

Type of specification method for synaptic conductance.

16.11.3.53 vector<int> NNmodel::synapseHostID

The ID of the cluster node which the synapse groups are computed on.

16.11.3.54 vector<vector<double>> NNmodel::synapselni

Initial values of synapse variables.

 $16.11.3.55 \quad \text{vector}{<} \text{unsigned int}{>} \, \text{NNmodel::synapseInSynNo}$

IDs of the target neurons' incoming synapse variables for each synapse group.

16.11.3.56 vector < string > NNmodel::synapseKernelParameters

16.11.3.57 vector<string> NNmodel::synapseKernelParameterTypes

16.11.3.58 vector < string > NNmodel::synapseName

Names of synapse groups.

16.11.3.59 vector<unsigned int> NNmodel::synapseOutSynNo

The target neurons' outgoing synapse for each synapse group.

16.11.3.60 vector<vector<double>> NNmodel::synapsePara

parameters of synapses

16.11.3.61 vector<unsigned int> NNmodel::synapseSource

Presynaptic neuron groups.

16.11.3.62 vector<unsigned int> NNmodel::synapseSpanType

Execution order of synapses in the kernel. It determines whether synapses are executed in parallel for every postsynaptic neuron (0, default), or for every presynaptic neuron (1).

16.11.3.63 vector<unsigned int> NNmodel::synapseTarget

Postsynaptic neuron groups.

16.11.3.64 vector < unsigned int > NNmodel::synapseType

Types of synapses.

16.11.3.65 vector < bool > NNmodel::synapseUsesPostLearning

Defines if anything is done in case of postsynaptic neuron spiking before presynaptic neuron (punishment in STDP etc.)

16.11.3.66 vector
bool> NNmodel::synapseUsesSpikeEvents

Defines if synapse update is done after detection of spike events (every point above threshold)

16.11.3.67 vector < bool > NN model::synapseUsesSynapseDynamics

Defines if there is any continuos synapse dynamics defined.

16.11.3.68 vector
bool> NNmodel::synapseUsesTrueSpikes

Defines if synapse update is done after detection of real spikes (only one point after threshold)

16.11.3.69 unsigned int NNmodel::synDynGroups

Number of synapse groups that define continuous synapse dynamics.

16.11.3.70 vector<unsigned int> NNmodel::synDynGrp

Enumeration of the IDs of synapse groups that have synapse Dynamics.

16.11.3.71 bool NNmodel::timing

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

- · modelSpec.h
- · src/modelSpec.cc

16.12 Parameter Struct Reference

Public Attributes

- · string name
- · string value

16.12.1 Member Data Documentation

16.12.1.1 string Parameter::name

16.12.1.2 string Parameter::value

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

experiment.cc

16.13 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.

16.13.1 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".

16.13.2 Constructor & Destructor Documentation

16.13.2.1 postSynModel::postSynModel()

Constructor for postSynModel objects.

16.13.2.2 postSynModel::~postSynModel()

Destructor for postSynModel objects.

16.13.3 Member Data Documentation

16.13.3.1 vector<string> postSynModel::dpNames

Names of dependent parameters of the model.

16.13.3.2 dpclass* postSynModel::dps

Derived parameters.

16.13.3.3 vector<string> postSynModel::pNames

Names of (independent) parameters of the model.

16.13.3.4 string postSynModel::postSynDecay

Code that defines how postsynaptic current decays.

16.13.3.5 string postSynModel::postSyntoCurrent

Code that defines how postsynaptic update is translated to current.

16.13.3.6 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.

16.13.3.7 vector<string> postSynModel::varNames

Names of the variables in the postsynaptic model.

16.13.3.8 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

16.14 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 dt=1.0)
- double lim0 (vector< double > pars, double dt)
- double lim1 (vector< double > pars, double dt)
- double slope0 (vector< double > pars, double dt)
- double slope1 (vector< double > pars, double dt)
- double off0 (vector< double > pars, double dt)
- double off1 (vector< double > pars, double dt)
- double off2 (vector< double > pars, double dt)

16.14.1 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

16.14.2 Member Function Documentation

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

Reimplemented from dpclass.

```
16.14.2.2 double pwSTDP::lim0 ( vector < double > pars, double dt ) [inline]
16.14.2.3 double pwSTDP::lim1 ( vector < double > pars, double dt ) [inline]
16.14.2.4 double pwSTDP::off0 ( vector < double > pars, double dt ) [inline]
16.14.2.5 double pwSTDP::off1 ( vector < double > pars, double dt ) [inline]
16.14.2.6 double pwSTDP::off2 ( vector < double > pars, double dt ) [inline]
16.14.2.7 double pwSTDP::slope0 ( vector < double > pars, double dt ) [inline]
```

```
16.14.2.8 double pwSTDP::slope1 (vector < double > pars, double dt ) [inline]
```

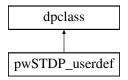
The documentation for this class was generated from the following file:

synapseModels.h

16.15 pwSTDP_userdef Class Reference

TODO This class definition may be code-generated in a future release.

Inheritance diagram for pwSTDP userdef:



Public Member Functions

- double calculateDerivedParameter (int index, vector< double > pars, double dt)
- double lim0 (vector< double > pars, double dt)
- double lim1 (vector< double > pars, double dt)
- double slope0 (vector< double > pars, double dt)
- double slope1 (vector< double > pars, double dt)
- double off0 (vector< double > pars, double dt)
- double off1 (vector< double > pars, double dt)
- double off2 (vector< double > pars, double dt)

16.15.1 Detailed Description

TODO This class definition may be code-generated in a future release.

16.15.2 Member Function Documentation

```
16.15.2.1 double pwSTDP_userdef::calculateDerivedParameter ( int index, vector < double > pars, double dt ) [inline], [virtual]
```

Reimplemented from dpclass.

```
16.15.2.2 double pwSTDP_userdef::lim0 ( vector< double > pars, double dt ) [inline]
16.15.2.3 double pwSTDP_userdef::lim1 ( vector< double > pars, double dt ) [inline]
16.15.2.4 double pwSTDP_userdef::off0 ( vector< double > pars, double dt ) [inline]
16.15.2.5 double pwSTDP_userdef::off1 ( vector< double > pars, double dt ) [inline]
16.15.2.6 double pwSTDP_userdef::off2 ( vector< double > pars, double dt ) [inline]
16.15.2.7 double pwSTDP_userdef::slope0 ( vector< double > pars, double dt ) [inline]
16.15.2.8 double pwSTDP_userdef::slope1 ( vector< double > pars, double dt ) [inline]
```

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

• MBody_userdef.cc

16.16 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)
- 16.16.1 Member Typedef Documentation
- 16.16.1.1 template < int ALPHA = (8), class T = ISAAC_INT> typedef unsigned char QTIsaac < ALPHA, T >::byte
- 16.16.2 Member Enumeration Documentation
- 16.16.2.1 template < int ALPHA = (8), class T = ISAAC_INT> anonymous enum

Enumerator

N

- 16.16.3 Constructor & Destructor Documentation
- 16.16.3.1 template < int ALPHA, class T> QTIsaac < ALPHA, T>::QTIsaac (T a = 0, T b = 0, T c = 0)
- 16.16.3.2 template < int ALPHA, class T > QTIsaac < ALPHA, T >:: ~QTIsaac (void) [virtual]
- 16.16.4 Member Function Documentation
- 16.16.4.1 template < int ALPHA, class T > T QTIsaac < ALPHA, T >::ind (T * mm, T x) [inline], [protected]
- 16.16.4.2 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::isaac (randctx * ctx) [protected], [virtual]
- 16.16.4.3 template < int ALPHA, class T > T QTIsaac < ALPHA, T >::rand (void) [inline]
- 16.16.4.4 template < int ALPHA, class T > void QTIsaac < ALPHA, T >::randinit (randctx * ctx, bool bUseSeed) [virtual]

```
16.16.4.5 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]
```

```
16.16.4.6 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]
```

```
16.16.4.7 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

16.17 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

16.17.1 Constructor & Destructor Documentation

```
16.17.1.1 template < int ALPHA = (8), class T = ISAAC_INT> QTIsaac < ALPHA, T >::randctx::randctx ( void ) [inline]
```

```
16.17.1.2 template < int ALPHA = (8), class T = ISAAC_INT> QTIsaac < ALPHA, T >::randctx::~randctx ( void ) [inline]
```

16.17.2 Member Data Documentation

```
16.17.2.1 \quad template < int ALPHA = (8), class \ T = ISAAC\_INT > T \ QTIsaac < ALPHA, \ T > :: randctx:: randa
```

- 16.17.2.2 template < int ALPHA = (8), class T = ISAAC_INT> T QTIsaac < ALPHA, T >::randctx::randb
- 16.17.2.3 template < int ALPHA = (8), class T = ISAAC_INT> T QTIsaac < ALPHA, T >::randctx::randc
- 16.17.2.4 template < int ALPHA = (8), class T = ISAAC_INT> T QTIsaac < ALPHA, T >::randctx::randcnt
- 16.17.2.5 template<int ALPHA = (8), class T = ISAAC_INT> T* QTIsaac< ALPHA, T >::randctx::randmem
- 16.17.2.6 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

16.18 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.

16.18.1 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().

16.18.2 Constructor & Destructor Documentation

```
16.18.2.1 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.

```
16.18.2.2 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.

```
16.18.2.3 randomGauss::∼randomGauss( ) [inline]16.18.3 Member Function Documentation16.18.3.1 double randomGauss::n( )
```

Method for obtaining a random number with Gaussian distribution.

Function for generating a pseudo random number from a Gaussian distribution.

16.18.3.2 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

16.19 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.

16.19.1 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.

16.19.2 Constructor & Destructor Documentation

```
16.19.2.1 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.

```
16.19.2.2 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.

```
16.19.2.3 randomGen::∼randomGen( ) [inline]
16.19.3 Member Function Documentation
```

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].

16.19.3.2 void randomGen::srand (unsigned long seed1, unsigned long seed2, unsigned long seed3)

Function for seeding with fixed seeds.

16.19.3.1 double randomGen::n ()

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

- · randomGen.h
- randomGen.cc

16.20 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 dt=1.0)
- double ip0 (vector< double > pars)
- double ip1 (vector< double > pars)
- double ip2 (vector< double > pars)

16.20.1 Detailed Description

Class defining the dependent parameters of the Rulkov map neuron.

16.20.2 Member Function Documentation

16.20.2.1 double rulkovdp::calculateDerivedParameter (int *index*, vector < double > pars, double dt = 1.0) [inline], [virtual]

Reimplemented from dpclass.

```
16.20.2.2 double rulkovdp::ip0 ( vector < double > pars ) [inline]
16.20.2.3 double rulkovdp::ip1 ( vector < double > pars ) [inline]
16.20.2.4 double rulkovdp::ip2 ( vector < double > pars ) [inline]
```

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

• neuronModels.h

16.21 Schmuker2014_classifier Class Reference

This class cpontains the methods for running the Schmuker_2014_classifier example model.

```
#include <Schmuker2014_classifier.h>
```

Public Types

enum data_type { data_type_int, data_type_uint, data_type_float, data_type_double }

Public Member Functions

- Schmuker2014 classifier ()
- ∼Schmuker2014 classifier ()
- · void allocateHostAndDeviceMemory ()
- void populateDeviceMemory ()
- void update_input_data_on_device ()
- void clearDownDevice ()
- void run (float runtime, string filename_rasterPlot, bool usePlasticity)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void outputSpikes (FILE *, string delim)
- void initialiseWeights_SPARSE_RN_PN ()
- void initialiseWeights_WTA_PN_PN ()
- void initialiseWeights DENSE PN AN ()
- void initialiseWeights WTA AN AN ()
- void createWTAConnectivity (float *synapse, UINT populationSize, UINT clusterSize, float synapseWeight, float probability)
- bool randomEventOccurred (float probability)
- void updateWeights_PN_AN_on_device ()
- · void generate or load inputrates dataset (unsigned int recordingIdx)
- void generate_inputrates_dataset (unsigned int recordingldx)
- FILE * openRecordingFile (UINT recordingIndex)
- void applyLearningRuleSynapses (float *synapsesPNAN)
- void initialiseInputData ()
- void load VR data ()
- void setCorrectClass (UINT recordingIdx)
- UINT getClassCluster (UINT anldx)
- void loadClassLabels ()
- void addInputRate (float *samplePoint, UINT timeStep)
- uint64_t convertToRateCode (float inputRateHz)
- float calculateVrResponse (float *samplePoint, float *vrPoint)
- void setMaxMinSampleDistances ()
- void findMaxMinSampleDistances (float *samples, UINT startAt, UINT totalSamples)
- float getSampleDistance (UINT max0_min1)
- float getManhattanDistance (float *pointA, float *pointB, UINT numElements)
- float getRand0to1 ()
- UINT calculateOverallWinner ()
- UINT calculateWinner (unsigned int *clusterSpikeCount)
- UINT calculateCurrentWindowWinner ()
- · void updateIndividualSpikeCountPN ()
- · void resetIndividualSpikeCountPN ()
- void updateClusterSpikeCountAN ()
- void resetClusterSpikeCountAN ()
- void resetOverallWinner ()
- void updateWeights_PN_AN ()
- UINT getClusterIndex (UINT neuronIndex, UINT clusterSize)
- void generateSimulatedTimeSeriesData ()
- string getRecordingFilename (UINT recordingIdx)
- bool loadArrayFromTextFile (string path, void *array, string delim, UINT arrayLen, data_type dataType)
- void checkContents (string title, void *array, UINT howMany, UINT displayPerLine, data_type dataType, UINT decimalPoints)
- void checkContents (string title, void *array, UINT howMany, UINT displayPerLine, data_type dataType, UINT decimalPoints, string delim)
- void printSeparator ()
- void resetDevice ()
- void startLog ()

Public Attributes

- double d_maxRandomNumber
- NNmodel model
- uint64_t * inputRates
- · unsigned int inputRatesSize
- float * vrData
- unsigned int * classLabel
- unsigned int * individualSpikeCountPN
- unsigned int * overallWinnerSpikeCountAN
- unsigned int * clusterSpikeCountAN
- float * plasticWeights
- uint64_t * d_inputRates
- unsigned int countRN
- · unsigned int countPN
- · unsigned int countAN
- unsigned int countPNAN
- float * sampleDistance
- · string recordingsDir
- string cacheDir
- · string outputDir
- · string datasetName
- · string uniqueRunId
- UINT correctClass
- int winningClass
- FILE * log
- UINT param_SPIKING_ACTIVITY_THRESHOLD_HZ
- UINT param_MAX_FIRING_RATE_HZ
- · UINT param MIN FIRING RATE HZ
- float param_GLOBAL_WEIGHT_SCALING
- · float param_WEIGHT_RN_PN
- float param_CONNECTIVITY_RN_PN
- float param_WEIGHT_WTA_PN_PN
- float param WEIGHT WTA AN AN
- float param_CONNECTIVITY_PN_PN
- · float param_CONNECTIVITY_AN_AN
- float param CONNECTIVITY PN AN
- float param MIN WEIGHT PN AN
- float param_MAX_WEIGHT_PN_AN
- · float param WEIGHT DELTA PN AN
- float param_PLASTICITY_INTERVAL_MS
- · bool clearedDownDevice

Static Public Attributes

• static const unsigned int timestepsPerRecording = RECORDING_TIME_MS / DT

16.21.1 Detailed Description

This class cpontains the methods for running the Schmuker_2014_classifier example model.

```
16.21.2 Member Enumeration Documentation
16.21.2.1 enum Schmuker2014_classifier::data_type
Enumerator
      data_type_int
      data_type_uint
      data_type_float
      data_type_double
16.21.3 Constructor & Destructor Documentation
16.21.3.1 Schmuker2014_classifier::Schmuker2014_classifier()
16.21.3.2 Schmuker2014_classifier:: ~Schmuker2014_classifier ( )
16.21.4 Member Function Documentation
16.21.4.1 void Schmuker2014_classifier::addInputRate ( float * samplePoint, UINT timeStep )
16.21.4.2 void Schmuker2014_classifier::allocateHostAndDeviceMemory ( )
16.21.4.3 void Schmuker2014_classifier::applyLearningRuleSynapses ( float * synapsesPNAN )
16.21.4.4 UINT Schmuker2014_classifier::calculateCurrentWindowWinner()
16.21.4.5 UINT Schmuker2014_classifier::calculateOverallWinner( )
16.21.4.6 float Schmuker2014 classifier::calculateVrResponse ( float * samplePoint, float * vrPoint )
16.21.4.7 UINT Schmuker2014_classifier::calculateWinner ( unsigned int * clusterSpikeCount )
16.21.4.8 void Schmuker2014_classifier::checkContents ( string title, void * array, UINT howMany, UINT displayPerLine,
          data_type dataType, UINT decimalPoints )
16.21.4.9 void Schmuker2014_classifier::checkContents ( string title, void * array, UINT howMany, UINT displayPerLine,
          data_type dataType, UINT decimalPoints, string delim )
16.21.4.10 void Schmuker2014_classifier::clearDownDevice ( )
16.21.4.11 uint64_t Schmuker2014_classifier::convertToRateCode ( float inputRateHz )
16.21.4.12 void Schmuker2014_classifier::createWTAConnectivity ( float * synapse, UINT populationSize, UINT clusterSize,
           float synapseWeight, float probability )
16.21.4.13 void Schmuker2014_classifier::findMaxMinSampleDistances ( float * samples, UINT startAt, UINT totalSamples )
16.21.4.14 void Schmuker2014_classifier::generate_inputrates_dataset ( unsigned int recordingldx )
16.21.4.15 void Schmuker2014_classifier::generate_or_load_inputrates_dataset ( unsigned int recordingldx )
16.21.4.16 void Schmuker2014_classifier::generateSimulatedTimeSeriesData ( )
16.21.4.17 UINT Schmuker2014_classifier::getClassCluster ( UINT anldx )
16.21.4.18 UINT Schmuker2014_classifier::getClusterIndex ( UINT neuronIndex, UINT clusterSize )
16.21.4.19 float Schmuker2014_classifier::getManhattanDistance (float * pointA, float * pointB, UINT numElements)
```

```
16.21.4.20
           float Schmuker2014_classifier::getRand0to1 ( )
16.21.4.21
           string Schmuker2014_classifier::getRecordingFilename ( UINT recordingIdx )
16.21.4.22
           float Schmuker2014_classifier::getSampleDistance ( UINT max0_min1 )
16.21.4.23
           void Schmuker2014_classifier::getSpikeNumbersFromGPU ( )
16.21.4.24
           void Schmuker2014_classifier::getSpikesFromGPU ( )
16.21.4.25 void Schmuker2014_classifier::initialiseInputData ( )
16.21.4.26 void Schmuker2014_classifier::initialiseWeights_DENSE_PN_AN ( )
16.21.4.27 void Schmuker2014_classifier::initialiseWeights_SPARSE_RN_PN ( )
16.21.4.28 void Schmuker2014_classifier::initialiseWeights_WTA_AN_AN ( )
16.21.4.29 void Schmuker2014_classifier::initialiseWeights_WTA_PN_PN()
16.21.4.30 void Schmuker2014_classifier::load_VR_data( )
           bool Schmuker2014_classifier::loadArrayFromTextFile ( string path, void * array, string delim, UINT arrayLen,
16.21.4.31
           data_type dataType )
16.21.4.32 void Schmuker2014_classifier::loadClassLabels ( )
16.21.4.33 FILE * Schmuker2014_classifier::openRecordingFile ( UINT recordingIndex )
16.21.4.34 void Schmuker2014_classifier::outputSpikes (FILE * f, string delim)
16.21.4.35 void Schmuker2014_classifier::populateDeviceMemory ( )
16.21.4.36 void Schmuker2014_classifier::printSeparator()
16.21.4.37 bool Schmuker2014_classifier::randomEventOccurred ( float probability )
16.21.4.38 void Schmuker2014_classifier::resetClusterSpikeCountAN ( )
16.21.4.39 void Schmuker2014_classifier::resetDevice ( )
16.21.4.40 void Schmuker2014_classifier::resetIndividualSpikeCountPN()
16.21.4.41 void Schmuker2014_classifier::resetOverallWinner()
16.21.4.42 void Schmuker2014_classifier::run ( float runtime, string filename_rasterPlot, bool usePlasticity )
16.21.4.43 void Schmuker2014_classifier::setCorrectClass ( UINT recordingldx )
16.21.4.44 void Schmuker2014_classifier::setMaxMinSampleDistances ( )
16.21.4.45 void Schmuker2014_classifier::startLog ( )
16.21.4.46 void Schmuker2014_classifier::update_input_data_on_device ( )
16.21.4.47 void Schmuker2014_classifier::updateClusterSpikeCountAN ( )
16.21.4.48 void Schmuker2014_classifier::updateIndividualSpikeCountPN ( )
16.21.4.49 void Schmuker2014_classifier::updateWeights_PN_AN ( )
16.21.4.50 void Schmuker2014_classifier::updateWeights_PN_AN_on_device ( )
```

16.21.5 Member Data Documentation

6.21.5.1	string Schmuker2014_classifier::cacheDir
6.21.5.2	unsigned int* Schmuker2014_classifier::classLabel
6.21.5.3	bool Schmuker2014_classifier::clearedDownDevice
6.21.5.4	unsigned int* Schmuker2014_classifier::clusterSpikeCountAN
6.21.5.5	UINT Schmuker2014_classifier::correctClass
6.21.5.6	unsigned int Schmuker2014_classifier::countAN
6.21.5.7	unsigned int Schmuker2014_classifier::countPN
6.21.5.8	unsigned int Schmuker2014_classifier::countPNAN
6.21.5.9	unsigned int Schmuker2014_classifier::countRN
6.21.5.10	uint64_t* Schmuker2014_classifier::d_inputRates
6.21.5.11	double Schmuker2014_classifier::d_maxRandomNumber
6.21.5.12	string Schmuker2014_classifier::datasetName
6.21.5.13	unsigned int* Schmuker2014_classifier::individualSpikeCountPN
6.21.5.14	uint64_t* Schmuker2014_classifier::inputRates
6.21.5.15	unsigned int Schmuker2014_classifier::inputRatesSize
6.21.5.16	FILE* Schmuker2014_classifier::log
6.21.5.17	NNmodel Schmuker2014_classifier::model
6.21.5.18	string Schmuker2014_classifier::outputDir
6.21.5.19	unsigned int* Schmuker2014_classifier::overallWinnerSpikeCountAN
6.21.5.20	float Schmuker2014_classifier::param_CONNECTIVITY_AN_AN
6.21.5.21	float Schmuker2014_classifier::param_CONNECTIVITY_PN_AN
6.21.5.22	float Schmuker2014_classifier::param_CONNECTIVITY_PN_PN
6.21.5.23	float Schmuker2014_classifier::param_CONNECTIVITY_RN_PN
6.21.5.24	float Schmuker2014_classifier::param_GLOBAL_WEIGHT_SCALING
6.21.5.25	UINT Schmuker2014_classifier::param_MAX_FIRING_RATE_HZ
6.21.5.26	float Schmuker2014_classifier::param_MAX_WEIGHT_PN_AN
6.21.5.27	UINT Schmuker2014_classifier::param_MIN_FIRING_RATE_HZ
6.21.5.28	float Schmuker2014_classifier::param_MIN_WEIGHT_PN_AN
6.21.5.29	float Schmuker2014_classifier::param_PLASTICITY_INTERVAL_MS
6.21.5.30	UINT Schmuker2014_classifier::param_SPIKING_ACTIVITY_THRESHOLD_HZ

```
16.21.5.31 float Schmuker2014_classifier::param_WEIGHT_DELTA_PN_AN

16.21.5.32 float Schmuker2014_classifier::param_WEIGHT_RN_PN

16.21.5.33 float Schmuker2014_classifier::param_WEIGHT_WTA_AN_AN

16.21.5.34 float Schmuker2014_classifier::param_WEIGHT_WTA_PN_PN

16.21.5.35 float* Schmuker2014_classifier::plasticWeights

16.21.5.36 string Schmuker2014_classifier::recordingsDir

16.21.5.37 float* Schmuker2014_classifier::sampleDistance

16.21.5.38 const unsigned int Schmuker2014_classifier::timestepsPerRecording = RECORDING_TIME_MS / DT [static]

16.21.5.39 string Schmuker2014_classifier::uniqueRunld

16.21.5.40 float* Schmuker2014_classifier::vrData

16.21.5.41 int Schmuker2014_classifier::winningClass
```

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

- · Schmuker2014 classifier.h
- Schmuker2014_classifier.cc

16.22 SimulationNeuronPolicyPrePostVar Class Reference

```
#include <simulation_neuron_policy_pre_post_var.h>
```

Public Member Functions

· void Init ()

16.22.1 Member Function Documentation

```
16.22.1.1 void SimulationNeuronPolicyPrePostVar::Init() [inline]
```

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

simulation_neuron_policy_pre_post_var.h

16.23 SimulationNeuronPolicyPreVar Class Reference

```
#include <simulation_neuron_policy_pre_var.h>
```

Public Member Functions

• void Init ()

16.23.1 Member Function Documentation

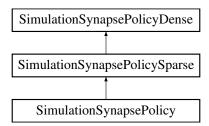
16.23.1.1 void SimulationNeuronPolicyPreVar::Init() [inline]

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

• simulation_neuron_policy_pre_var.h

16.24 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

16.24.1 Member Function Documentation

16.24.1.1 void SimulationSynapsePolicy::Init() [inline]

16.24.1.2 template<typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicy::Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn) [inline]

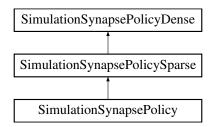
The documentation for this class was generated from the following file:

extra_global_params_in_sim_code_event_spare_inv/test.cc

16.25 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)

16.25.1 Member Function Documentation

```
16.25.1.1 float* SimulationSynapsePolicyDense::GetTheW (unsigned int delay ) const [inline], [protected]
```

```
16.25.1.2 void SimulationSynapsePolicyDense::Init() [inline]
```

- 16.25.1.4 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

16.26 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)

16.26.1 Member Function Documentation

```
16.26.1.1 void SimulationSynapsePolicyNone::Init() [inline]
```

16.26.1.2 template<typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicyNone::Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn) [inline]

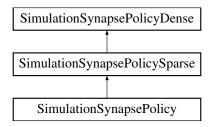
The documentation for this class was generated from the following file:

• simulation_synapse_policy_none.h

16.27 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

16.27.1 Member Function Documentation

- 16.27.1.1 void SimulationSynapsePolicySparse::Init() [inline]
- 16.27.1.2 template < typename UpdateFn , typename StepGeNNFn > float SimulationSynapsePolicySparse::Simulate (UpdateFn updateFn, StepGeNNFn stepGeNNFn) [inline]

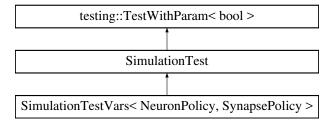
The documentation for this class was generated from the following file:

• simulation_synapse_policy_sparse.h

16.28 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 ()

16.28.1 Member Function Documentation

16.28.1.1 virtual void SimulationTest::Init() [protected], [pure virtual]

Implemented in SimulationTestVars< NeuronPolicy, SynapsePolicy >.

```
16.28.1.2 virtual void SimulationTest::SetUp( ) [inline], [protected], [virtual]
16.28.1.3 void SimulationTest::StepGeNN( ) [inline], [protected]
16.28.1.4 virtual void SimulationTest::TearDown( ) [inline], [protected], [virtual]
```

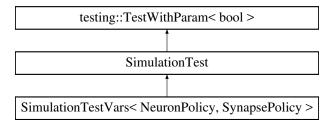
The documentation for this class was generated from the following file:

• simulation_test.h

16.29 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)

16.29.1 Member Function Documentation

16.29.1.1 template<typename NeuronPolicy, typename SynapsePolicy > virtual void SimulationTestVars< NeuronPolicy, SynapsePolicy >::Init() [inline], [protected], [virtual]

Implements SimulationTest.

16.29.1.2 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

16.30 SparseProjection Struct Reference

class (struct) for defining a spars connectivity projection

```
#include <sparseProjection.h>
```

Public Attributes

- unsigned int * indlnG
- unsigned int * ind

- unsigned int * preInd
- unsigned int * revIndInG
- unsigned int * revInd
- unsigned int * remap
- unsigned int connN

16.30.1 Detailed Description

class (struct) for defining a spars connectivity projection

16.30.2 Member Data Documentation

```
16.30.2.1 unsigned int SparseProjection::connN
```

- 16.30.2.2 unsigned int* SparseProjection::ind
- 16.30.2.3 unsigned int* SparseProjection::indlnG
- 16.30.2.4 unsigned int* SparseProjection::preInd
- 16.30.2.5 unsigned int* SparseProjection::remap
- 16.30.2.6 unsigned int* SparseProjection::revInd
- 16.30.2.7 unsigned int* SparseProjection::revIndInG

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

• sparseProjection.h

16.31 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 ()

16.31.1 Constructor & Destructor Documentation

```
16.31.1.1 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.

```
16.31.1.2 stdRG::stdRG ( unsigned int seed )
```

Constructor of the standard random number generator class with explicit seed.

The seed is an arbitrary unsigned int

```
16.31.1.3 stdRG::~stdRG( ) [inline]
16.31.2 Member Function Documentation
16.31.2.1 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[

```
16.31.2.2 unsigned long stdRG::nlong ( )
```

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

- · randomGen.h
- · randomGen.cc

16.32 stopWatch Struct Reference

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

Public Attributes

- timeval start
- timeval stop

16.32.1 Member Data Documentation

16.32.1.1 timeval stopWatch::start

16.32.1.2 timeval stopWatch::stop

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

• hr_time.h

16.33 SynDelay Class Reference

```
#include <SynDelaySim.h>
```

Public Member Functions

- SynDelay (bool usingGPU)
- ∼SynDelay ()
- void run (float t)

16.33.1 Constructor & Destructor Documentation

```
16.33.1.1 SynDelay::SynDelay (bool usingGPU)
```

16.33.1.2 SynDelay::∼SynDelay ()

16.33.2 Member Function Documentation

16.33.2.1 void SynDelay::run (float *t*)

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

- SynDelaySim.h
- SynDelaySim.cc

16.34 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.

16.34.1 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.

```
16.34.2 Constructor & Destructor Documentation
```

16.34.2.1 weightUpdateModel::weightUpdateModel()

Constructor for weightUpdateModel objects.

16.34.2.2 weightUpdateModel::~weightUpdateModel()

Destructor for weightUpdateModel objects.

16.34.3 Member Data Documentation

16.34.3.1 vector<string> weightUpdateModel::dpNames

Names of dependent parameters of the model.

16.34.3.2 dpclass* weightUpdateModel::dps

16.34.3.3 string weightUpdateModel::evntThreshold

Simulation code for spike event detection.

16.34.3.4 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.

16.34.3.5 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.

16.34.3.6 bool weightUpdateModel::needPostSt

Whether postsynaptic spike times are needed or not.

16.34.3.7 bool weightUpdateModel::needPreSt

Whether presynaptic spike times are needed or not.

16.34.3.8 vector<string> weightUpdateModel::pNames

Names of (independent) parameters of the model.

16.34.3.9 string weightUpdateModel::simCode

Simulation code that is used for true spikes (only one time step after spike detection)

16.34.3.10 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.

16.34.3.11 string weightUpdateModel::simCodeEvnt

Simulation code that is used for spike events (all the instances where event threshold condition is met)

16.34.3.12 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.

16.34.3.13 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.

16.34.3.14 string weightUpdateModel::synapseDynamics

Simulation code for synapse dynamics independent of spike detection.

16.34.3.15 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.

17 File Documentation 121

16.34.3.16 vector < string > weightUpdateModel::varNames

Names of the variables in the postsynaptic model.

16.34.3.17 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

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

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.

17.14.1 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.

17.14.2 Function Documentation

```
17.14.2.1 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.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

17.15.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody1 model.

```
17.15.2 Macro Definition Documentation

17.15.2.1 #define MYRAND( Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);

17.15.2.2 #define PAT_TIME 100.0

17.15.2.3 #define PATFTIME 1.5

17.15.2.4 #define PATTERNNO 100

17.15.2.5 #define SYN_OUT_TME 20000.0

17.15.2.6 #define T_REPORT_TME 10000.0

17.15.2.7 #define TOTAL_TME 5000.0

17.15.3.1 variable Documentation

17.15.3.2 int patFireTime

17.15.3.3 int patSetTime

17.15.3.4 CStopWatch timer
```

17.16 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

17.16.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody_delayedSyn model.

```
17.16.2 Macro Definition Documentation

17.16.2.1 #define DBG_SIZE 10000

17.16.2.2 #define MYRAND( Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);

17.16.2.3 #define PAT_TIME 100.0

17.16.2.4 #define PATFTIME 1.5

17.16.2.5 #define PATTERNNO 100

17.16.2.6 #define SYN_OUT_TME 20000.0

17.16.2.7 #define T_REPORT_TME 10000.0

17.16.2.8 #define TOTAL_TME 5000.0

17.16.3.1 scalar InputBaseRate = 2e-04

17.16.3.2 int patFireTime

17.16.3.3 int patSetTime
```

17.17 classol_sim.h File Reference

17.16.3.4 CStopWatch timer

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

17.17.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody_individualID model.

```
17.17.2 Macro Definition Documentation
```

```
17.17.2.1 #define DBG_SIZE 10000
```

```
17.17.2.2 #define MYRAND( Y, X) Y = Y * 1103515245 + 12345; X= (Y >> 16);
```

- 17.17.2.3 #define PAT_TIME 100.0
- 17.17.2.4 #define PATFTIME 1.5
- 17.17.2.5 #define PATTERNNO 100
- 17.17.2.6 #define SYN_OUT_TME 20000.0
- 17.17.2.7 #define T_REPORT_TME 10000.0
- 17.17.2.8 #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 / 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

17.18.1 Detailed Description

Header file containing global variables and macros used in running the classol / MBody1 model.

17.18.2 Macro Definition Documentation

```
17.18.2.1 #define MYRAND( Y, X) Y = Y * 1103515245 + 12345; X= (<math>Y >> 16);
```

17.18.2.2 #define PAT_TIME 100.0

17.18.2.3 #define PATFTIME 1.5

17.18.2.4 #define PATTERNNO 100

17.18.2.5 #define SYN_OUT_TME 20000.0

17.18.2.6 #define T_REPORT_TME 10000.0

17.18.2.7 #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 CodeHelper.h File Reference

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <string>
#include <stream>
#include <vector>
```

Classes

· class CodeHelper

Macros

- #define SAVEP(X) "(" << X << ")"
- #define OB(X) hlp.openBrace(X)
- #define CB(X) hlp.closeBrace(X)
- #define ENDL hlp.endl()

Variables

· CodeHelper hlp

17.19.1 Macro Definition Documentation

```
17.19.1.1 #define CB( X ) hlp.closeBrace(X)
```

- 17.19.1.2 #define ENDL hlp.endl()
- 17.19.1.3 #define OB(X) hlp.openBrace(X)
- 17.19.1.4 #define SAVEP(X) "(" << X << ")"
- 17.19.2 Variable Documentation
- 17.19.2.1 CodeHelper hlp

17.20 command line processing.h File Reference

This file contains some tools for parsing the argy 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)

17.20.1 Detailed Description

This file contains some tools for parsing the argv array which contains the command line options.

17.20.2 Function Documentation

- 17.20.2.1 int extract_bool_value (char * op, unsigned int & val)
- 17.20.2.2 int extract_option (char * op, string & option)

```
17.20.2.3 int extract_string_value ( char * op, string & val )
17.20.2.4 string toLower ( string s )
17.20.2.5 string toUpper ( string s )
17.21 dpclass.h File Reference
#include <vector>
```

Classes

· class dpclass

17.22 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

```
17.22.1 Macro Definition Documentation
17.22.1.1 #define RAND( Y, X ) Y = Y * 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767
17.22.1.2 #define S_ISDIR( mode ) (((mode) & S_IFMT) == S_IFDIR)
17.22.2 Typedef Documentation
17.22.2.1 typedef struct Parameter Parameter
17.22.3 Function Documentation
17.22.3.1 bool applyInputToClassifier ( UINT recordingIdx, bool usePlasticity )
17.22.3.2 bool createDirectory ( string path )
17.22.3.3 bool directoryExists ( string const & path )
17.22.3.4 float getAverage (vector < float > & v)
17.22.3.5 float getStdDev ( vector < float > & v, float avg )
17.22.3.6 string getUniqueRunld ( )
17.22.3.7 int main ( int argc, char * argv[] )
17.22.3.8 void outputRunParameters ( )
17.22.3.9 bool printTextFile ( string path )
17.22.3.10 void setDefaultParamValues ( )
17.22.3.11 bool vectorContains ( vector< int > & vec, int lookingFor )
17.22.4 Variable Documentation
17.22.4.1 Schmuker2014 classifier classifier
17.23 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
- 17.23.1 Macro Definition Documentation
- 17.23.1.1 #define CACHE_DIR "cached_iris_data"
- 17.23.1.2 #define CONNECTIVITY_AN_AN 0.5
- 17.23.1.3 #define CONNECTIVITY_PN_AN 0.5
- 17.23.1.4 #define CONNECTIVITY_PN_PN 0.5
- 17.23.1.5 #define CONNECTIVITY_RN_PN 0.5
- 17.23.1.6 #define D_MAX_RANDOM_NUM 32767
- 17.23.1.7 #define DATASET_NAME "Iris"
- 17.23.1.8 #define divi "/"
- 17.23.1.9 #define FLAG_RUN_ON_CPU 1
- 17.23.1.10 #define GLOBAL_WEIGHT_SCALING 1.0
- 17.23.1.11 #define MAX_FIRING_RATE_HZ 70
- 17.23.1.12 #define MAX_WEIGHT_PN_AN 0.4
- 17.23.1.13 #define MIN_FIRING_RATE_HZ 20
- 17.23.1.14 #define MIN_WEIGHT_PN_AN 0.1
- 17.23.1.15 #define N_FOLDING 5

17.23.1.16 #define OUTPUT_DIR "output_iris" 17.23.1.17 #define PLASTICITY_INTERVAL_MS 330 17.23.1.18 #define RECORDING_TIME_MS 1000 17.23.1.19 #define RECORDINGS_DIR "recordings_iris_data" 17.23.1.20 #define REPEAT_LEARNING_SET 2 17.23.1.21 #define SPIKING_ACTIVITY_THRESHOLD_HZ 5 17.23.1.22 #define TOTAL_RECORDINGS 150 17.23.1.23 #define VR_DATA_FILENAME "VR-recordings-iris.data" 17.23.1.24 #define WEIGHT_DELTA_PN_AN 0.04 17.23.1.25 #define WEIGHT_RN_PN 0.5 17.23.1.26 #define WEIGHT_WTA_AN_AN 0.01 17.23.1.27 #define WEIGHT_WTA_PN_PN 0.01 17.23.2 Typedef Documentation 17.23.2.1 typedef unsigned int UINT 17.23.3 Variable Documentation 17.23.3.1 CStopWatch timer 17.24 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

```
17.24.1 Function Documentation
17.24.1.1 ps dpNames clear ( )
17.24.1.2 n varNames push_back ( "V" )
17.24.1.3 ps varTypes push_back ( "float" )
17.24.1.4 n varNames push_back ( "V_NB" )
17.24.1.5 n varNames push_back ( "tSpike_NB" )
17.24.1.6 n varNames push_back ( "__regime_val" )
17.24.1.7 n varTypes push_back ( "int" )
17.24.1.8 n pNames push_back ( "VReset_NB" )
17.24.1.9 n pNames push_back ( "VThresh_NB" )
17.24.1.10 n pNames push_back ( "tRefrac_NB" )
17.24.1.11 n pNames push_back ( "VRest_NB" )
17.24.1.12 n pNames push_back ( "TAUm_NB" )
17.24.1.13 n pNames push_back ( "Cm_NB" )
17.24.1.14 nModels push_back ( n )
17.24.1.15 n varNames push_back ( "count_t_NB" )
17.24.2 Variable Documentation
17.24.2.1 n simCode
Initial value:
           (V) = -1000000; \
           if ($(__regime_val)==1) { \n \
if ($(V_NB)>$(VThresh_NB)) { \n
(V_NB) = (VReset_NB); \ \ \
$(tSpike_NB) = t; \n \
$(V) = 100000; \
$(__regime_val) = 2; \n \
} \n \
} \n \
if ($(_
       _{regime\_val)} == 2) { <math>n \
if (t-$(tSpike_NB) > $(tRefrac_NB)) { \n \
$(__regime_val) = 1; \n \
} \n \
} \n \
```

17.25 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

17.25.1 Function Documentation

```
17.25.1.1 ps varNames clear ( )
```

- 17.25.1.2 ps varNames push_back ("g_PS")
- 17.25.1.3 ps varTypes push_back ("float")
- 17.25.1.4 ps pNames push_back ("tau_syn_PS")
- 17.25.1.5 ps pNames push_back ("E_PS")
- 17.25.1.6 postSynModels push_back (ps)
- 17.25.2 Variable Documentation
- 17.25.2.1 ps postSynDecay

Initial value:

17.25.2.2 ps postSyntoCurrent

Initial value:

17.26 extra_weightupdates.h File Reference

17.27 GA.cc File Reference

```
#include <algorithm>
```

Classes

struct errTupel

Functions

```
    int compareErrTupel (const void *x, const void *y)
```

```
    void procreatePop (FILE *osb)
```

17.27.1 Function Documentation

```
17.27.1.1 int compareErrTupel ( const void *x, const void *y )
17.27.1.2 void procreatePop ( FILE *osb )
```

17.28 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

17.28.1 Detailed Description

Contains the implementation of the Gaussian random number generator class randomGauss.

17.28.2 Macro Definition Documentation

```
17.28.2.1 #define GAUSS_CC
```

macro for avoiding multiple inclusion during compilation

17.29 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

17.29.1 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.).

17.29.2 Macro Definition Documentation

```
17.29.2.1 #define GAUSS_H
```

macro for avoiding multiple inclusion during compilation

17.30 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

17.30.1 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.

```
17.30.2 Function Documentation
17.30.2.1 int main (int argc, char * argv[])
17.30.3 Variable Documentation
17.30.3.1 randomGen R
```

17.31 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

17.31.1 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.

```
17.31.2 Function Documentation
17.31.2.1 int main (int argc, char * argv[])
17.31.3 Variable Documentation
17.31.3.1 randomGen R
17.31.3.2 randomGauss RG
```

17.32 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

```
17.32.1 Function Documentation

17.32.1.1 int main (int argc, char * argv[])

17.32.2 Variable Documentation

17.32.2.1 randomGen R

17.32.2.2 randomGen R2

17.32.2.3 randomGauss RG
```

17.33 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

17.33.1 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.

```
17.33.2 Function Documentation
17.33.2.1 int main ( int argc, char * argv[] )
17.33.3 Variable Documentation
17.33.3.1 randomGen R
17.33.3.2 randomGauss RG
```

17.34 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) \mid (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

17.34.1 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.

17.34.2 Macro Definition Documentation

```
17.34.2.1 #define B( x, i) ((x) & (0x800000000 >> (i)))
```

Extract the bit at the specified position i from \boldsymbol{x} .

```
17.34.2.2 #define delB( x, i) x= ((x) & (\sim(0x800000000 >> (i))))
```

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

```
17.34.2.3 #define setB( x, i) x= ((x) | (0x800000000 >> (i)))
```

Set the bit at the specified position i in x to 1.

17.34.3 Function Documentation

```
17.34.3.1 int main ( int argc, char * argv[] )
```

17.34.4 Variable Documentation

```
17.34.4.1 randomGen R
```

17.34.4.2 randomGauss RG

17.35 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[])

17.35.1 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.

17.35.2 Function Documentation

```
17.35.2.1 int main ( int argc, char * argv[] )
```

17.36 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

17.36.1 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.

```
17.36.2 Function Documentation

17.36.2.1 int main ( int argc, char * argv[] )

17.36.3 Variable Documentation

17.36.3.1 randomGen R

17.36.3.2 randomGauss RG
```

17.37 gen_syns_sparse_izhModel.cc File Reference

This file is part of a tool chain for running the Izhikevich network model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <cmath>
#include <vector>
#include "randomGen.h"
#include "randomGen.cc"
```

Functions

- int printVector (vector< unsigned int > &)
- int printVector (vector< double > &)
- int main (int argc, char *argv[])

Variables

- · randomGen R
- · randomGen Rind
- double gsyn
- double * garray
- unsigned int * ind
- double * garray_ee
- std::vector< double > g_ee
- std::vector< unsigned int > indlnG_ee
- $std::vector < unsigned int > ind_ee$
- double * garray_ei
- std::vector< double > g ei
- std::vector< unsigned int > indlnG_ei
- std::vector< unsigned int > ind_ei
- double * garray_ie
- $std::vector < double > g_ie$
- std::vector< unsigned int > indlnG_ie
- std::vector< unsigned int > ind_ie
- double * garray_ii

- std::vector< double > g_ii
- std::vector< unsigned int > indlnG_ii
- std::vector< unsigned int > ind_ii

17.37.1 Detailed Description

This file is part of a tool chain for running the Izhikevich network model.

```
17.37.2 Function Documentation
```

- 17.37.2.1 int main (int argc, char * argv[])
- 17.37.2.2 int printVector (vector< unsigned int > & ν)
- 17.37.2.3 int printVector (vector < double > & v)
- 17.37.3 Variable Documentation
- 17.37.3.1 std::vector<double> g_ee
- 17.37.3.2 std::vector<double> g_ei
- 17.37.3.3 std::vector<double> g_ie
- 17.37.3.4 std::vector<double> g_ii
- 17.37.3.5 double * garray
- 17.37.3.6 double* garray_ee
- 17.37.3.7 double* garray_ei
- 17.37.3.8 double* garray_ie
- 17.37.3.9 double* garray_ii
- 17.37.3.10 double gsyn
- 17.37.3.11 unsigned int* ind
- 17.37.3.12 std::vector<unsigned int> ind_ee
- 17.37.3.13 std::vector<unsigned int> ind_ei
- 17.37.3.14 std::vector<unsigned int> ind_ie
- 17.37.3.15 std::vector<unsigned int> ind_ii
- 17.37.3.16 std::vector<unsigned int> indlnG_ee
- 17.37.3.17 std::vector<unsigned int> indlnG_ei
- 17.37.3.18 std::vector<unsigned int> indlnG_ie
- 17.37.3.19 std::vector<unsigned int> indlnG_ii
- 17.37.3.20 randomGen R
- 17.37.3.21 randomGen Rind

17.38 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.

17.38.1 Detailed Description

This file is used to run the HHVclampGA model with a single command line.

17.38.2 Function Documentation

```
17.38.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.39 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.

17.39.1 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.

```
17.39.2 Function Documentation
```

```
17.39.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.39.2.2 unsigned int openFileGetMax (unsigned int * array, unsigned int size, string name)

17.40 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.

17.40.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool 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.

```
17.40.2 Function Documentation
```

```
17.40.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.41 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.

17.41.1 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.

```
17.41.2 Function Documentation
```

```
17.41.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.42 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.

17.42.1 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.

```
17.42.2 Function Documentation
```

```
17.42.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.43 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.

17.43.1 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.

17.43.2 Function Documentation

```
17.43.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.44 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.

17.44.1 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.

```
17.44.2 Function Documentation
```

```
17.44.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.45 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.

17.45.1 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.

17.45.2 Function Documentation

```
17.45.2.1 int main ( int argc, char * argv[] )
```

Main entry point for generate_run.

17.46 generateALL.cc File Reference

Main file combining the code for code generation. Part of the code generation section.

```
#include "global.h"
#include "generateALL.h"
#include "generateCPU.h"
#include "generateKernels.h"
#include "modelSpec.h"
#include "utils.h"
#include "stringUtils.h"
#include "CodeHelper.h"
#include <cmath>
#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

17.46.1 Detailed Description

Main file combining the code for code generation. Part of the code generation section.

The file includes separate files for generating kernels (generateKernels.cc), generating the CPU side code for running simulations on either the CPU or GPU (generateRunner.cc) and for CPU-only simulation code (generateCP← U.cc).

17.46.2 Function Documentation

17.46.2.1 void chooseDevice (NNmodel & model, const string & path)

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

The main tasks in this function are the detection and characterization of the GPU device present (if any), choosing which GPU device to use, finding and appropriate block size, taking note of the major and minor version of the C← 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

17.46.2.2 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

17.46.2.3 int main (int *argc*, char * *argv*[])

Main entry point for the generateALL executable that generates the code for GPU and CPU.

The main function is the entry point for the code generation engine. It prepares the system and then invokes generate_model_runner to initiate the different parts of actual code generation.

argc	number of arguments; expected to be 2
	J , 1

Parameters

argv	Arguments; expected to contain the target directory for code generation.
------	--

17.46.3 Variable Documentation

17.46.3.1 CodeHelper hlp

17.47 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.

17.47.1 Function Documentation

17.47.1.1 void chooseDevice (NNmodel & model, const string & path)

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

The main tasks in this function are the detection and characterization of the GPU device present (if any), choosing which GPU device to use, finding and appropriate block size, taking note of the major and minor version of the $C \leftarrow UDA$ enabled device chosen for use, and populating the list of standard neuron models. The chosen device number is returned.

Parameters

model	the nn model we are generating code for
path	path the generated code will be deposited

17.47.1.2 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.

model	Model description
path	Path where the generated code will be deposited

17.48 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 "stringUtils.h"
#include "CodeHelper.h"
#include <algorithm>
```

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 generate_process_presynaptic_events_code_CPU (ostream &os, const NNmodel &model, unsigned int src, unsigned int trg, int i, string &localID, unsigned int inSynNo, const string &postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

void genSynapseFunction (const NNmodel &model, const string &path)

Function that generates code that will simulate all synapses of the model on the CPU.

17.48.1 Detailed Description

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

17.48.2 Function Documentation

17.48.2.1 void generate_process_presynaptic_events_code_CPU (ostream & os, const NNmodel & model, unsigned int src, unsigned int trg, int i, string & localID, unsigned int inSynNo, const string & postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

Parameters

os	output stream for code
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed
localID	the variable name of the local ID of the thread within the synapse group
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron population
postfix	whether to generate code for true spikes or spike type events

17.48.2.2 void genNeuronFunction (const NNmodel & model, const string & path)

Function that generates the code of the function the will simulate all neurons on the CPU.

m	odel	Model description

Parameters

n for code generation	path
-----------------------	------

17.48.2.3 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

17.49 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 generate_process_presynaptic_events_code_CPU (ostream &os, const NNmodel &model, unsigned int src, unsigned int trg, int i, string &localID, unsigned int inSynNo, const string &postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

void genSynapseFunction (const NNmodel &model, const string &path)

Function that generates code that will simulate all synapses of the model on the CPU.

17.49.1 Detailed Description

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

17.49.2 Function Documentation

17.49.2.1 void generate_process_presynaptic_events_code_CPU (ostream & os, const NNmodel & model, unsigned int src, unsigned int trg, int i, string & localID, unsigned int inSynNo, const string & postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

os	output stream for code
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed

Parameters

localID	the variable name of the local ID of the thread within the synapse group	
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron population	
postfix	whether to generate code for true spikes or spike type events	

17.49.2.2 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

17.49.2.3 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

17.50 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 "stringUtils.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 generate_process_presynaptic_events_code (ostream &os, const NNmodel &model, unsigned int src, unsigned int trg, int i, const string &localID, unsigned int inSynNo, const string &postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

• void genSynapseKernel (const NNmodel &model, const string &path)

Function for generating a CUDA kernel for simulating all synapses.

Variables

• short * isGrpVarNeeded

17.50.1 Detailed Description

Contains functions that generate code for CUDA kernels. Part of the code generation section.

17.50.2 Function Documentation

17.50.2.1 void generate_process_presynaptic_events_code (ostream & os, const NNmodel & model, unsigned int src, unsigned int trg, int i, const string & localID, unsigned int inSynNo, const string & postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

Parameters

os	output stream for code
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed
localID	the variable name of the local ID of the thread within the synapse group
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron population
postfix	whether to generate code for true spikes or spike type events

17.50.2.2 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. Binary flag for the sparse synapses to use atomic operations when the number of connections is bigger than the block size, and shared variables otherwise

Binary flag for the sparse synapses to use atomic operations when the number of connections is bigger than the block size, and shared variables otherwise

Parameters

model	Model description
path	Path for code generation

17.50.2.3 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)

model	Model description
path	Path for code generation

17.50.3 Variable Documentation

17.50.3.1 short* isGrpVarNeeded

17.51 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 generate_process_presynaptic_events_code (ostream &os, const NNmodel &model, unsigned int src, unsigned int trg, int i, const string &localID, unsigned int inSynNo, const string &postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

• void genSynapseKernel (const NNmodel &model, const string &path)

Function for generating a CUDA kernel for simulating all synapses.

17.51.1 Detailed Description

Contains functions that generate code for CUDA kernels. Part of the code generation section.

17.51.2 Function Documentation

17.51.2.1 void generate_process_presynaptic_events_code (ostream & os, const NNmodel & model, unsigned int src, unsigned int trg, int i, const string & localID, unsigned int inSynNo, const string & postfix)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

Parameters

os	output stream for code
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed
localID	the variable name of the local ID of the thread within the synapse group
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron
	population
postfix	whether to generate code for true spikes or spike type events

17.51.2.2 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. Binary flag for the sparse synapses to use atomic operations when the number of connections is bigger than the block size, and shared variables otherwise

Binary flag for the sparse synapses to use atomic operations when the number of connections is bigger than the block size, and shared variables otherwise

Parameters

model	Model description
path	Path for code generation

17.51.2.3 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

17.52 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 "stringUtils.h"
#include "CodeHelper.h"
#include <stdint.h>
#include <cfloat>
```

Functions

void variable_def (ofstream &os, const string &type, const string &name)

This function generates host and device variable definitions, of the given type and name.

• void extern_variable_def (ofstream &os, const string &type, const string &name)

This function generates host extern variable definitions, of the given type and name.

• 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.

17.52.1 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.

17.52.2 Function Documentation

17.52.2.1 void extern_variable_def (ofstream & os, const string & type, const string & name)

This function generates host extern variable definitions, of the given type and name.

This fucntion generates host extern variable definitions, of the given type and name.

17.52.2.2 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

17.52.2.3 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

17.52.2.4 void genRunnerGPU (const NNmodel & model, const string & path)

A function to generate the code that simulates the model on the GPU.

The function generates functions that will spawn kernel grids onto the GPU (but not the actual kernel code which is generated in "genNeuronKernel()" and "genSynpaseKernel()"). Generated functions include "copyGToDevice()", "copyGFromDevice()", "copySpikesFromDevice()", "copySpi

Parameters

model	Model description
path	Path for code generation

17.52.2.5 void variable_def (ofstream & os, const string & type, const string & name)

This function generates host and device variable definitions, of the given type and name.

This fucntion generates host and device variable definitions, of the given type and name.

17.53 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 variable_def (ofstream &os, const string &type, const string &name)

This fucntion generates host and device variable definitions, of the given type and name.

void extern_variable_def (ofstream &os, const string &type, const string &name)

This fucntion generates host extern variable definitions, of the given type and name.

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.

17.53.1 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.

17.53.2 Function Documentation

```
17.53.2.1 void extern_variable_def ( ofstream & os, const string & type, const string & name )
```

This fucntion generates host extern variable definitions, of the given type and name.

This fucntion generates host extern variable definitions, of the given type and name.

17.53.2.2 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

17.53.2.3 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

17.53.2.4 void genRunnerGPU (const NNmodel & model, const string & path)

A function to generate the code that simulates the model on the GPU.

The function generates functions that will spawn kernel grids onto the GPU (but not the actual kernel code which is generated in "genNeuronKernel()" and "genSynpaseKernel()"). Generated functions include "copyGToDevice()", "copyGFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()", "copySpikesFromDevice()". The last mentioned function is the function that will initialize the execution on the GPU in the generated simulation engine. All other generated functions are "convenience functions" to handle data transfer from and to the GPU.

Parameters

model	Model description
path	Path for code generation

17.53.2.5 void variable_def (ofstream & os, const string & type, const string & name)

This fucntion generates host and device variable definitions, of the given type and name.

This fucntion generates host and device variable definitions, of the given type and name.

17.54 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_gpulnput_xorwow (curandState *state, T *result, int sizeofResult, T Rstrength, T Rshift)
- template<typename T >
 void generate_random_gpuInput_xorwow (curandState *state, T *result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads)
- 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)

17.54.1 Function Documentation

- 17.54.1.1 template<typename T > __global__ void generate_random_gpulnput_xorwow (curandState * state, T * result, int sizeofResult, T Rstrength, T Rshift)
- 17.54.1.2 template < typename T > void generate_random_gpulnput_xorwow (curandState * state, T * result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads)

- 17.54.1.3 template void generate_random_gpuInput_xorwow< double > (curandState * state, double * result, int sizeofResult, double Rstrength, double Rshift, dim3 sGrid, dim3 sThreads)
- 17.54.1.4 template void generate_random_gpuInput_xorwow< float > (curandState * state, float * result, int sizeofResult, float Rstrength, float Rshift, dim3 sGrid, dim3 sThreads)
- 17.54.1.5 __global__ void setup_kernel (curandState * state, unsigned long seed, int sizeofResult)
- 17.54.1.6 void xorwow_setup (curandState * devStates, long int sampleSize, long long int seed)

17.55 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
- 17.55.1 Function Documentation
- 17.55.1.1 template<typename T > __global__ void generate_random_gpulnput_xorwow (curandState * state, T * result, int sizeofResult, T Rstrength, T Rshift)
- 17.55.1.2 template < typename T > void generate_random_gpulnput_xorwow (curandState * state, T * result, int sizeofResult, T Rstrength, T Rshift, dim3 sGrid, dim3 sThreads)
- 17.55.1.3 __global__ void setup_kernel (curandState * state, unsigned long seed, int sizeofResult)
- $17.55.1.4 \quad \text{void xorwow_setup (curandState} * \textit{devStates, } long int \textit{sampleSize, } long long int \textit{seed)}$
- 17.55.2 Variable Documentation
- 17.55.2.1 const int BlkSz = 256

17.56 global.cc File Reference

```
#include "global.h"
```

Namespaces

- GENN FLAGS
- GENN_PREFERENCES

Macros

• #define GLOBAL CC

Variables

· int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

· int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

· int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

· 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.

17.56.1 Macro Definition Documentation

17.56.1.1 #define GLOBAL_CC

17.56.2 Variable Documentation

17.56.2.1 int deviceCount

Global variable containing the number of CUDA devices on this host.

17.56.2.2 cudaDeviceProp* deviceProp

17.56.2.3 int hostCount

Global variable containing the number of hosts within the local compute cluster.

17.56.2.4 int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

17.56.2.5 int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

17.56.2.6 int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

17.56.2.7 int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

17.56.2.8 int theDevice

Global variable containing the currently selected CUDA device's number.

17.57 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

Macros

- #define TRUE true
- #define FALSE false

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)

· int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

· int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

· 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.

17.57.1 Detailed Description

Global header file containing a few global variables. Part of the code generation section.

17.57.2 Macro Definition Documentation

17.57.2.1 #define FALSE false

17.57.2.2 #define TRUE true

17.57.3 Variable Documentation

17.57.3.1 int deviceCount

Global variable containing the number of CUDA devices on this host.

17.57.3.2 cudaDeviceProp* deviceProp

17.57.3.3 int hostCount

Global variable containing the number of hosts within the local compute cluster.

17.57.3.4 int learnBlkSz

Global variable containing the GPU block size for the learn kernel.

17.57.3.5 int neuronBlkSz

Global variable containing the GPU block size for the neuron kernel.

17.57.3.6 int synapseBlkSz

Global variable containing the GPU block size for the synapse kernel.

17.57.3.7 int synDynBlkSz

Global variable containing the GPU block size for the synapse dynamics kernel.

17.57.3.8 int theDevice

Global variable containing the currently selected CUDA device's number.

17.58 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 initl (inputSpec &I)

Variables

- double sigGNa = 0.1
- double sigENa = 10.0
- double sigGK = 0.1
- double sigEK = 10.0
- double sigGl = 0.1
- double sigEl = 10.0
- double sigC = 0.1
- const double limit [7][2]
- double Vexp
- double mexp
- double hexp
- double nexp
- double gNaexp
- double ENaexp
- double gKexp
- double EKexp
- · double glexp
- · double Elexp
- double Cexp

```
17.58.1 Function Documentation
17.58.1.1 void copy_var ( int src, int trg )
17.58.1.2 void initexpHH ( )
17.58.1.3 void initl (inputSpec & I)
17.58.1.4 ostream& operator << ( ostream & os, inputSpec & I )
17.58.1.5 void runexpHH ( float t )
17.58.1.6 void single_var_init_fullrange ( int n )
17.58.1.7 void single_var_reinit ( int n, double fac )
17.58.1.8 void truevar_init ( )
17.58.1.9 void truevar_initexpHH ( )
17.58.1.10 void var_init_fullrange ( )
17.58.1.11 void var_reinit ( double fac )
17.58.1.12 void write_para ( )
17.58.2 Variable Documentation
17.58.2.1 double Cexp
17.58.2.2 double EKexp
17.58.2.3 double Elexp
17.58.2.4 double ENaexp
17.58.2.5 double gKexp
17.58.2.6 double glexp
17.58.2.7 double gNaexp
17.58.2.8 double hexp
17.58.2.9 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}}
17.58.2.10 double mexp
17.58.2.11 double nexp
17.58.2.12 double sigC = 0.1
17.58.2.13 double sigEK = 10.0
```

```
17.58.2.14 double sigEl = 10.0

17.58.2.15 double sigENa = 10.0

17.58.2.16 double sigGK = 0.1

17.58.2.17 double sigGl = 0.1

17.58.2.18 double sigGNa = 0.1

17.58.2.19 double Vexp
```

17.59 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"
```

Functions

• void modelDefinition (NNmodel &model)

This function defines the HH model with variable parameters.

Variables

```
double myHH_ini [11]double * myHH p = NULL
```

17.59.1 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.

```
17.59.2 Function Documentation
```

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

This function defines the HH model with variable parameters.

```
17.59.3 Variable Documentation
```

```
17.59.3.1 double myHH_ini[11]
```

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207,
    120.0,
    55.0,
```

```
36.0,
-72.0,
0.3,
-50.0,
1.0
```

17.59.3.2 double * myHH_p = NULL

17.60 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

17.60.1 Macro Definition Documentation

```
17.60.1.1 #define _FTYPE GENN_FLOAT
```

17.60.1.2 #define NPOP 12

17.60.1.3 #define scalar float

17.60.1.4 #define SCALAR_MAX 3.40282e+38f

17.60.1.5 #define SCALAR_MIN 1.17549e-38f

17.60.1.6 #define TOTALT 200000

17.61 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

17.61.1 Detailed Description

This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock.

17.61.2 Macro Definition Documentation

17.61.2.1 #define HR_TIMER

17.62 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

17.62.1 Detailed Description

This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock.

17.63 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)

17.63.1 Detailed Description

Header file and implementation of the ISAAC random number generator.

 $\hbox{C++} \ \hbox{TEMPLATE VERSION OF Robert J. Jenkins Jr.'s ISAAC Random Number Generator.}$

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
17.63.2 Macro Definition Documentation
17.63.2.1 #define __ISAAC_HPP
macro for avoiding multiple inclusion during compilation
17.63.3 Typedef Documentation
17.63.3.1 typedef unsigned long int ISAAC_INT
17.63.4 Variable Documentation
17.63.4.1 const ISAAC_INT GOLDEN_RATIO = ISAAC_INT(0x9e3779b9)
17.64 Izh_sim_sparse.cc File Reference
#include <iostream>
#include <fstream>
#include "Izh_sparse_sim.h"
#include <cuda_runtime.h>
#include "GeNNHelperKrnls.h"
Functions
    • int main (int argc, char *argv[])
17.64.1 Function Documentation
17.64.1.1 int main ( int argc, char * argv[] )
17.65 Izh_sparse.cc File Reference
#include "modelSpec.h"
#include "global.h"
```

Functions

void modelDefinition (NNmodel &model)

#include "stringUtils.h"

#include <vector>
#include "sizes.h"

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
    • double * exclzh_p = NULL
    double * inhlzh p = NULL
    • double <a href="mailto:lzhExc_ini">lzhExc_ini</a> [7]
    • double lzhlnh_ini [7]
    double * SynIzh_p = NULL
    double * postExpP = NULL

    double * postSynV = NULL

    • double SynIzh_ini [1]
17.65.1 Function Documentation
17.65.1.1 void modelDefinition ( NNmodel & model )
17.65.2 Variable Documentation
17.65.2.1 double* exclzh_p = NULL
17.65.2.2 double* inhlzh_p = NULL
17.65.2.3 double lzhExc_ini[7]
Initial value:
= {
        -65.0,
         0.02,
        0.2,
        8.0,
        0.0
17.65.2.4 double lzhInh_ini[7]
Initial value:
= {
         -65,
         0.0,
         0.02,
        0.25,
        -65.0,
        2.0,
        0.0
17.65.2.5 scalar meanInpExc = 5.0*inputFac
17.65.2.6 scalar meanInplnh = 2.0*inputFac
```

17.65.2.7 std::vector<unsigned int> neuronPSize

```
17.65.2.8 std::vector<unsigned int> neuronVSize
17.65.2.9 double* postExpP = NULL
17.65.2.10 double* postSynV = NULL
17.65.2.11 std::vector<unsigned int> synapsePSize
17.65.2.12 double Synlzh_ini[1]
Initial value:
   0.0
17.65.2.13 double* Synlzh_p = NULL
17.66 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
17.66.1 Macro Definition Documentation
17.66.1.1 #define _IZH_SPARSE_MODEL_CC_
17.66.2 Variable Documentation
17.66.2.1 randomGen R
17.66.2.2 randomGauss RG
17.67 Izh_sparse_model.h File Reference
Classes
    · class classIzh
17.68 Izh_sparse_sim.h File Reference
```

#include <cassert>

```
#include "hr_time.h"
#include "utils.h"
#include <cuda_runtime.h>
#include "Izh_sparse.cc"
#include "Izh_sparse_model.cc"
```

Macros

- #define DBG_SIZE 5000
- #define T REPORT TME 5000.0
- #define TOTAL_TME 1000.0

Variables

· CStopWatch timer

```
17.68.1 Macro Definition Documentation
```

17.68.1.1 #define DBG_SIZE 5000

17.68.1.2 #define T_REPORT_TME 5000.0

17.68.1.3 #define TOTAL_TME 1000.0

17.68.2 Variable Documentation

17.68.2.1 CStopWatch timer

17.69 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

17.69.1 Detailed Description

Implementation of the classol class.

17.69.2 Macro Definition Documentation

17.69.2.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

17.70 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

17.70.1 Detailed Description

Implementation of the classol class.

17.70.2 Macro Definition Documentation

```
17.70.2.1 #define _MAP_CLASSOL_CC_
```

macro for avoiding multiple inclusion during compilation

17.71 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

17.71.1 Detailed Description

Implementation of the classol class.

17.71.2 Macro Definition Documentation

17.71.2.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

17.72 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

17.72.1 Detailed Description

Implementation of the classol class.

17.72.2 Macro Definition Documentation

```
17.72.2.1 #define MAP CLASSOL CC
```

macro for avoiding multiple inclusion during compilation

17.73 map_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

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

Classes

class classol

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

17.73.1 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

17.74 map_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

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

Classes

· class classol

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

17.74.1 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

17.75 map_classol.h File Reference

Header file containing the class definition for classol (CLASSification OLfaction model), which contains the methods for setting up, initialising, simulating and saving results of a model of the insect mushroom body.

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

Classes

class classol

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

17.75.1 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

17.76 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.

17.76.1 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

17.77 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 "modelSpec.cc"
```

Macros

#define DT 0.1

This defines the global time step at which the simulation will run.

Functions

· void modelDefinition (NNmodel &model)

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

Variables

- double myPOI_p [4]
- double myPOI_ini [4]
- double stdTM p [7]
- double stdTM_ini [4]
- double myPNKC_p [3]
- double postExpPNKC [2]
- double myPNLHI p [3]
- double postExpPNLHI [2]
- double myLHIKC_p [4]
- double gLHIKC = 0.006
- double postExpLHIKC [2]
- double myKCDN_p [13]
- double postExpKCDN [2]
- double myDNDN_p [4]
- double gDNDN = 0.01
- double postExpDNDN [2]
- double * postSynV = NULL

17.77.1 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).

17.77.2 Macro Definition Documentation

17.77.2.1 #define DT 0.1

This defines the global time step at which the simulation will run.

17.77.3 Function Documentation

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

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

17.77.4 Variable Documentation

```
17.77.4.1 double gDNDN = 0.01
```

17.77.4.2 double gLHIKC = 0.006

17.77.4.3 double myDNDN_p[4]

Initial value:

```
= {
    -92.0,
    -30.0,
    8.0,
    50.0
```

17.77.4.4 double myKCDN_p[13]

Initial value:

```
= {
    0.0,
    -20.0,
    5.0,
    25.0,
    100.0,
    50000.0,
    10000.0,
    0.06,
    0.03,
    33.33,
    10.0,
    0.00006
```

17.77.4.5 double myLHIKC_p[4]

Initial value:

```
= {
    -92.0,
    -40.0,
    3.0,
    50.0
```

17.77.4.6 double myPNKC_p[3]

Initial value:

```
= {
    0.0,
    -20.0,
    1.0
```

17.77.4.7 double myPNLHI_p[3]

Initial value:

```
= {
    0.0,
    -20.0,
    1.0
```

17.77.4.8 double myPOI_ini[4]

Initial value:

```
= {
-60.0,
0,
-10.0,
```

17.77.4.9 double myPOI_p[4]

Initial value:

```
= {
    0.1,
    2.5,
    20.0,
    -60.0
```

17.77.4.10 double postExpDNDN[2]

Initial value:

```
= {
8.0,
-92.0
```

17.77.4.11 double postExpKCDN[2]

Initial value:

```
= {
    5.0,
    0.0
}
```

17.77.4.12 double postExpLHIKC[2]

Initial value:

```
= {
3.0,
-92.0
```

17.77.4.13 double postExpPNKC[2]

Initial value:

```
= {
1.0,
0.0
```

17.77.4.14 double postExpPNLHI[2]

Initial value:

```
= {
    1.0,
    0.0
}
```

17.77.4.15 double* postSynV = NULL

17.77.4.16 double stdTM_ini[4]

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207
```

17.77.4.17 double stdTM_p[7]

Initial value:

```
= {
    7.15,
    50.0,
    1.43,
    -95.0,
    0.02672,
    -63.563,
    0.143
```

17.78 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

• void modelDefinition (NNmodel &model)

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

Variables

```
• double myPOI_p [4]
```

- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [10]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

17.78.1 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).

17.78.2 Function Documentation

17.78.2.1 void modelDefinition (NNmodel & model)

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

17.78.3 Variable Documentation

17.78.3.1 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
```

17.78.3.2 double myDNDN_p[2]

Initial value:

```
= {
    -30.0,
50.0
```

17.78.3.3 double myKCDN_ini[2]

```
Initial value:
```

```
={
    0.01,
    0.01,
}
```

17.78.3.4 double myKCDN_p[10]

Initial value:

```
= {
    50.0,
    50.0,
    50000.0,
    100000.0,
    200.0,
    0.015,
    0.0075,
    33.33,
    10.0,
    0.00006
}
```

17.78.3.5 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

17.78.3.6 double myLHIKC_p[2]

Initial value:

```
= {
 -40.0,
 50.0
```

17.78.3.7 double myPNKC_ini[1]

Initial value:

```
= {
 0.01
}
```

17.78.3.8 double* myPNKC_p = NULL

17.78.3.9 double myPNLHI_ini[1]

17.78.3.10 double* myPNLHI_p = NULL

17.78.3.11 double myPOI_ini[3]

```
Initial value:
= {
-60.0,
 0,
-10.0
17.78.3.12 double myPOI_p[4]
Initial value:
= {
    0.1,
    2.5,
    20.0,
    -60.0
17.78.3.13 double postExpDNDN[2]
Initial value:
={
2.5,
-92.0
}
17.78.3.14 double postExpKCDN[2]
Initial value:
= {
    5.0,
    0.0
}
17.78.3.15 double postExpLHIKC[2]
Initial value:
={
1.5,
 -92.0
17.78.3.16 double postExpPNKC[2]
Initial value:
= {
    1.0,
    0.0
}
17.78.3.17 double postExpPNLHI[2]
Initial value:
= {
1.0,
0.0
```

```
17.78.3.18 double* postSynV = NULL
```

17.78.3.19 double stdTM_ini[4]

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207
```

17.78.3.20 double stdTM_p[7]

Initial value:

```
= {
    7.15,
    50.0,
    1.43,
    -95.0,
    0.02672,
    -63.563,
    0.143
```

17.79 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

• void modelDefinition (NNmodel &model)

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

Variables

- double myPOI_p [4]
- double myPOI ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [10]
- double myKCDN_ini [2]
- double postExpKCDN [2]

- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

17.79.1 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).

17.79.2 Function Documentation

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

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

17.79.3 Variable Documentation

```
17.79.3.1 double myDNDN_ini[1]
```

Initial value:

```
={
    5.0/_NLB
}
```

17.79.3.2 double myDNDN_p[2]

Initial value:

```
= {
 -30.0,
50.0
```

17.79.3.3 double myKCDN_ini[2]

Initial value:

```
= {
    0.01,
    0.01,
}
```

17.79.3.4 double myKCDN_p[10]

```
= {
    50.0,
    50.0,
    50000.0,
    100000.0,
    200.0,
    0.015,
    0.0075,
    33.33,
    10.0,
    0.00006
```

```
17.79.3.5 double myLHIKC_ini[1]
```

```
Initial value:
```

```
= {
    1.0/_NLHI
}
```

17.79.3.6 double myLHIKC_p[2]

Initial value:

```
= {
    -40.0,
    50.0
}
```

17.79.3.7 double myPNKC_ini[1]

Initial value:

```
= {
 0.01
}
```

17.79.3.8 double* myPNKC_p = NULL

17.79.3.9 double myPNLHI_ini[1]

Initial value:

```
= {
0.0
```

17.79.3.10 double* myPNLHI_p = NULL

17.79.3.11 double myPOI_ini[3]

Initial value:

```
= {
-60.0,
0,
-10.0
}
```

17.79.3.12 double myPOI_p[4]

Initial value:

```
= {
    0.1,
    2.5,
    20.0,
    -60.0
```

17.79.3.13 double postExpDNDN[2]

```
= {
2.5,
-92.0
```

```
17.79.3.14 double postExpKCDN[2]
```

Initial value:

```
={
5.0,
0.0
}
```

17.79.3.15 double postExpLHIKC[2]

Initial value:

```
= {
1.5,
-92.0
```

17.79.3.16 double postExpPNKC[2]

Initial value:

```
= {
1.0,
0.0
}
```

17.79.3.17 double postExpPNLHI[2]

Initial value:

```
= {
1.0,
0.0
```

17.79.3.18 double* postSynV = NULL

17.79.3.19 double stdTM_ini[4]

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207
}
```

17.79.3.20 double stdTM_p[7]

```
= {
    7.15,
    50.0,
    1.43,
    -95.0,
    0.02672,
    -63.563,
    0.143
```

17.80 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

· void modelDefinition (NNmodel &model)

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

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [11]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

17.80.1 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.

```
17.80.2 Function Documentation
```

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

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

```
17.80.3 Variable Documentation
```

17.80.3.1 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
}
```

17.80.3.2 double myDNDN_p[2]

Initial value:

```
= {
  -30.0,
  50.0
}
```

17.80.3.3 double myKCDN_ini[2]

Initial value:

```
= {
    0.01,
    0.01,
}
```

17.80.3.4 double myKCDN_p[11]

Initial value:

17.80.3.5 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

17.80.3.6 double myLHIKC_p[2]

```
= {
-40.0,
50.0
```

```
17.80.3.7 double myPNKC_ini[1]
```

```
Initial value:
```

```
= {
   gPNKC_GLOBAL
}
```

17.80.3.8 double* myPNKC_p = NULL

17.80.3.9 double myPNLHI_ini[1]

Initial value:

```
= {
 0.0
}
```

17.80.3.10 double* myPNLHI_p = NULL

17.80.3.11 double myPOI_ini[3]

Initial value:

```
= {
-60.0,
0,
-10.0
```

17.80.3.12 double myPOI_p[4]

Initial value:

```
= {
    0.1,
    2.5,
    20.0,
    -60.0
```

17.80.3.13 double postExpDNDN[2]

Initial value:

```
= {
2.5,
-92.0
```

17.80.3.14 double postExpKCDN[2]

Initial value:

```
= {
5.0,
0.0
```

17.80.3.15 double postExpLHIKC[2]

```
={
1.5,
-92.0
```

17.80.3.16 double postExpPNKC[2]

Initial value:

```
= {
1.0,
0.0
}
```

17.80.3.17 double postExpPNLHI[2]

Initial value:

```
= {
1.0,
0.0
}
```

17.80.3.18 double* postSynV = NULL

17.80.3.19 double stdTM_ini[4]

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207
}
```

17.80.3.20 double stdTM_p[7]

Initial value:

```
= {
    7.15,
    50.0,
    1.43,
    -95.0,
    0.02672,
    -63.563,
    0.143
}
```

17.81 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 pwSTDP userdef

TODO This class definition may be code-generated in a future release.

Macros

• #define TIMING

Functions

· void modelDefinition (NNmodel &model)

This function defines the MBody1 model with user defined synapses.

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [11]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL
- double postSynV_EXPDECAY_EVAR [1]
- scalar * gpPNKC = new scalar[_NAL*_NMB]
- scalar * gpKCDN = new scalar[_NMB*_NLB]

17.81.1 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).

```
17.81.2 Macro Definition Documentation
```

```
17.81.2.1 #define TIMING
```

17.81.3 Function Documentation

17.81.3.1 void modelDefinition (NNmodel & model)

This function defines the MBody1 model with user defined synapses.

```
17.81.4 Variable Documentation
```

```
17.81.4.1 scalar* gpKCDN = new scalar[_NMB*_NLB]
```

17.81.4.2 scalar* gpPNKC = new scalar[_NAL*_NMB]

17.81.4.3 double myDNDN_ini[1]

```
={
    5.0/_NLB
}
```

17.81.4.4 double myDNDN_p[2]

Initial value:

```
= {
    -30.0,
    50.0
}
```

17.81.4.5 double myKCDN_ini[2]

Initial value:

```
={
    0.01,
    0.01,
}
```

17.81.4.6 double myKCDN_p[11]

Initial value:

```
= {
    -20.0,
    50.0,
    50.0,
    50.000.0,
    100000.0,
    200.0,
    0.015,
    0.0075,
    33.33,
    10.0,
    0.00006
```

17.81.4.7 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

17.81.4.8 double myLHIKC_p[2]

Initial value:

```
= {
    -40.0,
    50.0
}
```

17.81.4.9 double myPNKC_ini[1]

```
= {
 0.01
}
```

```
17.81.4.10 double* myPNKC_p = NULL
```

17.81.4.11 double myPNLHI_ini[1]

Initial value:

```
= {
 0.0
```

17.81.4.12 double* myPNLHI_p = NULL

17.81.4.13 double myPOI_ini[3]

Initial value:

```
= {
-60.0,
0,
-10.0
```

17.81.4.14 double myPOI_p[4]

Initial value:

```
= {
    0.1,
    2.5,
    20.0,
    -60.0
}
```

17.81.4.15 double postExpDNDN[2]

Initial value:

```
={
2.5,
-92.0
```

17.81.4.16 double postExpKCDN[2]

Initial value:

```
= {
    5.0,
    0.0
```

17.81.4.17 double postExpLHIKC[2]

Initial value:

```
={
1.5,
-92.0
}
```

17.81.4.18 double postExpPNKC[2]

```
= {
1.0,
0.0
}
```

```
17.81.4.19 double postExpPNLHI[2]
```

Initial value:

Initial value:

```
= {
0
}
```

17.81.4.22 double stdTM_ini[4]

Initial value:

```
= {
    -60.0,
    0.0529324,
    0.3176767,
    0.5961207
}
```

17.81.4.23 double stdTM_p[7]

Initial value:

```
= {
    7.15,
    50.0,
    1.43,
    -95.0,
    0.02672,
    -63.563,
    0.143
```

17.82 model.cc File Reference

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

Functions

void modelDefinition (NNmodel &model)

Variables

• double neuron_ini [2]

17.82.1 Function Documentation

17.82.1.1 void modelDefinition (NNmodel & model)

```
17.82.2 Variable Documentation
```

17.82.2.1 double neuron_ini[2]

Initial value:

17.83 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

- double neuron_ini [2]
- double synapses_ini [1]

17.83.1 Function Documentation

17.83.1.1 void modelDefinition (NNmodel & model)

17.83.2 Variable Documentation

17.83.2.1 double neuron_ini[2]

Initial value:

17.83.2.2 double synapses_ini[1]

Initial value:

17.84 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

17.84.1 Function Documentation

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

17.84.2 Variable Documentation

17.84.2.1 double neuron_ini[2]

Initial value:

17.84.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.85 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

- double neuron_ini [2]
- double synapses_ini [1]

17.85.1 Function Documentation

17.85.1.1 void modelDefinition (NNmodel & model)

17.85.2 Variable Documentation

17.85.2.1 double neuron_ini[2]

```
= {
0.0,
0.0
```

17.85.2.2 double synapses_ini[1]

Initial value:

```
= {
0.0
```

17.86 model.cc File Reference

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

Functions

· void modelDefinition (NNmodel &model)

Variables

- double neuron_ini [2]
- double synapses_ini [1]

17.86.1 Function Documentation

17.86.1.1 void modelDefinition (NNmodel & model)

17.86.2 Variable Documentation

17.86.2.1 double neuron_ini[2]

Initial value:

17.86.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.87 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

```
Variables
```

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

17.87.1 Function Documentation

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

17.87.2 Variable Documentation

```
17.87.2.1 double neuron_ini[2]
```

Initial value:

17.87.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.88 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]

17.88.1 Function Documentation

17.88.1.1 void modelDefinition (NNmodel & model)

17.88.2 Variable Documentation

17.88.2.1 double neuron_ini[2]

```
0.0,
```

```
17.88.2.2 double neuron_p[1]
```

```
Initial value:
```

```
={
1.0
```

17.88.2.3 double neuron_p2[1]

Initial value:

```
={
2.0
}
```

17.88.2.4 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.89 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]

17.89.1 Function Documentation

17.89.1.1 void modelDefinition (NNmodel & model)

17.89.2 Variable Documentation

17.89.2.1 double neuron_ini[2]

```
17.89.2.2 double neuron_p[1]
Initial value:
={
1.0
17.89.2.3 double neuron_p2[1]
Initial value:
={
2.0
17.89.2.4 double synapses_ini[1]
Initial value:
= {
17.90 model.cc File Reference
#include "modelSpec.h"
Functions

    void modelDefinition (NNmodel &model)

Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.90.1 Function Documentation
17.90.1.1 void modelDefinition ( NNmodel & model )
17.90.2 Variable Documentation
17.90.2.1 double neuron_ini[2]
Initial value:
= {
0.0,
   0.0
17.90.2.2 double synapses_ini[1]
Initial value:
= { 0.0
```

17.91 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

- double neuron_ini [2]
- double synapses_ini [1]

17.91.1 Function Documentation

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

17.91.2 Variable Documentation

17.91.2.1 double neuron_ini[2]

Initial value:

17.91.2.2 double synapses_ini[1]

Initial value:

```
= {
```

17.92 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

- double neuron_ini [2]
- double synapses_ini [1]

```
17.92.1 Function Documentation
17.92.1.1 void modelDefinition ( NNmodel & model )
17.92.2 Variable Documentation
17.92.2.1 double neuron_ini[2]
Initial value:
    0.0,
17.92.2.2 double synapses_ini[1]
Initial value:
= { 0.0
17.93 model.cc File Reference
#include "modelSpec.h"
Functions
    • void modelDefinition (NNmodel &model)
Variables
    • double neuron_ini [2]
    • double synapses_ini [1]
17.93.1 Function Documentation
17.93.1.1 void modelDefinition ( NNmodel & model )
17.93.2 Variable Documentation
17.93.2.1 double neuron_ini[2]
Initial value:
   0.0,
    0.0
17.93.2.2 double synapses_ini[1]
Initial value:
= {
```

17.94 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]

17.94.1 Function Documentation

17.94.1.1 void modelDefinition (NNmodel & model)

17.94.2 Variable Documentation

17.94.2.1 double neuron_ini[2]

Initial value:

17.94.2.2 double neuron_p[1]

Initial value:

```
={
1.0
}
```

17.94.2.3 double neuron_p2[1]

Initial value:

```
={
2.0
}
```

17.94.2.4 double synapses_ini[1]

Initial value:

```
= { 0.0
```

17.95 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]

17.95.1 Function Documentation

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

17.95.2 Variable Documentation

17.95.2.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
```

17.95.2.2 double neuron_p[1]

Initial value:

```
={
1.0
}
```

17.95.2.3 double neuron_p2[1]

Initial value:

```
= { 2.0
```

17.95.2.4 double synapses_ini[1]

Initial value:

17.96 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

```
Variables
```

```
• double neuron_ini [2]
```

```
• double synapses_ini [1]
```

17.96.1 Function Documentation

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

17.96.2 Variable Documentation

```
17.96.2.1 double neuron_ini[2]
```

Initial value:

17.96.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.97 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]

17.97.1 Function Documentation

17.97.1.1 void modelDefinition (NNmodel & model)

17.97.2 Variable Documentation

17.97.2.1 double neuron_ini[2]

```
17.97.2.2 double synapses_ini[1]
Initial value:
= { 0.0
17.97.2.3 double synapses_p[1]
Initial value:
= { 0.0
17.98 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]
17.98.1 Function Documentation
17.98.1.1 void modelDefinition ( NNmodel & model )
17.98.2 Variable Documentation
17.98.2.1 double neuron_ini[2]
Initial value:
= {
    0.0,
    0.0
17.98.2.2 double synapses_ini[1]
Initial value:
= {
17.98.2.3 double synapses_p[1]
Initial value:
= { 0.0
```

17.99 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]

17.99.1 Function Documentation

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

17.99.2 Variable Documentation

17.99.2.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
```

17.99.2.2 double synapses_ini[1]

Initial value:

17.99.2.3 double synapses_p[1]

Initial value:

17.100 model.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

Variables

```
• double neuron_ini [2]
```

```
• 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[2]

Initial value:

17.100.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.101 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]

17.101.1 Function Documentation

17.101.1.1 void modelDefinition (NNmodel & model)

17.101.2 Variable Documentation

17.101.2.1 double neuron_ini[2]

```
0.0,
```

```
17.101.2.2 double neuron_p[1]
```

```
Initial value:
```

```
={
1.0
```

17.101.2.3 double neuron_p2[1]

Initial value:

```
={
2.0
```

17.101.2.4 double synapses_ini[1]

Initial value:

```
= {
 0.0
```

17.102 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]

17.102.1 Function Documentation

17.102.1.1 void modelDefinition (NNmodel & model)

17.102.2 Variable Documentation

17.102.2.1 double neuron_ini[2]

```
17.102.2.2 double neuron_p[1]
Initial value:
    1.0
17.102.2.3 double neuron_p2[1]
Initial value:
   2.0
17.102.2.4 double synapses_ini[1]
Initial value:
= {
17.103 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]
17.103.1 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:
```

```
17.103.2.2 double synapses_ini[1]
```

```
Initial value:
```

```
= {
 0.0
}
```

17.103.2.3 double synapses_p[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_p [1]
- double synapses_ini [1]

17.104.1 Function Documentation

17.104.1.1 void modelDefinition (NNmodel & model)

17.104.2 Variable Documentation

17.104.2.1 double neuron_ini[2]

Initial value:

17.104.2.2 double synapses_ini[1]

Initial value:

```
= {
 0.0
}
```

17.104.2.3 double synapses_p[1]

```
= {
 0.0
```

17.105 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]

17.105.1 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:

17.105.2.2 double neuron_p[1]

Initial value:

```
= {
```

17.105.2.3 double neuron_p2[1]

Initial value:

```
={
2.0
}
```

17.105.2.4 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:

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]

```
0.0,
    0.0
17.107.2.2 double synapses_ini[1]
Initial value:
```

```
0.0
```

17.108 Model Schmuker 2014 classifier.cc File Reference

```
#include "modelSpec.h"
#include <iostream>
```

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

void modelDefinition (NNmodel &model)

```
17.108.1 Macro Definition Documentation
```

```
17.108.1.1 #define CLUST_SIZE_AN (NETWORK_SCALE * 6)
```

17.108.1.2 #define CLUST_SIZE_PN (NETWORK_SCALE * 6)

17.108.1.3 #define CLUST_SIZE_RN (NETWORK_SCALE * 6)

17.108.1.4 #define DT 0.5

17.108.1.5 #define NETWORK_SCALE 10

17.108.1.6 #define NUM_CLASSES 3

17.108.1.7 #define NUM_FEATURES 4

17.108.1.8 #define NUM_VR 10

17.108.1.9 #define SYNAPSE_TAU_ANAN 8.0

```
17.108.1.10 #define SYNAPSE_TAU_PNAN 1.0

17.108.1.11 #define SYNAPSE_TAU_PNPN 5.5

17.108.1.12 #define SYNAPSE_TAU_RNPN 1.0

17.108.2 Function Documentation

17.108.2.1 void modelDefinition ( NNmodel & model )

17.109 modelSpec.cc File Reference

17.110 modelSpec.cc File Reference

#include "modelSpec.h"
#include "global.h"
#include "stringUtils.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

```
17.110.1 Macro Definition Documentation
17.110.1.1 #define MODELSPEC_CC
17.110.2 Function Documentation
17.110.2.1 void initGeNN ( )
```

Method for GeNN initialisation (by preparing standard models)

```
17.110.3 Variable Documentation
```

17.110.3.1 unsigned int GeNNReady = 0

17.111 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 "neuronModels.h"
#include "synapseModels.h"
#include "postSynapseModels.h"
#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

17.111.1 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.

17.111.2 Macro Definition Documentation

17.111.2.1 #define _MODELSPEC_H_

macro for avoiding multiple inclusion during compilation

17.111.2.2 #define AUTODEVICE -1

Macro attaching the label AUTODEVICE to flag -1. Used by setGPUDevice.

17.111.2.3 #define CPU 0

Macro attaching the label "CPU" to flag 0.

17.111.2.4 #define EXITSYN 0

Macro attaching the label "EXITSYN" to flag 0 (excitatory synapse)

17.111.2.5 #define GPU 1

Macro attaching the label "GPU" to flag 1.

17.111.2.6 #define INHIBSYN 1

Macro attaching the label "INHIBSYN" to flag 1 (inhibitory synapse)

17.111.2.7 #define LEARNING 1

Macro attaching the label "LEARNING" to flag 1.

17.111.2.8 #define NO_DELAY 0

Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

17.111.2.9 #define NOLEARNING 0

Macro attaching the label "NOLEARNING" to flag 0.

17.111.3 Enumeration Type Documentation

17.111.3.1 enum FloatType

Enumerator

GENN_LONG_DOUBLE

17.111.3.2 enum SynapseConnType

Enumerator

ALLTOALL

DENSE

SPARSE

```
17.111.3.3 enum SynapseGType Enumerator
```

INDIVIDUALG GLOBALG INDIVIDUALID

17.111.4 Function Documentation

```
17.111.4.1 void initGeNN ( )
```

Method for GeNN initialisation (by preparing standard models)

17.111.5 Variable Documentation

17.111.5.1 unsigned int GeNNReady

17.112 neuronModels.cc File Reference

```
#include "neuronModels.h"
#include "stringUtils.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"

17.112.1 Macro Definition Documentation

17.112.1.1 #define NEURONMODELS_CC

17.112.2 Function Documentation

17.112.2.1 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).

17.112.3 Variable Documentation

17.112.3.1 unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

17.112.3.2 unsigned int IZHIKEVICH_V

variable attaching the name "IZHIKEVICH_V"

17.112.3.3 unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

17.112.3.4 vector<neuronModel> nModels

Global C++ vector containing all neuron model descriptions.

17.112.3.5 unsigned int POISSONNEURON

variable attaching the name "POISSONNEURON"

17.112.3.6 unsigned int SPIKESOURCE

variable attaching the name "SPIKESOURCE"

17.112.3.7 unsigned int TRAUBMILES

variable attaching the name "TRAUBMILES"

17.112.3.8 unsigned int TRAUBMILES_ALTERNATIVE

variable attaching the name "TRAUBMILES_ALTERNATIVE"

17.112.3.9 unsigned int TRAUBMILES_FAST

variable attaching the name "TRAUBMILES_FAST"

```
17.112.3.10 unsigned int TRAUBMILES_PSTEP
```

variable attaching the name "TRAUBMILES_PSTEP"

17.112.3.11 unsigned int TRAUBMILES_SAFE

variable attaching the name "TRAUBMILES_SAFE"

17.113 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

17.113.1 Function Documentation

17.113.1.1 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).

17.113.2 Variable Documentation

17.113.2.1 unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

17.113.2.2 unsigned int IZHIKEVICH_V

variable attaching the name "IZHIKEVICH_V"

17.113.2.3 unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

17.113.2.4 const unsigned int MAXNRN = 7

17.113.2.5 vector<neuronModel> nModels

Global C++ vector containing all neuron model descriptions.

17.113.2.6 unsigned int POISSONNEURON

variable attaching the name "POISSONNEURON"

17.113.2.7 unsigned int SPIKESOURCE

variable attaching the name "SPIKESOURCE"

17.113.2.8 unsigned int TRAUBMILES

variable attaching the name "TRAUBMILES"

17.113.2.9 unsigned int TRAUBMILES_ALTERNATIVE

variable attaching the name "TRAUBMILES_ALTERNATIVE"

17.113.2.10 unsigned int TRAUBMILES_FAST

variable attaching the name "TRAUBMILES_FAST"

17.113.2.11 unsigned int TRAUBMILES_PSTEP

variable attaching the name "TRAUBMILES_PSTEP"

17.113.2.12 unsigned int TRAUBMILES_SAFE

variable attaching the name "TRAUBMILES_SAFE"

17.114 OneComp.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

Functions

void modelDefinition (NNmodel &model)

Variables

```
• double exlzh_p [5]
```

- double exlzh_ini [2]
- double mySyn_p [3]
- double postExp [2]
- double * postSynV = NULL

17.114.1 Function Documentation

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

17.114.2 Variable Documentation

17.114.2.1 double exlzh_ini[2]

Initial value:

```
={
-65,
-20
```

17.114.2.2 double exlzh_p[5]

Initial value:

```
={
0.02,
0.2,
-65,
6,
4.0
```

17.114.2.3 double mySyn_p[3]

Initial value:

```
= {
    0.0,
    -20.0,
    1.0
```

```
17.114.2.4 double postExp[2]
```

```
Initial value:
```

```
= {
1.0,
0.0
```

17.114.2.5 double* postSynV = NULL

17.115 OneComp_model.cc File Reference

```
#include "OneComp_CODE/definitions.h"
```

Macros

• #define ONECOMP MODEL CC

17.115.1 Macro Definition Documentation

17.115.1.1 #define ONECOMP_MODEL_CC_

17.116 OneComp_model.h File Reference

```
#include "OneComp.cc"
```

Classes

• class neuronpop

17.117 OneComp_sim.cc File Reference

```
#include "OneComp_sim.h"
```

Functions

• int main (int argc, char *argv[])

17.117.1 Function Documentation

17.117.1.1 int main (int argc, char * argv[])

17.118 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

```
17.118.1 Macro Definition Documentation
```

```
17.118.1.1 #define DBG_SIZE 10000
```

17.118.1.2 #define T_REPORT_TME 100.0

17.118.1.3 #define TOTAL_TME 5000

17.118.2 Variable Documentation

17.118.2.1 CStopWatch timer

17.119 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

17.119.1 Function Documentation

```
17.119.1.1 for ( )
```

- 17.119.1.2 if (cpu_only && which==1)
- 17.119.2 Variable Documentation
- 17.119.2.1 unsigned int cpu_only = 0
- 17.119.2.2 unsigned int dbgMode = 0
- 17.119.2.3 unsigned int fixsynapse = 0
- 17.119.2.4 string ftype = "FLOAT"

17.119.2.5 string option

17.120 PoissonIzh-model.cc File Reference

```
#include "PoissonIzh-model.h"
#include "PoissonIzh_CODE/definitions.h"
#include "modelSpec.h"
```

Macros

• #define _POISSONIZHMODEL_CC_

17.120.1 Macro Definition Documentation

```
17.120.1.1 #define _POISSONIZHMODEL_CC_
```

17.121 PoissonIzh-model.h File Reference

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

Classes

· class classol

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

17.122 PoissonIzh.cc File Reference

```
#include "modelSpec.h"
#include "global.h"
#include "sizes.h"
```

Functions

• void modelDefinition (NNmodel &model)

Variables

- double myPOI_p [4]
- double myPOI_ini [4]
- double exlzh_p [4]
- double exlzh_ini [2]
- double mySyn_p [3]
- double mySyn_ini [1]
- double postExp [2]
- double * postSynV = NULL

17.122.1 Function Documentation

17.122.2 Variable Documentation

17.122.1.1 void modelDefinition (NNmodel & model)

```
17.122.2.1 double exlzh_ini[2]
 Initial value:
          -65,
-20
 17.122.2.2 double exlzh_p[4]
 Initial value:
 = {
          0.02,
0.2,
-65,
 17.122.2.3 double myPOI_ini[4]
 Initial value:
 = {
-60.0,
00.0,
0,
-10.0
 17.122.2.4 double myPOI_p[4]
 Initial value:
 = {
   1,
2.5,
20.0,
    -60.0
 17.122.2.5 double mySyn_ini[1]
 Initial value:
 = {
0.0
}
 17.122.2.6 double mySyn_p[3]
 Initial value:
- {
0.0,
-20.0,
1.0
```

17.122.2.7 double postExp[2]

```
Initial value:
```

```
= {
1.0,
0.0
```

17.122.2.8 double* postSynV = NULL

17.123 PoissonIzh_sim.cc File Reference

```
#include "PoissonIzh_sim.h"
```

Functions

• int main (int argc, char *argv[])

17.123.1 Function Documentation

```
17.123.1.1 int main ( int argc, char * argv[] )
```

17.124 Poissonlzh_sim.h File Reference

```
#include "utils.h"
#include "stringUtils.h"
#include "hr_time.h"
#include <cuda_runtime.h>
#include "PoissonIzh.cc"
#include <cassert>
#include "PoissonIzh-model.h"
#include "PoissonIzh-model.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);
- #define T_REPORT_TME 1000.0
- #define SYN_OUT_TME 2000.0
- #define TOTAL_TME 5000

Variables

- scalar InputBaseRate = 2e-02
- · CStopWatch timer

17.124.1 Macro Definition Documentation

```
17.124.1.1 #define MYRAND( \mbox{ } \
```

17.124.1.2 #define SYN_OUT_TME 2000.0

17.124.1.3 #define T_REPORT_TME 1000.0

```
17.124.1.4 #define TOTAL_TME 5000
```

17.124.2 Variable Documentation

17.124.2.1 scalar InputBaseRate = 2e-02

17.124.2.2 CStopWatch timer

17.125 postSynapseModels.cc File Reference

```
#include "postSynapseModels.h"
#include "stringUtils.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
- 17.125.1 Macro Definition Documentation
- 17.125.1.1 #define POSTSYNAPSEMODELS CC
- 17.125.2 Function Documentation
- 17.125.2.1 void preparePostSynModels ()

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

17.125.3 Variable Documentation

17.125.3.1 unsigned int EXPDECAY

17.125.3.2 unsigned int IZHIKEVICH_PS

17.125.3.3 vector<postSynModel> postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

17.126 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

17.126.1 Function Documentation

```
17.126.1.1 void preparePostSynModels ( )
```

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

```
17.126.2 Variable Documentation
```

17.126.2.1 unsigned int EXPDECAY

17.126.2.2 unsigned int IZHIKEVICH_PS

17.126.2.3 const unsigned int MAXPOSTSYN = 2

17.126.2.4 vector<postSynModel> postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

17.127 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

17.127.1 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().

17.127.2 Macro Definition Documentation

17.127.2.1 #define RANDOMGEN_CC

macro for avoiding multiple inclusion during compilation

17.128 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

17.128.1 Detailed Description

header file containing the class definition for a uniform random generator based on the ISAAC random number generator

17.128.2 Macro Definition Documentation

17.128.2.1 #define RANDOMGEN_H

macro for avoiding multiple inclusion during compilation

17.129 Schmuker2014_classifier.cc File Reference

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <cstdlib>
#include <cstring>
#include <time.h>
#include "Schmuker2014_classifier.h"
#include "schmuker_2014_classifier_CODE/definitions.h"
#include "sparseUtils.h"
#include "sparseUtils.h"
```

Macros

• #define _SCHMUKER2014_CLASSIFIER_

macro for avoiding multiple inclusion during compilation

17.129.1 Macro Definition Documentation

```
17.129.1.1 #define _SCHMUKER2014_CLASSIFIER_
```

macro for avoiding multiple inclusion during compilation

17.130 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.

17.130.1 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".

17.131 simulation_neuron_policy_pre_post_var.h File Reference

Classes

• class SimulationNeuronPolicyPrePostVar

17.132 simulation_neuron_policy_pre_var.h File Reference

Classes

• class SimulationNeuronPolicyPreVar

17.133 simulation_synapse_policy_dense.h File Reference

```
#include <functional>
#include <numeric>
```

Classes

class SimulationSynapsePolicyDense

17.134 simulation_synapse_policy_none.h File Reference

```
#include <numeric>
```

Classes

• class SimulationSynapsePolicyNone

17.135 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)

```
17.135.1 Macro Definition Documentation
```

```
17.135.1.1 #define SETUP_THE_C( / )
```

Value:

17.136 simulation_test.h File Reference

```
#include "gtest/gtest.h"
```

Classes

class SimulationTest

17.137 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)

17.137.1 Macro Definition Documentation

17.137.1.1 #define ASSIGN_ARRAY_VARS(ARRAY_NAME, VAR_PREFIX, COUNT)

Value:

17.138 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
- 17.138.1 Macro Definition Documentation
- 17.138.1.1 #define _FTYPE GENN_FLOAT
- 17.138.1.2 #define _NExc 8000
- 17.138.1.3 #define _Nlnh 2000
- 17.138.1.4 #define _NMaxConnP0 845
- 17.138.1.5 #define _NMaxConnP1 301
- 17.138.1.6 #define _NMaxConnP2 834
- 17.138.1.7 #define _NMaxConnP3 237
- 17.138.1.8 #define inputFac 1
- 17.138.1.9 #define scalar float
- 17.138.1.10 #define SCALAR_MAX FLT_MAX
- 17.138.1.11 #define SCALAR_MIN FLT_MIN
- 17.139 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
- 17.139.1 Macro Definition Documentation
- 17.139.1.1 #define _FTYPE GENN_FLOAT
- 17.139.1.2 #define _NAL 100
- 17.139.1.3 #define _NLB 100
- 17.139.1.4 #define _NLHI 20
- 17.139.1.5 #define _NMB 1000
- 17.139.1.6 #define scalar float
- 17.139.1.7 #define SCALAR_MAX 3.40282e+38f

17.139.1.8 #define SCALAR_MIN 1.17549e-38f

17.140 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
- 17.140.1 Macro Definition Documentation
- 17.140.1.1 #define _FTYPE GENN_FLOAT
- 17.140.1.2 #define _NAL 100
- 17.140.1.3 #define _NLB 100
- 17.140.1.4 #define _NLHI 20
- 17.140.1.5 #define _NMB 1000
- 17.140.1.6 #define scalar float
- 17.140.1.7 #define SCALAR_MAX 3.40282e+38f
- 17.140.1.8 #define SCALAR_MIN 1.17549e-38f

17.141 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
- 17.141.1 Macro Definition Documentation
- 17.141.1.1 #define _FTYPE GENN_FLOAT
- 17.141.1.2 #define _NAL 100
- 17.141.1.3 #define _NLB 100
- 17.141.1.4 #define _NLHI 20

17.141.1.5 #define _NMB 1000

17.141.1.6 #define gPNKC_GLOBAL 0.0025 17.141.1.7 #define scalar float 17.141.1.8 #define SCALAR_MAX 3.40282e+38f 17.141.1.9 #define SCALAR_MIN 1.17549e-38f 17.142 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 17.142.1 Macro Definition Documentation 17.142.1.1 #define _FTYPE GENN_FLOAT 17.142.1.2 #define _NAL 100 17.142.1.3 #define _NLB 100 17.142.1.4 #define _NLHI 20 17.142.1.5 #define _NMB 1000 17.142.1.6 #define scalar float 17.142.1.7 #define SCALAR_MAX 3.40282e+38f 17.142.1.8 #define SCALAR_MIN 1.17549e-38f 17.143 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 17.143.1 Macro Definition Documentation 17.143.1.1 #define _FTYPE GENN_FLOAT 17.143.1.2 #define _NC1 1

```
17.143.1.3 #define scalar float
17.143.1.4 #define SCALAR_MAX 3.40282e+38f
17.143.1.5 #define SCALAR_MIN 1.17549e-38f
```

Macros

• #define _NPoisson 100

17.144 sizes.h File Reference

- #define NIzh 10
- #define _FTYPE GENN_FLOAT
- · #define scalar float
- #define SCALAR_MIN 1.17549e-38f
- #define SCALAR_MAX 3.40282e+38f

```
17.144.1 Macro Definition Documentation
```

```
17.144.1.1 #define _FTYPE GENN_FLOAT
```

17.144.1.2 #define _NIzh 10

17.144.1.3 #define _NPoisson 100

17.144.1.4 #define scalar float

17.144.1.5 #define SCALAR_MAX 3.40282e+38f

17.144.1.6 #define SCALAR_MIN 1.17549e-38f

17.145 sparseProjection.h File Reference

Classes

struct SparseProjection

class (struct) for defining a spars connectivity projection

17.146 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)

- 17.146.1 Macro Definition Documentation
- 17.146.1.1 #define SPARSEUTILS_CC
- 17.146.2 Function Documentation
- 17.146.2.1 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)

17.146.2.2 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.

17.146.2.3 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)

17.146.2.4 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)

17.146.2.5 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)

17.147 sparseUtils.h File Reference

```
#include "sparseProjection.h"
#include "global.h"
#include <cstdlib>
#include <cstdio>
#include <string>
#include <cmath>
```

Functions

template < class DATATYPE >
 unsigned int countEntriesAbove (DATATYPE *Array, int sz, double includeAbove)

Utility to count how many entries above a specified value exist in a float array.

template < class DATATYPE >

DATATYPE getG (DATATYPE *wuvar, SparseProjection *sparseStruct, int x, int y)

DEPRECATED Utility to get a synapse weight from a SPARSE structure by x,y coordinates NB: as the Sparse—Projection struct doesnt hold the preN size (it should!) it is not possible to check the parameter validity. This fn may therefore crash unless user knows max poss X.

• template<class DATATYPE >

float getSparseVar (DATATYPE *wuvar, SparseProjection *sparseStruct, int x, int y)

template < class DATATYPE >

void setSparseConnectivityFromDense (DATATYPE *wuvar, int preN, int postN, DATATYPE *tmp_gRNPN, SparseProjection *sparseStruct)

Function for setting the values of SPARSE connectivity matrix.

template < class DATATYPE >

void createSparseConnectivityFromDense (DATATYPE *wuvar, int preN, int postN, DATATYPE *tmp_gR← NPN, SparseProjection *sparseStruct, bool runTest)

Utility to generate the SPARSE connectivity structure from a simple all-to-all array.

void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection *C)

Utility to generate the SPARSE array structure with post-to-pre arrangement from the original pre-to-post arrangement where postsynaptic feedback is necessary (learning etc)

• void createPreIndices (unsigned int preN, unsigned int postN, SparseProjection *C)

Function to create the mapping from the normal index array "ind" to the "reverse" array revInd, i.e. the inverse mapping of remap. This is needed if SynapseDynamics accesses pre-synaptic variables.

- void initializeSparseArray (SparseProjection C, unsigned int *dInd, unsigned int *dIndInG, unsigned int preN)
 Function for initializing conductance array indices for sparse matrices on the GPU (by copying the values from the host)
- void initializeSparseArrayRev (SparseProjection C, unsigned int *dRevInd, unsigned int *dRevIndInG, unsigned int *dRemap, unsigned int postN)

Function for initializing reversed conductance array indices for sparse matrices on the GPU (by copying the values from the host)

void initializeSparseArrayPreInd (SparseProjection C, unsigned int *dPreInd)

Function for initializing reversed conductance arrays presynaptic indices for sparse matrices on the GPU (by copying the values from the host)

17.147.1 Function Documentation

17.147.1.1 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.

17.147.1.2 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)

17.147.1.3 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.

17.147.1.4 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.

17.147.1.5 template < class DATATYPE > DATATYPE getG (DATATYPE * wuvar, SparseProjection * sparseStruct, int x, int y)

DEPRECATED Utility to get a synapse weight from a SPARSE structure by x,y coordinates NB: as the Sparse Projection struct doesnt hold the preN size (it should!) it is not possible to check the parameter validity. This fn may therefore crash unless user knows max poss X.

- 17.147.1.6 template < class DATATYPE > float getSparseVar (DATATYPE * wuvar, SparseProjection * sparseStruct, int x, int y)
- 17.147.1.7 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)

17.147.1.8 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)

17.147.1.9 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)

17.147.1.10 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.

17.148 stringUtils.cc File Reference

```
#include "modelSpec.h"
#include "stringUtils.h"
#include "utils.h"
#include <limits>
#include <regex>
```

Macros

- #define STRINGUTILS_CC
- #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.
- void name_substitutions (string &code, const string &prefix, const vector< string > &names, const string &postfix)

This function performs a list of name substitutions for variables in code snippets.

- void value_substitutions (string &code, const vector< string > &names, const vector< double > &values)

 This function performs a list of value substitutions for parameters in code snippets.
- void extended_name_substitutions (string &code, const string &prefix, const vector< string > &names, const string &ext, const string &postfix)

This function performs a list of name substitutions for variables in code snippets where the variables have an extension in their names (e.g. "_pre").

 void extended_value_substitutions (string &code, const vector< string > &names, const string &ext, const vector< double > &values)

This function performs a list of value substitutions for parameters in code snippets where the parameters have an extension in their names (e.g. "_pre").

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 (string code, 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 NNmodel &model, unsigned int src, unsigned int trg, unsigned int nt_pre, unsigned int nt_post, const string &offsetPre, const string &offsetPost, const string &preldx, const string &devPrefix)

Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

- 17.148.1 Macro Definition Documentation
- 17.148.1.1 #define REGEX OPERATIONAL
- 17.148.1.2 #define STRINGUTILS CC
- 17.148.2 Function Documentation
- 17.148.2.1 void checkUnreplacedVariables (string code, string codeName)

This function checks for unknown variable definitions and returns a gennError if any are found.

17.148.2.2 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).

17.148.2.3 void extended_name_substitutions (string & code, const string & prefix, const vector< string > & names, const string & ext, const string & postfix)

This function performs a list of name substitutions for variables in code snippets where the variables have an extension in their names (e.g. " pre").

17.148.2.4 void extended_value_substitutions (string & code, const vector< string > & names, const string & ext, const vector< double > & values)

This function performs a list of value substitutions for parameters in code snippets where the parameters have an extension in their names (e.g. "_pre").

17.148.2.5 void name_substitutions (string & code, const string & prefix, const vector < string > & names, const string & postfix)

This function performs a list of name substitutions for variables in code snippets.

17.148.2.6 void neuron_substitutions_in_synaptic_code (string & wCode, const NNmodel & model, unsigned int src, unsigned int trg, unsigned int nt_pre, unsigned int nt_post, const string & offsetPre, const string & offsetPost, const string & preldx, const string & postldx, 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
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
nt_pre	the neuron type of the pre-synaptic neuron
nt_post	the neuron type of the post-synaptic neuron
offsetPre	delay slot offset expression for pre-synaptic vars
offsetPost	delay slot offset expression for post-synaptic vars
preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)
devPrefix	device prefix, "dd_" for GPU, nothing for CPU

17.148.2.7 void substitute (string & s, const string & trg, const string & rep)

Tool for substituting strings in the neuron code strings or other templates.

17.148.2.8 void value_substitutions (string & code, const vector < string > & names, const vector < double > & values)

This function performs a list of value substitutions for parameters in code snippets.

17.149 stringUtils.h File Reference

```
#include <string>
#include <sstream>
#include <vector>
```

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

void substitute (string &s, const string &trg, const string &rep)

Tool for substituting strings in the neuron code strings or other templates.

void name_substitutions (string &code, const string &prefix, const vector< string > &names, const string &postfix="")

This function performs a list of name substitutions for variables in code snippets.

void value_substitutions (string &code, const vector < string > &names, const vector < double > &values)

This function performs a list of value substitutions for parameters in code snippets.

• void extended_name_substitutions (string &code, const string &prefix, const vector< string > &names, const string &ext, const string &postfix="")

This function performs a list of name substitutions for variables in code snippets where the variables have an extension in their names (e.g. "_pre").

 void extended_value_substitutions (string &code, const vector< string > &names, const string &ext, const vector< double > &values) This function performs a list of value substitutions for parameters in code snippets where the parameters have an extension in their names (e.g. "_pre").

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 (string code, 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 NNmodel &model, unsigned int src, unsigned int trg, unsigned int nt_pre, unsigned int nt_post, const string &offsetPre, const string &offsetPost, const string &preldx, const string &devPrefix)

Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

17.149.1 Macro Definition Documentation

17.149.1.1 #define tS(X) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

17.149.2 Function Documentation

17.149.2.1 void checkUnreplacedVariables (string code, string codeName)

This function checks for unknown variable definitions and returns a gennError if any are found.

17.149.2.2 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).

17.149.2.3 void extended_name_substitutions (string & code, const string & prefix, const vector < string > & names, const string & ext, const string & postfix = " ")

This function performs a list of name substitutions for variables in code snippets where the variables have an extension in their names (e.g. "_pre").

17.149.2.4 void extended_value_substitutions (string & code, const vector< string > & names, const string & ext, const vector< double > & values)

This function performs a list of value substitutions for parameters in code snippets where the parameters have an extension in their names (e.g. "_pre").

17.149.2.5 void name_substitutions (string & code, const string & prefix, const vector < string > & names, const string & postfix = " ")

This function performs a list of name substitutions for variables in code snippets.

17.149.2.6 void neuron_substitutions_in_synaptic_code (string & wCode, const NNmodel & model, unsigned int src, unsigned int trg, unsigned int nt_pre, unsigned int nt_post, const string & offsetPre, const string & offsetPost, const string & postldx, 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
model	the neuronal network model to generate code for

Parameters

src	the number of the src neuron population
trg	the number of the target neuron population
nt_pre	the neuron type of the pre-synaptic neuron
nt_post	the neuron type of the post-synaptic neuron
offsetPre	delay slot offset expression for pre-synaptic vars
offsetPost	delay slot offset expression for post-synaptic vars
preldx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postldx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)
devPrefix	device prefix, "dd_" for GPU, nothing for CPU

17.149.2.7 void substitute (string & s, const string & trg, const string & rep)

Tool for substituting strings in the neuron code strings or other templates.

```
17.149.2.8 template < class T > std::string to String ( T t )
```

template functions for conversion of various types to C++ strings

17.149.2.9 void value_substitutions (string & code, const vector < string > & names, const vector < double > & values)

This function performs a list of value substitutions for parameters in code snippets.

17.150 synapseModels.cc File Reference

```
#include "synapseModels.h"
#include "stringUtils.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.

```
17.150.1 Macro Definition Documentation
```

17.150.1.1 #define SYNAPSEMODELS_CC

17.150.2 Function Documentation

17.150.2.1 void prepareWeightUpdateModels ()

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

17.150.3 Variable Documentation

17.150.3.1 unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

17.150.3.2 unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

17.150.3.3 unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

17.150.3.4 vector<weightUpdateModel> weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

17.151 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

17.151.1 Function Documentation

```
17.151.1.1 void prepareWeightUpdateModels ( )
```

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

17.151.2 Variable Documentation

17.151.2.1 unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the the primitive STDP model for learning.

17.151.2.2 unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to the graded synapse wrt the presynaptic voltage.

17.151.2.3 unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to the non-learning synapse.

17.151.2.4 const unsigned int SYNTYPENO = 4

17.151.2.5 vector<weightUpdateModel> weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

17.152 SynDelay.cc File Reference

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

Functions

• void modelDefinition (NNmodel &model)

17.152.1 Function Documentation

17.152.1.1 void modelDefinition (NNmodel & model)

17.153 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[])
```

17.153.1 Macro Definition Documentation

17.153.1.1 #define SYNDELAYSIM_CU

17.153.2 Function Documentation

17.153.2.1 int main (int argc, char * argv[])

17.154 SynDelaySim.h File Reference

Classes

class SynDelay

Macros

- #define TOTAL_TIME 5000.0f
- #define REPORT TIME 1000.0f
- 17.154.1 Macro Definition Documentation
- 17.154.1.1 #define REPORT_TIME 1000.0f
- 17.154.1.2 #define TOTAL_TIME 5000.0f

17.155 test.cc File Reference

```
#include "gtest/gtest.h"
#include "extra_global_params_in_sim_code_CODE/definitions.h"
#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 > Simulation ← TestExtraGlobalParams

Functions

- TEST_P (SimulationTestExtraGlobalParams, AcceptableError)
- INSTANTIATE TEST CASE P (SimCode, SimulationTestExtraGlobalParams, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

17.155.1 Typedef Documentation

- 17.155.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyNone > SimulationTestExtraGlobalParams
- 17.155.2 Function Documentation
- 17.155.2.1 INSTANTIATE_TEST_CASE_P (SimCode , SimulationTestExtraGlobalParams , simulatorBackends)
- 17.155.2.2 TEST_P (SimulationTestExtraGlobalParams , AcceptableError)
- 17.155.3 Variable Documentation
- 17.155.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.156 test.cc File Reference

```
#include "gtest/gtest.h"
#include "extra_global_params_in_sim_code_event_sparse_inv_CODE/definitions.
h"
#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

Functions

- TEST_P (SimulationTestExtraGlobalParams, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SimCodeEventSparseInv, SimulationTestExtraGlobalParams, simulator
 — Backends)

Variables

auto simulatorBackends = ::testing::Values(true, false)

```
17.156.1 Typedef Documentation

17.156.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicy>SimulationTestExtraGlobalParams

17.156.2 Function Documentation

17.156.2.1 INSTANTIATE_TEST_CASE_P ( SimCodeEventSparseInv , SimulationTestExtraGlobalParams , simulatorBackends )

17.156.2.2 TEST_P ( SimulationTestExtraGlobalParams , AcceptableError )

17.156.3 Variable Documentation

17.156.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.157 test.cc File Reference

#include "gtest/gtest.h"
#include "extra_global_post_param_in_sim_code_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
```

#include "../../utils/simulation_synapse_policy_dense.h"

Typedefs

Functions

- TEST_P (SimTest, AcceptableError)
- INSTANTIATE_TEST_CASE_P (extra_global_post_param_in_sim_code, SimTest, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.157.1 Typedef Documentation
- 17.157.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Dense> SimTest
- 17.157.2 Function Documentation
- 17.157.2.1 INSTANTIATE_TEST_CASE_P (extra_global_post_param_in_sim_code , SimTest , simulatorBackends)
- 17.157.2.2 TEST_P (SimTest , AcceptableError)
- 17.157.3 Variable Documentation
- 17.157.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.158 test.cc File Reference

```
#include "gtest/gtest.h"
#include "extra_global_pre_param_in_sim_code_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

Typedefs

Functions

- TEST P (SimTest, AcceptableError)
- INSTANTIATE_TEST_CASE_P (extra_global_pre_param_in_sim_code, SimTest, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.158.1 Typedef Documentation
- 17.158.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy

 Dense> SimTest
- 17.158.2 Function Documentation
- 17.158.2.1 INSTANTIATE TEST CASE P (extra global pre param in sim code, SimTest, simulatorBackends)
- 17.158.2.2 TEST_P (SimTest , AcceptableError)
- 17.158.3 Variable Documentation
- 17.158.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.159 test.cc File Reference

```
#include "gtest/gtest.h"
#include "neuron_support_code_sim_CODE/definitions.h"
#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 > Simulation
 — TestNeuronSupportCode

Functions

- TEST P (SimulationTestNeuronSupportCode, AcceptableError)
- INSTANTIATE_TEST_CASE_P (Sim, SimulationTestNeuronSupportCode, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.159.1 Typedef Documentation
- 17.159.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestNeuronSupportCode
- 17.159.2 Function Documentation
- 17.159.2.1 INSTANTIATE_TEST_CASE_P (Sim , SimulationTestNeuronSupportCode , simulatorBackends)
- 17.159.2.2 TEST_P (SimulationTestNeuronSupportCode , AcceptableError)
- 17.159.3 Variable Documentation
- 17.159.3.1 auto simulatorBackends = ::testing::Values(true, false)
- 17.160 test.cc File Reference

```
#include "gtest/gtest.h"
#include "neuron_support_code_threshold_CODE/definitions.h"
#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 > Simulation
 — TestNeuronSupportCode

Functions

- TEST_P (SimulationTestNeuronSupportCode, AcceptableError)
- INSTANTIATE TEST CASE P (Threshold, SimulationTestNeuronSupportCode, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.160.1 Typedef Documentation
- 17.160.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestNeuronSupportCode
- 17.160.2 Function Documentation

```
17.160.2.1 INSTANTIATE_TEST_CASE_P ( Threshold , SimulationTestNeuronSupportCode , simulatorBackends )
17.160.2.2 TEST_P ( SimulationTestNeuronSupportCode , AcceptableError )
17.160.3 Variable Documentation
17.160.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.161 test.cc File Reference
#include "gtest/gtest.h"
#include "post_vars_in_post_learn_CODE/definitions.h"
#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 >

     SimulationTestPostVars
Functions
   • TEST P (SimulationTestPostVars, AcceptableError)

    INSTANTIATE_TEST_CASE_P (PostLearn, SimulationTestPostVars, simulatorBackends)

Variables

    auto simulatorBackends = ::testing::Values(true, false)

17.161.1 Typedef Documentation
17.161.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy ←
         Dense > SimulationTestPostVars
17.161.2 Function Documentation
17.161.2.1 INSTANTIATE TEST_CASE P ( PostLearn , SimulationTestPostVars , simulatorBackends )
17.161.2.2 TEST_P ( SimulationTestPostVars , AcceptableError )
17.161.3 Variable Documentation
17.161.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.162 test.cc File Reference
#include "gtest/gtest.h"
#include "post_vars_in_post_learn_sparse_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
```

#include "../../utils/simulation_synapse_policy_sparse.h"

typedef SimulationTestVars
 SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicySparse
 SimulationTestPostVarsSparse

Functions

- TEST_P (SimulationTestPostVarsSparse, AcceptableError)
- INSTANTIATE_TEST_CASE_P (PostLearn, SimulationTestPostVarsSparse, simulatorBackends)

Variables

auto simulatorBackends = ::testing::Values(true, false)

17.162.1 Typedef Documentation

- 17.162.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Sparse> SimulationTestPostVarsSparse
- 17.162.2 Function Documentation
- 17.162.2.1 INSTANTIATE_TEST_CASE_P (PostLearn , SimulationTestPostVarsSparse , simulatorBackends)
- 17.162.2.2 TEST_P (SimulationTestPostVarsSparse , AcceptableError)
- 17.162.3 Variable Documentation
- 17.162.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.163 test.cc File Reference

```
#include "gtest/gtest.h"
#include "post_vars_in_sim_code_CODE/definitions.h"
#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 > SimulationTestPostVars

Functions

- TEST_P (SimulationTestPostVars, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SimCode, SimulationTestPostVars, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.163.1 Typedef Documentation
17.163.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
          Dense > SimulationTestPostVars
17.163.2 Function Documentation
17.163.2.1 INSTANTIATE_TEST_CASE_P ( SimCode , SimulationTestPostVars , simulatorBackends )
17.163.2.2 TEST_P ( SimulationTestPostVars , AcceptableError )
17.163.3 Variable Documentation
17.163.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.164 test.cc File Reference
#include "gtest/gtest.h"
#include "post vars in sim code sparse CODE/definitions.h"
#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

     SimulationTestPostVarsSparse
Functions
    • TEST_P (SimulationTestPostVarsSparse, AcceptableError)

    INSTANTIATE TEST CASE P (SimCode, SimulationTestPostVarsSparse, simulatorBackends)

Variables
    • auto simulatorBackends = ::testing::Values(true, false)
17.164.1 Typedef Documentation
17.164.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
          Sparse > SimulationTestPostVarsSparse
17.164.2 Function Documentation
17.164.2.1 INSTANTIATE_TEST_CASE_P ( SimCode , SimulationTestPostVarsSparse , simulatorBackends )
17.164.2.2 TEST_P ( SimulationTestPostVarsSparse , AcceptableError )
17.164.3 Variable Documentation
```

17.164.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.165 test.cc File Reference

#include "gtest/gtest.h"

```
#include "post_vars_in_synapse_dynamics_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_post_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

typedef SimulationTestVars
 SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicyDense > SimulationTestPostVars

Functions

- TEST P (SimulationTestPostVars, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SynapseDynamics, SimulationTestPostVars, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

17.165.1 Typedef Documentation

- 17.165.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Dense> SimulationTestPostVars
- 17.165.2 Function Documentation
- 17.165.2.1 INSTANTIATE_TEST_CASE_P (SynapseDynamics , SimulationTestPostVars , simulatorBackends)
- 17.165.2.2 TEST_P (SimulationTestPostVars , AcceptableError)
- 17.165.3 Variable Documentation
- 17.165.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.166 test.cc File Reference

```
#include <functional>
#include <numeric>
#include "gtest/gtest.h"
#include "post_vars_in_synapse_dynamics_sparse_CODE/definitions.h"
#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
 SimulationTestPostVarsSparse

Functions

- TEST P (SimulationTestPostVarsSparse, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SynapseDynamics, SimulationTestPostVarsSparse, simulatorBackends)

Variables

```
    auto simulatorBackends = ::testing::Values(true, false)
```

```
17.166.1 Typedef Documentation
```

- 17.166.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy← Sparse> SimulationTestPostVarsSparse
- 17.166.2 Function Documentation
- 17.166.2.1 INSTANTIATE_TEST_CASE_P (SynapseDynamics , SimulationTestPostVarsSparse , simulatorBackends)
- 17.166.2.2 TEST_P (SimulationTestPostVarsSparse , AcceptableError)
- 17.166.3 Variable Documentation
- 17.166.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.167 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_post_learn_CODE/definitions.h"
#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 > Simulation
 — TestPreVars

Functions

- TEST_P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE TEST CASE P (PostLearn, SimulationTestPreVars, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.167.1 Typedef Documentation
- 17.167.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestPreVars
- 17.167.2 Function Documentation
- 17.167.2.1 INSTANTIATE_TEST_CASE_P (PostLearn , SimulationTestPreVars , simulatorBackends)
- 17.167.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.167.3 Variable Documentation

```
17.167.3.1 auto simulatorBackends = ::testing::Values(true, false)
```

17.168 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_post_learn_sparse_CODE/definitions.h"
#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 > Simulation
 — TestPreVarsSparse

Functions

- TEST_P (SimulationTestPreVarsSparse, AcceptableError)
- INSTANTIATE TEST CASE P (PostLearn, SimulationTestPreVarsSparse, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.168.1 Typedef Documentation
- 17.168.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimulationTestPreVarsSparse
- 17.168.2 Function Documentation
- 17.168.2.1 INSTANTIATE_TEST_CASE_P (PostLearn , SimulationTestPreVarsSparse , simulatorBackends)
- 17.168.2.2 TEST_P (SimulationTestPreVarsSparse , AcceptableError)
- 17.168.3 Variable Documentation
- 17.168.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.169 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_sim_code_CODE/definitions.h"
#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 > Simulation
 —
 TestPreVars

Functions

- TEST P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE TEST CASE P (SimCode, SimulationTestPreVars, simulatorBackends)

Variables

```
• auto simulatorBackends = ::testing::Values(true, false)
```

17.169.1 Typedef Documentation

- 17.169.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestPreVars
- 17.169.2 Function Documentation
- 17.169.2.1 INSTANTIATE_TEST_CASE_P (SimCode , SimulationTestPreVars , simulatorBackends)
- 17.169.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.169.3 Variable Documentation
- 17.169.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.170 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_sim_code_event_CODE/definitions.h"
#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 > Simulation
 —
 TestPreVars

Functions

- TEST_P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE TEST CASE P (SimCodeEvent, SimulationTestPreVars, simulatorBackends)

Variables

auto simulatorBackends = ::testing::Values(true, false)

17.170.1 Typedef Documentation

- 17.170.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestPreVars
- 17.170.2 Function Documentation

```
17.170.2.1 INSTANTIATE_TEST_CASE_P(SimCodeEvent, SimulationTestPreVars, simulatorBackends)
17.170.2.2 TEST_P ( SimulationTestPreVars , AcceptableError )
17.170.3 Variable Documentation
17.170.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.171 test.cc File Reference
#include "gtest/gtest.h"
#include "pre_vars_in_sim_code_event_sparse_CODE/definitions.h"
#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 > Simulation←

     TestPreVars
Functions

    TEST P (SimulationTestPreVars, AcceptableError)

    INSTANTIATE_TEST_CASE_P (SimCodeEventSparse, SimulationTestPreVars, simulatorBackends)

Variables

    auto simulatorBackends = ::testing::Values(true, false)

17.171.1 Typedef Documentation
17.171.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse >
         SimulationTestPreVars
```

- 17.171.2 Function Documentation
- 17.171.2.1 INSTANTIATE_TEST_CASE_P (SimCodeEventSparse , SimulationTestPreVars , simulatorBackends)
- 17.171.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.171.3 Variable Documentation
- 17.171.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.172 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_sim_code_event_sparse_inv_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > Simulation
 — TestPreVars

Functions

- TEST P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SimCodeEventSparseInv, SimulationTestPreVars, simulatorBackends)

Variables

auto simulatorBackends = ::testing::Values(true, false)

17.172.1 Typedef Documentation

- 17.172.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimulationTestPreVars
- 17.172.2 Function Documentation
- 17.172.2.1 INSTANTIATE_TEST_CASE_P (SimCodeEventSparseInv , SimulationTestPreVars , simulatorBackends)
- 17.172.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.172.3 Variable Documentation
- 17.172.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.173 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_sim_code_sparse_CODE/definitions.h"
#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 > Simulation
 — TestPreVars

Functions

- TEST_P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE TEST CASE P (SimCodeSparse, SimulationTestPreVars, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.173.1 Typedef Documentation
```

- 17.173.1.1 typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > SimulationTestPreVars
- 17.173.2 Function Documentation
- 17.173.2.1 INSTANTIATE_TEST_CASE_P (SimCodeSparse , SimulationTestPreVars , simulatorBackends)
- 17.173.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.173.3 Variable Documentation
- 17.173.3.1 auto simulatorBackends = ::testing::Values(true, false)
- 17.174 test.cc File Reference

```
#include "gtest/gtest.h"
#include "pre_vars_in_synapse_dynamics_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > Simulation
 — TestPreVars

Functions

- TEST_P (SimulationTestPreVars, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SynapseDynamics, SimulationTestPreVars, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.174.1 Typedef Documentation
- 17.174.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestPreVars
- 17.174.2 Function Documentation
- $17.174.2.1 \quad INSTANTIATE_TEST_CASE_P \left(\begin{array}{ccc} SynapseDynamics \,, \, SimulationTestPreVars \,, \, simulatorBackends \end{array} \right)$
- 17.174.2.2 TEST_P (SimulationTestPreVars , AcceptableError)
- 17.174.3 Variable Documentation
- 17.174.3.1 auto simulatorBackends = ::testing::Values(true, false)
- 17.175 test.cc File Reference

```
#include "gtest/gtest.h"
```

```
#include "pre_vars_in_synapse_dynamics_sparse_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_sparse.h"
```

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse > Simulation
 — TestPreVarsSparse

Functions

- TEST_P (SimulationTestPreVarsSparse, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SynapseDynamics, SimulationTestPreVarsSparse, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.175.1 Typedef Documentation
```

- 17.175.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicySparse> SimulationTestPreVarsSparse
- 17.175.2 Function Documentation
- 17.175.2.1 INSTANTIATE_TEST_CASE_P (SynapseDynamics , SimulationTestPreVarsSparse , simulatorBackends)
- 17.175.2.2 TEST_P (SimulationTestPreVarsSparse , AcceptableError)
- 17.175.3 Variable Documentation
- 17.175.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.176 test.cc File Reference

```
#include "gtest/gtest.h"
#include "synapse_support_code_event_sim_code_CODE/definitions.h"
#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 > Simulation
 — TestSynapseSupportCode

Functions

- TEST_P (SimulationTestSynapseSupportCode, AcceptableError)
- INSTANTIATE_TEST_CASE_P (EventSimCode, SimulationTestSynapseSupportCode, simulatorBackends)

Variables

```
• auto simulatorBackends = ::testing::Values(true, false)
```

17.176.1 Typedef Documentation

- 17.176.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestSynapseSupportCode
- 17.176.2 Function Documentation
- 17.176.2.1 INSTANTIATE_TEST_CASE_P (EventSimCode , SimulationTestSynapseSupportCode , simulatorBackends)
- 17.176.2.2 TEST_P (SimulationTestSynapseSupportCode , AcceptableError)
- 17.176.3 Variable Documentation
- 17.176.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.177 test.cc File Reference

```
#include "gtest/gtest.h"
#include "synapse_support_code_event_threshold_CODE/definitions.h"
#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 > Simulation
 — TestSynapseSupportCode

Functions

- TEST_P (SimulationTestSynapseSupportCode, AcceptableError)
- INSTANTIATE_TEST_CASE_P (EventThreshold, SimulationTestSynapseSupportCode, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

17.177.1 Typedef Documentation

- 17.177.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestSynapseSupportCode
- 17.177.2 Function Documentation
- 17.177.2.1 INSTANTIATE_TEST_CASE_P (EventThreshold , SimulationTestSynapseSupportCode , simulatorBackends)
- $17.177.2.2 \quad TEST_P \ (\ Simulation Test Synapse Support Code \ , \ Acceptable Error \)$

17.177.3 Variable Documentation

17.177.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.178 test.cc File Reference

```
#include "gtest/gtest.h"
#include "synapse_support_code_post_learn_CODE/definitions.h"
#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 > Simulation
 — TestSynapseSupportCode

Functions

- TEST_P (SimulationTestSynapseSupportCode, AcceptableError)
- INSTANTIATE TEST CASE P (PostLearn, SimulationTestSynapseSupportCode, simulatorBackends)

Variables

- auto simulatorBackends = ::testing::Values(true, false)
- 17.178.1 Typedef Documentation
- 17.178.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestSynapseSupportCode
- 17.178.2 Function Documentation
- 17.178.2.1 INSTANTIATE_TEST_CASE_P (PostLearn , SimulationTestSynapseSupportCode , simulatorBackends)
- 17.178.2.2 TEST_P (SimulationTestSynapseSupportCode , AcceptableError)
- 17.178.3 Variable Documentation
- 17.178.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.179 test.cc File Reference

```
#include "gtest/gtest.h"
#include "synapse_support_code_sim_code_CODE/definitions.h"
#include "../../utils/simulation_test_vars.h"
#include "../../utils/simulation_neuron_policy_pre_var.h"
#include "../../utils/simulation_synapse_policy_dense.h"
```

typedef SimulationTestVars < SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense > Simulation
 — TestSynapseSupportCode

Functions

- TEST_P (SimulationTestSynapseSupportCode, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SimCode, SimulationTestSynapseSupportCode, simulatorBackends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
17.179.1 Typedef Documentation
```

- 17.179.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPreVar, SimulationSynapsePolicyDense> SimulationTestSynapseSupportCode
- 17.179.2 Function Documentation
- 17.179.2.1 INSTANTIATE_TEST_CASE_P (SimCode , SimulationTestSynapseSupportCode , simulatorBackends)
- 17.179.2.2 TEST_P (SimulationTestSynapseSupportCode , AcceptableError)
- 17.179.3 Variable Documentation
- 17.179.3.1 auto simulatorBackends = ::testing::Values(true, false)

17.180 test.cc File Reference

```
#include "gtest/gtest.h"
#include "synapse_support_code_synapse_dynamics_CODE/definitions.h"
#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 > SimulationTestSynapseSupportCode

Functions

- TEST_P (SimulationTestSynapseSupportCode, AcceptableError)
- INSTANTIATE_TEST_CASE_P (SynapseDynamics, SimulationTestSynapseSupportCode, simulator
 — Backends)

Variables

• auto simulatorBackends = ::testing::Values(true, false)

```
266
17.180.1 Typedef Documentation
17.180.1.1 typedef SimulationTestVars<SimulationNeuronPolicyPrePostVar, SimulationSynapsePolicy←
                             Dense > SimulationTestSynapseSupportCode
17.180.2 Function Documentation
17.180.2.1 \quad INSTANTIATE\_TEST\_CASE\_P \ ( \ SynapseDynamics \ , \ SimulationTestSynapseSupportCode \ , \ A support \ ( \ SynapseDynamics \ ) \ ( \ 
                             simulatorBackends )
17.180.2.2 TEST_P ( SimulationTestSynapseSupportCode , AcceptableError )
17.180.3 Variable Documentation
17.180.3.1 auto simulatorBackends = ::testing::Values(true, false)
17.181 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 (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 (string type)

                           Tool for determining the size of variable types on the current architecture.
17.181.1 Macro Definition Documentation
```

17.181.1.1 #define UTILS CC

17.181.2 Function Documentation

17.181.2.1 CUresult cudaFuncGetAttributesDriver (cudaFuncAttributes * attr, CUfunction kern)

Function for getting the capabilities of a CUDA device via the driver API.

17.181.2.2 void gennError (string error)

Function called upon the detection of an error. Outputs an error message and then exits.

17.181.2.3 unsigned int the Size (string type)

Tool for determining the size of variable types on the current architecture.

```
17.181.2.4 void writeHeader (ostream & os)
```

Function to write the comment header denoting file authorship and contact details into the generated code.

17.182 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) $x = ((x) \mid (0x800000000 >> (i)))$

Set the bit at the specified position i in x to 1.

• #define delB(x, i) x= ((x) & (\sim (0x80000000 >> (i))))

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

Functions

• CUresult cudaFuncGetAttributesDriver (cudaFuncAttributes *attr, CUfunction kern)

Function for getting the capabilities of a CUDA device via the driver API.

void gennError (string error)

Function called upon the detection of an error. Outputs an error message and then exits.

• unsigned int theSize (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.

17.182.1 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.

17.182.2 Macro Definition Documentation

```
17.182.2.1 #define _UTILS_H_
```

macro for avoiding multiple inclusion during compilation

```
17.182.2.2 #define B( x, i) ((x) & (0x800000000 >> (i)))
```

Bit tool macros.

Extract the bit at the specified position i from x

```
17.182.2.3 #define CHECK_CU_ERRORS( call ) call
```

Macros for catching errors returned by the CUDA driver and runtime APIs.

```
17.182.2.4 #define CHECK_CUDA_ERRORS( call )
```

Value:

```
{
  cudaError_t error = call;
  if (error != cudaSuccess)
  {
    cerr << __FILE__ << ": " << __LINE__;
    cerr << ": cuda runtime error " << error << ": ";
    cerr << cudaGetErrorString(error) << endl;
    exit(EXIT_FAILURE);
  }
}</pre>
```

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

```
17.182.2.6 #define setB( x, i) x = ((x) | (0x800000000 >> (i)))
```

Set the bit at the specified position i in x to 1.

17.182.3 Function Documentation

```
17.182.3.1 CUresult cudaFuncGetAttributesDriver ( cudaFuncAttributes * attr, CUfunction kern )
```

Function for getting the capabilities of a CUDA device via the driver API.

```
17.182.3.2 void gennError ( string error )
```

Function called upon the detection of an error. Outputs an error message and then exits.

```
17.182.3.3 unsigned int the Size ( string type )
```

Tool for determining the size of variable types on the current architecture.

```
17.182.3.4 void writeHeader (ostream & os)
```

Function to write the comment header denoting file authorship and contact details into the generated code.

17.183 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.

17.183.1 Detailed Description

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU.

17.183.2 Function Documentation

```
17.183.2.1 int main ( int argc, char * argv[] )
```

This function is the entry point for running the project.

17.184 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

17.184.1 Detailed Description

Header file containing global variables and macros used in running the HHVClamp/VClampGA model.

```
17.184.2 Macro Definition Documentation
```

```
17.184.2.1 #define RAND( Y, X) Y = Y * 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767
```

17.184.3 Variable Documentation

17.184.3.1 randomGen R

17.184.3.2 randomGauss RG

17.184.3.3 CStopWatch timer

REFERENCES 271

References

[1] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. 8, 9, 10, 11, 29

- [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. 36
- [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, 27
- [4] Nikolai F Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65(4):041922, 2002. 27
- [5] R. D. Traub and R. Miles. *Neural Networks of the Hippocampus*. Cambridge University Press, New York, 1991. 12, 14, 15, 16, 17, 28, 29, 38, 39

Index

FTYPE	NMaxConnP3
HHVClampParameters.h, 167	lzh_sparse_project/model/sizes.h, 234
lzh_sparse_project/model/sizes.h, 234	_NPoisson
MBody1_project/model/sizes.h, 234	PoissonIzh_project/model/sizes.h, 237
MBody_delayedSyn_project/model/sizes.h, 235	_ONECOMP_MODEL_CC_
MBody_individualID_project/model/sizes.h, 235	OneComp_model.cc, 223
MBody_userdef_project/model/sizes.h, 236	_POISSONIZHMODEL_CC_
OneComp_project/model/sizes.h, 236	PoissonIzh-model.cc, 225
PoissonIzh_project/model/sizes.h, 237	_SCHMUKER2014_CLASSIFIER_
_IZH_SPARSE_MODEL_CC_	Schmuker2014_classifier.cc, 231
lzh_sparse_model.cc, 171	_UTILS_H_
_MAP_CLASSOL_CC_	utils.h, 267
MBody1_project/model/map_classol.cc, 172	ISAAC_HPP
MBody_delayedSyn_project/model/map_classol. ↔	isaac.cc, 169
cc, 173	~NNmodel
MBody_individualID_project/model/map_classol. ←	NNmodel, 87
cc, 173	~QTIsaac
MBody_userdef_project/model/map_classol.cc, 174	QTIsaac, 101
MODELSPEC_H_	~Schmuker2014_classifier Schmuker2014_classifier, 108
modelSpec.h, 217	~SynDelay
NAL	SynDelay, 118
MBody1_project/model/sizes.h, 234	~classIzh
MBody_delayedSyn_project/model/sizes.h, 235	classIzh, 65
MBody_individualID_project/model/sizes.h, 235	~classol
MBody_userdef_project/model/sizes.h, 236	classol, 69, 70
_NC1	~neuronModel
OneComp_project/model/sizes.h, 236	neuronModel, 80
NExc	~neuronpop
Izh_sparse_project/model/sizes.h, 234	neuronpop, 82
_NInh	~postSynModel
lzh_sparse_project/model/sizes.h, 234	postSynModel, 98
_Nlzh	\sim randctx
PoissonIzh_project/model/sizes.h, 237	QTIsaac::randctx, 102
_NLB	\sim randomGauss
MBody1_project/model/sizes.h, 234	randomGauss, 103
MBody_delayedSyn_project/model/sizes.h, 235	\sim random Gen
MBody_individualID_project/model/sizes.h, 235	randomGen, 104
MBody_userdef_project/model/sizes.h, 236	∼stdRG
_NLHI	stdRG, 117
MBody1_project/model/sizes.h, 234	~weightUpdateModel
MBody_delayedSyn_project/model/sizes.h, 235	weightUpdateModel, 119
MBody_individualID_project/model/sizes.h, 235	00_MainPage.dox, 121
MBody_userdef_project/model/sizes.h, 236	01_Installation.dox, 121
_NMB	02_Quickstart.dox, 121
MBody1_project/model/sizes.h, 234	03_Examples.dox, 121
MBody_delayedSyn_project/model/sizes.h, 235	09_ReleaseNotes.dox, 121
MBody_usordef_project/model/sizes.h, 235	10_UserManual.dox, 121
MBody_userdef_project/model/sizes.h, 236 NMaxConnP0	11_Tutorial.dox, 121 12_Tutorial.dox, 121
Izh_sparse_project/model/sizes.h, 234	13_UserGuide.dox, 121
NMaxConnP1	13_OserGuide.dox, 121 14_Credits.dox, 121
Izh_sparse_project/model/sizes.h, 234	17_0160113.00X, 121
NMaxConnP2	ALLTOALL
Izh_sparse_project/model/sizes.h, 234	modelSpec.h, 217

ASSIGN_ARRAY_VARS	CONNECTIVITY_PN_PN
simulation_test_vars.h, 233	experiment.h, 131
AUTODEVICE	CONNECTIVITY_RN_PN
modelSpec.h, 217	experiment.h, 131
activateDirectInput	CPU
NNmodel, 87	modelSpec.h, 217
addInputRate	CStopWatch, 77
Schmuker2014_classifier, 108	CStopWatch, 77
addNeuronPopulation	getElapsedTime, 77
NNmodel, 87, 88	startTimer, 77
addSynapsePopulation	stopTimer, 77
NNmodel, 88, 89	cacheDir
allocate_device_mem_input	Schmuker2014_classifier, 110
classIzh, 65	calcNeurons
classol, 70	GENN_FLAGS, 62
allocate_device_mem_patterns	calcSynapseDynamics
classIzh, 65	GENN_FLAGS, 62
classol, 70	calcSynapses
allocateHostAndDeviceMemory	GENN_FLAGS, 62
Schmuker2014_classifier, 108	calculateCurrentWindowWinner
applyInputToClassifier	Schmuker2014_classifier, 108
experiment.cc, 130	calculateDerivedParameter
applyLearningRuleSynapses	dpclass, 78
Schmuker2014_classifier, 108	expDecayDp, 79
asGoodAsZero	pwSTDP_userdef, 100
GENN_PREFERENCES, 63	pwSTDP, 99
autoChooseDevice	rulkovdp, 105
GENN_PREFERENCES, 63	calculateOverallWinner
autoRefractory	Schmuker2014_classifier, 108
GENN_PREFERENCES, 63	calculateVrResponse
В	Schmuker2014_classifier, 108
	calculateWinner
gen_pnkc_syns_indivID.cc, 139 utils.h, 267	Schmuker2014_classifier, 108
baserates	CB
classol, 75	CodeHelper.h, 128
baseV	Cexp
inputSpec, 79	helper.h, 165
BlkSz	checkContents
GeNNHelperKrnls.h, 160	Schmuker2014_classifier, 108
byte	checkSizes
QTIsaac, 101	NNmodel, 90
a ribado, ror	checkUnreplacedVariables
CACHE DIR	stringUtils.cc, 241
experiment.h, 131	stringUtils.h, 243
CHECK_CU_ERRORS	chooseDevice
utils.h, 268	generateALL.cc, 149
CHECK_CUDA_ERRORS	generateALL.h, 150
utils.h, 268	classIzh, 64
CLUST_SIZE_AN	\sim classIzh, 65
Model_Schmuker_2014_classifier.cc, 214	allocate_device_mem_input, 65
CLUST_SIZE_PN	allocate_device_mem_patterns, 65
Model_Schmuker_2014_classifier.cc, 214	classIzh, 65
CLUST_SIZE_RN	copy_device_mem_input, 65
Model_Schmuker_2014_classifier.cc, 214	create_input_values, 65
CONNECTIVITY_AN_AN	d_input1, 66
experiment.h, 131	d_input2, 66
CONNECTIVITY_PN_AN	free_device_mem, 65
experiment.h, 131	gen_alltoall_syns, 65

getSpikeNumbersFromGPU, 65	sumLHI, 76
getSpikesFromGPU, 65	sumPN, 76
init, 65	theRates, 76
initializeAllVars, 65	write_kcdnsyns, 75
input1, 66	write_pnkcsyns, 75
input2, 66	write_pnlhisyns, 75
model, 66	classol_sim.cc, 121-123
output_params, 65	classol_sim.h, 123-126
output_spikes, 65	clear
output_state, 65	extra_neurons.h, 133
randomizeVar, 66	extra_postsynapses.h, 134
randomizeVarSq, 66	clearDownDevice
read_input_values, 66	Schmuker2014_classifier, 108
read_sparsesyns_par, 66	clearedDownDevice
run, 66	Schmuker2014_classifier, 110
setInput, 66	closeBrace
sum spikes, 66	CodeHelper, 76
sumPExc, 66	clusterSpikeCountAN
sumPlnh, 66	Schmuker2014_classifier, 110
write input to file, 66	CodeHelper, 76
classLabel	closeBrace, 76
Schmuker2014_classifier, 110	CodeHelper, 76
classifier	endl, 76
experiment.cc, 130	openBrace, 76
classol, 66	setVerbose, 76
\sim classol, 69, 70	CodeHelper.h, 127
	•
allocate_device_mem_input, 70	CB, 128
allocate_device_mem_patterns, 70	ENDL, 128
baserates, 75	hlp, 128
classol, 69, 70	OB, 128
d_baserates, 75	SAVEP, 128
d_pattern, 75	command_line_processing.h, 128
free_device_mem, 70	extract_bool_value, 128
generate_baserates, 70	extract_option, 128
get_kcdnsyns, 70, 71	extract_string_value, 128
getSpikeNumbersFromGPU, 71	toLower, 129
getSpikesFromGPU, 71	toUpper, 129
init, 71	compareErrTupel
model, 76	GA.cc, 135
offset, 76	connN
output_spikes, 71, 72	SparseProjection, 116
output_state, 72	convertToRateCode
p_pattern, 76	Schmuker2014_classifier, 108
pattern, 76	copy_device_mem_input
read_PNIzh1syns, 72	classIzh, 65
read_input_patterns, 72	copy_var
read kcdnsyns, 72	helper.h, 165
read_pnkcsyns, 72, 73	correctClass
read_pnlhisyns, 73	Schmuker2014_classifier, 110
read_sparsesyns_par, 73, 74	countAN
run, 74	Schmuker2014_classifier, 110
runCPU, 74	countEntriesAbove
runGPU, 74	sparseUtils.h, 239
size_g, 76	countPNAN
sum_spikes, 74, 75	Schmuker2014_classifier, 110
sumDN, 76	countPN
sumIzh1, 76	Schmuker2014_classifier, 110
sumKC, 76	countRN
Junito, 10	oodini ii v

Schmuker2014_classifier, 110	parse_options.h, 224
cpu_only	debugCode
parse_options.h, 224	GENN_PREFERENCES, 63
create_input_values	defaultDevice
classIzh, 65	GENN_PREFERENCES, 63
createDirectory	delB
experiment.cc, 130	gen_pnkc_syns_indivID.cc, 139
createPosttoPreArray	utils.h, 268
sparseUtils.cc, 238	deviceCount
sparseUtils.h, 239	global.cc, 161
createPreIndices	global.h, 163
sparseUtils.cc, 238	deviceProp
sparseUtils.h, 239	global.cc, 161
createSparseConnectivityFromDense	global.h, 163
sparseUtils.h, 239	directoryExists
createWTAConnectivity	experiment.cc, 130
Schmuker2014_classifier, 108	divi
cudaFuncGetAttributesDriver	experiment.h, 131
utils.cc, 266	dnp
utils.h, 268	NNmodel, 92
,	dpNames
D_MAX_RANDOM_NUM	neuronModel, 81
experiment.h, 131	postSynModel, 98
d_baserates	weightUpdateModel, 119
classol, 75	dpclass, 77
d_input1	calculateDerivedParameter, 78
classIzh, 66	dpclass.h, 129
d_input2	dps
classIzh, 66	neuronModel, 81
d_inputRates	postSynModel, 98
Schmuker2014_classifier, 110	weightUpdateModel, 119
d_maxRandomNumber	dpsp
Schmuker2014_classifier, 110	NNmodel, 92
d_pattern	dsp_w
classol, 75	NNmodel, 92
DATASET_NAME	DT
experiment.h, 131	HHVclampGA_project/model/MBody1.cc, 177
DBG_SIZE	Model_Schmuker_2014_classifier.cc, 214
Izh_sparse_sim.h, 172	dt
MBody_delayedSyn_project/model/classol_sim.h,	NNmodel, 92
125	
MBody_individualID_project/model/classol_sim.h,	EKexp
126	helper.h, 165
OneComp_sim.h, 224	ENDL
DENSE	CodeHelper.h, 128
modelSpec.h, 217	ENaexp
data_type	helper.h, 165
Schmuker2014_classifier, 108	EXITSYN
data_type_double	modelSpec.h, 217
Schmuker2014_classifier, 108	EXPDECAY
data_type_float	postSynapseModels.cc, 228
Schmuker2014_classifier, 108	postSynapseModels.h, 229
data_type_int	Elexp
Schmuker2014_classifier, 108	helper.h, 165
data_type_uint	endl
Schmuker2014_classifier, 108	CodeHelper, 76
datasetName	ensureFtype
Schmuker2014_classifier, 110	stringUtils.cc, 241
dbgMode	stringUtils.h, 243

err	TOTAL_RECORDINGS, 132
errTupel, 78	timer, 132
errTupel, 78	UINT, 132
err, 78	VR_DATA_FILENAME, 132
id, 78	WEIGHT_DELTA_PN_AN, 132
evntThreshold	WEIGHT_RN_PN, 132
weightUpdateModel, 119	WEIGHT_WTA_AN_AN, 132
exlzh_ini	WEIGHT_WTA_PN_PN, 132
OneComp.cc, 222	extended_name_substitutions
PoissonIzh.cc, 226	stringUtils.cc, 241
exlzh_p	stringUtils.h, 243
OneComp.cc, 222	extended_value_substitutions
PoissonIzh.cc, 226	stringUtils.cc, 241
exclzh_p	stringUtils.h, 243
Izh_sparse.cc, 170	extern_variable_def
expDecay	generateRunner.cc, 157
expDecayDp, 79	generateRunner.h, 158
expDecayDp, 78	extra_global_params_in_sim_code/model.cc
calculateDerivedParameter, 79	modelDefinition, 194
expDecay, 79	neuron_ini, 195
experiment.cc, 129	extra_global_params_in_sim_code/test.cc
applyInputToClassifier, 130	INSTANTIATE_TEST_CASE_P, 248
classifier, 130	SimulationTestExtraGlobalParams, 248
createDirectory, 130	simulatorBackends, 248
directoryExists, 130	TEST_P, 248
getAverage, 130	$extra_global_params_in_sim_code_event_spare_{\leftarrow}$
getStdDev, 130	inv/model.cc
getUniqueRunId, 130	modelDefinition, 195
main, 130	neuron_ini, 195
outputRunParameters, 130	synapses_ini, 195
Parameter, 130	$extra_global_params_in_sim_code_event_spare_{\leftarrow}$
printTextFile, 130	inv/test.cc
RAND, 130	INSTANTIATE_TEST_CASE_P, 249
S_ISDIR, 130	SimulationTestExtraGlobalParams, 249
setDefaultParamValues, 130	simulatorBackends, 249
vectorContains, 130	TEST_P, 249
experiment.h, 130	extra_global_post_param_in_sim_code/model.cc
CACHE_DIR, 131	modelDefinition, 196
CONNECTIVITY_AN_AN, 131	neuron_ini, 196
CONNECTIVITY_PN_AN, 131	synapses_ini, 196
CONNECTIVITY_PN_PN, 131	extra_global_post_param_in_sim_code/test.cc
CONNECTIVITY_RN_PN, 131	INSTANTIATE_TEST_CASE_P, 249
D_MAX_RANDOM_NUM, 131	SimTest, 249
DATASET_NAME, 131	simulatorBackends, 249
divi, 131	TEST_P, 249
FLAG_RUN_ON_CPU, 131	extra_global_pre_param_in_sim_code/model.cc
GLOBAL_WEIGHT_SCALING, 131	modelDefinition, 196
MAX_FIRING_RATE_HZ, 131	neuron_ini, 196
MAX_WEIGHT_PN_AN, 131	synapses_ini, 196
MIN_FIRING_RATE_HZ, 131	extra_global_pre_param_in_sim_code/test.cc
MIN_WEIGHT_PN_AN, 131	INSTANTIATE_TEST_CASE_P, 250
N_FOLDING, 131	SimTest, 250
OUTPUT_DIR, 131	simulatorBackends, 250
PLASTICITY_INTERVAL_MS, 132	TEST_P, 250
RECORDING_TIME_MS, 132	extra_neurons.h, 132
RECORDINGS_DIR, 132	clear, 133
REPEAT_LEARNING_SET, 132	push_back, 133
SPIKING_ACTIVITY_THRESHOLD_HZ, 132	simCode, 133

clear, 134 postSynDeay, 134 postSynDeay, 134 postSynDeay, 134 postSynDeay, 134 postSynDeay, 134 push_back, 134 extra_weightupdatesh, 134 extra_weightupdatesh, 134 extra_lobalNeuronKernelParameters neuronModel, 81 extraGlobalNeuronKernelParameters weightUpdateModel, 120 extract bool value command_line_processing.h, 128 extrac_tstring_value deloxport.processing.h, 128 extrac_tstring_value deloxport.processing.h, 128 extrac_tstring_value deloxport.processing.h, 128 extrac_tstring_valu		GAUSS_CC
postSyntoCurrent, 134 push_back, 134 extra_weighrupdates.h, 134 extra_weighrupdates.h, 134 extraGlobalNeuronKernelParameterTypes neuronModel, 81 extraGlobalSynapseKernelParameters neuronModel, 81 extraGlobalSynapseKernelParameters weightUpdateModel, 120 extraGlobalSynapseKernelParameters weightUpdateModel, 120 extract bool value command_line_processing.h, 128 extract_option deblufcate, 23 extract_op		-
push back, 134 extra deightupdates h, 120 extra deightupdates h, 120 extra deightupdates h, 120 extra deightupdates h, 121 extra deightupdates h, 120 extra deightupdates h, 120 extra deightupdates h, 120 extra deightupdates h, 121 extra deightupdates h, 121 extra deightupdates h, 121 extra deightupdates h, 122 extra deightupdates h, 128 extra deightupdates h, 129 extra deightupdates h, 120 extra deightupdates h, 121 extra		GAUSS_H
extra weightupdates.h. 134 extraGlobalNeuronKernelParameterTypes neuronModel, 81 extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalSynapseKernelParameterTypes weightupdateModel, 120 extraGlobalSynapseKernelParameters weightupdateModel, 120 extraGlobalSynapseKernelParameters weightupdateModel, 119 extraClooly value command_line_processing.h, 128 extract_toolion command_line_processing.h, 128 extract_toolion command_line_processing.h, 128 extract_toolion command_line_processing.h, 128 extract_string_value delSpec.h, 217 extract_string_value delSpec.h, 217 extract_string_value	•	
extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalNeuronKernelParameters weightUpdateModel, 119 extract_bool_value command_line_processing.h, 128 extract_obiol_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_option default_belobel, 163 extract_option command_line_processing.h, 128 extract_option default_belobel, 163 extract_option command_line_processing.h, 128 extract_option default_belobel, 63 extract_option, 63 extract	• —	_
neuromModel, 81 extraGlobalNeuronKernelParameters neuromModel, 81 extraGlobalSynapseKernelParameterTypes weightUpdateModel, 120 extraGlobalSynapseKernelParameters weightUpdateModel, 119 extraClobul yalue command line processing.h, 128 extract_polion command_line_processing.h, 128 extract_polion command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 ext		
extraGlobalNeuronKernelParameters neuronModel, 81 extraGlobalSynapseKernelParameterTypes weightUpdateModel, 120 extracLbool_value command_line_processing.h, 128 extracLbool_value command_line_processing.h, 128 extracLstring_value command_line_processing.h, 128 extracLoption develoption.h, 63 extracloption develoption.h, 63 extracloption develoption.h, 63 extracloption develoption.h, 63 extracloption.h, 63 extracloptio		GENN_FLAGS, 62
neuronModel, 81 extraGlobalSynapseKernelParameterTypes weightUpdateModel, 120 extraGlobalSynapseKernelParameters weightUpdateModel, 119 extract_bool_value command_line_processing.h, 128 extract_bool_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_obol_value command_line_processing.h, 128 extract_obol_value command_line_processing.h, 128 extract_obol_value command_line_processing.h, 128 extract_bool_value looksize, 63 autoChooseDevice, 63 autoChooseDevice, 63 autoChooseDevice, 63 autoChooseDevice, 63 autoChooseDevice, 63 autoChooseDevice, 63 extract_bool_value looksize, 63 optimiseBlockSize, 63 optimiseBlockSize, 63 optimiseBlockSize, 63 everingBlocksize, 63 optimiseBlockSize, 63 everingBlocksize, 63 e		
extraGlobalSynapseKernelParameterTypes weightUpdateModel, 120 extraGlobalSynapseKernelParameters weightUpdateModel, 119 extract_bool_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_option develoption.h, 28 extract_option develoption.h, 23 expansellockSize, 63 optimiseBlockSize, 63 optimiseB		calcSynapseDynamics, 62
weightUpdateModel, 120 extraGlobalSynapseKernelParameters weightUpdateModel, 119 extract_bool_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findMeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 free_device_mem classizh, 65 classol, 70 ftype NNmodel, 92 gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syn		calcSynapses, 62
extraclobalSynapseKernelParameters weightUpdateModel, 119 extract_bool_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FLAG RUN_ON_CPU experiment.h, 131 final NNmodel, 90 findlMaxMinSampleDistances Schmuker2014_classifier, 108 findlNeuronGrp NNmodel, 90 findSynapseCrp		learnSynapsesPost, 62
extract_bool_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 learningBlockSize, 63 debugCode, 64 debugCod	- ·	GENN_LONG_DOUBLE
extract bool_value command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findSynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 FloatType classith, 65 classol, 70 flype NNmodel, 92 parse_options.h, 224 Floe_device_mem classith, 65 classol, 70 flype gen_syns_sparse_izhModel.cc, 142 gei gen_syns_sparse_izhModel.cc, 142 gerray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie	- •	modelSpec.h, 217
command_line_processing.h, 128 extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classizh, 55 classol, 70 ftype NNmodel, 92 en_syns_sparse_izhModel.cc, 142 gele gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garay_ie gen_syns_sparse_izhModel.cc, 142 garay_ie gen_syns_sparse_izhModel.cc, 142 garay_ie gen_syns_sparse_izhModel.cc, 142	- ·	GENN_PREFERENCES, 62
extract_option command_line_processing.h, 128 extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FFALSE_global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classizh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 ge_e gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142		asGoodAsZero, 63
command_line_processing.h, 128 extract_string_value	command_line_processing.h, 128	autoChooseDevice, 63
extract_string_value command_line_processing.h, 128 FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNouronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapseBoptins.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classizh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ei gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142	extract_option	autoRefractory, 63
command_line_processing.h, 128 learningBiockSize, 63 neuronBlockSize, 63 neuronBlockSize, 63 optimizeCode, 63 showPtxInfo, 63 synapseBlockSize, 63 showPtxInfo, 63 synapseBlockSize, 63 showPtxInfo, 63 synapseBlockSize, 63 showPtxInfo, 63 synapseBlockSize, 63 showPtxInfo, 63 synapseBlockSize, 64 userCxxFlagsBNI, 64 userCxxFlagsBnI userCxxF	command_line_processing.h, 128	debugCode, 63
FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 nNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findsynapseGrp NNmodel, 90 findsynapseGrp Nnmodel, 90 findsynapseGrp Nnmodel, 90 findsynapse, 20 finalize Nomodel, 90 findSynapseGrp Nnmodel, 90 findSynapseGrp Nnmodel, 90 findSynapse, 20 findSynapse,	extract_string_value	defaultDevice, 63
FALSE global.h, 163 FLAG, RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ei gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142	command_line_processing.h, 128	learningBlockSize, 63
FALSE global.h, 163 FLAG_RUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findSynapseGrp NNmodel, 90 fixsynapse garse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ei gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142		-
Section Sect	FALSE	
FLAG_HUN_ON_CPU experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapseGrp Nnmodel, 90 findSynapseGrp nodelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_gen_syns_sparse_izhModel.cc, 142 gei gen_syns_sparse_izhModel.cc, 142 geri gen_syns_sparse_izhModel.cc, 142		•
experiment.h, 131 final NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNouronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii		•
Infall NNmodel, 92 finalize NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapse Grp NNmodel, 90 findSynapse parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_er gen_syns_sparse_izhModel.cc, 142	experiment.h, 131	
NNmodel, 92 finalize NNmodel, 90 finalizes Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findSynapseGrp NNmodel, 90 findsynapseGrp Nnmodel, 90 findsynapseGrp Nnmodel, 90 findsynapse parse_options.h, 224 floatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 flype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 GA.cc, 134 compareErrTupel, 135 userCxxFlagsWIN, 64 userNvccFlagsWIN, 64 userNvccFlagsWIN elper.h, 165 GLOBAL_CC global-model/MBod.cc, 177 GLOBAL_CC global-modelspec.h, 218 gNaexp helper.h, 165 GLOBAL_CC global-modelspec.h, 218 gNae	final	•
NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNouronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType classlzh, 65 classol, 70 fivpe NNmodel, 92 parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 FloatType classol, 70 ftype NNmodel, 92 parse_options.h, 224 FloatType classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 FloatType classol, 70 ftype clobal, Welight, 135 ftype clobal, CC clobal, Melght, 131 ftype clobal, CC clobal, Melght, 131 ftype clobal, CC clobal, Melght, 131 ftype clobal, CC clobal, Melght, 165 classol, 70 ftype clobal, CC clobal, CC clobal, CC clobal, CC clobal, CC clobal, CC clobal	NNmodel, 92	
NNmodel, 90 findMaxMinSampleDistances Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 Nnmodel, 90 fixynapse Grp Nnmodel, 90 fixynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 flype NNmodel, 92 parse_options.h, 224 FloatType modelSpec.b, 217 five classol, 70 flype Nnmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	finalize	
Schmuker2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 floatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 flype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii compareErrTupel, 135 gKexp helper.h, 165 gLHIKC HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global_CC, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG global.cc, 161 GLOBAL_MEGHT_SCALING experiment.h, 131 GLOBAL global.cc, 161 GLOBAL_MEGHT_SCALING experiment.h, 131 GLOBAL_G global.cc, 161 GLOBAL_G modelSpec.h, 218 global.cc, 161 GLOBAL_G global.cc, 161 GLOBAL_G global.cc, 161 GLOBAL_G GLOBAL MBody_individuallD_project/model/sizes.h, 236 GPU modelSpec.h, 217 gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	NNmodel, 90	
Schmurer2014_classifier, 108 findNeuronGrp NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 Roll DEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_iii gen_syns_sparse_izhModel.cc, 142 gersyns_sparse_izhModel.cc, 142	findMaxMinSampleDistances	_
Intokerrorrap NNmodel, 90 findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 Gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_array_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii	Schmuker2014_classifier, 108	
INNMOdel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classizh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 Gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	findNeuronGrp	•
NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 RNmodel, 92 parse_options.h, 224 GPU modelSpec.h, 217 gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142	NNmodel 90	ULTING
fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142	Miniodei, 30	
FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNMmodel, 92 parse_options.h, 224 Gen_syns_sparse_izhModel.cc, 142		HHVclampGA_project/model/MBody1.cc, 177
FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC
Float type modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 GPU modelSpec.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161
modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype	findSynapseGrp NNmodel, 90 fixsynapse	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING
free_device_mem classIzh, 65 classol, 70 ftype gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 gee gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131
helper.h, 165 free_device_mem	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG
free_device_mem	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218
classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp
classol, 70 ftype NNmodel, 92 parse_options.h, 224 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165
ftype NNmodel, 92 parse_options.h, 224 gee gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO
NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169
parse_options.h, 224 gee gen_syns_sparse_izhModel.cc, 142 gei gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL
modelSpec.h, 217 g_ee	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL
gen_syns_sparse_izhModel.cc, 142 g_ei	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236
gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU
g_ei garray_ee gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142 garray_ii gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217
gen_syns_sparse_izhModel.cc, 142 g_ie	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray
g_ie garray_ei gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 GA.cc, 134 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142
gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 g_ray_ie gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee
g_ii garray_ie gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classIzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142
gen_syns_sparse_izhModel.cc, 142 gen_syns_sparse_izhModel.cc, 142 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ie	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei
GA.cc, 134 garray_ii compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142
compareErrTupel, 135 gen_syns_sparse_izhModel.cc, 142	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ie
	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142
	findSynapseGrp NNmodel, 90 fixsynapse parse_options.h, 224 FloatType modelSpec.h, 217 for parse_options.h, 224 free_device_mem classlzh, 65 classol, 70 ftype NNmodel, 92 parse_options.h, 224 g_ee gen_syns_sparse_izhModel.cc, 142 g_ei gen_syns_sparse_izhModel.cc, 142 g_ie gen_syns_sparse_izhModel.cc, 142 g_ii gen_syns_sparse_izhModel.cc, 142 GA.cc, 134	HHVclampGA_project/model/MBody1.cc, 177 GLOBAL_CC global.cc, 161 GLOBAL_WEIGHT_SCALING experiment.h, 131 GLOBALG modelSpec.h, 218 gNaexp helper.h, 165 GOLDEN_RATIO isaac.cc, 169 gPNKC_GLOBAL MBody_individualID_project/model/sizes.h, 236 GPU modelSpec.h, 217 garray gen_syns_sparse_izhModel.cc, 142 garray_ee gen_syns_sparse_izhModel.cc, 142 garray_ei gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie gen_syns_sparse_izhModel.cc, 142 garray_ie

GAUSS_CC, 135	garray_ii, 142
gauss.h, 135	gsyn, 142
GAUSS_H, 136	ind, 142
GeNNHelperKrnls.cu, 159	ind_ee, 142
generate_random_gpuInput_xorwow, 159	ind_ei, 142
generate random gpulnput xorwow< double >,	ind_ie, 142
159	ind_ii, 142
generate_random_gpuInput_xorwow< float >, 160	indlnG_ee, 142
setup_kernel, 160	indlnG_ei, 142
xorwow setup, 160	indlnG_ie, 142
GeNNHelperKrnls.h, 160	indlnG ii, 142
BlkSz, 160	main, 142
generate_random_gpuInput_xorwow, 160	printVector, 142
setup_kernel, 160	R, 142
xorwow_setup, 160	Rind, 142
GeNNReady	genMakefile
modelSpec.h, 218	generateRunner.cc, 157
·	
src/modelSpec.cc, 215	generateRunner.h, 158
gen_alltoall_syns	genNeuronFunction
classIzh, 65	generateCPU.cc, 151
gen_input_structured.cc, 136	generateCPU.h, 153
main, 136	genNeuronKernel
R, 136	generateKernels.cc, 154
gen_kcdn_syns.cc, 136	generateKernels.h, 155
main, 137	genRunner
R, 137	generateRunner.cc, 157
RG, 137	generateRunner.h, 158
gen_kcdn_syns_fixto10K.cc, 137	genRunnerGPU
main, 138	generateRunner.cc, 157
R, 138	generateRunner.h, 159
R2, 138	genSynapseFunction
RG, 138	generateCPU.cc, 152
gen_pnkc_syns.cc, 138	generateCPU.h, 153
main, 138	genSynapseKernel
R, 138	generateKernels.cc, 154
RG, 138	generateKernels.h, 156
gen_pnkc_syns_indivID.cc, 138	generate baserates
	S =
B, 139	classol, 70
delB, 139	generate_inputrates_dataset
main, 139	Schmuker2014_classifier, 108
R, 140	generate_model_runner
RG, 140	generateALL.cc, 149
setB, 139	generateALL.h, 150
gen_pnlhi_syns.cc, 140	generate_or_load_inputrates_dataset
main, 140	Schmuker2014_classifier, 108
gen_syns_sparse.cc, 140	generate_process_presynaptic_events_code
main, 141	generateKernels.cc, 154
R, 141	generateKernels.h, 155
RG, 141	generate_process_presynaptic_events_code_CPU
gen_syns_sparse_izhModel.cc, 141	generateCPU.cc, 151
g_ee, 142	generateCPU.h, 152
g_ei, 142	generate_random_gpuInput_xorwow
g_ie, 142	GeNNHelperKrnls.cu, 159
g_ii, 142	GeNNHelperKrnls.h, 160
garray, 142	generate_random_gpuInput_xorwow< double >
garray_ee, 142	GeNNHelperKrnls.cu, 159
garray_ei, 142	generate_random_gpuInput_xorwow< float >
garray_ie, 142	GeNNHelperKrnls.cu, 160
3	

generate_run.cc, 143–147	Schmuker2014_classifier, 108
generateALL.cc, 148	getRecordingFilename
chooseDevice, 149	Schmuker2014_classifier, 109
generate_model_runner, 149	getSampleDistance
hlp, 150	Schmuker2014_classifier, 109
main, 149	getSparseVar
generateALL.h, 150	sparseUtils.h, 240
chooseDevice, 150	getSpikeNumbersFromGPU
generate_model_runner, 150	classIzh, 65
generateCPU.cc, 151	classol, 71
genNeuronFunction, 151	neuronpop, 82
genSynapseFunction, 152	Schmuker2014_classifier, 109
generate_process_presynaptic_events_code_C←	getSpikesFromGPU
	classIzh, 65
PU, 151	classol, 71
generateCPU.h, 152	neuronpop, 82
genNeuronFunction, 153	• •
genSynapseFunction, 153	Schmuker2014_classifier, 109
generate_process_presynaptic_events_code_C	getStdDev
PU, 152	experiment.cc, 130
generateKernels.cc, 153	GetTheW
genNeuronKernel, 154	SimulationSynapsePolicyDense, 113
genSynapseKernel, 154	getUniqueRunId
generate_process_presynaptic_events_code, 154	experiment.cc, 130
isGrpVarNeeded, 155	getG
generateKernels.h, 155	sparseUtils.h, 239
genNeuronKernel, 155	glexp
genSynapseKernel, 156	helper.h, 165
generate_process_presynaptic_events_code, 155	global.cc, 160
generateRunner.cc, 156	deviceCount, 161
extern_variable_def, 157	deviceProp, 161
	GLOBAL_CC, 161
genMakefile, 157	hostCount, 161
genRunner, 157	learnBlkSz, 161
genRunnerGPU, 157	neuronBlkSz, 161
variable_def, 157	synDynBlkSz, 161
generateRunner.h, 158	synapseBlkSz, 161
extern_variable_def, 158	theDevice, 161
genMakefile, 158	global.h, 162
genRunner, 158	deviceCount, 163
genRunnerGPU, 159	deviceProp, 163
variable_def, 159	FALSE, 163
generateSimulatedTimeSeriesData	
Schmuker2014_classifier, 108	hostCount, 163
gennError	learnBlkSz, 163
utils.cc, 266	neuronBlkSz, 163
utils.h, 268	synDynBlkSz, 163
get_kcdnsyns	synapseBlkSz, 163
classol, 70, 71	TRUE, 163
getAverage	theDevice, 163
experiment.cc, 130	gpKCDN
•	MBody_userdef.cc, 191
getClassCluster	gpPNKC
Schmuker2014_classifier, 108	MBody_userdef.cc, 191
getClusterIndex	gsyn
Schmuker2014_classifier, 108	gen_syns_sparse_izhModel.cc, 142
getElapsedTime	
CStopWatch, 77	HHVClamp.cc, 166
getManhattanDistance	modelDefinition, 166
Schmuker2014_classifier, 108	myHH_ini, 166
getRand0to1	myHH_p, 167

HHVClampParameters.h, 167	truevar_initexpHH, 165
_FTYPE, 167	var_init_fullrange, 165
NPOP, 167	var_reinit, 165
SCALAR_MAX, 167	Vexp, 166
SCALAR_MIN, 167	write_para, 165
scalar, 167	hexp
TOTALT, 167	helper.h, 165
HHVclampGA_project/generate_run.cc	hlp
main, 143	CodeHelper.h, 128
HHVclampGA_project/model/MBody1.cc	generateALL.cc, 150
DT, 177	hostCount
gDNDN, 177	global.cc, 161
gLHIKC, 177	global.h, 163
modelDefinition, 177	hr_time.cc, 167
myDNDN_p, 177	HR_TIMER, 167
myKCDN_p, 177	hr_time.h, 168
myLHIKC_p, 177	INDIVIDUALID
myPNKC_p, 177	modelSpec.h, 218
myPNLHI_p, 178	INDIVIDUALG
myPOI_ini, 178	
myPOI_p, 178	modelSpec.h, 218
postExpDNDN, 178	INHIBSYN
postExpKCDN, 178	modelSpec.h, 217
postExpLHIKC, 178	INSTANTIATE_TEST_CASE_P
postExpPNKC, 178	extra_global_params_in_sim_code/test.cc, 248
postExpPNLHI, 179	extra_global_params_in_sim_code_event_spare↔
postSynV, 179	_inv/test.cc, 249
stdTM ini, 179	extra_global_post_param_in_sim_code/test.cc,
stdTM_p, 179	249
HR TIMER	extra_global_pre_param_in_sim_code/test.cc, 250
hr time.cc, 167	neuron_support_code_sim/test.cc, 251
helper.h, 164	neuron_support_code_threshold/test.cc, 251
Cexp, 165	post_vars_in_post_learn/test.cc, 252
copy_var, 165	post_vars_in_post_learn_sparse/test.cc, 253
EKexp, 165	post_vars_in_sim_code/test.cc, 254
ENaexp, 165	post_vars_in_sim_code_sparse/test.cc, 254
Elexp, 165	post_vars_in_synapse_dynamics/test.cc, 255
gKexp, 165	post_vars_in_synapse_dynamics_sparse/test.cc,
gNaexp, 165	256
glexp, 165	pre_vars_in_post_learn/test.cc, 256
hexp, 165	pre_vars_in_post_learn_sparse/test.cc, 257
•	pre_vars_in_sim_code/test.cc, 258
initexpHH, 165	pre_vars_in_sim_code_event/test.cc, 258
initl, 165	pre_vars_in_sim_code_event_sparse/test.cc, 259
limit, 165	pre_vars_in_sim_code_event_sparse_inv/test.cc,
mexp, 165	260
nexp, 165	pre_vars_in_sim_code_sparse/test.cc, 261
operator<<, 165	pre_vars_in_synapse_dynamics/test.cc, 261
runexpHH, 165	pre_vars_in_synapse_dynamics_sparse/test.cc,
sigENa, 166	262
sigEK, 165	synapse_support_code_event_sim_code/test.cc,
sigEl, 165	263
sigGNa, 166	synapse_support_code_event_threshold/test.cc,
sigGK, 166	263
sigGl, 166	synapse_support_code_post_learn/test.cc, 264
sigC, 165	synapse_support_code_sim_code/test.cc, 265
single_var_init_fullrange, 165	synapse_support_code_synapse_dynamics/test. ←
single_var_reinit, 165	cc, 266
truevar_init, 165	ISAAC_INT

isaac.cc, 169	initLearnGrps
IZHIKEVICH_PS	NNmodel, 90
postSynapseModels.cc, 228	initexpHH
postSynapseModels.h, 229	helper.h, 165
IZHIKEVICH_V	initl
neuronModels.cc, 219	helper.h, 165
neuronModels.h, 221	initialiseInputData
IZHIKEVICH	Schmuker2014_classifier, 109
neuronModels.cc, 219	initialiseWeights_DENSE_PN_AN
neuronModels.h, 221	Schmuker2014_classifier, 109
id	initialiseWeights_SPARSE_RN_PN
errTupel, 78	Schmuker2014_classifier, 109
if	initialiseWeights_WTA_AN_AN
parse_options.h, 224	Schmuker2014_classifier, 109
inSyn	initialiseWeights_WTA_PN_PN
NNmodel, 93	Schmuker2014_classifier, 109
ind	initializeAllVars
gen_syns_sparse_izhModel.cc, 142	classIzh, 65
QTIsaac, 101	initializeSparseArray
SparseProjection, 116	sparseUtils.cc, 238
ind_ee	sparseUtils.h, 240
gen_syns_sparse_izhModel.cc, 142	initializeSparseArrayPreInd
ind_ei	sparseUtils.cc, 238
gen syns sparse izhModel.cc, 142	sparseUtils.h, 240
ind_ie	initializeSparseArrayRev
gen_syns_sparse_izhModel.cc, 142	sparseUtils.cc, 238
ind_ii	sparseUtils.h, 240
gen_syns_sparse_izhModel.cc, 142	input1
indlnG_ee	classIzh, 66
gen_syns_sparse_izhModel.cc, 142	input2
indlnG_ei	classIzh, 66
gen_syns_sparse_izhModel.cc, 142	InputBaseRate
indInG_ie	MBody1_project/model/classol_sim.h, 124
gen_syns_sparse_izhModel.cc, 142 indlnG_ii	MBody_delayedSyn_project/model/classol_sim.h.
gen_syns_sparse_izhModel.cc, 142	MBody_individualID_project/model/classol_sim.h.
indInG	126
SparseProjection, 116	MBody_userdef_project/model/classol_sim.h, 127
individualSpikeCountPN	PoissonIzh_sim.h, 228
Schmuker2014_classifier, 110	inputFac
inhlzh_p	lzh_sparse_project/model/sizes.h, 234
lzh_sparse.cc, 170	inputRates
Init	Schmuker2014_classifier, 110
SimulationNeuronPolicyPrePostVar, 111	inputRatesSize
SimulationNeuronPolicyPreVar, 111	Schmuker2014_classifier, 110
SimulationSynapsePolicy, 112	inputSpec, 79
	·
SimulationSynapsePolicyDense, 113	baseV, 79
SimulationSynapsePolicyNone, 113	N, 79
SimulationSynapsePolicySparse, 114	st, 79
SimulationTest, 114	t, 79
SimulationTestVars, 115	V, 79
init	ip0
classIzh, 65	rulkovdp, 105
classol, 71	ip1
neuronpop, 83	rulkovdp, 105
initGeNN	ip2
modelSpec.h, 218	rulkovdp, 105
src/modelSpec.cc, 215	isGrpVarNeeded

generateKernels.cc, 155	learnBlkSz
isaac	global.cc, 161
QTIsaac, 101	global.h, 163
isaac.cc, 168	IearnSynapsesPost
ISAAC_HPP, 169	GENN_FLAGS, 62
GOLDEN RATIO, 169	learningBlockSize
ISAAC INT, 169	GENN_PREFERENCES, 63
Izh_sim_sparse.cc, 169	lim0
main, 169	pwSTDP userdef, 100
Izh_sparse.cc, 169	pwSTDP, 99
exclzh_p, 170	lim1
inhlzh p, 170	pwSTDP_userdef, 100
IzhExc_ini, 170	pwSTDP, 99
IzhInh_ini, 170	limit
meanInpExc, 170	-
•	helper.h, 165
meanInpInh, 170	load_VR_data
modelDefinition, 170	Schmuker2014_classifier, 109
neuronPSize, 170	loadArrayFromTextFile
neuronVSize, 170	Schmuker2014_classifier, 109
postExpP, 171	loadClassLabels
postSynV, 171	Schmuker2014_classifier, 109
Synlzh_ini, 171	log
Synlzh_p, 171	Schmuker2014_classifier, 110
synapsePSize, 171	IrnGroups
Izh_sparse_model.cc, 171	NNmodel, 93
_IZH_SPARSE_MODEL_CC_, 171	IrnSynGrp
R, 171	NNmodel, 93
RG, 171	
Izh_sparse_model.h, 171	MAPNEURON
lzh_sparse_project/generate_run.cc	neuronModels.cc, 219
main, 144	neuronModels.h, 221
openFileGetMax, 144	MAX_FIRING_RATE_HZ
lzh_sparse_project/model/sizes.h	experiment.h, 131
FTYPE, 234	MAX_WEIGHT_PN_AN
NExc, 234	experiment.h, 131
NInh, 234	MAXNRN
_NMaxConnP0, 234	neuronModels.h, 221
NMaxConnP1, 234	MAXPOSTSYN
_NMaxConnP2, 234	postSynapseModels.h, 229
NMaxConnP3, 234	MBody1.cc, 176, 179
inputFac, 234	MBody1_project/generate_run.cc
SCALAR_MAX, 234	main, 144
SCALAR MIN, 234	MBody1 project/model/MBody1.cc
scalar, 234	modelDefinition, 180
	myDNDN_ini, 180
Izh_sparse_sim.h, 171	myDNDN_p, 180
DBG_SIZE, 172	myKCDN_ini, 180
T_REPORT_TME, 172	myKCDN_p, 181
TOTAL_TME, 172	myLHIKC_ini, 181
timer, 172	myLHIKC_p, 181
IzhExc_ini	myPNKC_ini, 181
lzh_sparse.cc, 170	myPNKC_p, 181
IzhInh_ini	-
Izh_sparse.cc, 170	myPNLHI_ini, 181
L FADALI CVALADOF	myPNLHI_p, 181
LEARN1SYNAPSE	myPOI_ini, 182
synapseModels.cc, 245	myPOI_p, 182
synapseModels.h, 246	postExpDNDN, 182
LEARNING	postExpKCDN, 182
modelSpec.h, 217	postExpLHIKC, 182

postExpPNKC, 182	InputBaseRate, 125
postExpPNLHI, 182	MYRAND, 125
postSynV, 182	PAT_TIME, 125
stdTM_ini, 183	PATFTIME, 125
stdTM_p, 183	PATTERNNO, 125
MBody1_project/model/classol_sim.cc	patFireTime, 125
main, 122	patSetTime, 125
MBody1 project/model/classol sim.h	SYN_OUT_TME, 125
InputBaseRate, 124	T_REPORT_TME, 125
MYRAND, 124	TOTAL_TME, 125
PAT TIME, 124	timer, 125
PATFTIME, 124	MBody_delayedSyn_project/model/map_classol.cc
PATTERNNO, 124	_MAP_CLASSOL_CC_, 173
patFireTime, 124	MBody_delayedSyn_project/model/sizes.h
patSetTime, 124	_FTYPE, 235
SYN_OUT_TME, 124	_NAL, 235
T_REPORT_TME, 124	_NLB, 235
TOTAL_TME, 124	_NLHI, 235
timer, 124	_NMB, 235
MBody1_project/model/map_classol.cc	SCALAR_MAX, 235
_MAP_CLASSOL_CC_, 172	SCALAR_MIN, 235
MBody1_project/model/sizes.h	scalar, 235
_FTYPE, 234	MBody_individualID.cc, 187
_NAL, 234	modelDefinition, 187
_NLB, 234	myDNDN_ini, 188
_NLHI, 234	myDNDN_p, 188
_NMB, 234	myKCDN_ini, 188
SCALAR_MAX, 234	myKCDN_p, 188
SCALAR_MIN, 234	myLHIKC_ini, 188
scalar, 234	myLHIKC_p, 188
MBody_delayedSyn.cc, 183	myPNKC_ini, 188
modelDefinition, 184	myPNKC_p, 189
myDNDN_ini, 184	myPNLHI ini, 189
myDNDN_p, 184	myPNLHI_p, 189
myKCDN_ini, 184	myPOI_ini, 189
myKCDN_p, 184	myPOI_p, 189
myLHIKC_ini, 184	postExpDNDN, 189
myLHIKC_p, 185	postExpKCDN, 189
myPNKC ini, 185	postExpLHIKC, 189
myPNKC_p, 185	postExpPNKC, 189
myPNLHI ini, 185	postExpPNLHI, 190
myPNLHI_p, 185	postSynV, 190
myPOI_ini, 185	stdTM_ini, 190
myPOI_p, 185	stdTM_nii, 190
postExpDNDN, 185	MBody_individualID_project/generate_run.cc
postExpKCDN, 185	main, 146
·	
postExpLHIKC, 186	MBody_individualID_project/model/classol_sim.cc
postExpPNKC, 186	main, 122
postExpPNLHI, 186	MBody_individualID_project/model/classol_sim.h
postSynV, 186	DBG_SIZE, 126
stdTM_ini, 186	InputBaseRate, 126
stdTM_p, 186	MYRAND, 126
MBody_delayedSyn_project/generate_run.cc	PAT_TIME, 126
main, 145	PATFTIME, 126
MBody_delayedSyn_project/model/classol_sim.cc	PATTERNNO, 126
main, 122	patFireTime, 126
MBody_delayedSyn_project/model/classol_sim.h	patSetTime, 126
DBG_SIZE, 125	SYN_OUT_TME, 126

T_REPORT_TME, 126 TOTAL_TME, 126 timer, 126 MBody_individualID_project/model/map_classol.cc	_MAP_CLASSOL_CC_, 174 MBody_userdef_project/model/sizes.h _FTYPE, 236 _NAL, 236
_MAP_CLASSOL_CC_, 173 MBody_individualID_project/model/sizes.h _FTYPE, 235 _NAL, 235 _NLB, 235	_NLB, 236 _NLHI, 236 _NMB, 236 SCALAR_MAX, 236 SCALAR_MIN, 236
_NLHI, 235 _NMB, 235 gPNKC_GLOBAL, 236	scalar, 236 MIN_FIRING_RATE_HZ experiment.h, 131
SCALAR_MAX, 236 SCALAR_MIN, 236	MIN_WEIGHT_PN_AN experiment.h, 131
scalar, 236 MBody_userdef.cc, 190 gpKCDN, 191	MODELSPEC_CC src/modelSpec.cc, 215 MYRAND
gpPNKC, 191 modelDefinition, 191 myDNDN_ini, 191	MBody1_project/model/classol_sim.h, 124 MBody_delayedSyn_project/model/classol_sim.h, 125
myDNDN_p, 192 myKCDN_ini, 192 myKCDN_p, 192	MBody_individualID_project/model/classol_sim.h, 126 MBody_userdef_project/model/classol_sim.h, 127
myLHIKC_ini, 192 myLHIKC_p, 192	PoissonIzh_sim.h, 227 main
myPNKC_ini, 192 myPNKC_p, 192 myPNLHI_ini, 193	experiment.cc, 130 gen_input_structured.cc, 136 gen_kcdn_syns.cc, 137
myPNLHI_p, 193 myPOI_ini, 193	gen_kcdn_syns_fixto10K.cc, 138 gen_pnkc_syns.cc, 138
myPOI_p, 193 postExpDNDN, 193	gen_pnkc_syns_indivID.cc, 139 gen_pnlhi_syns.cc, 140
postExpKCDN, 193 postExpLHIKC, 193 postExpPNKC, 193	gen_syns_sparse.cc, 141 gen_syns_sparse_izhModel.cc, 142 generateALL.cc, 149
postExpPNLHI, 193 postSynV_EXPDECAY_EVAR, 194	HHVclampGA_project/generate_run.cc, 143 lzh_sim_sparse.cc, 169
postSynV, 194 stdTM_ini, 194	Izh_sparse_project/generate_run.cc, 144 MBody1_project/generate_run.cc, 144
stdTM_p, 194 TIMING, 191 MBody_userdef_project/generate_run.cc	MBody1_project/model/classol_sim.cc, 122 MBody_delayedSyn_project/generate_run.cc, 145 MBody_delayedSyn_project/model/classol_sim.cc,
main, 147 MBody_userdef_project/model/classol_sim.cc	122 MBody_individualID_project/generate_run.cc, 146
main, 123 MBody_userdef_project/model/classol_sim.h	MBody_individualID_project/model/classol_sim.cc,
InputBaseRate, 127 MYRAND, 127 PAT_TIME, 127	MBody_userdef_project/generate_run.cc, 147 MBody_userdef_project/model/classol_sim.cc, 123 OneComp_project/generate_run.cc, 147
PATFTIME, 127 PATTERNNO, 127 patFireTime, 127	OneComp_sim.cc, 223 Poissonlzh_project/generate_run.cc, 148 Poissonlzh_sim.cc, 227
patSetTime, 127 SYN_OUT_TME, 127	SynDelaySim.cc, 247 VClampGA.cc, 269
T_REPORT_TME, 127 TOTAL_TME, 127 timer, 127	map_classol.cc, 172, 173 map_classol.h, 174, 175 maxConn
MBody_userdef_project/model/map_classol.cc	NNmodel, 93

meanInpExc	pre_vars_in_sim_code_event_sparse/model.cc,
Izh_sparse.cc, 170	206
meanInpInh	pre_vars_in_sim_code_event_sparse_inv/model.
lzh_sparse.cc, 170	cc, 207
mexp	pre_vars_in_sim_code_sparse/model.cc, 208
helper.h, 165	pre_vars_in_synapse_dynamics/model.cc, 208
model	pre_vars_in_synapse_dynamics_sparse/model.cc,
classIzh, 66	209
classol, 76	SynDelay.cc, 246
neuronpop, 83	$synapse_support_code_event_sim_code/model. {\leftarrow}$
Schmuker2014_classifier, 110	cc, 210
model.cc, 194–213	$synapse_support_code_event_threshold/model. \leftarrow$
Model_Schmuker_2014_classifier.cc, 214	cc, 211
CLUST_SIZE_AN, 214	synapse_support_code_post_learn/model.cc, 212
CLUST SIZE PN, 214	synapse_support_code_sim_code/model.cc, 213
CLUST_SIZE_RN, 214	synapse_support_code_synapse_dynamics/model.
DT, 214	cc, 213
modelDefinition, 215	modelSpec.cc, 215
NETWORK_SCALE, 214	modelSpec.h, 215
NUM_CLASSES, 214	_MODELSPEC_H_, 217
NUM FEATURES, 214	ALLTOALL, 217
NUM_VR, 214	AUTODEVICE, 217
SYNAPSE_TAU_ANAN, 214	CPU, 217
SYNAPSE TAU PNAN, 214	DENSE, 217
SYNAPSE_TAU_PNPN, 215	EXITSYN, 217
SYNAPSE_TAU_RNPN, 215	FloatType, 217
modelDefinition	GENN_LONG_DOUBLE, 217
extra_global_params_in_sim_code/model.cc, 194	GLOBALG, 218
extra_global_params_in_sim_code_event_spare	GPU, 217
_inv/model.cc, 195	GeNNReady, 218
extra_global_post_param_in_sim_code/model.cc,	INDIVIDUALID, 218
196	INDIVIDUALG, 218
	INHIBSYN, 217
extra_global_pre_param_in_sim_code/model.cc, 196	initGeNN, 218
HHVClamp.cc, 166	LEARNING, 217
•	NO_DELAY, 217
HHVclampGA_project/model/MBody1.cc, 177	NOLEARNING, 217
Izh_sparse.cc, 170	SPARSE, 217
MBody1_project/model/MBody1.cc, 180	SynapseConnType, 217
MBody_delayedSyn.cc, 184	SynapseGType, 217
MBody_individualID.cc, 187	myDNDN_ini
MBody_userdef.cc, 191	MBody1_project/model/MBody1.cc, 180
Model_Schmuker_2014_classifier.cc, 215	MBody_delayedSyn.cc, 184
neuron_support_code_sim/model.cc, 197	MBody_individualID.cc, 188
neuron_support_code_threshold/model.cc, 198	MBody_userdef.cc, 191
OneComp.cc, 222	myDNDN_p
PoissonIzh.cc, 226	HHVclampGA_project/model/MBody1.cc, 177
post_vars_in_post_learn/model.cc, 198	MBody1_project/model/MBody1.cc, 180
post_vars_in_post_learn_sparse/model.cc, 199	MBody_delayedSyn.cc, 184
post_vars_in_sim_code/model.cc, 200	MBody_individualID.cc, 188
post_vars_in_sim_code_sparse/model.cc, 201	MBody_userdef.cc, 192
post_vars_in_synapse_dynamics/model.cc, 202	myHH_ini
post_vars_in_synapse_dynamics_sparse/model.	HHVClamp.cc, 166
cc, 202	myHH_p
pre_vars_in_post_learn/model.cc, 203	HHVClamp.cc, 167
pre_vars_in_post_learn_sparse/model.cc, 204	myKCDN_ini
pre_vars_in_sim_code/model.cc, 205	MBody1_project/model/MBody1.cc, 180
pre_vars_in_sim_code_event/model.cc, 205	MBody_delayedSyn.cc, 184

MD-shalledistribus IID-s- 400	0.0000000000000000000000000000000000000
MBody_individualID.cc, 188	OneComp.cc, 222
MBody_userdef.cc, 192	PoissonIzh.cc, 226
myKCDN_p	N
HHVclampGA_project/model/MBody1.cc, 177	N
MBody1_project/model/MBody1.cc, 181	inputSpec, 79
MBody_delayedSyn.cc, 184	QTIsaac, 101
MBody_individualID.cc, 188	n
MBody_userdef.cc, 192	randomGauss, 103
myLHIKC_ini	randomGen, 104
MBody1_project/model/MBody1.cc, 181	stdRG, 117
MBody_delayedSyn.cc, 184	N_FOLDING
	experiment.h, 131
MBody_individualID.cc, 188	NETWORK SCALE
MBody_userdef.cc, 192	Model_Schmuker_2014_classifier.cc, 214
myLHIKC_p	NEURONMODELS CC
HHVclampGA_project/model/MBody1.cc, 177	neuronModels.cc, 219
MBody1_project/model/MBody1.cc, 181	NGRADSYNAPSE
MBody_delayedSyn.cc, 185	
MBody_individualID.cc, 188	synapseModels.cc, 245
MBody_userdef.cc, 192	synapseModels.h, 246
myPNKC_ini	nModels
MBody1_project/model/MBody1.cc, 181	neuronModels.cc, 219
MBody_delayedSyn.cc, 185	neuronModels.h, 221
MBody_individualID.cc, 188	NNmodel, 83
	\sim NNmodel, 87
MBody_userdef.cc, 192	activateDirectInput, 87
myPNKC_p	addNeuronPopulation, 87, 88
HHVclampGA_project/model/MBody1.cc, 177	addSynapsePopulation, 88, 89
MBody1_project/model/MBody1.cc, 181	checkSizes, 90
MBody_delayedSyn.cc, 185	dnp, 92
MBody_individualID.cc, 189	dpsp, 92
MBody_userdef.cc, 192	• •
myPNLHI_ini	dsp_w, 92
MBody1_project/model/MBody1.cc, 181	dt, 92
MBody_delayedSyn.cc, 185	final, 92
MBody_individualID.cc, 189	finalize, 90
•	findNeuronGrp, 90
MBody_userdef.cc, 193	findSynapseGrp, 90
myPNLHI_p	ftype, 92
HHVclampGA_project/model/MBody1.cc, 178	inSyn, 93
MBody1_project/model/MBody1.cc, 181	initLearnGrps, 90
MBody_delayedSyn.cc, 185	IrnGroups, 93
MBody_individualID.cc, 189	IrnSynGrp, 93
MBody_userdef.cc, 193	maxConn, 93
myPOI_ini	NNmodel, 87
HHVclampGA_project/model/MBody1.cc, 178	name, 93
MBody1_project/model/MBody1.cc, 182	
MBody_delayedSyn.cc, 185	needEvntThresholdReTest, 93
MBody_individualID.cc, 189	needSt, 93
• —	needSynapseDelay, 93
MBody_userdef.cc, 193	neuronDelaySlots, 93
PoissonIzh.cc, 226	neuronDeviceID, 93
myPOI_p	neuronGrpN, 93
HHVclampGA_project/model/MBody1.cc, 178	neuronHostID, 93
MBody1_project/model/MBody1.cc, 182	neuronIni, 93
MBody_delayedSyn.cc, 185	neuronKernelParameterTypes, 93
MBody_individualID.cc, 189	neuronKernelParameters, 93
MBody_userdef.cc, 193	neuronName, 94
PoissonIzh.cc, 226	neuronNeedSpkEvnt, 94
mySyn_ini	neuronNeedSt, 94
PoissonIzh.cc, 226	neuronNeedTrueSpk, 94
mySyn_p	neuronPara, 94

D 10 01	
neuronPostSyn, 94	timing, 96
neuronSpkEvntCondition, 94	NO_DELAY
neuronType, 94	modelSpec.h, 217
neuronVarNeedQueue, 94	NOLEARNING
neuronN, 93	modelSpec.h, 217
outSyn, 94	NPOP
padSumLearnN, 94	HHVClampParameters.h, 167
padSumNeuronN, 94	NSYNAPSE
padSumSynDynN, 94	synapseModels.cc, 245
padSumSynapseKrnl, 94	synapseModels.h, 246
postSynIni, 95	
postSynapsePara, 94	NUM_CLASSES
postSynapseType, 94	Model_Schmuker_2014_classifier.cc, 214
	NUM_FEATURES
RNtype, 95	Model_Schmuker_2014_classifier.cc, 214
resetKernel, 95	NUM_VR
scalarExpr, 90	Model_Schmuker_2014_classifier.cc, 214
seed, 95	name
setConstInp, 90	NNmodel, 93
setDT, 90	Parameter, 97
setGPUDevice, 91	name_substitutions
setMaxConn, 91	stringUtils.cc, 241
setName, 91	stringUtils.h, 243
setNeuronClusterIndex, 91	needEvntThresholdReTest
setPopulationSums, 91	
setPrecision, 91	NNmodel, 93
setSeed, 91	needPostSt
setSpanTypeToPre, 91	neuronModel, 81
setSynapseClusterIndex, 92	weightUpdateModel, 120
setSynapseG, 92	needPreSt
setTiming, 92	neuronModel, 81
simLearnPostKernelParameterTypes, 95	weightUpdateModel, 120
simLearnPostKernelParameters, 95	needSt
	NNmodel, 93
sumNeuronN, 95	needSynapseDelay
synDynGroups, 96	NNmodel, 93
synDynGrp, 96	neuron ini
synapseConnType, 95	extra_global_params_in_sim_code/model.cc, 195
synapseDelay, 95	
synapseDeviceID, 95	extra_global_params_in_sim_code_event_spare-
synapseDynamicsKernelParameterTypes, 95	_inv/model.cc, 195
synapseDynamicsKernelParameters, 95	extra_global_post_param_in_sim_code/model.cc,
synapseGType, 95	196
synapseGrpN, 95	extra_global_pre_param_in_sim_code/model.cc,
synapseHostID, 95	196
synapseInSynNo, 95	neuron_support_code_sim/model.cc, 197
synapselni, 95	neuron_support_code_threshold/model.cc, 198
synapseKernelParameterTypes, 96	post_vars_in_post_learn/model.cc, 198
synapseKernelParameters, 95	post_vars_in_post_learn_sparse/model.cc, 199
synapseName, 96	post_vars_in_sim_code/model.cc, 200
synapseOutSynNo, 96	post_vars_in_sim_code_sparse/model.cc, 201
synapsePara, 96	post_vars_in_synapse_dynamics/model.cc, 202
synapseSource, 96	post_vars_in_synapse_dynamics_sparse/model.
- •	cc, 202
synapseSpanType, 96	
synapseTarget, 96	pre_vars_in_post_learn/model.cc, 203
synapseType, 96	pre_vars_in_post_learn_sparse/model.cc, 204
synapseUsesPostLearning, 96	pre_vars_in_sim_code/model.cc, 205
synapseUsesSpikeEvents, 96	pre_vars_in_sim_code_event/model.cc, 205
synapseUsesSynapseDynamics, 96	pre_vars_in_sim_code_event_sparse/model.cc,
synapseUsesTrueSpikes, 96	206

was were in sine and award arrange in the sale.	nauwan Dalau Clata
pre_vars_in_sim_code_event_sparse_inv/model.	neuronDelaySlots
cc, 207	NNmodel, 93
pre_vars_in_sim_code_sparse/model.cc, 208	neuronDeviceID
pre_vars_in_synapse_dynamics/model.cc, 208	NNmodel, 93
pre_vars_in_synapse_dynamics_sparse/model.cc,	neuronGrpN
209	NNmodel, 93
synapse_support_code_event_sim_code/model.	neuronHostID
cc, 210	NNmodel, 93
synapse_support_code_event_threshold/model.↔	neuronIni
cc, 211	NNmodel, 93
synapse_support_code_post_learn/model.cc, 212	neuronKernelParameterTypes
synapse_support_code_sim_code/model.cc, 213	NNmodel, 93
synapse_support_code_synapse_dynamics/model.	neuronKernelParameters
cc, 213	NNmodel, 93
neuron_p	neuronModel, 79
post_vars_in_post_learn/model.cc, 198	\sim neuronModel, 80
post_vars_in_post_learn_sparse/model.cc, 199	dpNames, 81
pre_vars_in_post_learn/model.cc, 203	dps, 81
· – – – –	• •
pre_vars_in_post_learn_sparse/model.cc, 204	extraGlobalNeuronKernelParameterTypes, 81
pre_vars_in_synapse_dynamics/model.cc, 208	extraGlobalNeuronKernelParameters, 81
pre_vars_in_synapse_dynamics_sparse/model.cc,	needPostSt, 81
209	needPreSt, 81
synapse_support_code_post_learn/model.cc, 212	neuronModel, 80
neuron_p2	pNames, 81
post_vars_in_post_learn/model.cc, 199	resetCode, 81
post_vars_in_post_learn_sparse/model.cc, 200	simCode, 81
pre_vars_in_post_learn/model.cc, 203	supportCode, 81
pre_vars_in_post_learn_sparse/model.cc, 204	• •
	thresholdConditionCode, 81
pre_vars_in_synapse_dynamics/model.cc, 209	tmpVarNames, 81
pre_vars_in_synapse_dynamics_sparse/model.cc,	tmpVarTypes, 82
210	varNames, 82
synapse_support_code_post_learn/model.cc, 212	varTypes, 82
neuron_substitutions_in_synaptic_code	neuronModels.cc, 218
stringUtils.cc, 241	IZHIKEVICH_V, 219
stringUtils.h, 243	IZHIKEVICH, 219
neuron_support_code_sim/model.cc	MAPNEURON, 219
modelDefinition, 197	NEURONMODELS_CC, 219
neuron ini, 197	
- :	nModels, 219
synapses_ini, 197	POISSONNEURON, 219
neuron_support_code_sim/test.cc	prepareStandardModels, 219
INSTANTIATE_TEST_CASE_P, 251	SPIKESOURCE, 219
SimulationTestNeuronSupportCode, 251	TRAUBMILES_ALTERNATIVE, 219
simulatorBackends, 251	TRAUBMILES_FAST, 219
TEST_P, 251	TRAUBMILES PSTEP, 219
neuron_support_code_threshold/model.cc	TRAUBMILES_SAFE, 220
modelDefinition, 198	TRAUBMILES, 219
neuron_ini, 198	neuronModels.h, 220
synapses_ini, 198	
–	IZHIKEVICH_V, 221
neuron_support_code_threshold/test.cc	IZHIKEVICH, 221
INSTANTIATE_TEST_CASE_P, 251	MAPNEURON, 221
SimulationTestNeuronSupportCode, 251	MAXNRN, 221
simulatorBackends, 252	nModels, 221
TEST_P, 252	POISSONNEURON, 221
neuronBlkSz	prepareStandardModels, 221
global.cc, 161	SPIKESOURCE, 221
global.h, 163	TRAUBMILES ALTERNATIVE, 221
neuronBlockSize	TRAUBMILES FAST, 221
	-
GENN_PREFERENCES, 63	TRAUBMILES_PSTEP, 221

TRAUBMILES_SAFE, 221	exlzh_ini, 222
TRAUBMILES, 221	exlzh_p, 222
neuronName	modelDefinition, 222
NNmodel, 94	mySyn_p, 222
neuronNeedSpkEvnt	postExp, 222
NNmodel, 94	postSynV, 223
neuronNeedSt	OneComp_model.cc, 223
NNmodel, 94	_ONECOMP_MODEL_CC_, 223
neuronNeedTrueSpk	OneComp_model.h, 223
NNmodel, 94	OneComp_project/generate_run.cc
neuronPSize	main, 147
Izh_sparse.cc, 170	OneComp_project/model/sizes.h
neuronPara	_FTYPE, 236
NNmodel, 94	_NC1, 236
neuronPostSyn	SCALAR_MAX, 237
NNmodel, 94	SCALAR_MIN, 237
neuronSpkEvntCondition	scalar, 236
NNmodel, 94	OneComp_sim.cc, 223
neuronType	main, 223
NNmodel, 94	OneComp_sim.h, 223
neuronVSize	DBG_SIZE, 224
Izh_sparse.cc, 170	T_REPORT_TME, 224
neuronVarNeedQueue	TOTAL_TME, 224
NNmodel, 94	timer, 224
neuronN	openBrace
NNmodel, 93	CodeHelper, 76
neuronpop, 82	openFileGetMax
\sim neuronpop, 82	lzh_sparse_project/generate_run.cc, 144
getSpikeNumbersFromGPU, 82	openRecordingFile
getSpikesFromGPU, 82	Schmuker2014_classifier, 109
init, 83	operator<<
model, 83	helper.h, 165
neuronpop, 82	optimiseBlockSize
output_spikes, 83	GENN_PREFERENCES, 63
output_state, 83	optimizeCode
run, 83	GENN_PREFERENCES, 63
sum_spikes, 83	option
sumlzh1, 83	parse_options.h, 224
nexp	outSyn
helper.h, 165 nlong	NNmodel, 94
stdRG, 117	output_params
Starta, 117	classIzh, 65
OUTPUT_DIR	output_spikes
experiment.h, 131	classIzh, 65
OB	classol, 71, 72
CodeHelper.h, 128	neuronpop, 83
off0	output_state
pwSTDP_userdef, 100	classIzh, 65
pwSTDP, 99	classol, 72
off1	neuronpop, 83
pwSTDP_userdef, 100	outputDir
pwSTDP, 99	Schmuker2014_classifier, 110
off2	outputRunParameters
pwSTDP_userdef, 100	experiment.cc, 130
pwSTDP, 99	outputSpikes
offset	Schmuker2014_classifier, 109
classol, 76	overallWinnerSpikeCountAN
OneComp.cc, 222	Schmuker2014_classifier, 110

p_pattern	param_MIN_WEIGHT_PN_AN
classol, 76	Schmuker2014_classifier, 110
PAT_TIME	param_PLASTICITY_INTERVAL_MS
MBody1_project/model/classol_sim.h, 124	Schmuker2014_classifier, 110
MBody_delayedSyn_project/model/classol_sim.h,	param_SPIKING_ACTIVITY_THRESHOLD_HZ
125	Schmuker2014_classifier, 110
MBody_individualID_project/model/classol_sim.h,	param_WEIGHT_DELTA_PN_AN
126	Schmuker2014_classifier, 110
MBody_userdef_project/model/classol_sim.h, 127	param_WEIGHT_RN_PN
PATFTIME	Schmuker2014_classifier, 111
MBody1_project/model/classol_sim.h, 124	param_WEIGHT_WTA_AN_AN
MBody_delayedSyn_project/model/classol_sim.h,	Schmuker2014_classifier, 111
125	param_WEIGHT_WTA_PN_PN
MBody_individualID_project/model/classol_sim.h,	Schmuker2014_classifier, 111
126	Parameter, 97
MBody_userdef_project/model/classol_sim.h, 127	experiment.cc, 130
PATTERNNO	name, 97
MBody1 project/model/classol sim.h, 124	value, 97
MBody_delayedSyn_project/model/classol_sim.h,	parse_options.h, 224
125	cpu_only, 224
MBody_individualID_project/model/classol_sim.h,	dbgMode, 224
126	fixsynapse, 224
MBody_userdef_project/model/classol_sim.h, 127	for, 224
PLASTICITY INTERVAL MS	ftype, 224
experiment.h, 132	if, 224
pNames	option, 224
neuronModel, 81	patFireTime
postSynModel, 98	MBody1_project/model/classol_sim.h, 124
weightUpdateModel, 120	MBody_delayedSyn_project/model/classol_sim.h.
POISSONNEURON	125
neuronModels.cc, 219	MBody_individualID_project/model/classol_sim.h.
neuronModels.h, 221	126
POSTSYNAPSEMODELS_CC	MBody userdef project/model/classol sim.h, 127
postSynapseModels.cc, 228	patSetTime
padSumLearnN	MBody1_project/model/classol_sim.h, 124
NNmodel, 94	MBody_delayedSyn_project/model/classol_sim.h.
padSumNeuronN	125
NNmodel, 94	MBody_individualID_project/model/classol_sim.h.
padSumSynDynN	126
NNmodel, 94	MBody_userdef_project/model/classol_sim.h, 127
padSumSynapseKrnl	pattern
NNmodel, 94	classol, 76
param_CONNECTIVITY_AN_AN	plasticWeights
Schmuker2014_classifier, 110	Schmuker2014_classifier, 111
param_CONNECTIVITY_PN_AN	PoissonIzh-model.cc, 225
Schmuker2014 classifier, 110	_POISSONIZHMODEL_CC_, 225
param_CONNECTIVITY_PN_PN	PoissonIzh-model.h, 225
Schmuker2014_classifier, 110	PoissonIzh.cc, 225
param_CONNECTIVITY_RN_PN	exlzh_ini, 226
Schmuker2014_classifier, 110	exlzh_p, 226
param_GLOBAL_WEIGHT_SCALING	modelDefinition, 226
Schmuker2014_classifier, 110	myPOI_ini, 226
param_MAX_FIRING_RATE_HZ	myPOI_p, 226
Schmuker2014_classifier, 110	mySyn_ini, 226
param_MAX_WEIGHT_PN_AN	mySyn_p, 226
Schmuker2014_classifier, 110	postExp, 226
param_MIN_FIRING_RATE_HZ	postSynV, 227
Schmuker2014_classifier, 110	PoissonIzh_project/generate_run.cc

main, 148	TEST P, 254
PoissonIzh_project/model/sizes.h	post_vars_in_synapse_dynamics/model.cc
FTYPE, 237	modelDefinition, 202
_Nlzh, 237	neuron_ini, 202
_NPoisson, 237	synapses_ini, 202
SCALAR_MAX, 237	post_vars_in_synapse_dynamics/test.cc
SCALAR_MIN, 237	INSTANTIATE_TEST_CASE_P, 255
scalar, 237	SimulationTestPostVars, 255
PoissonIzh_sim.cc, 227	simulatorBackends, 255
main, 227	TEST_P, 255
PoissonIzh_sim.h, 227	post_vars_in_synapse_dynamics_sparse/model.cc
InputBaseRate, 228	modelDefinition, 202
MYRAND, 227	neuron_ini, 202
SYN_OUT_TME, 227	synapses_ini, 202
T_REPORT_TME, 227	post_vars_in_synapse_dynamics_sparse/test.cc
TOTAL_TME, 227	INSTANTIATE_TEST_CASE_P, 256
timer, 228	SimulationTestPostVarsSparse, 256
populateDeviceMemory	simulatorBackends, 256
Schmuker2014_classifier, 109	TEST_P, 256
post_vars_in_post_learn/model.cc	postExp
modelDefinition, 198	OneComp.cc, 222
neuron_ini, 198	PoissonIzh.cc, 226 postExpDNDN
neuron_p, 198	• •
neuron_p2, 199 synapses_ini, 199	HHVclampGA_project/model/MBody1.cc, 178 MBody1_project/model/MBody1.cc, 182
post_vars_in_post_learn/test.cc	MBody_delayedSyn.cc, 185
INSTANTIATE TEST CASE P, 252	MBody_individualID.cc, 189
SimulationTestPostVars, 252	MBody_userdef.cc, 193
simulatorBackends, 252	postExpKCDN
TEST_P, 252	HHVclampGA_project/model/MBody1.cc, 178
post_vars_in_post_learn_sparse/model.cc	MBody1_project/model/MBody1.cc, 182
modelDefinition, 199	MBody_delayedSyn.cc, 185
neuron ini, 199	MBody individualID.cc, 189
neuron_p, 199	MBody userdef.cc, 193
neuron_p2, 200	postExpLHIKC
synapses_ini, 200	HHVclampGA_project/model/MBody1.cc, 178
post_vars_in_post_learn_sparse/test.cc	MBody1_project/model/MBody1.cc, 182
INSTANTIATE_TEST_CASE_P, 253	MBody_delayedSyn.cc, 186
SimulationTestPostVarsSparse, 253	MBody individualID.cc, 189
simulatorBackends, 253	MBody userdef.cc, 193
TEST_P, 253	postExpPNKC
post_vars_in_sim_code/model.cc	HHVclampGA_project/model/MBody1.cc, 178
modelDefinition, 200	MBody1_project/model/MBody1.cc, 182
neuron_ini, 200	MBody_delayedSyn.cc, 186
synapses_ini, 200	MBody_individualID.cc, 189
post_vars_in_sim_code/test.cc	MBody_userdef.cc, 193
INSTANTIATE_TEST_CASE_P, 254	postExpPNLHI
SimulationTestPostVars, 254	HHVclampGA_project/model/MBody1.cc, 179
simulatorBackends, 254	MBody1_project/model/MBody1.cc, 182
TEST_P, 254	MBody_delayedSyn.cc, 186
post_vars_in_sim_code_sparse/model.cc	MBody_individualID.cc, 190
modelDefinition, 201	MBody_userdef.cc, 193
neuron_ini, 201	postExpP
synapses_ini, 201	Izh_sparse.cc, 171
post_vars_in_sim_code_sparse/test.cc	postSynDecay
INSTANTIATE_TEST_CASE_P, 254	extra_postsynapses.h, 134
SimulationTestPostVarsSparse, 254	postSynModel, 98
simulatorBackends, 254	postSynIni

NNmodel, 95	neuron_ini, 204
postSynModel, 97	neuron_p, 204
\sim postSynModel, 98	neuron_p2, 204
dpNames, 98	synapses_ini, <mark>204</mark>
dps, 98	pre_vars_in_post_learn_sparse/test.cc
pNames, 98	INSTANTIATE_TEST_CASE_P, 257
postSynDecay, 98	SimulationTestPreVarsSparse, 257
postSynModel, 98	simulatorBackends, 257
postSyntoCurrent, 98	TEST P, 257
supportCode, 98	pre_vars_in_sim_code/model.cc
varNames, 98	modelDefinition, 205
varTypes, 98	neuron_ini, 205
postSynModels	synapses_ini, 205
postSynapseModels.cc, 228	pre_vars_in_sim_code/test.cc
postSynapseModels.h, 229	INSTANTIATE_TEST_CASE_P, 258
postSynV_EXPDECAY_EVAR	SimulationTestPreVars, 258
MBody_userdef.cc, 194	simulatorBackends, 258
postSynapseModels.cc, 228	TEST_P, 258
EXPDECAY, 228	
	pre_vars_in_sim_code_event/model.cc
IZHIKEVICH_PS, 228	modelDefinition, 205
POSTSYNAPSEMODELS_CC, 228	neuron_ini, 205
postSynModels, 228	synapses_ini, 205
preparePostSynModels, 228	synapses_p, 206
postSynapseModels.h, 229	pre_vars_in_sim_code_event/test.cc
EXPDECAY, 229	INSTANTIATE_TEST_CASE_P, 258
IZHIKEVICH_PS, 229	SimulationTestPreVars, 258
MAXPOSTSYN, 229	simulatorBackends, 259
postSynModels, 229	TEST_P, 259
preparePostSynModels, 229	pre_vars_in_sim_code_event_sparse/model.cc
postSynapsePara	modelDefinition, 206
NNmodel, 94	neuron_ini, 206
postSynapseType	synapses_ini, 206
NNmodel, 94	synapses_p, 206
postSyntoCurrent	pre_vars_in_sim_code_event_sparse/test.cc
extra_postsynapses.h, 134	INSTANTIATE_TEST_CASE_P, 259
postSynModel, 98	SimulationTestPreVars, 259
postSynV	simulatorBackends, 259
HHVclampGA_project/model/MBody1.cc, 179	TEST P, 259
Izh_sparse.cc, 171	pre_vars_in_sim_code_event_sparse_inv/model.cc
MBody1 project/model/MBody1.cc, 182	modelDefinition, 207
MBody_delayedSyn.cc, 186	neuron_ini, 207
MBody_individualID.cc, 190	synapses ini, 207
MBody_userdef.cc, 194	synapses p, 207
OneComp.cc, 223	pre_vars_in_sim_code_event_sparse_inv/test.cc
PoissonIzh.cc, 227	INSTANTIATE TEST CASE P, 260
pre_vars_in_post_learn/model.cc	SimulationTestPreVars, 260
modelDefinition, 203	simulatorBackends, 260
	TEST_P, 260
neuron_ini, 203	
neuron_p, 203	pre_vars_in_sim_code_sparse/model.cc
neuron_p2, 203	modelDefinition, 208
synapses_ini, 203	neuron_ini, 208
pre_vars_in_post_learn/test.cc	synapses_ini, 208
INSTANTIATE_TEST_CASE_P, 256	pre_vars_in_sim_code_sparse/test.cc
SimulationTestPreVars, 256	INSTANTIATE_TEST_CASE_P, 261
simulatorBackends, 256	SimulationTestPreVars, 261
TEST_P, 256	simulatorBackends, 261
pre_vars_in_post_learn_sparse/model.cc	TEST_P, 261
modelDefinition, 204	pre_vars_in_synapse_dynamics/model.cc

modelDefinition, 208	off2, 99
neuron_ini, 208	slope0, 99
neuron_p, 208	slope1, 99
neuron_p2, 209	OTICLE
synapses_ini, 209	QTIsaac
pre_vars_in_synapse_dynamics/test.cc	~QTIsaac, 101
INSTANTIATE_TEST_CASE_P, 261	byte, 101
SimulationTestPreVars, 261	ind, 101
simulatorBackends, 261	isaac, 101
TEST_P, 261	N, 101 QTIsaac, 101
pre_vars_in_synapse_dynamics_sparse/model.cc	
modelDefinition, 209	rand, 101 randinit, 101
neuron_ini, 209	rngstep, 101
neuron_p, 209	shuffle, 102
neuron_p2, 210	srand, 102
synapses_ini, 210	QTIsaac< ALPHA, T >, 101
pre_vars_in_synapse_dynamics_sparse/test.cc	QTIsaac< ALPHA, T >::randctx, 102
INSTANTIATE_TEST_CASE_P, 262	QTIsaac::randctx
SimulationTestPreVarsSparse, 262	~randetx, 102
simulatorBackends, 262	randa, 102
TEST_P, 262	randb, 102
preInd	rando, 102
SparseProjection, 116	randont, 102
preparePostSynModels	randctx, 102
postSynapseModels.cc, 228	randmem, 102
postSynapseModels.h, 229	randrsl, 102
prepareStandardModels	
neuronModels.cc, 219	R
neuronModels.h, 221	gen_input_structured.cc, 136
prepareWeightUpdateModels	gen_kcdn_syns.cc, 137
synapseModels.cc, 245	gen_kcdn_syns_fixto10K.cc, 138
synapseModels.h, 246	gen_pnkc_syns.cc, 138
printSeparator	gen_pnkc_syns_indivID.cc, 140
Schmuker2014_classifier, 109	gen_syns_sparse.cc, 141
printTextFile	gen_syns_sparse_izhModel.cc, 142
experiment.cc, 130	Izh_sparse_model.cc, 171
printVector	VClampGA.h, 269
gen_syns_sparse_izhModel.cc, 142	R2
procreatePop	gen_kcdn_syns_fixto10K.cc, 138
GA.cc, 135	RANDOMGEN_CC
push_back	randomGen.cc, 230
extra_neurons.h, 133	RANDOMGEN_H
extra_postsynapses.h, 134	randomGen.h, 231
pwSTDP_userdef, 100	RAND
calculateDerivedParameter, 100	experiment.cc, 130
lim0, 100	VClampGA.h, 269
lim1, 100	RECORDING_TIME_MS
off0, 100	experiment.h, 132
off1, 100	RECORDINGS_DIR
off2, 100	experiment.h, 132
slope0, 100	REGEX_OPERATIONAL
slope1, 100 pwSTDP, 99	stringUtils.cc, 241
•	REPEAT_LEARNING_SET
calculateDerivedParameter, 99	experiment.h, 132
lim0, 99 lim1, 99	REPORT_TIME SynDolaySim b. 247
off0, 99	SynDelaySim.h, 247 RNtype
off1, 99	NNmodel, 95
OII 1, 00	MINITOGEI, 30

rand	Schmuker2014 classifier, 109
QTIsaac, 101	resetCode
randa	neuronModel, 81
QTIsaac::randctx, 102	resetDevice
randb	Schmuker2014_classifier, 109
QTIsaac::randctx, 102	resetIndividualSpikeCountPN
rando	Schmuker2014 classifier, 109
QTIsaac::randctx, 102	resetKernel
randent	NNmodel, 95
QTIsaac::randctx, 102	resetOverallWinner
randctx	Schmuker2014 classifier, 109
QTIsaac::randctx, 102	revInd
randinit	SparseProjection, 116
QTIsaac, 101	revIndInG
randmem	SparseProjection, 116
QTIsaac::randctx, 102	RG
randomEventOccurred	gen kcdn syns.cc, 137
Schmuker2014_classifier, 109	gen_kcdn_syns_fixto10K.cc, 138
randomGauss, 103	gen_pnkc_syns.cc, 138
~randomGauss, 103	gen_pnkc_syns_indivID.cc, 140
	gen_syns_sparse.cc, 141
n, 103	lzh_sparse_model.cc, 171
randomGauss, 103	VClampGA.h, 269
srand, 103	Rind
randomGen, 104	gen_syns_sparse_izhModel.cc, 142
∼randomGen, 104	rngstep
n, 104	QTIsaac, 101
randomGen, 104	rulkovdp, 105
srand, 104	calculateDerivedParameter, 105
randomGen.cc, 230	ip0, 105
RANDOMGEN_CC, 230	ip1, 105
randomGen.h, 230	ip2, 105
RANDOMGEN_H, 231	run
randomizeVar	classIzh, 66
classIzh, 66	classol, 74
randomizeVarSq	neuronpop, 83
classIzh, 66	Schmuker2014 classifier, 109
randrsl	SynDelay, 118
QTIsaac::randctx, 102	runCPU
read_PNIzh1syns	classol, 74
classol, 72	runGPU
read_input_patterns	classol, 74
classol, 72	runexpHH
read_input_values	helper.h, 165
classIzh, 66	Helperin, 100
read_kcdnsyns	S ISDIR
classol, 72	experiment.cc, 130
read_pnkcsyns	SAVEP
classol, 72, 73	CodeHelper.h, 128
read_pnlhisyns	SCALAR_MAX
classol, 73	HHVClampParameters.h, 167
read_sparsesyns_par	lzh_sparse_project/model/sizes.h, 234
classIzh, 66	MBody1_project/model/sizes.h, 234
classol, 73, 74	MBody_delayedSyn_project/model/sizes.h, 235
recordingsDir	MBody_individualID_project/model/sizes.h, 236
Schmuker2014_classifier, 111	MBody_userdef_project/model/sizes.h, 236
remap	OneComp_project/model/sizes.h, 237
SparseProjection, 116	PoissonIzh_project/model/sizes.h, 237
resetClusterSpikeCountAN	SCALAR_MIN
. 355. Sidotoi opiito oodiilii ii i	

HHVClampParameters.h, 167	addInputRate, 108
Izh sparse project/model/sizes.h, 234	allocateHostAndDeviceMemory, 108
MBody1_project/model/sizes.h, 234	applyLearningRuleSynapses, 108
MBody_delayedSyn_project/model/sizes.h, 235	cacheDir, 110
MBody_individualID_project/model/sizes.h, 236	calculateCurrentWindowWinner, 108
MBody_userdef_project/model/sizes.h, 236	calculateOverallWinner, 108
OneComp_project/model/sizes.h, 237	calculateVrResponse, 108
PoissonIzh_project/model/sizes.h, 237	calculateWinner, 108
SETUP_THE_C	checkContents, 108
simulation_synapse_policy_sparse.h, 233	classLabel, 110
SPARSEUTILS_CC	clearDownDevice, 108
sparseUtils.cc, 238	clearedDownDevice, 110
SPARSE	clusterSpikeCountAN, 110
modelSpec.h, 217	convertToRateCode, 108
SPIKESOURCE	correctClass, 110
neuronModels.cc, 219	countAN, 110
neuronModels.h, 221	countPNAN, 110
SPIKING ACTIVITY THRESHOLD HZ	countPN, 110
experiment.h, 132	countRN, 110
STRINGUTILS CC	createWTAConnectivity, 108
stringUtils.cc, 241	d_inputRates, 110
SYN_OUT_TME	d_maxRandomNumber, 110
MBody1_project/model/classol_sim.h, 124	data_type, 108
MBody_delayedSyn_project/model/classol_sim.h,	data_type_double, 108
125	data_type_float, 108
MBody_individualID_project/model/classol_sim.h,	data_type_int, 108
126	data_type_uint, 108
MBody_userdef_project/model/classol_sim.h, 127	datasetName, 110
PoissonIzh_sim.h, 227	findMaxMinSampleDistances, 108
SYNAPSE_TAU_ANAN	generate_inputrates_dataset, 108
Model_Schmuker_2014_classifier.cc, 214	generate_or_load_inputrates_dataset, 108
SYNAPSE_TAU_PNAN	generateSimulatedTimeSeriesData, 108
Model_Schmuker_2014_classifier.cc, 214	getClassCluster, 108
SYNAPSE_TAU_PNPN	getClusterIndex, 108
Model_Schmuker_2014_classifier.cc, 215	getManhattanDistance, 108
SYNAPSE_TAU_RNPN	getRand0to1, 108
Model_Schmuker_2014_classifier.cc, 215	getRecordingFilename, 109
SYNAPSEMODELS_CC	getSampleDistance, 109
synapseModels.cc, 245	getSpikeNumbersFromGPU, 109
SYNDELAYSIM_CU	getSpikesFromGPU, 109
SynDelaySim.cc, 247	individualSpikeCountPN, 110
SYNTYPENO	initialiseInputData, 109
synapseModels.h, 246	initialiseWeights DENSE PN AN, 109
sampleDistance	initialiseWeights_SPARSE_RN_PN, 109
Schmuker2014_classifier, 111	initialiseWeights_WTA_AN_AN, 109
scalar	initialiseWeights_WTA_PN_PN, 109
HHVClampParameters.h, 167	inputRates, 110
•	•
Izh_sparse_project/model/sizes.h, 234	inputRatesSize, 110
MBody1_project/model/sizes.h, 234	load_VR_data, 109
MBody_delayedSyn_project/model/sizes.h, 235	loadArrayFromTextFile, 109
MBody_individualID_project/model/sizes.h, 236	loadClassLabels, 109
MBody_userdef_project/model/sizes.h, 236	log, 110
OneComp_project/model/sizes.h, 236	model, 110
PoissonIzh_project/model/sizes.h, 237	openRecordingFile, 109
scalarExpr	outputDir, 110
NNmodel, 90	outputSpikes, 109
Schmuker2014_classifier, 105	overallWinnerSpikeCountAN, 110
\sim Schmuker2014_classifier, 108	param_CONNECTIVITY_AN_AN, 110

param_CONNECTIVITY_PN_AN, 110	setMaxMinSampleDistances
param_CONNECTIVITY_PN_PN, 110	Schmuker2014_classifier, 109
param_CONNECTIVITY_RN_PN, 110	setName
param_GLOBAL_WEIGHT_SCALING, 110	NNmodel, 91
param_MAX_FIRING_RATE_HZ, 110	setNeuronClusterIndex
param_MAX_WEIGHT_PN_AN, 110	NNmodel, 91
param_MIN_FIRING_RATE_HZ, 110	setPopulationSums
param_MIN_WEIGHT_PN_AN, 110	NNmodel, 91
param_PLASTICITY_INTERVAL_MS, 110	setPrecision
param SPIKING ACTIVITY THRESHOLD HZ,	NNmodel, 91
110	
	setSeed
param_WEIGHT_DELTA_PN_AN, 110	NNmodel, 91
param_WEIGHT_RN_PN, 111	setSpanTypeToPre
param_WEIGHT_WTA_AN_AN, 111	NNmodel, 91
param_WEIGHT_WTA_PN_PN, 111	setSparseConnectivityFromDense
plasticWeights, 111	sparseUtils.h, 240
populateDeviceMemory, 109	setSynapseClusterIndex
printSeparator, 109	NNmodel, 92
randomEventOccurred, 109	setSynapseG
recordingsDir, 111	NNmodel, 92
resetClusterSpikeCountAN, 109	SetTheW
resetDevice, 109	SimulationSynapsePolicyDense, 113
resetIndividualSpikeCountPN, 109	setTiming
resetOverallWinner, 109	NNmodel, 92
	SetUp
run, 109	•
sampleDistance, 111	SimulationTest, 114
Schmuker2014_classifier, 108	setVerbose
setCorrectClass, 109	CodeHelper, 76
setMaxMinSampleDistances, 109	setB
startLog, 109	gen_pnkc_syns_indivID.cc, 139
timestepsPerRecording, 111	utils.h, 268
uniqueRunId, 111	setup_kernel
update_input_data_on_device, 109	GeNNHelperKrnls.cu, 160
updateClusterSpikeCountAN, 109	GeNNHelperKrnls.h, 160
updateIndividualSpikeCountPN, 109	showPtxInfo
updateWeights PN AN on device, 109	GENN PREFERENCES, 63
updateWeights_PN_AN, 109	shuffle
vrData, 111	QTIsaac, 102
winningClass, 111	sigENa
Schmuker2014_classifier.cc, 231	helper.h, 166
_SCHMUKER2014_CLASSIFIER_, 231	sigEK
Schmuker2014 classifier.h, 231	helper.h, 165
	•
seed NIN-seed-1.05	sigEl
NNmodel, 95	helper.h, 165
setConstInp	sigGNa
NNmodel, 90	helper.h, 166
setCorrectClass	sigGK
Schmuker2014_classifier, 109	helper.h, 166
setDefaultParamValues	sigGl
experiment.cc, 130	helper.h, 166
setDT	sigC
NNmodel, 90	helper.h, 165
setGPUDevice	simCode
NNmodel, 91	extra_neurons.h, 133
setInput	neuronModel, 81
classIzh, 66	weightUpdateModel, 120
setMaxConn	simCode_supportCode
NNmodel, 91	weightUpdateModel, 120

simCodeEvnt	neuron_support_code_threshold/test.cc, 251
weightUpdateModel, 120	SimulationTestPostVars
simLearnPost	post_vars_in_post_learn/test.cc, 252
weightUpdateModel, 120	post_vars_in_sim_code/test.cc, 254
simLearnPost_supportCode	post_vars_in_synapse_dynamics/test.cc, 255
weightUpdateModel, 120	SimulationTestPostVarsSparse
simLearnPostKernelParameterTypes	post_vars_in_post_learn_sparse/test.cc, 253
NNmodel, 95 simLearnPostKernelParameters	post_vars_in_sim_code_sparse/test.cc, 254
NNmodel, 95	<pre>post_vars_in_synapse_dynamics_sparse/test.cc, 256</pre>
SimTest	SimulationTestPreVars
extra_global_post_param_in_sim_code/test.cc,	pre_vars_in_post_learn/test.cc, 256
249	pre_vars_in_sim_code/test.cc, 258
extra_global_pre_param_in_sim_code/test.cc, 250	pre_vars_in_sim_code_event/test.cc, 258
Simulate	pre_vars_in_sim_code_event_sparse/test.cc, 259
SimulationSynapsePolicy, 112	pre_vars_in_sim_code_event_sparse_inv/test.cc,
SimulationSynapsePolicyDense, 113	260
SimulationSynapsePolicyNone, 113	pre_vars_in_sim_code_sparse/test.cc, 261
SimulationSynapsePolicySparse, 114	pre_vars_in_synapse_dynamics/test.cc, 261
SimulationTestVars, 115	SimulationTestPreVarsSparse
simulation_neuron_policy_pre_post_var.h, 232	pre_vars_in_post_learn_sparse/test.cc, 257
simulation_neuron_policy_pre_var.h, 232	pre_vars_in_synapse_dynamics_sparse/test.cc,
simulation_synapse_policy_dense.h, 232	262
simulation_synapse_policy_none.h, 232	SimulationTestSynapseSupportCode
simulation_synapse_policy_sparse.h, 232	synapse_support_code_event_sim_code/test.cc,
SETUP_THE_C, 233	263
simulation_test.h, 233	synapse_support_code_event_threshold/test.cc,
simulation_test_vars.h, 233	263
ASSIGN_ARRAY_VARS, 233	synapse_support_code_post_learn/test.cc, 264
SimulationNeuronPolicyPrePostVar, 111	synapse_support_code_sim_code/test.cc, 265
Init, 111	synapse_support_code_synapse_dynamics/test. ←
SimulationNeuronPolicyPreVar, 111	cc, 266
Init, 111	SimulationTestVars
SimulationSynapsePolicy, 112	Init, 115
Init, 112	Simulate, 115
Simulate, 112	SimulationTestVars< NeuronPolicy, SynapsePolicy >,
SimulationSynapsePolicyDense, 112	115
GetTheW, 113	simulatorBackends
Init, 113	extra_global_params_in_sim_code/test.cc, 248
SetTheW, 113 Simulate, 113	extra_global_params_in_sim_code_event_spare ← _inv/test.cc, 249
SimulationSynapsePolicyNone, 113	extra_global_post_param_in_sim_code/test.cc,
Init, 113	249
Simulate, 113	extra_global_pre_param_in_sim_code/test.cc, 250
SimulationSynapsePolicySparse, 113	neuron_support_code_sim/test.cc, 251
Init, 114	neuron_support_code_threshold/test.cc, 252
Simulate, 114	post_vars_in_post_learn/test.cc, 252
SimulationTest, 114	post vars in post learn sparse/test.cc, 253
Init, 114	post_vars_in_sim_code/test.cc, 254
SetUp, 114	post_vars_in_sim_code_sparse/test.cc, 254
StepGeNN, 115	post_vars_in_synapse_dynamics/test.cc, 255
TearDown, 115	post_vars_in_synapse_dynamics_sparse/test.cc,
SimulationTestExtraGlobalParams	256
extra_global_params_in_sim_code/test.cc, 248	pre_vars_in_post_learn/test.cc, 256
extra_global_params_in_sim_code_event_spare ←	pre_vars_in_post_learn_sparse/test.cc, 257
_inv/test.cc, 249	pre_vars_in_sim_code/test.cc, 258
SimulationTestNeuronSupportCode	pre_vars_in_sim_code_event/test.cc, 259
neuron_support_code_sim/test.cc, 251	pre_vars_in_sim_code_event_sparse/test.cc, 259

pre_vars_in_sim_code_event_sparse_inv/test.cc,	src/modelSpec.cc
260	GeNNReady, 215
pre_vars_in_sim_code_sparse/test.cc, 261	initGeNN, 215
pre_vars_in_synapse_dynamics/test.cc, 261	MODELSPEC_CC, 215
pre_vars_in_synapse_dynamics_sparse/test.cc,	st
262	inputSpec, 79
synapse_support_code_event_sim_code/test.cc,	start
263	stopWatch, 117
synapse_support_code_event_threshold/test.cc,	startLog
264	Schmuker2014_classifier, 109
synapse_support_code_post_learn/test.cc, 264	startTimer
synapse_support_code_sim_code/test.cc, 265	CStopWatch, 77
synapse_support_code_synapse_dynamics/test.	stdRG, 116
cc, 266	\sim stdRG, 117
single_var_init_fullrange	n, 117
helper.h, 165	nlong, 117
single_var_reinit	stdRG, 116
helper.h, 165	stdTM_ini
size_g	HHVclampGA_project/model/MBody1.cc, 179
classol, 76	MBody1_project/model/MBody1.cc, 183
sizes.h, 233–237	MBody delayedSyn.cc, 186
slope0	MBody_individualID.cc, 190
pwSTDP_userdef, 100	MBody_userdef.cc, 194
pwSTDP, 99	stdTM_p
slope1	HHVclampGA_project/model/MBody1.cc, 179
pwSTDP_userdef, 100	MBody1_project/model/MBody1.cc, 183
pwSTDP, 99	MBody_delayedSyn.cc, 186
SparseProjection, 115	MBody_individualID.cc, 190
connN, 116	MBody_userdef.cc, 194
ind, 116	StepGeNN
indlnG, 116	SimulationTest, 115
preInd, 116	stop
remap, 116	stopWatch, 117
revInd, 116	stopTimer
revind, 116	CStopWatch, 77
sparseProjection.h, 237	stopWatch, 117
sparseUtils.cc, 237	start, 117
createPosttoPreArray, 238	stop, 117
createPreIndices, 238	stop, 117 stringUtils.cc, 240
initializeSparseArray, 238	checkUnreplacedVariables, 241
initializeSparseArrayPreInd, 238	ensureFtype, 241
initializeSparseArrayRev, 238	extended_name_substitutions, 241
SPARSEUTILS_CC, 238	extended_value_substitutions, 241
sparseUtils.h, 238	name_substitutions, 241
countEntriesAbove, 239	neuron_substitutions_in_synaptic_code, 241
createPosttoPreArray, 239	REGEX_OPERATIONAL, 241
createPreIndices, 239	STRINGUTILS_CC, 241
createSparseConnectivityFromDense, 239	substitute, 242
getSparseVar, 240	value_substitutions, 242
getG, 239	stringUtils.h, 242
initializeSparseArray, 240	checkUnreplacedVariables, 243
initializeSparseArrayPreInd, 240	ensureFtype, 243
initializeSparseArrayRev, 240	extended_name_substitutions, 243
setSparseConnectivityFromDense, 240	extended_value_substitutions, 243
srand	name_substitutions, 243
QTIsaac, 102	neuron_substitutions_in_synaptic_code, 243
randomGauss, 103	substitute, 244
randomGen, 104	toString, 244

tS, 243	INSTANTIATE TEST CASE P, 263
value_substitutions, 244	SimulationTestSynapseSupportCode, 263
substitute	simulatorBackends, 263
stringUtils.cc, 242	TEST_P, 263
stringUtils.h, 244	synapse_support_code_event_threshold/model.cc
sum_spikes	modelDefinition, 211
classIzh, 66	neuron_ini, 211
classol, 74, 75	synapses_ini, 211
neuronpop, 83	synapses_p, 211
sumDN	synapse_support_code_event_threshold/test.cc
classol, 76	INSTANTIATE_TEST_CASE_P, 263
sumlzh1	SimulationTestSynapseSupportCode, 263
classol, 76	simulatorBackends, 264
neuronpop, 83	TEST_P, 263
sumKC	synapse_support_code_post_learn/model.cc
classol, 76	modelDefinition, 212
sumLHI	neuron_ini, 212
classol, 76	neuron p, 212
sumNeuronN	neuron_p2, 212
NNmodel, 95	synapses_ini, 212
sumPExc	synapse_support_code_post_learn/test.cc
classIzh, 66	INSTANTIATE TEST CASE P, 264
sumPInh	SimulationTestSynapseSupportCode, 264
classIzh, 66	simulatorBackends, 264
sumPN	TEST P, 264
classol, 76	synapse_support_code_sim_code/model.cc
supportCode	modelDefinition, 213
neuronModel, 81	neuron_ini, 213
postSynModel, 98	synapses_ini, 213
SynDelay, 117	synapse_support_code_sim_code/test.cc
∼SynDelay, 118	INSTANTIATE_TEST_CASE_P, 265
run, 118	SimulationTestSynapseSupportCode, 265
SynDelay, 118	simulatorBackends, 265
SynDelay.cc, 246	TEST P, 265
modelDefinition, 246	synapse_support_code_synapse_dynamics/model.cc
SynDelaySim.cc, 246	modelDefinition, 213
main, 247	neuron_ini, 213
SYNDELAYSIM_CU, 247	synapses_ini, 214
SynDelaySim.h, 247	synapse_support_code_synapse_dynamics/test.cc
REPORT_TIME, 247	INSTANTIATE_TEST_CASE_P, 266
TOTAL_TIME, 247	SimulationTestSynapseSupportCode, 266
synDynBlkSz	simulatorBackends, 266
global.cc, 161	TEST_P, 266
global.h, 163	synapseBlkSz
synDynGroups	global.cc, 161
NNmodel, 96	global.h, 163
synDynGrp	synapseBlockSize
NNmodel, 96	GENN_PREFERENCES, 63
Synlzh_ini	SynapseConnType
lzh_sparse.cc, 171	modelSpec.h, 217
Synlzh_p	synapseConnType
Izh_sparse.cc, 171	NNmodel, 95
synapse_support_code_event_sim_code/model.cc	synapseDelay
modelDefinition, 210	NNmodel, 95
neuron_ini, 210	synapseDeviceID
synapses_ini, 210	NNmodel, 95
synapses_p, 211	synapseDynamics
synapse_support_code_event_sim_code/test.cc	weightUpdateModel, 120

synapseDynamics_supportCode	synapseUsesSynapseDynamics
weightUpdateModel, 120	NNmodel, 96
synapseDynamicsBlockSize	synapseUsesTrueSpikes
GENN_PREFERENCES, 64	NNmodel, 96
synapseDynamicsKernelParameterTypes	synapses_ini
NNmodel, 95	extra_global_params_in_sim_code_event_spare ←
synapseDynamicsKernelParameters	_inv/model.cc, 195
NNmodel, 95	extra_global_post_param_in_sim_code/model.cc,
SynapseGType	196
modelSpec.h, 217	extra_global_pre_param_in_sim_code/model.cc,
synapseGType	196
NNmodel, 95	neuron_support_code_sim/model.cc, 197
synapseGrpN	neuron_support_code_threshold/model.cc, 198
NNmodel, 95	post_vars_in_post_learn/model.cc, 199
synapseHostID	post_vars_in_post_learn_sparse/model.cc, 200
NNmodel, 95	post_vars_in_sim_code/model.cc, 200
synapseInSynNo	post_vars_in_sim_code_sparse/model.cc, 201
NNmodel, 95	post_vars_in_synapse_dynamics/model.cc, 202
synapselni	post_vars_in_synapse_dynamics_sparse/model.
NNmodel, 95	cc, 202
synapseKernelParameterTypes	pre_vars_in_post_learn/model.cc, 203
NNmodel, 96	pre_vars_in_post_learn_sparse/model.cc, 204
synapseKernelParameters	pre_vars_in_sim_code/model.cc, 205
NNmodel, 95	pre_vars_in_sim_code_event/model.cc, 205
synapseModels.cc, 244	pre_vars_in_sim_code_event_sparse/model.cc,
LEARN1SYNAPSE, 245	206
NGRADSYNAPSE, 245	pre_vars_in_sim_code_event_sparse_inv/model.
NSYNAPSE, 245	cc, 207
prepareWeightUpdateModels, 245	pre_vars_in_sim_code_sparse/model.cc, 208
SYNAPSEMODELS_CC, 245	pre_vars_in_synapse_dynamics/model.cc, 209
weightUpdateModels, 245	pre_vars_in_synapse_dynamics_sparse/model.cc,
synapseModels.h, 245	210
LEARN1SYNAPSE, 246	synapse_support_code_event_sim_code/model.
NGRADSYNAPSE, 246	cc, 210
NSYNAPSE, 246	synapse_support_code_event_threshold/model.
prepareWeightUpdateModels, 246	cc, 211
SYNTYPENO, 246	synapse_support_code_post_learn/model.cc, 212
weightUpdateModels, 246	synapse_support_code_sim_code/model.cc, 213
synapseName	synapse_support_code_synapse_dynamics/model.
NNmodel, 96	cc, 214
synapseOutSynNo	synapses_p
NNmodel, 96	pre_vars_in_sim_code_event/model.cc, 206
synapsePSize	pre_vars_in_sim_code_event_sparse/model.cc,
Izh_sparse.cc, 171	206
synapsePara	pre_vars_in_sim_code_event_sparse_inv/model.
NNmodel, 96	cc, 207
synapseSource	synapse_support_code_event_sim_code/model.←
NNmodel, 96	cc, 211
synapseSpanType	synapse_support_code_event_threshold/model.←
NNmodel, 96	cc, 211
synapseTarget	
NNmodel, 96	t
synapseType	inputSpec, 79
NNmodel, 96	T_REPORT_TME
synapseUsesPostLearning	lzh_sparse_sim.h, 172
NNmodel, 96	MBody1_project/model/classol_sim.h, 124
synapseUsesSpikeEvents	MBody_delayedSyn_project/model/classol_sim.h,
NNmodel, 96	125

MBody_individualID_project/model/classol_sim.h,	TRAUBMILES_ALTERNATIVE neuronModels.cc, 219
MBody_userdef_project/model/classol_sim.h, 127	neuronModels.h, 221
OneComp_sim.h, 224	TRAUBMILES FAST
PoissonIzh_sim.h, 227	neuronModels.cc, 219
TEST P	neuronModels.h, 221
extra_global_params_in_sim_code/test.cc, 248	TRAUBMILES_PSTEP
extra_global_params_in_sim_code_event_spare←	neuronModels.cc, 219
_inv/test.cc, 249	neuronModels.h, 221
extra_global_post_param_in_sim_code/test.cc,	TRAUBMILES SAFE
249	neuronModels.cc, 220
extra_global_pre_param_in_sim_code/test.cc, 250	neuronModels.h, 221
neuron_support_code_sim/test.cc, 251	TRAUBMILES
neuron_support_code_threshold/test.cc, 252	neuronModels.cc, 219
post_vars_in_post_learn/test.cc, 252	neuronModels.h, 221
post_vars_in_post_learn_sparse/test.cc, 253	TRUE
post_vars_in_sim_code/test.cc, 254	global.h, 163
post_vars_in_sim_code_sparse/test.cc, 254	TearDown
post_vars_in_synapse_dynamics/test.cc, 255	SimulationTest, 115
post_vars_in_synapse_dynamics_sparse/test.cc,	test.cc, 247–265
256	theDevice
pre_vars_in_post_learn/test.cc, 256	global.cc, 161
pre_vars_in_post_learn_sparse/test.cc, 257	global.h, 163
pre_vars_in_sim_code/test.cc, 258	theRates
pre_vars_in_sim_code_event/test.cc, 259	classol, 76
pre_vars_in_sim_code_event_sparse/test.cc, 259	theSize
pre_vars_in_sim_code_event_sparse_inv/test.cc,	utils.cc, 266
260	utils.h, 268
pre_vars_in_sim_code_sparse/test.cc, 261	thresholdConditionCode
pre_vars_in_synapse_dynamics/test.cc, 261	neuronModel, 81
pre_vars_in_synapse_dynamics_sparse/test.cc,	timer
262	experiment.h, 132
synapse_support_code_event_sim_code/test.cc,	Izh_sparse_sim.h, 172
263	MBody1_project/model/classol_sim.h, 124
synapse_support_code_event_threshold/test.cc, 263	MBody_delayedSyn_project/model/classol_sim.h, 125
synapse_support_code_post_learn/test.cc, 264	MBody_individualID_project/model/classol_sim.h,
synapse_support_code_sim_code/test.cc, 265	126
synapse_support_code_synapse_dynamics/test.↔	MBody_userdef_project/model/classol_sim.h, 127
cc, 266	OneComp_sim.h, 224
TIMING	PoissonIzh_sim.h, 228
MBody_userdef.cc, 191	VClampGA.h, 269
TOTAL_RECORDINGS	timestepsPerRecording
experiment.h, 132	Schmuker2014_classifier, 111
TOTAL_TIME	timing
SynDelaySim.h, 247	NNmodel, 96
TOTAL_TME	tmpVarNames
Izh_sparse_sim.h, 172	neuronModel, 81
MBody1_project/model/classol_sim.h, 124	tmpVarTypes
MBody_delayedSyn_project/model/classol_sim.h,	neuronModel, 82
125	toLower
MBody_individualID_project/model/classol_sim.h,	command_line_processing.h, 129
126 MPady upardef project/model/classed sim b 127	toString
MBody_userdef_project/model/classol_sim.h, 127	stringUtils.h, 244
OneComp_sim.h, 224 PoissonIzh_sim.h, 227	toUpper
TOTALT	command_line_processing.h, 129
HHVClampParameters.h, 167	truevar_init helper.h, 165
r ciampi aramotorom, 101	

truevar_initexpHH	stringUtils.cc, 242
helper.h, 165	stringUtils.h, 244
tS	var_init_fullrange
stringUtils.h, 243	helper.h, 165
LUNIT	var_reinit
UINT	helper.h, 165
experiment.h, 132	varNames
UTILS_CC	neuronModel, 82
utils.cc, 266	postSynModel, 98
uniqueRunId	weightUpdateModel, 120
Schmuker2014_classifier, 111	varTypes
update_input_data_on_device	neuronModel, 82
Schmuker2014_classifier, 109	postSynModel, 98
updateClusterSpikeCountAN	weightUpdateModel, 121
Schmuker2014_classifier, 109	variable_def
updateIndividualSpikeCountPN	generateRunner.cc, 157
Schmuker2014_classifier, 109	generateRunner.h, 159
updateWeights_PN_AN_on_device	vectorContains
Schmuker2014_classifier, 109	experiment.cc, 130
updateWeights_PN_AN	Vexp
Schmuker2014_classifier, 109	helper.h, 166
userCxxFlagsGNU	vrData
GENN_PREFERENCES, 64	Schmuker2014_classifier, 111
userCxxFlagsWIN	_ ,
GENN PREFERENCES, 64	WEIGHT_DELTA_PN_AN
userNvccFlags	experiment.h, 132
GENN_PREFERENCES, 64	WEIGHT_RN_PN
utils.cc, 266	experiment.h, 132
cudaFuncGetAttributesDriver, 266	WEIGHT_WTA_AN_AN
gennError, 266	experiment.h, 132
theSize, 266	WEIGHT_WTA_PN_PN
UTILS_CC, 266	experiment.h, 132
writeHeader, 266	weightUpdateModel, 118
utils.h, 267	~weightUpdateModel, 119
_UTILS_H_, 267	dpNames, 119
B, 267	dps, 119
CHECK_CU_ERRORS, 268	evntThreshold, 119
CHECK CUDA ERRORS, 268	extraGlobalSynapseKernelParameterTypes, 120
cudaFuncGetAttributesDriver, 268	extraGlobalSynapseKernelParameters, 119
delB, 268	needPostSt, 120
	needPreSt, 120
gennError, 268 setB, 268	pNames, 120
	simCode, 120
theSize, 268	simCode_supportCode, 120
writeHeader, 268	simCodeEvnt, 120
V	simLearnPost, 120
inputSpec, 79	simLearnPost supportCode, 120
VClampGA.cc, 268	synapseDynamics, 120
main, 269	· · · · ·
	synapseDynamics_supportCode, 120
VClampGA.h, 269	varNames, 120
R, 269	varTypes, 121
RAND, 269	weight UpdateModels
RG, 269	weightUpdateModels
timer, 269	synapseModels.cc, 245
VR_DATA_FILENAME	synapseModels.h, 246
experiment.h, 132	winningClass
value	Schmuker2014_classifier, 111
Parameter, 97	write_input_to_file
value_substitutions	classIzh, 66

```
write_kcdnsyns
classol, 75
write_para
helper.h, 165
write_pnkcsyns
classol, 75
write_pnlhisyns
classol, 75
writeHeader
utils.cc, 266
utils.h, 268

xorwow_setup
GeNNHelperKrnls.cu, 160
GeNNHelperKrnls.h, 160
```