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 interfaces for SpineML and SpineCreator and for Brian via Brian2GeNN.

GeNN is currently developed and maintained by

Dr James Knight (contact James)
James Turner (contact James)
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 numbers EP/P006094/1 - Brains on Board and 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 next release will be based. 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.

Note

While GeNN models are normally simulated using CUDA on NVIDIA GPUs, if you want to use GeNN on a machine without an NVIDIA GPU, you can skip steps v and vi and use GeNN in "CPU_ONLY" mode.

(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) Add GeNN's "bin" directory to your path, e.g. if you are running Linux or Mac OS X and extracted/downloaded GeNN to $HOME/GeNN, then you can add:

```
export PATH=$PATH:$HOME/GeNN/bin
```

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 PATH "C:\Users\me\GeNN\bin;%PATH%"
```

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

(iv) Install the C++ compiler on the machine, if not already present. For Windows, download Microsoft Visual Studio Community Edition from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx. When installing Visual Studio, one should select the 'Desktop development with C++' configuration' and the 'Windows 8.1 SDK' and 'Windows Universal CRT' individual components. 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 your machine has a GPU and you haven't installed CUDA already, obtain a fresh installation of the NVIDIA CUDA toolkit from https://developer.nvidia.com/cuda-downloads Again, be sure to pick CUDA and C++ compiler versions which are compatible with each other. The latest C++ compiler is not necessarily compatible with the latest CUDA toolkit.

(vi) Set the `CUDA_PATH` variable if it is not already set by the system, by putting

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

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

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

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

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 - Microsoft Visual Studio - Visual Studio Tools - x64 Native Tools Command Prompt. You may wish to create a shortcut for this tool on the desktop, for convenience.

Top | Next

3 Quickstart

GeNN is based on the idea of code generation for the involved GPU or CPU simulation code for neuronal network models but leaves a lot of freedom how to use the generated code in the final application. To facilitate the use of GeNN on the background of this philosophy, it comes with a number of complete examples containing both the model description code that is used by GeNN for code generation and the "user side code" to run the generated model and safe the results. Some of the example models such as the Insect olfaction model use an `generate_run` executable which automates the building and simulation of the model. Using these executables, running these complete examples should be achievable in a few minutes. The necessary steps are described below.

3.1 Running an Example Model

3.1.1 Unix

In order to build the `generate_run` executable as well as any additional tools required for the model, open a shell and navigate to the `userproject/MBody1_project` directory. Then type

```
make
```

to generate an executable that you can invoke with

```
./generate_run test1
```

or, if you don't have an NVIDIA GPU and are running GeNN in CPU_ONLY mode, you can instead invoke this executable with

```
./generate_run --cpu-only test1
```

3.1.2 Windows

While GeNN can be used from within Visual Studio, in this example we will use a `cmd` window. Open a Visual Studio `cmd` window via Start: All Programs: Visual Studio: Tools: x86 Native Tools Command Prompt, and navigate to the `userproject\tools` directory. Then compile the additional tools and the `generate_run` executable for creating and running the project:

```
msbuild ..\userprojects.sln /t:generate_mbody1_runner /p:Configuration=Release
```

to generate an executable that you can invoke with

```
generate_run test1
```

or, if you don't have an NVIDIA GPU and are running GeNN in CPU_ONLY mode, you can instead invoke this executable with

```
generate_run --cpu-only test1
```

3.1.3 Visualising results

These steps will build and simulate a model of the locust olfactory system with default parameters of 100 projection neurons, 1000 Kenyon cells, 20 lateral horn interneurons and 100 output neurons in the mushroom body lobes.

Note

If the model isn't build in CPU_ONLY mode it will be simulated on an automatically chosen GPU.

The generate_run tool generates input patterns and writes them to file, compiles and runs the model using these files as inputs and finally output the resulting spiking activity. For more information of the options passed to this command see the Insect olfaction model section. The results of the simulation can be plotted with

```
python plot.py test1
```

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 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++ model file (see below) directly to the genn-buildmodel script. Another way to use GeNN is to create or modify a script or executable such as `userproject/MBody1_project/generate_run.cc` that wraps around the other programs that are used for each of the steps listed above. In more detail, the GeNN workflow consists of:

1. Either use external programs to generate connectivity and input files to be loaded into the user side code at runtime or generate these matrices directly inside the user side code.
2. Generating the model simulation code using `genn-buildmodel.sh` (On Linux or Mac) or `genn-buildmodel.bat` (on Windows). For example, inside the `generate_run` engine used by the MBody1_project, the following command is executed on Linux:

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

   or, if you don't have an NVIDIA GPU and are running GeNN in CPU_ONLY mode, the following command is executed:

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

   The `genn-buildmodel` script compiles the GeNN code generator in conjunction with the user-provided model description `model/MBody1.cc`. It then executes the GeNN code generator to generate the complete model simulation code for the model.
3. Provide a build script to compile the generated model simulation and the user side code into a simulator executable (in the case of the MBody1 example this consists the file `MBody1Sim.cc`). On Linux or Mac a suitable GNU makefile can be created by running:

   ```
   genn-create-user-project.sh MBody1 MBody1Sim.cc
   ```

   And on Windows an MSBuild project can be created by running:

   ```
   genn-create-user-project.bat MBody1 MBody1Sim.cc
   ```
4. Compile the simulator executable by invoking GNU make on Linux or Mac:

   ```
   make clean all
   ```

   or MSbuild on Windows:

   ```
   msbuild MBody1.sln /t:MBody1 /p:Configuration=Release
   ```
5. Finally, run the resulting stand-alone simulator executable. In the MBody1 example, this is called `MBody1` on Linux and `MBody1_Release.exe` on Windows.

3.3 Defining a New Model in GeNN

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

1. The neuronal network of interest is defined in a model definition file, e.g. `Example1.cc`.
2. Within the the model definition file `Example1.cc`, the following tasks need to be completed:

   a) The GeNN file `modelSpec.h` needs to be included,

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

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

   ```
   NeuronModels::PoissonNew::ParamValues poissonParams(
     10.0);      // 0 - firing rate
   ```

   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 required for a particular population of neurons (or synapses), they need to be defined as "variables" rather than parameters. See the User Manual for how to define new neuron (or synapse) types and the Defining a new variable initialisation snippet section for more information on initialising these variables to hetererogenous values.

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

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

   Note

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

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

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

   Note

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

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

3. The programmer defines their own "user-side" modeling code similar to the code in `userproject/MBody1_project/model/MBody1Sim.cc`. In this code,

   a) They manually define the connectivity matrices between neuron groups. Refer to the Synaptic matrix types section 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 `stepTime()` to run one time step on either the CPU or GPU depending on the options passed to genn-buildmodel.

d) They use functions like `copyStateFromDevice()` etc to transfer the results from GPU calculations to the main memory of the host computer for further processing.

e) They analyze the results. In the most simple case this could just be writing the relevant data to output files.

Previous | Top | Next

4 Examples

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

4.1 Single compartment Izhikevich neuron(s)

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

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

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

msbuild ..\userprojects.sln /t:generate_one_comp_runner /p:Configuration=Release

for Windows users, or:

make

for Linux, Mac and other UNIX users.


USAGE
-----

generate_run [OPTIONS] <outname>

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

Optional arguments:
--debug: Builds a debug version of the simulation and attaches the debugger
--cpu-only: Uses CPU rather than CUDA backend for GeNN
--timing: Uses GeNN's timing mechanism to measure performance and displays it at the end of the simulation
--ftype: Sets the floating point precision of the model to either float or double (defaults to float)
--gpu-device: Sets which GPU device to use for the simulation (defaults to -1 which picks automatically)
--num-neurons: Number of neurons to simulate (defaults to 1)

For a first minimal test, using these defaults and recording results with a base name of 'test',the system may

generate_run.exe test

for Windows users, or:
```

```
./generate_run test

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.

Another example of an invocation that runs the simulation using the CPU rather than GPU,
records timing information and 4 neurons would be:

generate_run.exe --cpu-only --timing --num_neurons=4 test

for Windows users, or:

./generate_run --cpu-only --timing --num_neurons=4 test

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 compiles and
executes the model.

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

msbuild ..\userprojects.sln /t:generate_poisson_izh_runner /p:Configuration=Release

for Windows users, or:

make

for Linux, Mac and other UNIX users.


USAGE
-----

generate_run [OPTIONS] <outname>

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

Optional arguments:
--debug: Builds a debug version of the simulation and attaches the debugger
--cpu-only: Uses CPU rather than CUDA backend for GeNN
--timing: Uses GeNN's timing mechanism to measure performance and displays it at the end of the simulation
--ftype: Sets the floating point precision of the model to either float or double (defaults to float)
--gpu-device: Sets which GPU device to use for the simulation (defaults to -1 which picks automatically)
--num-poisson:  Number of Poisson sources to simulate (defaults to 100)
--num-izh: Number of Izhikievich neurons to simulate (defaults to 10)
--pconn: Probability of connection between each pair of poisson sources and neurons (defaults to 0.5)
--gscale: Scaling of synaptic conductances (defaults to 2)
--sparse: Use sparse rather than dense data structure to represent connectivity

An example invocation of generate_run using these defaults and recording results with a base name of 'test':

generate_run.exe test

for Windows users, or:

./generate_run test
```

```
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 --sparse flag to the command line.

Another example of an invocation that runs the simulation using the CPU rather than GPU,
records timing information and uses sparse connectivity would be:

generate_run.exe --cpu-only --timing --sparse test

for Windows users, or:

./generate_run --cpu-only --timing --sparse test

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 a fixed number of neurons with sparse connectivity.

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

msbuild ..\userprojects.sln /t:generate_izh_sparse_runner /p:Configuration=Release

for Windows users, or:

make

for Linux, Mac and other UNIX users.


USAGE
-----

generate_run [OPTIONS] <outname>

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

Optional arguments:
--debug: Builds a debug version of the simulation and attaches the debugger
--cpu-only: Uses CPU rather than CUDA backend for GeNN
--timing: Uses GeNN's timing mechanism to measure performance and displays it at the end of the simulation
--ftype: Sets the floating point precision of the model to either float or double (defaults to float)
--gpu-device: Sets which GPU device to use for the simulation (defaults to -1 which picks automatically)
--num-neurons: Number of neurons (defaults to 10000)
--num-connections: Number of connections per neuron (defaults to 1000)
--gscale: General scaling of synaptic conductances (defaults to 1.0)

An example invocation of generate_run using these defaults and recording results with a base name of 'test' wo

generate_run.exe test

for Windows users, or:

./generate_run test

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`.

Another example of an invocation that runs the simulation using the CPU rather than GPU,
records timing information and doubles the number of neurons would be:

generate_run.exe --cpu-only --timing --num_neurons=20000 test

for Windows users, or:

./generate_run --cpu-only --timing --num_neurons=20000 test

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:

msbuild SynDelay.sln /t:SynDelay /p:Configuration=Release


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 [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
prepares input pattern data, before compiling and
executing the model.

To compile it, navigate to genn/userproject/MBody1_project and type:

msbuild ..\userprojects.sln /t:generate_mbody1_runner /p:Configuration=Release

for Windows users, or:

make

for Linux, Mac and other UNIX users.


USAGE
-----

generate_run [OPTIONS] <outname>

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

Optional arguments:
--debug: Builds a debug version of the simulation and attaches the debugger
--cpu-only: Uses CPU rather than CUDA backend for GeNN
--timing: Uses GeNN's timing mechanism to measure performance and displays it at the end of the simulation
--ftype: Sets the floating point precision of the model to either float or double (defaults to float)
--gpu-device: Sets which GPU device to use for the simulation (defaults to -1 which picks automatically)
--num-al: Number of neurons in the antennal lobe (AL), the input neurons to this model (defaults to 100)
--num-kc: Number of Kenyon cells (KC) in the "hidden layer" (defaults to 1000)
--num-lhi: Number of lateral horn interneurons, implementing gain control (defaults to 20)
--num-dn: Number of decision neurons (DN) in the output layer (defaults to 100)
--gscale: A general rescaling factor for synaptic strength (defaults to 0.0025)
--bitmask: Use bitmasks to represent sparse PN->KC connectivity rather than dense connectivity
--delayed-synapses: Rather than use constant delays of DT throughough, use delays of (5 * DT) ms on KC->DN and

An example invocation of generate_run using these defaults and recording results with a base name of 'test' wo

generate_run.exe test

for Windows users, or:

./generate_run test

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 "test"
and will be created under the "test" directory. The model that is run is defined
in 'model/MBody1.cc', debugging is switched off and the model would be simulated using
float (single precision floating point) variables.

In more details, what generate_run program does is:
a) use another tools to generate input patterns.

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".

Another example of an invocation that runs the simulation using the CPU rather than GPU,
records timing information and uses bitmask connectivity would be:
```

```
generate_run.exe --cpu-only --timing --bitmask test

for Windows users, or:

./generate_run --cpu-only --timing --bitmask test

for Linux, Mac and other UNIX users.

As provided, the model outputs  `test.dn.st', `test.kc.st', `test.lhi.st' and `test.pn.st' files which contain
the spiking activity observed in each population inthe 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 and users of python can use the plot_spikes.py script in userproject/python.
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: [4]; Traub-Miles Hodgkin-Huxley neuron model: [7]

4.6 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 using GeNN and evolves them with a simple
guided random search (simple GA) to mimic the dynamics of a separate Hodgkin-Huxley
neuron that is simulated on the CPU. The parameters of the CPU simulated "true cell" are drifting
according to a user-chosen protocol: Either one of the parameters gNa, ENa, gKd, EKd, gleak,
Eleak, Cmem are modified by a sinusoidal addition (voltage parameters) or factor (conductance or capacitance)
protocol 0-6. For protocol 7 all 7 parameters undergo a random walk concurrently.

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

msbuild ..\userproject.sln /t:generate_hhvclamp_runner /p:Configuration=Release

for Windows users, or:s

make

for Linux, Mac and other UNIX users.


USAGE
-----

generate_run [OPTIONS] <outname>

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

Optional arguments:
--debug: Builds a debug version of the simulation and attaches the debugger
--cpu-only: Uses CPU rather than CUDA backend for GeNN
--timing: Uses GeNN's timing mechanism to measure performance and displays it at the end of the simulation
--ftype: Sets the floating point precision of the model to either float or double (defaults to float)
--gpu-device: Sets which GPU device to use for the simulation (defaults to -1 which picks automatically)
--protocol: Which changes to apply during the run to the parameters of the "true cell" (defaults to -1 which m
--num-pops: Number of neurons in the tracking population (defaults to 5000)
--total-time: Time in ms how long to run the simulation  (defaults to 1000 ms)
```

```
An example invocation of generate_run is:

generate_run.exe test1

for Windows users, or:

./generate_run test1

for Linux, Mac and other UNIX users.

This will simulate 5000 Hodgkin-Huxley neurons on the GPU which will, for 1000 ms, be matched to a
Hodgkin-Huxley neuron. 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 that records timing information for the the simulation and runs it for 10000

generate_run.exe --timing --total-time 10000

for Windows users, or:

./generate_run --timing --total-time 10000

for Linux, Mac and other UNIX users.
```

Traub-Miles Hodgkin-Huxley neuron model: [7]

4.7 A neuromorphic network for generic multivariate data classification

```
Author: Alan Diamond, University of Sussex, 2014

This project recreates using GeNN the spiking classifier design used in the paper

"A neuromorphic network for generic multivariate data classification"
 Authors: Michael Schmuker, Thomas Pfeil, Martin Paul Nawrota

The classifier design is based on an abstraction of the insect olfactory system.
This example uses the IRIS stadard data set as a test for the classifier

BUILD / RUN INSTRUCTIONS

Install GeNN from the internet released build, following instruction on setting your PATH etc

Start a terminal session

cd to this project directory (userproject/Model_Schmuker_2014_project)

To build the model using the GENN meta compiler type:

genn-buildmodel.sh Model_Schmuker_2014_classifier.cc

for Linux, Mac and other UNIX systems, or:

genn-buildmodel.bat Model_Schmuker_2014_classifier.cc

for Windows systems (add -d for a debug build).

You should only have to do this at the start, or when you change your actual network model  (i.e. editing the

Then to compile the experiment plus the GeNN created C/CUDA code type:-

make

for Linux, Mac and other UNIX users (add DEBUG=1 if using debug mode), or:

msbuild Schmuker2014_classifier.vcxproj /p:Configuration=Release

for Windows users (change Release to Debug if using debug mode).

Once it compiles you should be able to run the classifier against the included Iris dataset.
```

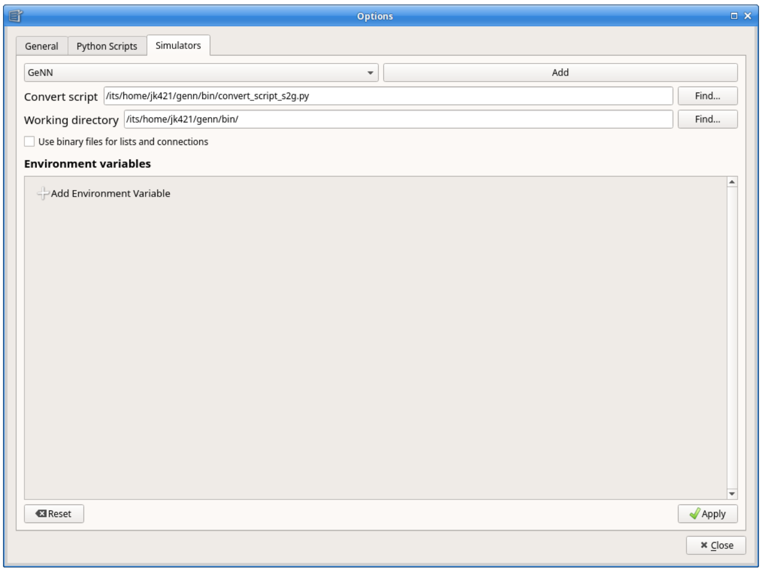
```
type

./experiment .

for Linux, Mac and other UNIX systems, or:

Schmuker2014_classifier .

for Windows systems.

This is how it works roughly.
The experiment (experiment.cu) controls the experiment at a high level. It mostly does this by instructing the

So the experiment first tells the classifier to set up the GPU with the model and synapse data.

Then it chooses the training and test set data.

It runs through the training set , with plasticity ON , telling the classifier to run with the specfied observ

Then it runs through the test set with plasticity OFF  and collects the results in various reporting files.

At the highest level it also has a loop where you can cycle through a list of parameter values e.g. some thres

You should also note there is no option currently to run on CPU, this is not due to the demanding task, it jus
```

Previous | Top | Next

5 SpineML and SpineCreator

GeNN now supports simulating models built using SpineML and includes scripts to fully integrate it with the SpineCreator graphical editor on Linux, Mac and Windows. After installing GeNN using the instructions in Installation, build SpineCreator for your platform.

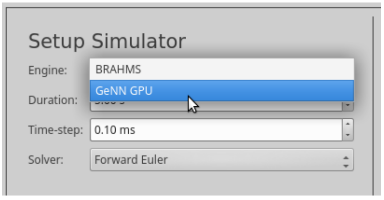
From SpineCreator, select Edit->Settings->Simulators and add a new simulator using the following settings (replacing "/home/j/jk/jk421/genn" with the GeNN installation directory on your own system):



If you would like SpineCreator to use GeNN in CPU only mode, add an environment variable called "GENN_SPINEML_CPU_ONLY".

The best way to get started using SpineML with GeNN is to experiment with some example models. A number are available here although the "Striatal model" uses features not currently supported by GeNN and the two "Brette

Benchmark" models use a legacy syntax no longer supported by SpineCreator (or GeNN). Once you have loaded a model, click "Expts" from the menu on the left hand side of SpineCreator, choose the experiment you would like to run and then select your newly created GeNN simulator in the "Setup Simulator" panel:



Now click "Run experiment" and, after a short time, the results of your GeNN simulation will be available for plotting by clicking the "Graphs" option in the menu on the left hand side of SpineCreator.

Previous | Top | Next

6 Brian interface (Brian2GeNN)

GeNN can simulate models written for the `Brian simulator` via the `Brian2GeNN` interface [6] . The easiest way to install everything needed is to install the `Anaconda` or `Miniconda` Python distribution and then follow the `instructions to install Brian2GeNN` with the conda package manager. When Brian2GeNN is installed in this way, it comes with a bundled version of GeNN and no further configuration is required. In all other cases (e.g. an installation from source), the path to GeNN and the CUDA libraries has to be configured via the `GENN_PATH` and `CUDA_PATH` environment variables as described in Installation or via the `devices.genn.path` and `devices.genn.cuda_path` `Brian preferences`.

To use GeNN to simulate a Brian script, import the `brian2genn` package and switch Brian to the `genn` device. As an example, the following Python script will simulate Leaky-integrate-and-fire neurons with varying input currents to construct an f/I curve:

```
from brian2 import *
import brian2genn
set_device('genn')

n = 1000
duration = 1*second
tau = 10*ms
eqs = '''
dv/dt = (v0 - v) / tau : volt (unless refractory)
v0 : volt
'''
group = NeuronGroup(n, eqs, threshold='v > 10*mV', reset='v = 0*mV',
                    refractory=5*ms, method='exact')
group.v = 0*mV
group.v0 = '20*mV * i / (n-1)'
monitor = SpikeMonitor(group)

run(duration)
```

Of course, your simulation should be more complex than the example above to actually benefit from the performance gains of using a GPU via GeNN.

Previous | Top | Next

7 Python interface (PyGeNN)

As well as being able to build GeNN models and user code directly from C++, you can also access all GeNN features from Python. The `pygenn.genn_model.GeNNModel` class provides a thin wrapper around `ModelSpec` as well as providing support for loading and running simulations; and accessing their state. `SynapseGroup`, `NeuronGroup` and `CurrentSource` are similarly wrapped by the `pygenn.genn_groups.SynapseGroup`, `pygenn.genn_groups.NeuronGroup` and `pygenn.genn_groups.CurrentSource` classes respectively.

PyGeNN can be built from source on Windows, Mac and Linux following the instructions in the README file in the pygenn directory of the GeNN repository. However, if you have a relatively recent version of Python and CUDA, we recommend that you instead downloading a suitable 'wheel' from our releases page. These can then be installed using e.g. `pip install cuda10-pygenn-0.2-cp27-cp27mu-linux_x86_64.whl` for a Linux system with CUDA 10 and Python 2.7. On Windows we recommend using the Python 3 version of Anaconda.

The following example shows how PyGeNN can be easily interfaced with standard Python packages such as numpy and matplotlib to plot 4 different Izhikevich neuron regimes:

```
import numpy as np
import matplotlib.pyplot as plt
from pygenn.genn_model import GeNNModel

# Create a single-precision GeNN model
model = GeNNModel("float", "pygenn")

# Set simulation timestep to 0.1ms
model.dT = 0.1

# Initialise IzhikevichVariable parameters - arrays will be automatically uploaded
izk_init = {"V": -65.0,
            "U": -20.0,
            "a": [0.02,     0.1,    0.02,   0.02],
            "b": [0.2,      0.2,    0.2,    0.2],
            "c": [-65.0,    -65.0,  -50.0,  -55.0],
            "d": [8.0,      2.0,    2.0,    4.0]}

# Add neuron populations and current source to model
pop = model.add_neuron_population("Neurons", 4, "IzhikevichVariable", {}, izk_init)
model.add_current_source("CurrentSource", "DC", "Neurons", {"amp": 10.0}, {})

# Build and load model
model.build()
model.load()

# Create a numpy view to efficiently access the membrane voltage from Python
voltage_view = pop.vars["V"].view

# Simulate
v = None
while model.t < 200.0:
    model.step_time()
    model.pull_state_from_device("Neurons")
    v = np.copy(voltage_view) if v is None else np.vstack((v, voltage_view))

# Create plot
figure, axes = plt.subplots(4, sharex=True)

# Plot voltages
for i, t in enumerate(["RS", "FS", "CH", "IB"]):
    axes[i].set_title(t)
    axes[i].set_ylabel("V [mV]")
    axes[i].plot(np.arange(0.0, 200.0, 0.1), v[:,i])

axes[-1].set_xlabel("Time [ms]")

# Show plot
plt.show()
```

Previous | Top | Next

8 Release Notes

Release Notes for GeNN v4.0.1

This release fixes several small bugs found in GeNN 4.0.0 and implements some small features:

User Side Changes

1. Improved detection and handling of errors when specifying model parameters and values in PyGeNN.
2. SpineML simulator is now implemented as a library which can be used directly from user applications as well as from command line tool.

Bug fixes:

1. Fixed typo in GeNNModel.push_var_to_device function in PyGeNN.
2. Fixed broken support for Visual C++ 2013.
3. Fixed zero-copy mode.
4. Fixed typo in tutorial 2.

Release Notes for GeNN v4.0.0

This release is the result of a second round of fairly major refactoring which we hope will make GeNN easier to use and allow it to be extended more easily in future. However, especially if you have been using GeNN 2.XX syntax, it breaks backward compatibility.

User Side Changes

1. Totally new build system - `make install` can be used to install GeNN to a system location on Linux and Mac and Windows projects work much better in the Visual Studio IDE.
2. Python interface now supports Windows and can be installed using binary 'wheels' (see Python interface (PyGeNN) for more details).
3. No need to call `initGeNN()` at start and `model.finalize()` at end of all models.
4. Initialisation system simplified - if you specify a value or initialiser for a variable or sparse connectivity, it will be initialised by your chosen backend. If you mark it as uninitialised, it is up to you to initialize it in user code between the calls to `initialize()` and `initializeSparse()` (where it will be copied to device).
5. `genn-create-user-project` helper scripts to create Makefiles or MSBuild projects for building user code
6. State variables can now be pushed and pulled individually using the `pull<var name><neuron or synapse name>FromDevice()` and `push<var name><neuron or synapse name>To←Device()` functions.
7. Management of extra global parameter arrays has been somewhat automated (see Extra Global Parameters for more details).
8. `GENN_PREFERENCES` is no longer a namespace - it's a global struct so members need to be accessed with . rather than ::.
9. `NeuronGroup`, `SynapseGroup`, `CurrentSource` and `NNmodel` all previously exposed a lot of methods that the user wasn't *supposed* to call but could. These have now all been made protected and are exposed to GeNN internals using derived classes (`NeuronGroupInternal`, `SynapseGroupInternal`, `CurrentSourceInternal`, `ModelSpecInternal`) that make them public using `using` directives.
10. Auto-refractory behaviour was controlled using `GENN_PREFERENCES::autoRefractory`, this is now controlled on a per-neuron-model basis using the `SET_NEEDS_AUTO_REFRACTORY` macro.
11. The functions used for pushing and pulling have been unified somewhat this means that `copyState←ToDevice` and `copyStateFromDevice` functions no longer copy spikes and `pus<neuron or synapse name>SpikesToDevice` and `pull<neuron or synapse name>SpikesFrom←Device` no longer copy spike times or spike-like events.
12. Standard models of leaky-integrate-and-fire neuron (`NeuronModels::LIF`) and of exponentially shaped postsynaptic current (`PostsynapticModels::ExpCurr`) have been added.
13. When a model is built using the CUDA backend, the device it was built for is stored using it's PCI bus ID so it will always use the same device.

Deprecations

1. Yale-format sparse matrices are no longer supported.
2. GeNN 2.X syntax for implementing neuron and synapse models is no longer supported.
3. $(addtoinSyn) = X; $(updatelinsyn); idiom in weight update models has been replaced by function style `$(addToInSyn, X);`.

Release Notes for GeNN v3.3.0

This release is intended as the last service release for GeNN 3.X.X. Fixes for serious bugs **may** be backported if requested but, otherwise, development will be switching to GeNN 4.

User Side Changes

1. Postsynaptic models can now have Extra Global Parameters.
2. Gamma distribution can now be sampled using `$(gennrand_gamma, a)`. This can be used to initialise variables using `InitVarSnippet::Gamma`.
3. Experimental Python interface - All features of GeNN are now exposed to Python through the `pygenn` module (see Python interface (PyGeNN) for more details).

Bug fixes:

1. Devices with Streaming Multiprocessor version 2.1 (compute capability 2.0) now work correctly in Windows.
2. Seeding of on-device RNGs now works correctly.
3. Improvements to accuracy of memory usage estimates provided by code generator.

Release Notes for GeNN v3.2.0

This release extends the initialisation system introduced in 3.1.0 to support the initialisation of sparse synaptic connectivity, adds support for networks with more sophisticated models of synaptic plasticity and delay as well as including several other small features, optimisations and bug fixes for certain system configurations. This release supports GCC >= 4.9.1 on Linux, Visual Studio >= 2013 on Windows and recent versions of Clang on Mac OS X.

User Side Changes

1. Sparse synaptic connectivity can now be initialised using small *snippets* of code run either on GPU or CPU. This can save significant amounts of initialisation time for large models. See Sparse connectivity initialisation for more details.
2. New 'ragged matrix' data structure for representing sparse synaptic connections – supports initialisation using new sparse synaptic connecivity initialisation system and enables future optimisations. See Synaptic matrix types for more details.
3. Added support for pre and postsynaptic state variables for weight update models to allow more efficient implementatation of trace based STDP rules. See Defining a new weight update model for more details.
4. Added support for devices with Compute Capability 7.0 (Volta) to block-size optimizer.
5. Added support for a new class of 'current source' model which allows non-synaptic input to be efficiently injected into neurons. See Current source models for more details.
6. Added support for heterogeneous dendritic delays. See Defining a new weight update model for more details.

7. Added support for (homogeneous) synaptic back propagation delays using SynapseGroup::setBackPropDelaySteps.

8. For long simulations, using single precision to represent simulation time does not work well. Added NNmodel::setTimePrecision to allow data type used to represent time to be set independently.

Optimisations

1. GENN_PREFERENCES::mergePostsynapticModels flag can be used to enable the merging together of postsynaptic models from a neuron population's incoming synapse populations - improves performance and saves memory.

2. On devices with compute capability $>$ 3.5 GeNN now uses the read only cache to improve performance of postsynaptic learning kernel.

Bug fixes:

1. Fixed bug enabling support for CUDA 9.1 and 9.2 on Windows.

2. Fixed bug in SynDelay example where membrane voltage went to NaN.

3. Fixed bug in code generation of SCALAR_MIN and SCALAR_MAX values.

4. Fixed bug in substitution of trancendental functions with single-precision variants.

5. Fixed various issues involving using spike times with delayed synapse projections.

Release Notes for GeNN v3.1.1

This release fixes several small bugs found in GeNN 3.1.0 and implements some small features:

User Side Changes

1. Added new synapse matrix types SPARSE_GLOBALG_INDIVIDUAL_PSM, DENSE_GLOBALG_INDIVIDUAL_PSM and BITMASK_GLOBALG_INDIVIDUAL_PSM to handle case where synapses with no individual state have a postsynaptic model with state variables e.g. an alpha synapse. See Synaptic matrix types for more details.

Bug fixes

1. Correctly handle aliases which refer to other aliases in SpineML models.

2. Fixed issues with presynaptically parallelised synapse populations where the postsynaptic population is small enough for input to be accumulated in shared memory.

Release Notes for GeNN v3.1.0

This release builds on the changes made in 3.0.0 to further streamline the process of building models with GeNN and includes several bug fixes for certain system configurations.

User Side Changes

1. Support for simulating models described using the SpineML model description language with GeNN (see SpineML and SpineCreator for more details).
2. Neuron models can now sample from uniform, normal, exponential or log-normal distributions - these calls are translated to cuRAND when run on GPUs and calls to the C++11 <random> library when run on CPU. See Defining your own neuron type for more details.
3. Model state variables can now be initialised using small *snippets* of code run either on GPU or CPU. This can save significant amounts of initialisation time for large models. See Defining a new variable initialisation snippet for more details.
4. New MSBuild build system for Windows - makes developing user code from within Visual Studio much more streamlined. See Debugging suggestions for more details.

Bug fixes:

1. Workaround for bug found in Glibc 2.23 and 2.24 which causes poor performance on some 64-bit Linux systems (namely on Ubuntu 16.04 LTS).
2. Fixed bug encountered when using extra global variables in weight updates.

Release Notes for GeNN v3.0.0

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

User Side Changes

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

Release Notes for GeNN v2.2.3

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

User Side Changes

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

Bug fixes:

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

Release Notes for GeNN v2.2.2

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

User Side Changes

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

Bug fixes:

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

Release Notes for GeNN v2.2.1

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

Bug fixes:

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

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

Release Notes for GeNN v2.2

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

User Side Changes

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

   Note

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

2. We introduced the namespace GENN_PREFERENCES which contains variables that determine the behaviour of GeNN.

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

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

   Note

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

4. The new convenience macros spikeCount_XX and spike_XX where "XX" is the name of the neuron group are now also available for events: spikeEventCount_XX and spikeEvent_XX. They access the values for the current time step even if there are synaptic delays and spikes events are stored in circular queues.

5. The old buildmodel.[sh|bat] scripts have been superseded by new genn-buildmodel.[sh|bat] scripts. These scripts accept UNIX style option switches, allow both relative and absolute model file paths, and allow the user to specify the directory in which all output files are placed (-o <path>). Debug (-d), CPU-only (-c) and show help (-h) are also defined.

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

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

7. The new genn-buildmodel "-o" switch allows the user to specify the output directory for all generated files - the default is the current directory. For example, a user project could be in '/home/genn_project', whilst the GeNN directory could be '/usr/local/genn'. The GeNN directory is kept clean, unless the user decides to build the sample projects inside of it without copying them elsewhere. This allows the deployment of GeNN to a read-only directory, like '/usr/local' or 'C:\Program Files'. It also allows multiple users - i.e. on a compute cluster - to use GeNN simultaneously, without overwriting each other's code-generation files, etcetera.

8. The ARM architecture is now supported - e.g. the NVIDIA Jetson development platform.

9. The NVIDIA CUDA SM_5* (Maxwell) architecture is now supported.

10. An error is now thrown when the user tries to use double precision floating-point numbers on devices with architecture older than SM_13, since these devices do not support double precision.

11. All GeNN helper functions and classes, such as `toString()` and `NNmodel`, are defined in the header files at `genn/lib/include/`, for example `stringUtils.h` and `modelSpec.h`, which should be individually included before the functions and classes may be used. The functions and classes are actually implementated in the static library `genn\lib\lib\genn.lib` (Windows) or `genn/lib/lib/libgenn.a` (Mac, Linux), which must be linked into the final executable if any GeNN functions or classes are used.

12. In the `modelDefinition()` file, only the header file `modelSpec.h` should be included - i.e. not the source file `modelSpec.cc`. This is because the declaration and definition of `NNmodel`, and associated functions, has been separated into `modelSpec.h` and `modelSpec.cc`, respectively. This is to enable NNmodel code to be precompiled separately. *Henceforth, only the header file `modelSpec.h` should be included in model definition files!*

13. In the `modelDefinition()` file, DT is now preferrably defined using `model.setDT(<val>);`, rather than `#define DT <val>`, in order to prevent problems with DT macro redefinition. For backward-compatibility reasons, the old `#define DT <val>` method may still be used, however users are advised to adopt the new method.

14. In preparation for multi-GPU support in GeNN, we have separated out the compilation of generated code from user-side code. This will eventually allow us to optimise and compile different parts of the model with different CUDA flags, depending on the CUDA device chosen to execute that particular part of the model. As such, we have had to use a header file `definitions.h` as the generated code interface, rather than the `runner.cc` file. In practice, this means that *user-side code should include `myModel_CODE/definitions.h`, rather than `myModel_CODE/runner.cc`. Including `runner.cc` will likely result in pages of linking errors at best!*

Developer Side Changes

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

2. The information about device choice is now stored in variables in the namespace `GENN_PREFERENCES`. This includes `chooseDevice`, `optimiseBlockSize`, `optimizeCode`, `debugCode`, `showPtxInfo`, `defaultDevice`. `asGoodAsZero` has also been moved into this namespace.

3. We have also introduced the namespace GENN_FLAGS that contains unsigned int variables that attach names to numeric flags that can be used within GeNN.

4. The definitions of all generated variables and functions such as pullXXXStateFromDevice etc, are now generated into definitions.h. This is useful where one wants to compile separate object files that cannot all include the full definitions in e.g. "runnerGPU.cc". One example where this is useful is the brian2genn interface.

5. A number of feature tests have been added that can be found in the `featureTests` directory. They can be run with the respective `runTests.sh` scripts. The `cleanTests.sh` scripts can be used to remove all generated code after testing.

Improvements

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

Bug fixes:

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

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

Release Notes for GeNN v2.1

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

User Side Changes

1. Block size debugging flag and the asGoodAsZero variables are moved into include/global.h.
2. NGRADSYNAPSES dynamics have changed (See Bug fix #4) and this change is applied to the example projects. If you are using this synapse model, you may want to consider changing model parameters.
3. The delay slots are now such that NO_DELAY is 0 delay slots (previously 1) and 1 means an actual delay of 1 time step.
4. The convenience function convertProbabilityToRandomNumberThreshold(float *, uint64_t *, int) was changed so that it actually converts firing probability/timestep into a threshold value for the GeNN random number generator (as its name always suggested). The previous functionality of converting a *rate* in kHz into a firing threshold number for the GeNN random number generator is now provided with the name convertRateToRandomNumberThreshold(float *, uint64_t *, int)
5. Every model definition function `modelDefinition()` now needs to end with calling `NNmodel::finalize()` for the defined network model. This will lock down the model and prevent any further changes to it by the supported methods. It also triggers necessary analysis of the model structure that should only be performed once. If the `finalize()` function is not called, GeNN will issue an error and exit before code generation.
6. To be more consistent in function naming the `pull\<SYNAPSENAME\>FromDevice` and `push\<SYNAPSENAME\>ToDevice` have been renamed to `pull\<SYNAPSENAME\>StateFromDevice` and `push\<SYNAPSENAME\>StateToDevice`. The old versions are still supported through macro definitions to make the transition easier.
7. New convenience macros are now provided to access the current spike numbers and identities of neurons that spiked. These are called spikeCount_XX and spike_XX where "XX" is the name of the neuron group. They access the values for the current time step even if there are synaptic delays and spikes are stored in circular queues.

8. There is now a pre-defined neuron type "SPIKECOURCE" which is empty and can be used to define PyNN style spike source arrays.
9. The macros FLOAT and DOUBLE were replaced with GENN_FLOAT and GENN_DOUBLE due to name clashes with typedefs in Windows that define FLOAT and DOUBLE.

Developer Side Changes

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

Improvements

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

Bug fixes:

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

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

Release Notes for GeNN v2.0

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

User Side Changes

1. Users are now required to call `initGeNN()` in the model definition function before adding any populations to the neuronal network model.
2. glbscnt is now call glbSpkCnt for consistency with glbSpkEvntCnt.
3. There is no longer a privileged parameter `Epre`. Spike type events are now defined by a code string `spkEvntThreshold`, the same way proper spikes are. The only difference is that Spike type events are specific to a synapse type rather than a neuron type.

4. The function setSynapseG has been deprecated. In a GLOBALG scenario, the variables of a synapse group are set to the initial values provided in the modeldefinition function.

5. Due to the split of synaptic models into weightUpdateModel and postSynModel, the parameter arrays used during model definition need to be carefully split as well so that each side gets the right parameters. For example, previously

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

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

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

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

Note

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

6. Advanced users can now define variables as type scalar when introducing new neuron or synapse types. This will at the code generation stage be translated to the model's floating point type (ftype), float or double. This works for defining variables as well as in all code snippets. Users can also use the expressions SCALAR_MAX and SCALAR_MIN for FLT_MIN, FLT_MAX, DBL_MIN and DBL_MAX, respectively. Corresponding definitions of scalar, SCALAR_MIN and SCALAR_MAX are also available for user-side code whenever the code-generated file runner.cc has been included.

7. The example projects have been re-organized so that wrapper scripts of the generate_run type are now all located together with the models they run instead of in a common tools directory. Generally the structure now is that each example project contains the wrapper script generate_run and a model subdirectory which contains the model description file and the user side code complete with Makefiles for Unix and Windows operating systems. The generated code will be deposited in the model subdirectory in its own modelname_CODE folder. Simulation results will always be deposited in a new sub-folder of the main project directory.

8. The addSynapsePopulation(...) function has now more mandatory parameters relating to the introduction of separate weightupdate models (pre-synaptic models) and postynaptic models. The correct syntax for the addSynapsePopulation(...) can be found with detailed explanations in teh User Manual.

9. We have introduced a simple performance profiling method that users can employ to get an overview over the differential use of time by different kernels. To enable the timers in GeNN generated code, one needs to declare

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

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

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

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

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

in `MBody1.cc`'s `modelDefinition` function and define the macro `TIMING` in `classol_sim.h`

```
#define TIMING
```

This will have the effect that timing information is output into `OUTNAME_output/OUTNAME.timingprofile`.

Developer Side Changes

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

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

Previous | Top | Next

9 User Manual

9.1 Contents

9.2 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

9.3 Defining a network model

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

```
void modelDefinition(ModelSpec &model)
```

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

1. The name of the model must be defined:

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

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

9.3.1 Defining neuron populations

Neuron populations are added using the function

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

where the arguments are:

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

- `NeuronModel::VarValues varInitialisers`: Initial values or initialisation snippets 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.

9.3.2 Defining synapse populations

Synapse populations are added with the function

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

where the arguments are

- `WeightUpdateModel`: Template parameter specifying the type of weight update model. These should be derived off WeightUpdateModels::Base and can either be one of the standard models or user-defined (see Weight update models).
- `PostsynapticModel`: Template parameter specifying the type of postsynaptic integration model. These should be derived off PostsynapticModels::Base and can either be one of the standard models or user-defined (see Postsynaptic integration methods).
- `const string &name`: The name of the synapse population
- `unsigned int mType`: How the synaptic matrix is stored. See Synaptic matrix types for available options.
- `unsigned int delay`: Homogeneous (axonal) delay for synapse population (in terms of the simulation time step `DT`).
- `const string preName`: Name of the (existing!) pre-synaptic neuron population.
- `const string postName`: Name of the (existing!) post-synaptic neuron population.
- `WeightUpdateModel::ParamValues weightParamValues`: The parameter values (common to all synapses of the population) for the weight update model.
- `WeightUpdateModel::VarValues weightVarInitialisers`: Initial values or initialisation snippets for the weight update model's state variables
- `WeightUpdateModel::PreVarValues weightPreVarInitialisers`: Initial values or initialisation snippets for the weight update model's presynaptic state variables
- `WeightUpdateModel::PostVarValues weightPostVarInitialisers`: Initial values or initialisation snippets for the weight update model's postsynaptic state variables
- `PostsynapticModel::ParamValues postsynapticParamValues`: The parameter values (common to all postsynaptic neurons) for the postsynaptic model.
- `PostsynapticModel::VarValues postsynapticVarInitialisers`: Initial values or initialisation snippets for variables for the postsynaptic model's state variables
- `InitSparseConnectivitySnippet::Init connectivityInitialiser`: Optional argument, specifying the initialisation snippet for synapse population's sparse connectivity (see Sparse connectivity initialisation).

The ModelSpec::addSynapsePopulation() function returns a pointer to the newly created SynapseGroup object which can be further configured, namely with:

- SynapseGroup::setMaxConnections() and SynapseGroup::setMaxSourceConnections() to configure the maximum number of rows and columns respectively allowed in the synaptic matrix - this can improve performance and reduce memory usage when using SynapseMatrixConnectivity::SPARSE connectivity (see Synaptic matrix types).

  Note

  When using a sparse connectivity initialisation snippet, these values are set automatically.

- SynapseGroup::setMaxDendriticDelayTimesteps() sets the maximum dendritic delay (in terms of the simulation time step `DT`) allowed for synapses in this population. No values larger than this should be passed to the delay parameter of the `addToDenDelay` function in user code (see Defining a new weight update model).

- SynapseGroup::setSpanType() sets how incoming spike processing is parallelised for this synapse group. The default SynapseGroup::SpanType::POSTSYNAPTIC is nearly always the best option, but SynapseGroup::SpanType::PRESYNAPTIC may perform better when there are large numbers of spikes every timestep or very few postsynaptic neurons.

Note

If the synapse matrix uses one of the "GLOBALG" types then the global value of the synapse parameters are taken from the initial value provided in `weightVarInitialisers` therefore these must be constant rather than sampled from a distribution etc.

Previous | Top | Next

9.4 Neuron models

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

- NeuronModels::RulkovMap
- NeuronModels::Izhikevich
- NeuronModels::IzhikevichVariable
- NeuronModels::LIF
- NeuronModels::SpikeSource
- NeuronModels::PoissonNew
- NeuronModels::TraubMiles
- NeuronModels::TraubMilesFast
- NeuronModels::TraubMilesAlt
- NeuronModels::TraubMilesNStep

9.4.1 Defining your own neuron type

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

- DECLARE_MODEL(TYPE, NUM_PARAMS, NUM_VARS): declared the boilerplate code required for the model e.g. the correct specialisations of NewModels::ValueBase used to wrap the neuron model parameters and values.

- SET_SIM_CODE(SIM_CODE): where SIM_CODE contains the code for executing the integration of the model for one time stepWithin this code string, variables need to be referred to by $(NAME), where NA↩ ME is the name of the variable as defined in the vector varNames. The code may refer to the predefined primitives `DT` for the time step size and for the total incoming synaptic current. It can also refer to a unique ID (within the population) using .
- SET_THRESHOLD_CONDITION_CODE(THRESHOLD_CONDITION_CODE) defines the condition for true spike detection.
- SET_PARAM_NAMES() defines the names of the model parameters. If defined as `NAME` here, they can then be referenced as $(NAME) in the code string. The length of this list should match the NUM_PARAM specified in DECLARE_MODEL. Parameters are assumed to be always of type double.
- SET_VARS() defines the names and type strings (e.g. "float", "double", etc) of the neuron state variables. The type string "scalar" can be used for variables which should be implemented using the precision set globally for the model with ModelSpec::setPrecision. The variables defined here as `NAME` can then be used in the syntax $(NAME) in the code string.
- SET_NEEDS_AUTO_REFRACTORY() defines whether the neuron should include an automatic refractory period to prevent it emitting spikes in successive timesteps.

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

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

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

    SET_THRESHOLD_CONDITION_CODE("$(V) >= 1.0");

    SET_PARAM_NAMES({"tau"});

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

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

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

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

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

    SET_THRESHOLD_CONDITION_CODE("$(V) >= 1.0");

    SET_PARAM_NAMES({"tau"});

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

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

GeNN provides several additional features that might be useful when defining more complex neuron models.

9.4.1.1 Support code

Support code enables a code block to be defined that contains supporting code that will be utilized in multiple pieces of user code. Typically, these are functions that are needed in the sim code or threshold condition code. If possible, these should be defined as `__host__ __device__` functions so that both GPU and CPU versions of GeNN code have an appropriate support code function available. The support code is protected with a namespace so that it is exclusively available for the neuron population whose neurons define it. Support code is added to a model using the SET_SUPPORT_CODE() macro, for example:

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

9.4.1.2 Extra global parameters

Extra global parameters are parameters common to all neurons in the population. However, unlike the standard neuron parameters, they can be varied at runtime meaning they could, for example, be used to provide a global reward signal. These parameters are defined by using the SET_EXTRA_GLOBAL_PARAMS() macro to specify a list of variable names and type strings (like the SET_VARS() macro). For example:

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

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

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

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

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

9.4.1.3 Additional input variables

Normally, neuron models receive the linear sum of the inputs coming from all of their synaptic inputs through the $(inSyn) variable. However neuron models can define additional input variables - allowing input from different synaptic inputs to be combined non-linearly. For example, if we wanted our leaky integrator to operate on the the product of two input currents, it could be defined as follows:

```
SET_ADDITIONAL_INPUT_VARS({{"Isyn2", "scalar", 1.0}});
SET_SIM_CODE("const scalar input = $(Isyn) * $(Isyn2);\n"
             "$(V) = input - $(ExpTC)*(input - $(V));");
```

Where the SET_ADDITIONAL_INPUT_VARS() macro defines the name, type and its initial value before postsynaptic inputs are applyed (see section Postsynaptic integration methods for more details).

9.4.1.4 Random number generation

Many neuron models have probabilistic terms, for example a source of noise or a probabilistic spiking mechanism. In GeNN this can be implemented by using the following functions in blocks of model code:

- `$(gennrand_uniform)` returns a number drawn uniformly from the interval $[0.0, 1.0]$
- `$(gennrand_normal)` returns a number drawn from a normal distribution with a mean of 0 and a standard deviation of 1.
- `$(gennrand_exponential)` returns a number drawn from an exponential distribution with $\lambda = 1$.
- `$(gennrand_log_normal, MEAN, STDDEV)` returns a number drawn from a log-normal distribution with the specified mean and standard deviation.
- `$(gennrand_gamma, ALPHA)` returns a number drawn from a gamma distribution with the specified shape.

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

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

Previous | Top | Next

9.5 Weight update models

Currently 4 predefined weight update models are available:

- WeightUpdateModels::StaticPulse
- WeightUpdateModels::StaticPulseDendriticDelay
- WeightUpdateModels::StaticGraded
- WeightUpdateModels::PiecewiseSTDP

For more details about these built-in synapse models, see [3].

9.5.1 Defining a new weight update model

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

- SET_DERIVED_PARAMS(), SET_PARAM_NAMES(), SET_VARS() and SET_EXTRA_GLOBAL_PARAMS() perform the same roles as they do in the neuron models discussed in Defining your own neuron type.
- DECLARE_WEIGHT_UPDATE_MODEL(TYPE, NUM_PARAMS, NUM_VARS, NUM_PRE_VARS, NUM_POST_VARS) is an extended version of `DECLARE_MODEL()` which declares the boilerplate code required for a weight update model with pre and postsynaptic as well as per-synapse state variables.
- SET_PRE_VARS() and SET_POST_VARS() define state variables associated with pre or postsynaptic neurons rather than synapses. These are typically used to efficiently implement *trace* variables for use in STDP learning rules [2]. Like other state variables, variables defined here as `NAME` can be accessed in weight update model code strings using the $(NAME) syntax.
- SET_SIM_CODE(SIM_CODE): defines the simulation code that is used when a true spike is detected. The update is performed only in timesteps after a neuron in the presynaptic population has fulfilled its threshold detection condition. Typically, spikes lead to update of synaptic variables that then lead to the activation of input into the post-synaptic neuron. Most of the time these inputs add linearly at the post-synaptic neuron. This is assumed in GeNN and the term to be added to the activation of the post-synaptic neuron should be applied using the the $(addToInSyn, weight) function. For example

  ```
  SET_SIM_CODE(
      "$(addToInSyn, $(inc));\n"
  ```

  where "inc" is the increment of the synaptic input to a post-synaptic neuron for each pre-synaptic spike. The simulation code also typically contains updates to the internal synapse variables that may have contributed to . For an example, see WeightUpdateModels::StaticPulse for a simple synapse update model and WeightUpdateModels::PiecewiseSTDP for a more complicated model that uses STDP. To apply input to the post-synaptic neuron with a dendritic (i.e. between the synapse and the postsynaptic neuron) delay you can instead use the $(addToInSynDelay, weight, delay) function. For example

  ```
  SET_SIM_CODE(
      "$(addToInSynDelay, $(inc), $(delay));");
  ```

  where, once again, `inc` is the magnitude of the input step to apply and `delay` is the length of the dendritic delay in timesteps. By implementing `delay` as a weight update model variable, heterogeneous synaptic delays can be implemented. For an example, see WeightUpdateModels::StaticPulseDendriticDelay for a simple synapse update model with heterogeneous dendritic delays.

Note

When using dendritic delays, the **maximum** dendritic delay for a synapse populations must be specified using the `SynapseGroup::setMaxDendriticDelayTimesteps()` function.

- SET_EVENT_THRESHOLD_CONDITION_CODE(EVENT_THRESHOLD_CONDITION_CODE) defines a condition for a synaptic event. This typically involves the pre-synaptic variables, e.g. the membrane potential:

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

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

- SET_EVENT_CODE(EVENT_CODE) defines the code that is used when the event threshold condition is met (as set using the SET_EVENT_THRESHOLD_CONDITION_CODE() macro).

- SET_LEARN_POST_CODE(LEARN_POST_CODE) defines the code which is used in the learnSynapses↩ Post kernel/function, which performs updates to synapses that are triggered by post-synaptic spikes. This is typically used in STDP-like models e.g. WeightUpdateModels::PiecewiseSTDP.

- SET_SYNAPSE_DYNAMICS_CODE(SYNAPSE_DYNAMICS_CODE) defines code that is run for each synapse, each timestep i.e. unlike the others it is not event driven. This can be used where synapses have internal variables and dynamics that are described in continuous time, e.g. by ODEs. However using this mechanism is typically computationally very costly because of the large number of synapses in a typical network. By using the $(addtoinsyn), $(updatelinsyn) and $(addToDenDelay) mechanisms discussed in the context of SET_SIM_CODE(), the synapse dynamics can also be used to implement continuous synapses for rate-based models.

- SET_PRE_SPIKE_CODE() and SET_POST_SPIKE_CODE() define code that is called whenever there is a pre or postsynaptic spike. Typically these code strings are used to update any pre or postsynaptic state variables.

- SET_NEEDS_PRE_SPIKE_TIME(PRE_SPIKE_TIME_REQUIRED) and SET_NEEDS_POST_SPIKE_TI↩ ME(POST_SPIKE_TIME_REQUIRED) define whether the weight update needs to know the times of the spikes emitted from the pre and postsynaptic populations. For example an STDP rule would be likely to require:

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

All code snippets, aside from those defined with `SET_PRE_SPIKE_CODE()` and `SET_POST_SPIKE_COD↩ E()`, can be used to manipulate any synapse variable and so learning rules can combine both time-drive and event-driven processes.

Previous | Top | Next

9.6 Postsynaptic integration methods

There are currently 3 built-in postsynaptic integration methods:

- PostsynapticModels::ExpCurr
- PostsynapticModels::ExpCond
- PostsynapticModels::DeltaCurr

9.6.1 Defining a new postsynaptic model

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

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

- DECLARE_MODEL(TYPE, NUM_PARAMS, NUM_VARS), SET_DERIVED_PARAMS(), SET_PARAM_NAMES(), SET_VARS() perform the same roles as they do in the neuron models discussed in Defining your own neuron type.
- SET_DECAY_CODE(DECAY_CODE) defines the code which provides the continuous time dynamics for the summed presynaptic inputs to the postsynaptic neuron. This usually consists of some kind of decay function.
- SET_APPLY_INPUT_CODE(APPLY_INPUT_CODE) defines the code specifying the conversion from synaptic inputs to a postsynaptic neuron input current. e.g. for a conductance model:

  ```
  SET_APPLY_INPUT_CODE("$(Isyn) += $(inSyn) * ($(E) - $(V))");
  ```

  where $(E) is a postsynaptic model parameter specifying reversal potential and $(V) is the variable containing the postsynaptic neuron's membrane potential. As discussed in Built-in Variables in GeNN, $(Isyn) is the built in variable used to sum neuron input. However additional input variables can be added to a neuron model using the SET_ADDITIONAL_INPUT_VARS() macro (see Defining your own neuron type for more details).

Previous | Top | Next

9.7 Current source models

There is a number of predefined models which can be used with the ModelSpec::addCurrentSource function:

- CurrentSourceModels::DC
- CurrentSourceModels::GaussianNoise

9.7.1 Defining your own current source model

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

- DECLARE_MODEL(TYPE, NUM_PARAMS, NUM_VARS), SET_DERIVED_PARAMS(), SET_PARAM_NAMES(), SET_VARS() perform the same roles as they do in the neuron models discussed in Defining your own neuron type.
- SET_INJECTION_CODE(INJECTION_CODE): where INJECTION_CODE contains the code for injecting current into the neuron every simulation timestep. The $(injectCurrent,) function is used to inject current.

For example, using these macros, we can define a uniformly distributed noisy current source:

```
class UniformNoise : public CurrentSourceModels::Base
{
public:
    DECLARE_MODEL(UniformNoise, 1, 0);

    SET_SIM_CODE("$(injectCurrent, $(gennrand_uniform) * $(magnitude));");

    SET_PARAM_NAMES({"magnitude"});
};
```

Previous | Top | Next

9.8 Synaptic matrix types

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

- SynapseMatrixConnectivity::DENSE stores synaptic matrices as a dense matrix. Large dense matrices require a large amount of memory and if they contain a lot of zeros it may be inefficient.
- SynapseMatrixConnectivity::SPARSE stores synaptic matrices in a(padded) 'ragged array' format. In general, this is less efficient to traverse using a GPU than the dense matrix format but does result in significant memory savings for large matrices. Ragged matrix connectivity is stored using several variables whose names, like state variables, have the name of the synapse population appended to them:
    1. `const unsigned int maxRowLength`: a constant set via the `SynapseGroup::setMaxConnections` method which specifies the maximum number of connections in any given row (this is the width the structure is padded to).
    2. `unsigned int *rowLength` (sized to number of presynaptic neurons): actual length of the row of connections associated with each presynaptic neuron
    3. `unsigned int *ind` (sized to `maxRowLength * number of presynaptic neurons`): Indices of corresponding postsynaptic neurons concatenated for each presynaptic neuron. 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 only to the 0th neuron. The struct RaggedProjection should have these members, with indexing from 0 (where X represents a padding value):

       ```
       maxRowLength = 2
       ind = [1 2 0 X]
       rowLength = [2 1]
       ```

       Weight update model variables associated with the sparsely connected synaptic population will be kept in an array using the same indexing as ind. For example, a variable caled `g` will be kept in an array such as: `g`=[g_Pre0-Post1 g_pre0-post2 g_pre1-post0 X]
- SynapseMatrixConnectivity::BITMASK is an alternative sparse matrix implementation where which synapses within the matrix are present is specified as a binary array (see Insect olfaction model). This structure is somewhat less efficient than the `SynapseMatrixConnectivity::SPARSE` and `SynapseMatrixConnectivity::RAGGED` formats and doesn't allow individual weights per synapse. However it does require the smallest amount of GPU memory for large networks.

Furthermore the SynapseMatrixWeight defines how

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

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

- SynapseMatrixType::SPARSE_GLOBALG
- SynapseMatrixType::SPARSE_GLOBALG_INDIVIDUAL_PSM

- SynapseMatrixType::SPARSE_INDIVIDUALG
- SynapseMatrixType::DENSE_GLOBALG
- SynapseMatrixType::DENSE_GLOBALG_INDIVIDUAL_PSM
- SynapseMatrixType::DENSE_INDIVIDUALG
- SynapseMatrixType::BITMASK_GLOBALG
- SynapseMatrixType::BITMASK_GLOBALG_INDIVIDUAL_PSM

Previous | Top | Next

9.9 Variable initialisation

Neuron, weight update and postsynaptic models all have state variables which GeNN can automatically initialise.

Previously we have shown variables being initialised to constant values such as:

```
NeuronModels::TraubMiles::VarValues ini(
    0.0529324,      // 1 - prob. for Na channel activation m
    ...
);
```

state variables can also be left *uninitialised* leaving it up to the user code to initialise them between the calls to `initialize()` and `initializeSparse()`:

```
NeuronModels::TraubMiles::VarValues ini(
    uninitialisedVar(),      // 1 - prob. for Na channel activation m
    ...
);
```

or initialised using one of a number of predefined *variable initialisation snippets*:

- InitVarSnippet::Uniform
- InitVarSnippet::Normal
- InitVarSnippet::Exponential
- InitVarSnippet::Gamma

For example, to initialise a parameter using values drawn from the normal distribution:

```
InitVarSnippet::Normal::ParamValues params(
    0.05,   // 0 - mean
    0.01);  // 1 - standard deviation

NeuronModels::TraubMiles::VarValues ini(
    initVar<InitVarSnippet::Normal>(params),      // 1 - prob. for Na channel activation m
    ...
);
```

9.9.1 Defining a new variable initialisation snippet

Similarly to neuron, weight update and postsynaptic models, new variable initialisation snippets can be created by simply defining a class in the model description. For example, when initialising excitatory (positive) synaptic weights with a normal distribution they should be clipped at 0 so the long tail of the normal distribution doesn't result in negative weights. This could be implemented using the following variable initialisation snippet which redraws until samples are within the desired bounds:

```
class NormalPositive : public InitVarSnippet::Base
{
public:
    DECLARE_SNIPPET(NormalPositive, 2);

    SET_CODE(
        "scalar normal;"
        "do\n"
        "{\n"
        "    normal = $(mean) + ($(gennrand_normal) * $(sd));\n"
        "} while (normal < 0.0);\n"
        "$(value) = normal;\n");

    SET_PARAM_NAMES({"mean", "sd"});
};
IMPLEMENT_SNIPPET(NormalPositive);
```

Within the snippet of code specified using the `SET_CODE()` macro, when initialisising neuron and postaynaptic model state variables , the $(id) variable can be used to access the id of the neuron being initialised. Similarly, when initialising weight update model state variables, the $(id_pre) and $(id_post) variables can used to access the ids of the pre and postsynaptic neurons connected by the synapse being initialised.

9.9.2 Variable locations

Once you have defined **how** your variables are going to be initialised you need to configure **where** they will be allocated. By default memory is allocated for variables on both the GPU and the host. However, the following alternative 'variable locations' are available:

- VarLocation::DEVICE - Variables are only allocated on the GPU, saving memory but meaning that they can't easily be copied to the host - best for internal state variables.
- VarLocation::HOST_DEVICE - Variables are allocated on both the GPU and the host - the default.
- VarLocation::HOST_DEVICE_ZERO_COPY - Variables are allocated as 'zero-copy' memory accessible to the host and GPU - useful on devices such as Jetson TX1 where physical memory is shared between the GPU and CPU.

Note

'Zero copy' memory is only supported on newer embedded systems such as the Jetson TX1 where there is no physical seperation between GPU and host memory and thus the same block of memory can be shared between them.

These modes can be set as a model default using `ModelSpec::setDefaultVarLocation` or on a per-variable basis using one of the following functions:

- NeuronGroup::setSpikeLocation
- NeuronGroup::setSpikeEventLocation
- NeuronGroup::setSpikeTimeLocation
- NeuronGroup::setVarLocation
- SynapseGroup::setWUVarLocation
- SynapseGroup::setWUPreVarLocation
- SynapseGroup::setWUPostVarLocation
- SynapseGroup::setPSVarLocation
- SynapseGroup::setInSynVarLocation

Previous | Top | Next

9.10 Sparse connectivity initialisation

Synaptic connectivity implemented using SynapseMatrixConnectivity::SPARSE and SynapseMatrixConnectivity::↩ BITMASK can be automatically initialised.

This can be done using one of a number of predefined *sparse connectivity initialisation snippets*:

- InitSparseConnectivitySnippet::OneToOne
- InitSparseConnectivitySnippet::FixedProbability
- InitSparseConnectivitySnippet::FixedProbabilityNoAutapse

For example, to initialise synaptic connectivity with a 10% connection probability (allowing connections between neurons with the same id):

```
InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1);

model.addSynapsePopulation<...>(
        ...
        initConnectivity<InitSparseConnectivitySnippet::FixedProbability>(fixedProb));
```

9.10.1 Defining a new sparse connectivity snippet

Similarly to variable initialisation snippets, sparse connectivity initialisation snippets can be created by simply defining a class in the model description.

For example, the following sparse connectivity initialisation snippet could be used to initialise a 'ring' of connectivity where each neuron is connected to a number of subsequent neurons specified using the `numNeighbours` parameter:

```
class Ring : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(Ring, 1);

    SET_ROW_BUILD_STATE_VARS({{"offset", {"unsigned int", 1}}}});
    SET_ROW_BUILD_CODE(
        "const unsigned int target = ($(id_pre) + offset) % $(num_post);\n"
        "$(addSynapse, target);\n"
        "offset++;\n"
        "if(offset > (unsigned int)$(numNeighbours)) {\n"
        "   $(endRow);\n"
        "}\n");

    SET_PARAM_NAMES({"numNeighbours"});
    SET_CALC_MAX_ROW_LENGTH_FUNC(
        [](unsigned int numPre, unsigned int numPost, const std::vector<double> &pars)
        {
            return (unsigned int)pars[0];
        });
    SET_CALC_MAX_COL_LENGTH_FUNC(
        [](unsigned int numPre, unsigned int numPost, const std::vector<double> &pars)
        {
            return (unsigned int)pars[0];
        });
};
IMPLEMENT_SNIPPET(Ring);
```

Each *row* of sparse connectivity is initialised independantly by running the snippet of code specified using the `S↩ ET_ROW_BUILD_CODE()` macro within a loop. The $(num_post) variable can be used to access the number of neurons in the postsynaptic population and the $(id_pre) variable can be used to access the index of the presynaptic neuron associated with the row being generated. The `SET_ROW_BUILD_STATE_VARS()` macro can be used to initialise state variables outside of the loop - in this case `offset` which is used to count the number of synapses created in each row. Synapses are added to the row using the $(addSynapse, target) function and iteration is stopped using the $(endRow) function. To avoid having to manually call SynapseGroup::setMaxConnections and SynapseGroup::setMaxSourceConnections, sparse connectivity snippets can also provide code to calculate the maximum row and column lengths this connectivity will result in using the SET_CALC_MAX_ROW_LENGTH_F↩ UNC() and SET_CALC_MAX_COL_LENGTH_FUNC() macros. Alternatively, if the maximum row or column length is constant, the `SET_MAX_ROW_LENGTH()` and `SET_MAX_COL_LENGTH()` shorthand macros can be used.

9.10.2 Sparse connectivity locations

Once you have defined **how** sparse connectivity is going to be initialised, similarly to variables, you can control **where** it is allocated. This is controlled using the same `VarLocations` options described in section Variable locations and can either be set using the model default specifiued with `ModelSpec::setDefaultSparseConnectivityLocation` or on a per-synapse group basis using `SynapseGroup::setSparseConnectivityLocation`.

Previous | Top | Next

10 Tutorial 1

In this tutorial we will go through step by step instructions how to create and run your first GeNN simulation from scratch.

10.1 The Model Definition

In this tutorial we will use a pre-defined Hodgkin-Huxley neuron model (NeuronModels::TraubMiles) and create a simulation consisting of ten such neurons without any synaptic connections. We will run this simulation on a GPU and save the results - firstly to stdout and then to file.

The first step is to write a model definition function in a model definition file. Create a new directory and, within that, create a new empty file called `tenHHModel.cc` using your favourite text 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 `ModelSpec`. Type in your `tenHHModel.cc` file:

```
// Model definintion file tenHHModel.cc

#include "modelSpec.h"

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

Two standard elements to the ‘modelDefinition function are setting the simulation step size and setting the name of the model:

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

Note

With this we have fixed the integration time step to `0.1` in the usual time units. The typical units in GeNN are `ms`, `mV`, `nF`, and μ`S`. Therefore, this defines `DT= 0.1 ms`.

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

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

We first create the parameter and initial variable arrays,

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

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

Note

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

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

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

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(ModelSpec &model)
{
    // definition of tenHHModel
    model.setDT(0.1);
    model.setName("tenHHModel");

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

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

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

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.

10.2 Building the model

To use GeNN to build your model description into simulation code, use a terminal to navigate to the directory containing your `tenHHModel.cc` file and, on Linux or Mac, type:

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

Alternatively, on Windows, type:

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

If you don't have an NVIDIA GPU and are running GeNN in CPU_ONLY mode, you can invoke `genn-buildmodel` with a `-c` option so, on Linux or Mac:

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

or on Windows:

```
>> genn-buildmodel.bat -c tenHHModel.cc
```

If GeNN has been added to your path and `CUDA_PATH` is correctly configured, you should see some compile output ending in `Model build complete ....`

10.3 User Code

GeNN will now have generated the code to simulate the model for one timestep using a function `stepTime()`. To make use of this code, we need to define a minimal C/C++ main function. 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. Open a new empty file `tenHHSimulation.cc` in an editor and type

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

int main()
{
    allocateMem();
    initialize();
    return 0;
}
```

This boiler plate code includes the header file for the generated code `definitions.h` in the subdirectory `ten↩ HHModel_CODE` where GeNN deposits all generated code (this corresponds to the name passed to the `Model↩ Spec::setName` function). Calling `allocateMem()` allocates the memory structures for all neuron variables and `initialize()` launches a GPU kernel which initialise all state variables to their initial values. Now we can use the generated code to integrate the neuron equations provided by GeNN for 1000ms. To do so, we add after `initialize();`

Note

The `t` variable is provided by GeNN to keep track of the current simulation time in milliseconds.

```
while (t < 1000.0f) {
    stepTime();
}
```

and we need to copy the result back to the host before outputting it to stdout (this will do nothing if you are running the model on a CPU),

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

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

Note

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

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

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

int main()
{
    allocateMem();
    initialize();

    while (t < 1000.0f) {
        stepTime();
    }
    pullPop1StateFromDevice();

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

10.4 Building the simulator (Linux or Mac)

On Linux and Mac, GeNN simulations are typically built using a simple Makefile which can be generated with the following command:

```
genn-create-user-project.sh tennHHModel tenHHSimulation.cc
```

This defines that the model is named tennHHModel and the simulation code is given in the file tenHH←Simulation.cc that we completed above. Now type

```
make
```

10.5 Building the simulator (Windows)

So that projects can be easily debugged within the Visual Studio IDE (see section Debugging suggestions for more details), Windows projects are built using an MSBuild script typically with the same title as the final executable. A suitable solution and project can be generated automatically with the following command:

```
genn-create-user-project.bat tennHHModel tenHHSimulation.cc
```

his defines that the model is named tennHHModel and the simulation code is given in the file tenHH←Simulation.cc that we completed above. Now type

```
msbuild tennHHModel.sln /p:Configuration=Release /t:tennHHModel
```

10.6 Running the Simulation

You can now execute your newly-built simulator on Linux or Mac with

```
./tennHHModel
```

Or on Windows with

```
tennHHModel_Release
```

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
```

10.7 Reading

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

```
std::ofstream os("tenHH_output.V.dat");
while (t < 1000.0f) {
    stepTime();

    pullVPop1FromDevice();

    os << t << " ";
    for (int j= 0; j < 10; j++) {
        os << VPop1[j] << " ";
    }
    os << std::endl;
}
os.close();
```

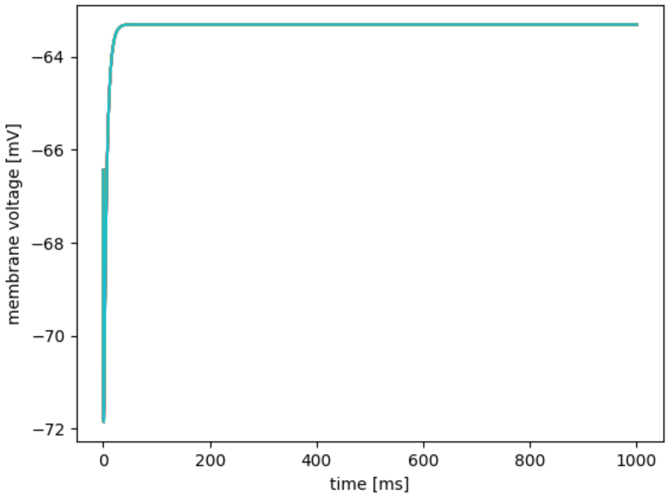
Note

t is a global variable updated by the GeNN code to keep track of elapsed simulation time in ms.
we switched from using `pullPop1StateFromDevice()` to `pullVPop1FromDevice()` as we are now only interested in the membrane voltage of the neuron.

You will also need to add:

```
#include <fstream>
```

to the top of tenHHSimulation.cc. After building the model; and building and running the simulator as described above there should be a file `tenHH_output.V.dat` in the same directory. If you plot column one (time) against the subsequent 10 columns (voltage of the 10 neurons), you should observe dynamics like this:



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

Previous | Top | Next

11 Tutorial 2

In this tutorial we will learn to add synapsePopulations to connect neurons in neuron groups to each other with synaptic 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 the `tenHHModel.cc` to `tenHHRingModel.cc` and `tenHHSimulation.cc` to `tenHHRingSimulation.cc`, e.g. on Linux or Mac:

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

Finally, to reduce confusion we should rename the model itself. Open `tenHHRingModel.cc`, change the model name inside,

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

11.1 Defining the Detailed Synaptic Connections

We want to connect our ten neurons into a ring where each neuron connects to its neighbours. In order to initialise this connectivity we need to add a sparse connectivity initialisation snippet at the top of `tenHHRingModel.cc`:

```
class Ring : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(Ring, 0);
    SET_ROW_BUILD_CODE(
        "$(addSynapse, ($(id_pre) + 1) % $(num_post));\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(Ring);
```

The SET_ROW_BUILD_CODE code string will be called to generate each row of the synaptic matrix (connections coming from a single presynaptic neuron) and, in this case, each row consists of a single synapses from the presynaptic neuron $(id_pre) to $(id_pre) + 1 (the modulus operator is used to ensure that the final connection between neuron 9 and 0 is made correctly). In order to allow GeNN to better optimise the generated code we also provide a maximum row length. In this case each row always contains only one synapse but, when more complex connectivity is used, the number of neurons in the pre and postsynaptic population as well as any parameters used to configure the snippet can be accessed from this function.

Note

When defining GeNN code strings, the $(VariableName) syntax is used to refer to variables provided by GeNN and the $(FunctionName, Parameter1,...) syntax is used to call functions provided by GeNN.

11.2 Adding Synaptic connections

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

```
WeightUpdateModels::StaticPulse::VarValues s_ini(
    -0.2); // 0 - g: the synaptic conductance value

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

Note

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

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

```
model.addSynapsePopulation<WeightUpdateModels::StaticPulse
      , PostsynapticModels::ExpCond>(
    "Pop1self", SynapseMatrixType::SPARSE_GLOBALG, 10,
    "Pop1", "Pop1",
    {}, s_ini,
    ps_p, {},
    initConnectivity<Ring>());
```

The addSynapsePopulation parameters are

- WeightUpdateModel: template parameter specifying the type of weight update model (derived from Weight↩ UpdateModels::Base).
- PostsynapticModel: template parameter specifying the type of postsynaptic model (derived from PostsynapticModels::Base).
- name string containing unique name of synapse population.
- mtype how the synaptic matrix associated with this synapse population should be represented. Here SynapseMatrixType::SPARSE_GLOBALG means that there will be sparse connectivity and each connection will have the same weight (-0.2 as specified previously).
- delayStep integer specifying number of timesteps of propagation delay that spikes travelling through this synapses population should incur (or NO_DELAY for none)
- src string specifying name of presynaptic (source) population
- trg string specifying name of postsynaptic (target) population
- weightParamValues parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.

- weightVarInitialisers initial values or initialisation snippets for the weight update model's state variables wrapped in a WeightUpdateModel::VarValues object.
- postsynapticParamValues parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.
- postsynapticVarInitialisers initial values or initialisation snippets for the postsynaptic model wrapped in PostsynapticModel::VarValues object.
- connectivityInitialiser snippet and any paramaters (in this case there are none) used to initialise the synapse population's sparse connectivity.

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

```
// Model definition file tenHHRing.cc
#include "modelSpec.h"

class Ring : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(Ring, 0);
    SET_ROW_BUILD_CODE(
        "$(addSynapse, ($(id_pre) + 1) % $(num_post));\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(Ring);

void modelDefinition(ModelSpec &model)
{
    // definition of tenHHRing
    model.setDT(0.1);
    model.setName("tenHHRing");

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

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

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

    WeightUpdateModels::StaticPulse::VarValues s_ini(
         -0.2); // 0 - g: the synaptic conductance value

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

    model.addSynapsePopulation<
      WeightUpdateModels::StaticPulse,
      PostsynapticModels::ExpCond>(
        "Pop1self", SynapseMatrixType::SPARSE_GLOBALG, 100,
        "Pop1", "Pop1",
        {}, s_ini,
        ps_p, {},
        initConnectivity<Ring>());
}
```

We can now build our new model:

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

Note

Again, if you don't have an NVIDIA GPU and are running GeNN in CPU_ONLY mode, you can instead build with the -c option as described in Tutorial 1.

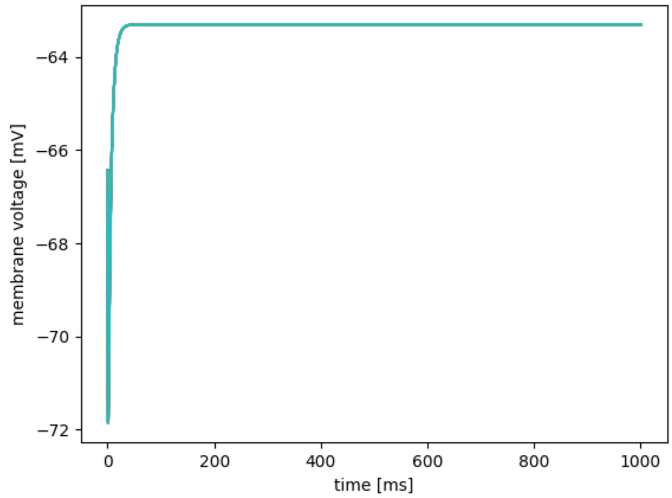
Now we can open the tenHHRingSimulation.cc file and update the file name of the model includes to match the name we set previously:

```
// tenHHRingModel simulation code
#include "tenHHRing_CODE/definitions.h"
```

Additionally, we need to add a call to a second initialisation function to main() after we call initialize():

```
initializeSparse();
```

This initializes any variables associated with the sparse connectivity we have added (and will also copy any manually initialised variables to the GPU). Then, after using the genn-create-user-project tool to create a new project with a model name of tenHHRing and using tenHHRingSimulation.cc rather than tenHHSimulation.cc, we can build and run our new simulator in the same way we did in Tutorial 1. However, even after all our hard work, if we plot the content of the first column against the subsequent 10 columns of tenHHexample.V.dat it looks very similar to the plot we obtained at the end of Tutorial 1.



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

11.3 Providing initial stimuli

We can use a NeuronModels::SpikeSource to inject an initial spike into the first neuron in the ring during the first timestep to start spikes propagating. Firstly we need to define another sparse connectivity initialisation snippet at the top of tenHHRingModel.cc which simply creates a single synapse on the first row of the synaptic matrix:

```
class FirstToFirst : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(FirstToFirst, 0);
    SET_ROW_BUILD_CODE(
        "if($(id_pre) == 0) {\n"
        "   $(addSynapse, $(id_pre));\n"
        "}\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(FirstToFirst);
```

We then need to add it to the network by adding the following to the end of the `modelDefinition(...)` function:

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

and finally inject a spike in the first timestep (in the same way that the `t` variable is provided by GeNN to keep track of the current simulation time in milliseconds, `iT` is provided to keep track of it in timesteps):

```
if(iT == 0) {
    spikeCount_Stim = 1;
    spike_Stim[0] = 0;
    pushStimCurrentSpikesToDevice();
}
```

Note

`spike_Stim[n]` is used to specify the indices of the neurons in population `Stim` spikes which should emit spikes where $n \in [0, \text{spikeCount\_Stim})$.

At this point our user code `tenHHRingModel.cc` should look like this

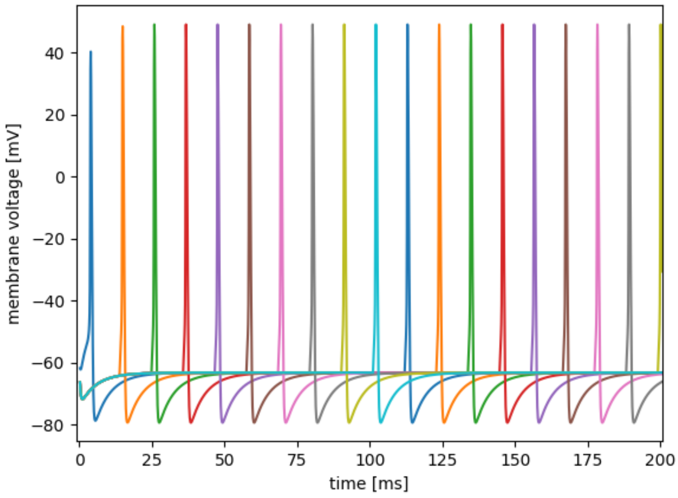
```
// Model definintion file tenHHRing.cc
#include "modelSpec.h"

class Ring : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(Ring, 0);
    SET_ROW_BUILD_CODE(
        "$(addSynapse, ($(id_pre) + 1) % $(num_post));\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(Ring);

class FirstToFirst : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(FirstToFirst, 0);
    SET_ROW_BUILD_CODE(
        "if($(id_pre) == 0) {\n"
        "    $(addSynapse, $(id_pre));\n"
        "}\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(FirstToFirst);

void modelDefinition(ModelSpec &model)
{
    // definition of tenHHRing
    model.setDT(0.1);
    model.setName("tenHHRing");

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

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

```
    model.addNeuronPopulation<NeuronModels::TraubMiles>("Pop1",
      10, p, ini);
    model.addNeuronPopulation<NeuronModels::SpikeSource>("Stim"
      , 1, {}, {});

    WeightUpdateModels::StaticPulse::VarValues s_ini(
         -0.2); // 0 - g: the synaptic conductance value

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

    model.addSynapsePopulation<
      WeightUpdateModels::StaticPulse,
      PostsynapticModels::ExpCond>(
        "Pop1self", SynapseMatrixType::SPARSE_GLOBALG, 100,
        "Pop1", "Pop1",
        {}, s_ini,
        ps_p, {},
        initConnectivity<Ring>());

    model.addSynapsePopulation<
      WeightUpdateModels::StaticPulse,
      PostsynapticModels::ExpCond>(
        "StimPop1", SynapseMatrixType::SPARSE_GLOBALG,
      NO_DELAY,
        "Stim", "Pop1",
        {}, s_ini,
        ps_p, {},
        initConnectivity<FirstToFirst>());
}
```

and `tenHHRingSimulation.cc`‘ should look like this:

```
// Standard C++ includes
#include <fstream>

// tenHHRing simulation code
#include "tenHHRing_CODE/definitions.h"

int main()
{
    allocateMem();
    initialize();
    initializeSparse();

    std::ofstream os("tenHHRing_output.V.dat");
    while(t < 200.0f) {
        if(iT == 0) {
            glbSpkStim[0] = 0;
            glbSpkCntStim[0] = 1;
            pushStimCurrentSpikesToDevice();
        }

        stepTimeU();
        pullVPop1FromDevice();

        os << t << " ";
        for (int j= 0; j < 10; j++) {
            os << VPop1[j] << " ";
        }
        os << std::endl;
    }
    os.close();
    return 0;
}
```

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



Previous | Top | Next

12 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.

12.1 Creating and simulating a network model

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

- ModelSpec::setDT();
- ModelSpec::setName();

Then add neuron populations by:

- ModelSpec::addNeuronPopulation();

for each neuron population. Add synapse populations by:

- ModelSpec::addSynapsePopulation();

for each synapse population.

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

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

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 .cc file which runs the simulation. This file should include <YourModelName>_CODE/definitions.h. 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:

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

Any variables marked as unintialised using the uninitialisedVar() function or sparse connectivity not initialised using a snippet must be initialised by the user between calls to initialize() and initializeSparse(). Core functions generated by GeNN to be included in the user code include:

- allocateMem()
- initialize()
- initializeSparse()
- push<neuron or synapse name>StateToDevice()
- pull<neuron or synapse name>StateFromDevice()
- push<neuron name>SpikesToDevice()
- pull<neuron name>SpikesFromDevice()
- push<neuron name>SpikesEventsToDevice()
- pull<neuron name>SpikesEventsFromDevice()
- push<neuron name>SpikeTimesToDevice()
- pull<neuron name>SpikeTimesFromDevice()
- push<neuron name>CurrentSpikesToDevice()
- pull<neuron name>CurrentSpikesFromDevice()
- push<neuron name>CurrentSpikesEventsToDevice()
- pull<neuron name>CurrentSpikesEventsFromDevice()
- pull<synapse name>ConnectivityFromDevice()
- push<synapse name>ConnectivityToDevice()
- pull<var name><neuron or synapse name>FromDevice()
- push<var name><neuron or synapse name>ToDevice()
- copyStateToDevice()
- copyStateFromDevice()
- copyCurrentSpikesFromDevice()
- copyCurrentSpikesEventsFromDevice()
- stepTime()
- freeMem()

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

12.1.1 Extra Global Parameters

If extra global parameters have a "scalar" type such as float they can be set directly from simulation code. For example the extra global parameter "reward" of population "Pop" could be set with:

```
rewardPop = 5.0f;
```

However, if extra global parameters have a pointer type such as float*, GeNN generates additional functions to allocate, free and copy these variables between host and device:

- allocate<var name><neuron or synapse name>
- free<var name><neuron or synapse name>
- push<var name><neuron or synapse name>ToDevice
- pull<var name><neuron or synapse name>FromDevice These operate in much the same manner as the functions for interacting with standard variables described above but the allocate, push and pull functions all take a "count" parameter specifying how many entries the extra global parameter array should be.

12.2 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 ModelSpec::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 ModelSpec::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.

12.3 Working with variables in GeNN

12.3.1 Model variables

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

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

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

```
pullPopStateFromDevice();
```

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

Note

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.

12.3.2 Built-in Variables in GeNN

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

- `DT` : Time step (typically in ms) for simulation; Neuron integration can be done in multiple sub-steps inside the neuron model for numerical stability (see Traub-Miles and Izhikevich neuron model variations in Neuron models).
- `inSyn:` This is an intermediary synapse variable which contains the summed input into a postsynaptic neuron (originating from the `addtoinSyn` variables of the incoming synapses) .
- `Isyn` : This is a local variable which contains the (summed) input current to a neuron. It is typically the sum of any explicit current input and all synaptic inputs. The way its value is calculated during the update of the postsynaptic neuron is defined by the code provided in the postsynaptic model. For example, the standard PostsynapticModels::ExpCond postsynaptic model defines

  ```
  SET_APPLY_INPUT_CODE("$(Isyn) += $(inSyn)*($(E)-$(V))");
  ```

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

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

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

- `sT` : This is a neuron variable containing the last spike time of each neuron and is automatically generated for pre and postsynaptic neuron groups if they are connected using a synapse population with a weight update model that has SET_NEEDS_PRE_SPIKE_TIME(true) or SET_NEEDS_POST_SPIKE_TIME(true) set.

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

12.4 Debugging suggestions

In Linux, users can call `cuda-gdb` to debug on the GPU. Example projects in the `userproject` directory come with a flag to enable debugging (–debug). 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.

Note

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

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).

On Windows models can also be debugged and developed by opening the sln file used to build the model in Visual Studio. From here files can be added to the project, build settings can be adjusted and the full suite of Visual Studio debugging and profiling tools can be used.

Note

When opening the models in the `userproject` directory in Visual Studio, right-click on the project in the solution explorer, select 'Properties'. Then, making sure the desired configuration is selected, navigate to 'Debugging' under 'Configuration Properties', set the 'Working Directory' to '..' and the 'Command Arguments' to match those passed to genn-buildmodel e.g. 'outdir' to use an output directory called outdir.

Previous | Top | Next

13 Credits

GeNN was created by Thomas Nowotny.

GeNN is currently maintained and developed by James Knight.

Current sources and PyGeNN were first implemented by Anton Komissarov.

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.

MPI support was largely developed by Mengchi Zhang.

Previous

14 Namespace Index

14.1 Namespace List

Here is a list of all namespaces with brief descriptions:

15 Hierarchical Index

15.1 Class Hierarchy

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

16 Class Index

16.1 Class List

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

17 File Index

17.1 File List

Here is a list of all files with brief descriptions:

18 Namespace Documentation

18.1 CodeGenerator Namespace Reference

Helper class for generating code - automatically inserts brackets, indents etc.

Namespaces

- CUDA
- SingleThreadedCPU

Classes

- class BackendBase
- class CodeStream
- struct FunctionTemplate

- class MemAlloc
- struct NameIterCtx
- struct PreferencesBase

    Base class for backend preferences - can be accessed via a global in 'classic' C++ code generator.
- class StructNameConstIter

    Custom iterator for iterating through the containers of structs with 'name' members.
- class Substitutions
- class TeeBuf
- class TeeStream

Typedefs

- typedef NameIterCtx< Snippet::Base::VarVec > VarNameIterCtx
- typedef NameIterCtx< Snippet::Base::DerivedParamVec > DerivedParamNameIterCtx
- typedef NameIterCtx< Snippet::Base::ParamValVec > ParamValIterCtx

Functions

- void substitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting strings in the neuron code strings or other templates.
- bool regexVarSubstitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting variable names in the neuron code strings or other templates using regular expressions.
- bool regexFuncSubstitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting function names in the neuron code strings or other templates using regular expressions.
- void functionSubstitute (std::string &code, const std::string &funcName, unsigned int numParams, const std::string &replaceFuncTemplate)

    This function substitutes function calls in the form:
- template<typename NameIter >
  void name_substitutions (std::string &code, const std::string &prefix, NameIter namesBegin, NameIter namesEnd, const std::string &postfix="", const std::string &ext="")

    This function performs a list of name substitutions for variables in code snippets.
- void name_substitutions (std::string &code, const std::string &prefix, const std::vector< std::string > &names, const std::string &postfix="", const std::string &ext="")

    This function performs a list of name substitutions for variables in code snippets.
- template<class T , typename std::enable_if< std::is_floating_point< T >::value >::type * = nullptr>
  void writePreciseString (std::ostream &os, T value)

    This function writes a floating point value to a stream -setting the precision so no digits are lost.
- template<class T , typename std::enable_if< std::is_floating_point< T >::value >::type * = nullptr>
  std::string writePreciseString (T value)

    This function writes a floating point value to a string - setting the precision so no digits are lost.
- template<typename NameIter >
  void value_substitutions (std::string &code, NameIter namesBegin, NameIter namesEnd, const std::vector< double > &values, const std::string &ext="")

    This function performs a list of value substitutions for parameters in code snippets.
- void value_substitutions (std::string &code, const std::vector< std::string > &names, const std::vector< double > &values, const std::string &ext="")

    This function performs a list of value substitutions for parameters in code snippets.
- std::string ensureFtype (const std::string &oldcode, const std::string &type)

    This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).
- void checkUnreplacedVariables (const std::string &code, const std::string &codeName)

    This function checks for unknown variable definitions and returns a gennError if any are found.

- void preNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &offset, const std::string &axonalDelayOffset, const std::string &postIdx, const std::string &devPrefix, const std::string &preVarPrefix="", const std::string &preVarSuffix="")

  suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
- void postNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &offset, const std::string &backPropDelayOffset, const std::string &preIdx, const std::string &devPrefix, const std::string &postVarPrefix="", const std::string &postVarSuffix="")

  suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
- void neuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &preIdx, const std::string &postIdx, const std::string &devPrefix, double dt, const std::string &preVarPrefix="", const std::string &preVarSuffix="", const std::string &postVarPrefix="", const std::string &postVarSuffix="")

  Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.
- GENN_EXPORT std::ostream & operator<< (std::ostream &s, const CodeStream::OB &ob)
- GENN_EXPORT std::ostream & operator<< (std::ostream &s, const CodeStream::CB &cb)
- GENN_EXPORT std::vector< std::string > generateAll (const ModelSpecInternal &model, const BackendBase &backend, const filesystem::path &outputPath, bool standaloneModules=false)
- void generateInit (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)
- void GENN_EXPORT generateMakefile (std::ostream &os, const BackendBase &backend, const std::vector< std::string > &moduleNames)
- void GENN_EXPORT generateMPI (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

  A function that generates predominantly MPI infrastructure code.
- void GENN_EXPORT generateMSBuild (std::ostream &os, const BackendBase &backend, const std::string &projectGUID, const std::vector< std::string > &moduleNames)
- void generateNeuronUpdate (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)
- MemAlloc generateRunner (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, const ModelSpecInternal &model, const BackendBase &backend, int localHostID)
- void generateSupportCode (CodeStream &os, const ModelSpecInternal &model)
- void generateSynapseUpdate (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

18.1.1 Detailed Description

Helper class for generating code - automatically inserts brackets, indents etc.

Based heavily on: `https://stackoverflow.com/questions/15053753/writing-a-manipulator-for-a-cus`

18.1.2 Typedef Documentation

18.1.2.1 DerivedParamNameIterCtx

```
typedef NameIterCtx<Snippet::Base::DerivedParamVec> CodeGenerator::DerivedParamNameIterCtx
```

18.1.2.2 ParamValIterCtx

```
typedef NameIterCtx<Snippet::Base::ParamValVec> CodeGenerator::ParamValIterCtx
```

18.1.2.3 VarNameIterCtx

```
typedef NameIterCtx<Snippet::Base::VarVec> CodeGenerator::VarNameIterCtx
```

18.1.3 Function Documentation

18.1.3.1 checkUnreplacedVariables()

```
void CodeGenerator::checkUnreplacedVariables (
            const std::string & code,
            const std::string & codeName )
```

This function checks for unknown variable definitions and returns a gennError if any are found.

18.1.3.2 ensureFtype()

```
std::string CodeGenerator::ensureFtype (
            const std::string & oldcode,
            const std::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).

18.1.3.3 functionSubstitute()

```
void CodeGenerator::functionSubstitute (
            std::string & code,
            const std::string & funcName,
            unsigned int numParams,
            const std::string & replaceFuncTemplate )
```

This function substitutes function calls in the form:

$(functionName, parameter1, param2Function(0.12, "string"))

with replacement templates in the form:

actualFunction(CONSTANT, $(0), $(1))

18.1.3.4 generateAll()

```
std::vector< std::string > CodeGenerator::generateAll (
            const ModelSpecInternal & model,
            const BackendBase & backend,
            const filesystem::path & outputPath,
            bool standaloneModules = false )
```

18.1.3.5 generateInit()

```
void CodeGenerator::generateInit (
            CodeStream & os,
            const ModelSpecInternal & model,
            const BackendBase & backend,
            bool standaloneModules )
```

18.1.3.6 generateMakefile()

```
void CodeGenerator::generateMakefile (
            std::ostream & os,
            const BackendBase & backend,
            const std::vector< std::string > & moduleNames )
```

18.1.3.7 generateMPI()

```
void CodeGenerator::generateMPI (
            CodeStream & os,
            const ModelSpecInternal & model,
            const BackendBase & backend,
            bool standaloneModules )
```

A function that generates predominantly MPI infrastructure code.

In this function MPI infrastructure code are generated, including: MPI send and receive functions.

18.1.3.8 generateMSBuild()

```
void CodeGenerator::generateMSBuild (
            std::ostream & os,
            const BackendBase & backend,
            const std::string & projectGUID,
            const std::vector< std::string > & moduleNames )
```

18.1.3.9 generateNeuronUpdate()

```
void CodeGenerator::generateNeuronUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            const BackendBase & backend,
            bool standaloneModules )
```

18.1.3.10 generateRunner()

```
CodeGenerator::MemAlloc CodeGenerator::generateRunner (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            const ModelSpecInternal & model,
            const BackendBase & backend,
            int localHostID )
```

18.1.3.11 generateSupportCode()

```
void CodeGenerator::generateSupportCode (
            CodeStream & os,
            const ModelSpecInternal & model )
```

18.1.3.12 generateSynapseUpdate()

```
void CodeGenerator::generateSynapseUpdate (
```

```
            CodeStream & os,
            const ModelSpecInternal & model,
            const BackendBase & backend,
            bool standaloneModules )
```

18.1.3.13 name_substitutions() [1/2]

```
template<typename NameIter >
void CodeGenerator::name_substitutions (
            std::string & code,
            const std::string & prefix,
            NameIter namesBegin,
            NameIter namesEnd,
            const std::string & postfix = "",
            const std::string & ext = "" ) [inline]
```

This function performs a list of name substitutions for variables in code snippets.

18.1.3.14 name_substitutions() [2/2]

```
void CodeGenerator::name_substitutions (
            std::string & code,
            const std::string & prefix,
            const std::vector< std::string > & names,
            const std::string & postfix = "",
            const std::string & ext = "" ) [inline]
```

This function performs a list of name substitutions for variables in code snippets.

18.1.3.15 neuronSubstitutionsInSynapticCode()

```
void CodeGenerator::neuronSubstitutionsInSynapticCode (
            std::string & wCode,
            const SynapseGroupInternal & sg,
            const std::string & preIdx,
            const std::string & postIdx,
            const std::string & devPrefix,
            double dt,
            const std::string & preVarPrefix = "",
            const std::string & preVarSuffix = "",
            const std::string & postVarPrefix = "",
            const std::string & postVarSuffix = "" )
```

Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

Parameters

wCode	the code string to work on
sg	the synapse group connecting the pre and postsynaptic neuron populations whose parameters might need to be substituted
preIdx	index of the pre-synaptic neuron to be accessed for _pre variables; differs for different Span)
postIdx	index of the post-synaptic neuron to be accessed for _post variables; differs for different Span)

Parameters

devPrefix	device prefix, "dd_" for GPU, nothing for CPU
dt	simulation timestep (ms)
preVarPrefix	prefix to be used for presynaptic variable accesses - typically combined with suffix to wrap in function call such as __ldg(&XXX)
preVarSuffix	suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
postVarPrefix	prefix to be used for postsynaptic variable accesses - typically combined with suffix to wrap in function call such as __ldg(&XXX)
postVarSuffix	suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

18.1.3.16 operator<<() [1/2]

```
std::ostream & CodeGenerator::operator<< (
            std::ostream & s,
            const CodeStream::OB & ob )
```

18.1.3.17 operator<<() [2/2]

```
std::ostream & CodeGenerator::operator<< (
            std::ostream & s,
            const CodeStream::CB & cb )
```

18.1.3.18 postNeuronSubstitutionsInSynapticCode()

```
void CodeGenerator::postNeuronSubstitutionsInSynapticCode (
            std::string & wCode,
            const SynapseGroupInternal & sg,
            const std::string & offset,
            const std::string & backPropDelayOffset,
            const std::string & preIdx,
            const std::string & devPrefix,
            const std::string & postVarPrefix = "",
            const std::string & postVarSuffix = "" )
```

suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

Parameters

wCode	the code string to work on
devPrefix	device prefix, "dd_" for GPU, nothing for CPU
postVarPrefix	prefix to be used for postsynaptic variable accesses - typically combined with suffix to wrap in function call such as __ldg(&XXX)
postVarSuffix	suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

18.1.3.19 preNeuronSubstitutionsInSynapticCode()

```
void CodeGenerator::preNeuronSubstitutionsInSynapticCode (
            std::string & wCode,
            const SynapseGroupInternal & sg,
            const std::string & offset,
            const std::string & axonalDelayOffset,
            const std::string & postIdx,
            const std::string & devPrefix,
            const std::string & preVarPrefix = "",
            const std::string & preVarSuffix = "" )
```

suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

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
devPrefix	device prefix, "dd_" for GPU, nothing for CPU
preVarPrefix	prefix to be used for presynaptic variable accesses - typically combined with suffix to wrap in function call such as __ldg(&XXX)
preVarSuffix	suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

18.1.3.20 regexFuncSubstitute()

```
bool CodeGenerator::regexFuncSubstitute (
            std::string & s,
            const std::string & trg,
            const std::string & rep )
```

Tool for substituting function names in the neuron code strings or other templates using regular expressions.

18.1.3.21 regexVarSubstitute()

```
bool CodeGenerator::regexVarSubstitute (
            std::string & s,
            const std::string & trg,
            const std::string & rep )
```

Tool for substituting variable names in the neuron code strings or other templates using regular expressions.

18.1.3.22 substitute()

```
void CodeGenerator::substitute (
            std::string & s,
            const std::string & trg,
            const std::string & rep )
```

Tool for substituting strings in the neuron code strings or other templates.

18.1.3.23 value_substitutions() [1/2]

```
template<typename NameIter >
void CodeGenerator::value_substitutions (
            std::string & code,
            NameIter namesBegin,
            NameIter namesEnd,
            const std::vector< double > & values,
            const std::string & ext = "" )  [inline]
```

This function performs a list of value substitutions for parameters in code snippets.

18.1.3.24 value_substitutions() [2/2]

```
void CodeGenerator::value_substitutions (
            std::string & code,
            const std::vector< std::string > & names,
            const std::vector< double > & values,
            const std::string & ext = "" )  [inline]
```

This function performs a list of value substitutions for parameters in code snippets.

18.1.3.25 writePreciseString() [1/2]

```
template<class T , typename std::enable_if< std::is_floating_point< T >::value >::type * =
nullptr>
void CodeGenerator::writePreciseString (
            std::ostream & os,
            T value )
```

This function writes a floating point value to a stream -setting the precision so no digits are lost.

18.1.3.26 writePreciseString() [2/2]

```
template<class T , typename std::enable_if< std::is_floating_point< T >::value >::type * =
nullptr>
std::string CodeGenerator::writePreciseString (
            T value )
```

This function writes a floating point value to a string - setting the precision so no digits are lost.

18.2 CodeGenerator::CUDA Namespace Reference

Namespaces

- Optimiser
- Utils

Classes

- class Backend
- struct Preferences

    Preferences for CUDA backend.

Typedefs

- using KernelBlockSize = std::array< size_t, KernelMax >

    Array of block sizes for each kernel.

Enumerations

- enum DeviceSelect { DeviceSelect::OPTIMAL, DeviceSelect::MOST_MEMORY, DeviceSelect::MANUAL }

    Methods for selecting CUDA device.
- enum BlockSizeSelect { BlockSizeSelect::OCCUPANCY, BlockSizeSelect::MANUAL }

    Methods for selecting CUDA kernel block size.
- enum Kernel {
  KernelNeuronUpdate, KernelPresynapticUpdate, KernelPostsynapticUpdate, KernelSynapseDynamics↩ Update,
  KernelInitialize, KernelInitializeSparse, KernelPreNeuronReset, KernelPreSynapseReset,
  KernelMax }

    Kernels generated by CUDA backend.

18.2.1 Typedef Documentation

18.2.1.1 KernelBlockSize

```
using CodeGenerator::CUDA::KernelBlockSize = typedef std::array<size_t, KernelMax>
```

Array of block sizes for each kernel.

18.2.2 Enumeration Type Documentation

18.2.2.1 BlockSizeSelect

```
enum CodeGenerator::CUDA::BlockSizeSelect  [strong]
```

Methods for selecting CUDA kernel block size.

Enumerator

OCCUPANCY	Pick optimal blocksize for each kernel based on occupancy.
MANUAL	Use block sizes specified by user.

18.2.2.2 DeviceSelect

```
enum CodeGenerator::CUDA::DeviceSelect  [strong]
```

Methods for selecting CUDA device.

Enumerator

OPTIMAL	Pick optimal device based on how well kernels can be simultaneously simulated and occupancy.
MOST_MEMORY	Pick device with most global memory.
MANUAL	Use device specified by user.

18.2.2.3 Kernel

```
enum CodeGenerator::CUDA::Kernel
```

Kernels generated by CUDA backend.

Enumerator

KernelNeuronUpdate	
KernelPresynapticUpdate	
KernelPostsynapticUpdate	
KernelSynapseDynamicsUpdate	
KernelInitialize	
KernelInitializeSparse	
KernelPreNeuronReset	
KernelPreSynapseReset	
KernelMax	

18.3 CodeGenerator::CUDA::Optimiser Namespace Reference

Functions

- BACKEND_EXPORT Backend createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

18.3.1 Function Documentation

18.3.1.1 createBackend()

```
Backend CodeGenerator::CUDA::Optimiser::createBackend (
            const ModelSpecInternal & model,
            const filesystem::path & outputPath,
            int localHostID,
            const Preferences & preferences )
```

18.4 CodeGenerator::CUDA::Utils Namespace Reference

Functions

- size_t ceilDivide (size_t numerator, size_t denominator)
- size_t padSize (size_t size, size_t blockSize)

18.4.1 Function Documentation

18.4.1.1 ceilDivide()

```
size_t CodeGenerator::CUDA::Utils::ceilDivide (
            size_t numerator,
            size_t denominator )  [inline]
```

18.4.1.2 padSize()

```
size_t CodeGenerator::CUDA::Utils::padSize (
            size_t size,
            size_t blockSize )  [inline]
```

18.5 CodeGenerator::SingleThreadedCPU Namespace Reference

Namespaces

- Optimiser

Classes

- class Backend
- struct Preferences

18.6 CodeGenerator::SingleThreadedCPU::Optimiser Namespace Reference

Functions

- BACKEND_EXPORT Backend createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

18.6.1 Function Documentation

18.6.1.1 createBackend()

```
Backend CodeGenerator::SingleThreadedCPU::Optimiser::createBackend (
            const ModelSpecInternal & model,
            const filesystem::path & outputPath,
            int localHostID,
            const Preferences & preferences )
```

18.7 CurrentSourceModels Namespace Reference

Classes

- class Base

  Base class for all current source models.
- class DC

  DC source.
- class GaussianNoise

  Noisy current source with noise drawn from normal distribution.

18.8 filesystem Namespace Reference

18.9 InitSparseConnectivitySnippet Namespace Reference

Base class for all sparse connectivity initialisation snippets.

Classes

- class Base
- class FixedProbability
- class FixedProbabilityBase
- class FixedProbabilityNoAutapse
- class Init
- class OneToOne

  Initialises connectivity to a 'one-to-one' diagonal matrix.
- class Uninitialised

  Used to mark connectivity as uninitialised - no initialisation code will be run.

18.9.1 Detailed Description

Base class for all sparse connectivity initialisation snippets.

18.10 InitVarSnippet Namespace Reference

Base class for all value initialisation snippets.

Classes

- class Base
- class Constant

  Initialises variable to a constant value.
- class Exponential

  Initialises variable by sampling from the exponential distribution.
- class Gamma

  Initialises variable by sampling from the exponential distribution.
- class Normal

  Initialises variable by sampling from the normal distribution.
- class Uniform

  Initialises variable by sampling from the uniform distribution.
- class Uninitialised

  Used to mark variables as uninitialised - no initialisation code will be run.

18.10.1 Detailed Description

Base class for all value initialisation snippets.

18.11 Models Namespace Reference

Classes

- class Base

  Base class for all models - in addition to the parameters snippets have, models can have state variables.
- class VarInit
- class VarInitContainerBase
- class VarInitContainerBase< 0 >

18.12 NeuronModels Namespace Reference

Classes

- class Base

  Base class for all neuron models.
- class Izhikevich

  Izhikevich neuron with fixed parameters [1].
- class IzhikevichVariable

  Izhikevich neuron with variable parameters [1].
- class LIF
- class Poisson

  Poisson neurons.
- class PoissonNew

  Poisson neurons.
- class RulkovMap

  Rulkov Map neuron.
- class SpikeSource

  Empty neuron which allows setting spikes from external sources.
- class SpikeSourceArray

  Spike source array.
- class TraubMiles

  Hodgkin-Huxley neurons with Traub & Miles algorithm.
- class TraubMilesAlt

  Hodgkin-Huxley neurons with Traub & Miles algorithm.
- class TraubMilesFast

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

  Hodgkin-Huxley neurons with Traub & Miles algorithm.

18.13 PostsynapticModels Namespace Reference

Classes

- class Base

  Base class for all postsynaptic models.
- class DeltaCurr

  Simple delta current synapse.
- class ExpCond

  Exponential decay with synaptic input treated as a conductance value.
- class ExpCurr

  Exponential decay with synaptic input treated as a current value.

18.14 pygenn Namespace Reference

Namespaces

- genn_groups
- genn_model
- model_preprocessor

18.15 pygenn.genn_groups Namespace Reference

Classes

- class CurrentSource

    Class representing a current injection into a group of neurons.
- class Group

    Parent class of NeuronGroup, SynapseGroup and CurrentSource.
- class NeuronGroup

    Class representing a group of neurons.
- class SynapseGroup

    Class representing synaptic connection between two groups of neurons.

Variables

- xrange = range

    GeNNGroups This module provides classes which automatize model checks and parameter convesions for GeNN Groups.

18.15.1 Variable Documentation

18.15.1.1 xrange

```
pygenn.genn_groups.xrange = range
```

GeNNGroups This module provides classes which automatize model checks and parameter convesions for GeNN Groups.

18.16 pygenn.genn_model Namespace Reference

Classes

- class GeNNModel

    GeNNModel class This class helps to define, build and run a GeNN model from python.

Functions

- def init_var (init_var_snippet, param_space)

    This helper function creates a VarInit object to easily initialise a variable using a snippet.
- def init_connectivity (init_sparse_connect_snippet, param_space)

    This helper function creates a InitSparseConnectivitySnippet::Init object to easily initialise connectivity using a snippet.

- def create_custom_neuron_class (class_name, param_names=None, var_name_types=None, derived_params=None, sim_code=None, threshold_condition_code=None, reset_code=None, support_code=None, extra_global_params=None, additional_input_vars=None, is_auto_refractory_required=None, custom_body=None)

  This helper function creates a custom NeuronModel class.
- def create_custom_postsynaptic_class (class_name, param_names=None, var_name_types=None, derived_params=None, decay_code=None, apply_input_code=None, support_code=None, custom_body=None)

  This helper function creates a custom PostsynapticModel class.
- def create_custom_weight_update_class (class_name, param_names=None, var_name_types=None, pre_var_name_types=None, post_var_name_types=None, derived_params=None, sim_code=None, event_code=None, learn_post_code=None, synapse_dynamics_code=None, event_threshold_condition_code=None, pre_spike_code=None, post_spike_code=None, sim_support_code=None, learn_post_support_code=None, synapse_dynamics_suppport_code=None, extra_global_params=None, is_pre_spike_time_required=None, is_post_spike_time_required=None, custom_body=None)

  This helper function creates a custom WeightUpdateModel class.
- def create_custom_current_source_class (class_name, param_names=None, var_name_types=None, derived_params=None, injection_code=None, extra_global_params=None, custom_body=None)

  This helper function creates a custom NeuronModel class.
- def create_custom_model_class (class_name, base, param_names, var_name_types, derived_params, custom_body)

  This helper function completes a custom model class creation.
- def create_dpf_class (dp_func)

  Helper function to create derived parameter function class.
- def create_cmlf_class (cml_func)

  Helper function to create function class for calculating sizes of matrices initialised with sparse connectivity initialisation snippet.
- def create_custom_init_var_snippet_class (class_name, param_names=None, derived_params=None, var_init_code=None, custom_body=None)

  This helper function creates a custom InitVarSnippet class.
- def create_custom_sparse_connect_init_snippet_class (class_name, param_names=None, derived_params=None, row_build_code=None, row_build_state_vars=None, calc_max_row_len_func=None, calc_max_col_len_func=None, extra_global_params=None, custom_body=None)

  This helper function creates a custom InitSparseConnectivitySnippet class.

Variables

- backend_modules = OrderedDict()
- m = import_module(".genn_wrapper." + b + "Backend", "pygenn")

18.16.1 Function Documentation

18.16.1.1 create_cmlf_class()

```
def pygenn.genn_model.create_cmlf_class (
              cml_func )
```

Helper function to create function class for calculating sizes of matrices initialised with sparse connectivity initialisation snippet.

Parameters

cml_func	a function which computes the length and takes three args "num_pre" (unsigned int), "num_post" (unsigned int) and "pars" (vector of double)

18.16.1.2 create_custom_current_source_class()

```
def pygenn.genn_model.create_custom_current_source_class (
              class_name,
              param_names = None,
              var_name_types = None,
              derived_params = None,
              injection_code = None,
              extra_global_params = None,
              custom_body = None )
```

This helper function creates a custom NeuronModel class.

See also

create_custom_neuron_class
create_custom_weight_update_class
create_custom_current_source_class
create_custom_init_var_snippet_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
var_name_types	list of pairs of strings with varible names and types of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of the class which inherits from `pygenn.genn_wrapper.DerivedParamFunc`
injection_code	string with the current injection code
extra_global_params	list of pairs of strings with names and types of additional parameters
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.3 create_custom_init_var_snippet_class()

```
def pygenn.genn_model.create_custom_init_var_snippet_class (
              class_name,
              param_names = None,
              derived_params = None,
              var_init_code = None,
              custom_body = None )
```

This helper function creates a custom InitVarSnippet class.

See also

create_custom_neuron_class
create_custom_weight_update_class
create_custom_postsynaptic_class
create_custom_current_source_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of the `pygenn.genn_wrapper.DerivedParamFunc`' class
var_init_code	string with the variable initialization code
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.4 create_custom_model_class()

```
def pygenn.genn_model.create_custom_model_class (
              class_name,
              base,
              param_names,
              var_name_types,
              derived_params,
              custom_body )
```

This helper function completes a custom model class creation.

This part is common for all model classes and is nearly useless on its own unless you specify custom_body.

See also

create_custom_neuron_class
create_custom_weight_update_class
create_custom_postsynaptic_class
create_custom_current_source_class
create_custom_init_var_snippet_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
base	base class
param_names	list of strings with param names of the model
var_name_types	list of pairs of strings with varible names and types of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of the class which inherits from the `pygenn.genn_wrapper.DerivedParamFunc` class
custom_body	dictionary with attributes and methods of the new class

18.16.1.5 create_custom_neuron_class()

```
def pygenn.genn_model.create_custom_neuron_class (
              class_name,
              param_names = None,
              var_name_types = None,
              derived_params = None,
              sim_code = None,
```

```
                threshold_condition_code = None,
                reset_code = None,
                support_code = None,
                extra_global_params = None,
                additional_input_vars = None,
                is_auto_refractory_required = None,
                custom_body = None )
```

This helper function creates a custom NeuronModel class.

See also

create_custom_postsynaptic_class
create_custom_weight_update_class
create_custom_current_source_class
create_custom_init_var_snippet_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
var_name_types	list of pairs of strings with varible names and types of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of a class which inherits from `pygenn.genn_wrapper.Snippet.DerivedParamFunc`
sim_code	string with the simulation code
threshold_condition_code	string with the threshold condition code
reset_code	string with the reset code
support_code	string with the support code
extra_global_params	list of pairs of strings with names and types of additional parameters
additional_input_vars	list of tuples with names and types as strings and initial values of additional local input variables
is_auto_refractory_required	does this model require auto-refractory logic to be generated?
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.6 create_custom_postsynaptic_class()

```
def pygenn.genn_model.create_custom_postsynaptic_class (
                class_name,
                param_names = None,
                var_name_types = None,
                derived_params = None,
                decay_code = None,
                apply_input_code = None,
                support_code = None,
                custom_body = None )
```

This helper function creates a custom PostsynapticModel class.

See also

create_custom_neuron_class
create_custom_weight_update_class
create_custom_current_source_class

create_custom_init_var_snippet_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
var_name_types	list of pairs of strings with varible names and types of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of a class which inherits from `pygenn.genn_wrapper.DerivedParamFunc`
decay_code	string with the decay code
apply_input_code	string with the apply input code
support_code	string with the support code
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.7 create_custom_sparse_connect_init_snippet_class()

```
def pygenn.genn_model.create_custom_sparse_connect_init_snippet_class (
              class_name,
              param_names = None,
              derived_params = None,
              row_build_code = None,
              row_build_state_vars = None,
              calc_max_row_len_func = None,
              calc_max_col_len_func = None,
              extra_global_params = None,
              custom_body = None )
```

This helper function creates a custom InitSparseConnectivitySnippet class.

See also

create_custom_neuron_class
create_custom_weight_update_class
create_custom_postsynaptic_class
create_custom_current_source_class
create_custom_init_var_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of the class which inherits from `pygenn.genn_wrapper.DerivedParamFunc`
row_build_code	string with row building initialization code
row_build_state_vars	list of tuples of state variables, their types and their initial values to use across row building loop
calc_max_row_len_func	instance of class inheriting from CalcMaxLengthFunc used to calculate maximum row length of synaptic matrix
calc_max_col_len_func	instance of class inheriting from CalcMaxLengthFunc used to calculate maximum col length of synaptic matrix

Parameters

extra_global_params	list of pairs of strings with names and types of additional parameters
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.8 create_custom_weight_update_class()

```
def pygenn.genn_model.create_custom_weight_update_class (
              class_name,
              param_names = None,
              var_name_types = None,
              pre_var_name_types = None,
              post_var_name_types = None,
              derived_params = None,
              sim_code = None,
              event_code = None,
              learn_post_code = None,
              synapse_dynamics_code = None,
              event_threshold_condition_code = None,
              pre_spike_code = None,
              post_spike_code = None,
              sim_support_code = None,
              learn_post_support_code = None,
              synapse_dynamics_suppport_code = None,
              extra_global_params = None,
              is_pre_spike_time_required = None,
              is_post_spike_time_required = None,
              custom_body = None )
```

This helper function creates a custom WeightUpdateModel class.

See also

create_custom_neuron_class
create_custom_postsynaptic_class
create_custom_current_source_class
create_custom_init_var_snippet_class
create_custom_sparse_connect_init_snippet_class

Parameters

class_name	name of the new class
param_names	list of strings with param names of the model
var_name_types	list of pairs of strings with variable names and types of the model
pre_var_name_types	list of pairs of strings with presynaptic variable names and types of the model
post_var_name_types	list of pairs of strings with postsynaptic variable names and types of the model
derived_params	list of pairs, where the first member is string with name of the derived parameter and the second MUST be an instance of a class which inherits from `pygenn.genn_wrapper.DerivedParamFunc`
sim_code	string with the simulation code
event_code	string with the event code
learn_post_code	string with the code to include in learn_synapse_post kernel/function
synapse_dynamics_code	string with the synapse dynamics code

Parameters

event_threshold_condition_code	string with the event threshold condition code
pre_spike_code	string with the code run once per spiking presynaptic neuron
post_spike_code	string with the code run once per spiking postsynaptic neuron
sim_support_code	string with simulation support code
learn_post_support_code	string with support code for learn_synapse_post kernel/function
synapse_dynamics_suppport_code	string with synapse dynamics support code
extra_global_params	list of pairs of strings with names and types of additional parameters
is_pre_spike_time_required	boolean, is presynaptic spike time required in any weight update kernels?
is_post_spike_time_required	boolean, is postsynaptic spike time required in any weight update kernels?
custom_body	dictionary with additional attributes and methods of the new class

18.16.1.9 create_dpf_class()

```
def pygenn.genn_model.create_dpf_class (
              dp_func )
```

Helper function to create derived parameter function class.

Parameters

dp_func	a function which computes the derived parameter and takes two args "pars" (vector of double) and "dt" (double)

18.16.1.10 init_connectivity()

```
def pygenn.genn_model.init_connectivity (
              init_sparse_connect_snippet,
              param_space )
```

This helper function creates a InitSparseConnectivitySnippet::Init object to easily initialise connectivity using a snippet.

Parameters

init_sparse_connect_snippet	type of the InitSparseConnectivitySnippet class as string or instance of class derived from InitSparseConnectivitySnippet::Custom.
param_space	dict with param values for the InitSparseConnectivitySnippet class

18.16.1.11 init_var()

```
def pygenn.genn_model.init_var (
              init_var_snippet,
              param_space )
```

This helper function creates a VarInit object to easily initialise a variable using a snippet.

Parameters

init_var_snippet	type of the InitVarSnippet class as string or instance of class derived from InitVarSnippet::Custom class.
param_space	dict with param values for the InitVarSnippet class

18.16.2 Variable Documentation

18.16.2.1 backend_modules

```
pygenn.genn_model.backend_modules = OrderedDict()
```

18.16.2.2 m

```
pygenn.genn_model.m = import_module(".genn_wrapper." + b + "Backend", "pygenn")
```

18.17 pygenn.model_preprocessor Namespace Reference

Classes

- class ExtraGlobalVariable

  Class holding information about GeNN extra global pointer variable.
- class Variable

  Class holding information about GeNN variables.

Functions

- def prepare_model (model, param_space, var_space, pre_var_space=None, post_var_space=None, model_family=None)

  Prepare a model by checking its validity and extracting information about variables and parameters.
- def prepare_snippet (snippet, param_space, snippet_family)

  Prepare a snippet by checking its validity and extracting information about parameters.
- def is_model_valid (model, model_family)

  Check whether the model is valid, i.e is native or derived from model_family.Custom.
- def param_space_to_vals (model, param_space)

  Convert a param_space dict to ParamValues.
- def param_space_to_val_vec (model, param_space)

  Convert a param_space dict to a std::vector<double>
- def var_space_to_vals (model, var_space)

  Convert a var_space dict to VarValues.
- def pre_var_space_to_vals (model, var_space)

  Convert a var_space dict to PreVarValues.
- def post_var_space_to_vals (model, var_space)

  Convert a var_space dict to PostVarValues.

Variables

- dictionary genn_to_numpy_types

18.17.1 Function Documentation

18.17.1.1 is_model_valid()

```
def pygenn.model_preprocessor.is_model_valid (
              model,
              model_family )
```

Check whether the model is valid, i.e is native or derived from model_family.Custom.

Parameters

model	string or instance of model_family.Custom
model_family	model family (NeuronModels, WeightUpdateModels or PostsynapticModels) to which model should belong to

Returns

instance of the model and its type as string

Raises ValueError if model is not valid (i.e. is not custom and is not natively available)

18.17.1.2 param_space_to_val_vec()

```
def pygenn.model_preprocessor.param_space_to_val_vec (
              model,
              param_space )
```

Convert a param_space dict to a std::vector<double>

Parameters

model	instance of the model
param_space	dict with parameters

Returns

native vector of parameters

18.17.1.3 param_space_to_vals()

```
def pygenn.model_preprocessor.param_space_to_vals (
              model,
              param_space )
```

Convert a param_space dict to ParamValues.

Parameters

model	instance of the model
param_space	dict with parameters

Returns

native model's ParamValues

18.17.1.4 post_var_space_to_vals()

```
def pygenn.model_preprocessor.post_var_space_to_vals (
              model,
              var_space )
```

Convert a var_space dict to PostVarValues.

Parameters

model	instance of the weight update model
var_space	dict with Variables

Returns

native model's VarValues

18.17.1.5 pre_var_space_to_vals()

```
def pygenn.model_preprocessor.pre_var_space_to_vals (
              model,
              var_space )
```

Convert a var_space dict to PreVarValues.

Parameters

model	instance of the weight update model
var_space	dict with Variables

Returns

native model's VarValues

18.17.1.6 prepare_model()

```
def pygenn.model_preprocessor.prepare_model (
              model,
              param_space,
              var_space,
              pre_var_space = None,
              post_var_space = None,
              model_family = None )
```

Prepare a model by checking its validity and extracting information about variables and parameters.

Parameters

model	string or instance of a class derived from `pygenn.genn_wrapper.NeuronModels.Custom` or `pygenn.genn_wrapper.WeightUpdateModels.Custom` or `pygenn.genn_wrapper.CurrentSourceModels.Custom`
param_space	dict with model parameters
var_space	dict with model variables
pre_var_space	optional dict with (weight update) model presynaptic variables
post_var_space	optional dict with (weight update) model postsynaptic variables
model_family	`pygenn.genn_wrapper.NeuronModels` or `pygenn.genn_wrapper.WeightUpdateModels` or `pygenn.genn_wrapper.CurrentSourceModels`

Returns

tuple consisting of (model instance, model type, model parameter names, model parameters, list of variable names, dict mapping names of variables to instances of class Variable)

18.17.1.7 prepare_snippet()

```
def pygenn.model_preprocessor.prepare_snippet (
              snippet,
              param_space,
              snippet_family )
```

Prepare a snippet by checking its validity and extracting information about parameters.

Parameters

snippet	string or instance of a class derived from `pygenn.genn_wrapper.InitVarSnippet.Custom` or `pygenn.genn_wrapper.InitSparseConnectivitySnippet.Custom`
param_space	dict with model parameters
snippet_family	`pygenn.genn_wrapper.InitVarSnippet` or `pygenn.genn_wrapper.InitSparseConnectivitySnippet`

Returns

tuple consisting of (snippet instance, snippet type, snippet parameter names, snippet parameters)

18.17.1.8 var_space_to_vals()

```
def pygenn.model_preprocessor.var_space_to_vals (
              model,
              var_space )
```

Convert a var_space dict to VarValues.

Parameters

model	instance of the model
var_space	dict with Variables

Returns

native model's VarValues

18.17.2 Variable Documentation

18.17.2.1 genn_to_numpy_types

`dictionary pygenn.model_preprocessor.genn_to_numpy_types`

Initial value:

```
= {
    "scalar":           np.float32,
    "float":            np.float32,
    "double":           np.float64,
    "int":              np.int32,
    "unsigned int":     np.uint32,
    "short":            np.int16,
    "unsigned short":   np.uint16,
    "char":             np.int8,
    "unsigned char":    np.uint8,
    "uint64_t":         np.uint64,
    "int64_t":          np.int64,
    "uint32_t":         np.uint32,
    "int32_t":          np.int32,
    "uint16_t":         np.uint16,
    "int16_t":          np.int16,
    "uint8_t":          np.uint8,
    "int8_t":           np.int8,
}
```

18.18 Snippet Namespace Reference

Classes

- class Base

  Base class for all code snippets.
- class Init
- class ValueBase
- class ValueBase< 0 >

18.18.1 Detailed Description

Wrapper to ensure at compile time that correct number of values are used when specifying the values of a model's parameters and initial state.

18.19 Utils Namespace Reference

Functions

- GENN_EXPORT bool isRNGRequired (const std::string &code)

  Does the code string contain any functions requiring random number generator.
- GENN_EXPORT bool isInitRNGRequired (const std::vector< Models::VarInit > &varInitialisers)

  Does the model with the vectors of variable initialisers and modes require an RNG for the specified init location i.e. host or device.
- GENN_EXPORT bool isTypePointer (const std::string &type)

  Function to determine whether a string containing a type is a pointer.
- GENN_EXPORT std::string getUnderlyingType (const std::string &type)

  Assuming type is a string containing a pointer type, function to return the underlying type.

18.19.1 Function Documentation

18.19.1.1 getUnderlyingType()

```
std::string Utils::getUnderlyingType (
            const std::string & type )
```

Assuming type is a string containing a pointer type, function to return the underlying type.

18.19.1.2 isInitRNGRequired()

```
bool Utils::isInitRNGRequired (
            const std::vector< Models::VarInit > & varInitialisers )
```

Does the model with the vectors of variable initialisers and modes require an RNG for the specified init location i.e. host or device.

18.19.1.3 isRNGRequired()

```
bool Utils::isRNGRequired (
            const std::string & code )
```

Does the code string contain any functions requiring random number generator.

18.19.1.4 isTypePointer()

```
bool Utils::isTypePointer (
            const std::string & type )
```

Function to determine whether a string containing a type is a pointer.

18.20 WeightUpdateModels Namespace Reference

Classes

- class Base

  Base class for all weight update models.
- class PiecewiseSTDP

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

  Graded-potential, static synapse.
- class StaticPulse

  Pulse-coupled, static synapse.
- class StaticPulseDendriticDelay

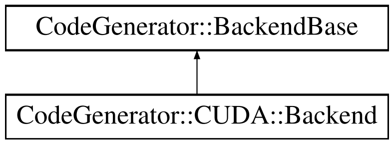
  Pulse-coupled, static synapse with heterogenous dendritic delays.

19 Class Documentation

19.1 CodeGenerator::CUDA::Backend Class Reference

```
#include <backend.h>
```

Inheritance diagram for CodeGenerator::CUDA::Backend:



Public Member Functions

- Backend (const KernelBlockSize &kernelBlockSizes, const Preferences &preferences, int localHostID, const std::string &scalarType, int device)
- virtual void genNeuronUpdate (CodeStream &os, const ModelSpecInternal &model, NeuronGroupSimHandler simHandler, NeuronGroupHandler wuVarUpdateHandler) const override

  Generate platform-specific function to update the state of all neurons.
- virtual void genSynapseUpdate (CodeStream &os, const ModelSpecInternal &model, SynapseGroupHandler wumThreshHandler, SynapseGroupHandler wumSimHandler, SynapseGroupHandler wumEventHandler, SynapseGroupHandler postLearnHandler, SynapseGroupHandler synapseDynamicsHandler) const override

  Generate platform-specific function to update the state of all synapses.
- virtual void genInit (CodeStream &os, const ModelSpecInternal &model, NeuronGroupHandler localNGHandler, NeuronGroupHandler remoteNGHandler, SynapseGroupHandler sgDenseInitHandler, SynapseGroupHandler sgSparseConnectHandler, SynapseGroupHandler sgSparseInitHandler) const override
- virtual void genDefinitionsPreamble (CodeStream &os) const override

  Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.
- virtual void genDefinitionsInternalPreamble (CodeStream &os) const override

  Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.
- virtual void genRunnerPreamble (CodeStream &os) const override
- virtual void genAllocateMemPreamble (CodeStream &os, const ModelSpecInternal &model) const override
- virtual void genStepTimeFinalisePreamble (CodeStream &os, const ModelSpecInternal &model) const override

  After all timestep logic is complete.
- virtual void genVariableDefinition (CodeStream &definitions, CodeStream &definitionsInternal, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genVariableImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual MemAlloc genVariableAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const override
- virtual void genVariableFree (CodeStream &os, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamDefinition (CodeStream &definitions, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamPush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamPull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genPopVariableInit (CodeStream &os, VarLocation loc, const Substitutions &kernelSubs, Handler handler) const override
- virtual void genVariableInit (CodeStream &os, VarLocation loc, size_t count, const std::string &indexVarName, const Substitutions &kernelSubs, Handler handler) const override

- virtual void genSynapseVariableRowInit (CodeStream &os, VarLocation loc, const SynapseGroupInternal &sg, const Substitutions &kernelSubs, Handler handler) const override
- virtual void genVariablePush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, bool autoInitialized, size_t count) const override
- virtual void genVariablePull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const override
- virtual void genCurrentTrueSpikePush (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentTrueSpikePull (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentSpikeLikeEventPush (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentSpikeLikeEventPull (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual MemAlloc genGlobalRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const ModelSpecInternal &model) const override
- virtual MemAlloc genPopulationRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const std::string &name, size_t count) const override
- virtual void genTimer (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, CodeStream &stepTimeFinalise, const std::string &name, bool updateInStepTime) const override
- virtual void genMakefilePreamble (std::ostream &os) const override

  This function can be used to generate a preamble for the GNU makefile used to build.
- virtual void genMakefileLinkRule (std::ostream &os) const override
- virtual void genMakefileCompileRule (std::ostream &os) const override
- virtual void genMSBuildConfigProperties (std::ostream &os) const override
- virtual void genMSBuildImportProps (std::ostream &os) const override
- virtual void genMSBuildItemDefinitions (std::ostream &os) const override
- virtual void genMSBuildCompileModule (const std::string &moduleName, std::ostream &os) const override
- virtual void genMSBuildImportTarget (std::ostream &os) const override
- virtual std::string getVarPrefix () const override
- virtual bool isGlobalRNGRequired (const ModelSpecInternal &model) const override

  Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?
- virtual bool isSynRemapRequired () const override
- virtual bool isPostsynapticRemapRequired () const override
- virtual size_t getDeviceMemoryBytes () const override

  How many bytes of memory does 'device' have.
- const cudaDeviceProp & getChosenCUDADevice () const
- int getChosenDeviceID () const
- int getRuntimeVersion () const
- std::string getNVCCFlags () const

Static Public Member Functions

- static size_t getNumPresynapticUpdateThreads (const SynapseGroupInternal &sg)
- static size_t getNumPostsynapticUpdateThreads (const SynapseGroupInternal &sg)
- static size_t getNumSynapseDynamicsThreads (const SynapseGroupInternal &sg)

Static Public Attributes

- static const char ∗ KernelNames [KernelMax]

Additional Inherited Members

19.1.1 Constructor & Destructor Documentation

19.1.1.1 Backend()

```
CodeGenerator::CUDA::Backend::Backend (
            const KernelBlockSize & kernelBlockSizes,
            const Preferences & preferences,
            int localHostID,
            const std::string & scalarType,
            int device )
```

19.1.2 Member Function Documentation

19.1.2.1 genAllocateMemPreamble()

```
void CodeGenerator::CUDA::Backend::genAllocateMemPreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const  [override], [virtual]
```

Allocate memory is the first function in GeNN generated code called by usercode and it should only ever be called once. Therefore it's a good place for any global initialisation. This function generates a 'preamble' to this function.

Implements CodeGenerator::BackendBase.

19.1.2.2 genCurrentSpikeLikeEventPull()

```
virtual void CodeGenerator::CUDA::Backend::genCurrentSpikeLikeEventPull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [inline], [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.3 genCurrentSpikeLikeEventPush()

```
virtual void CodeGenerator::CUDA::Backend::genCurrentSpikeLikeEventPush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [inline], [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.4 genCurrentTrueSpikePull()

```
virtual void CodeGenerator::CUDA::Backend::genCurrentTrueSpikePull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [inline], [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.5 genCurrentTrueSpikePush()

```
virtual void CodeGenerator::CUDA::Backend::genCurrentTrueSpikePush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [inline], [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.6 genDefinitionsInternalPreamble()

```
void CodeGenerator::CUDA::Backend::genDefinitionsInternalPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.

This will only be included by the platform-specific compiler used to build this backend so can include platform-specific types or headers

Implements CodeGenerator::BackendBase.

19.1.2.7 genDefinitionsPreamble()

```
void CodeGenerator::CUDA::Backend::genDefinitionsPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.

This will be included from a standard C++ compiler so shouldn't include any platform-specific types or headers

Implements CodeGenerator::BackendBase.

19.1.2.8 genExtraGlobalParamAllocation()

```
void CodeGenerator::CUDA::Backend::genExtraGlobalParamAllocation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.9 genExtraGlobalParamDefinition()

```
void CodeGenerator::CUDA::Backend::genExtraGlobalParamDefinition (
            CodeStream & definitions,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.10 genExtraGlobalParamImplementation()

```
void CodeGenerator::CUDA::Backend::genExtraGlobalParamImplementation (
            CodeStream & os,
```

```
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.11 genExtraGlobalParamPull()

```
void CodeGenerator::CUDA::Backend::genExtraGlobalParamPull (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.12 genExtraGlobalParamPush()

```
void CodeGenerator::CUDA::Backend::genExtraGlobalParamPush (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.13 genGlobalRNG()

```
MemAlloc CodeGenerator::CUDA::Backend::genGlobalRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const ModelSpecInternal & model ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.14 genInit()

```
void CodeGenerator::CUDA::Backend::genInit (
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupHandler localNGHandler,
            NeuronGroupHandler remoteNGHandler,
            SynapseGroupHandler sgDenseInitHandler,
            SynapseGroupHandler sgSparseConnectHandler,
            SynapseGroupHandler sgSparseInitHandler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.15 genMakefileCompileRule()

```
void CodeGenerator::CUDA::Backend::genMakefileCompileRule (
            std::ostream & os ) const  [override], [virtual]
```

The GNU make build system uses 'pattern rules' (https://www.gnu.org/software/make/manual/html←_node/Pattern-Intro.html) to build backend modules into objects. This function should generate a GNU make pattern rule capable of building each module (i.e. compiling .cc file $\$<$ into .o file $@).

Implements CodeGenerator::BackendBase.

19.1.2.16 genMakefileLinkRule()

```
void CodeGenerator::CUDA::Backend::genMakefileLinkRule (
            std::ostream & os ) const [override], [virtual]
```

The GNU make build system will populate a variable called with a list of objects to link. This function should generate a GNU make rule to build these objects into a shared library.

Implements CodeGenerator::BackendBase.

19.1.2.17 genMakefilePreamble()

```
void CodeGenerator::CUDA::Backend::genMakefilePreamble (
            std::ostream & os ) const [override], [virtual]
```

This function can be used to generate a preamble for the GNU makefile used to build.

Implements CodeGenerator::BackendBase.

19.1.2.18 genMSBuildCompileModule()

```
void CodeGenerator::CUDA::Backend::genMSBuildCompileModule (
            const std::string & moduleName,
            std::ostream & os ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.19 genMSBuildConfigProperties()

```
void CodeGenerator::CUDA::Backend::genMSBuildConfigProperties (
            std::ostream & os ) const [override], [virtual]
```

In MSBuild, 'properties' are used to configure global project settings e.g. whether the MSBuild project builds a static or dynamic library This function can be used to add additional XML properties to this section.

see https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-properties for more information.

Implements CodeGenerator::BackendBase.

19.1.2.20 genMSBuildImportProps()

```
void CodeGenerator::CUDA::Backend::genMSBuildImportProps (
            std::ostream & os ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.21 genMSBuildImportTarget()

```
void CodeGenerator::CUDA::Backend::genMSBuildImportTarget (
```

```
            std::ostream & os ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.22 genMSBuildItemDefinitions()

```
void CodeGenerator::CUDA::Backend::genMSBuildItemDefinitions (
            std::ostream & os ) const  [override], [virtual]
```

In MSBuild, the 'item definitions' are used to override the default properties of 'items' such as `<ClCompile>` or `<Link>`. This function should generate XML to correctly configure the 'items' required to build the generated code, taking into account etc.

see https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-items#item-definitions for more information.

Implements CodeGenerator::BackendBase.

19.1.2.23 genNeuronUpdate()

```
void CodeGenerator::CUDA::Backend::genNeuronUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupSimHandler simHandler,
            NeuronGroupHandler wuVarUpdateHandler ) const  [override], [virtual]
```

Generate platform-specific function to update the state of all neurons.

Parameters

os	CodeStream to write function to
model	model to generate code for
simHandler	callback to write platform-independent code to update an individual NeuronGroup
wuVarUpdateHandler	callback to write platform-independent code to update pre and postsynaptic weight update model variables when neuron spikes

Implements CodeGenerator::BackendBase.

19.1.2.24 genPopulationRNG()

```
MemAlloc CodeGenerator::CUDA::Backend::genPopulationRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const std::string & name,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.25 genPopVariableInit()

```
void CodeGenerator::CUDA::Backend::genPopVariableInit (
            CodeStream & os,
```

```
            VarLocation loc,
            const Substitutions & kernelSubs,
            Handler handler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.26 genRunnerPreamble()

```
void CodeGenerator::CUDA::Backend::genRunnerPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.27 genStepTimeFinalisePreamble()

```
void CodeGenerator::CUDA::Backend::genStepTimeFinalisePreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const  [override], [virtual]
```

After all timestep logic is complete.

Implements CodeGenerator::BackendBase.

19.1.2.28 genSynapseUpdate()

```
void CodeGenerator::CUDA::Backend::genSynapseUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            SynapseGroupHandler wumThreshHandler,
            SynapseGroupHandler wumSimHandler,
            SynapseGroupHandler wumEventHandler,
            SynapseGroupHandler postLearnHandler,
            SynapseGroupHandler synapseDynamicsHandler ) const  [override], [virtual]
```

Generate platform-specific function to update the state of all synapses.

Parameters

os	CodeStream to write function to
model	model to generate code for
wumThreshHandler	callback to write platform-independent code to update an individual NeuronGroup
wumSimHandler	callback to write platform-independent code to process presynaptic spikes. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
wumEventHandler	callback to write platform-independent code to process presynaptic spike-like events. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
postLearnHandler	callback to write platform-independent code to process postsynaptic spikes. "id_pre", "id_post" and "id_syn" variables will be provided to callback via Substitutions.
synapseDynamicsHandler	callback to write platform-independent code to update time-driven synapse dynamics. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.

Implements CodeGenerator::BackendBase.

19.1.2.29 genSynapseVariableRowInit()

```
void CodeGenerator::CUDA::Backend::genSynapseVariableRowInit (
            CodeStream & os,
            VarLocation loc,
            const SynapseGroupInternal & sg,
            const Substitutions & kernelSubs,
            Handler handler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.30 genTimer()

```
void CodeGenerator::CUDA::Backend::genTimer (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            CodeStream & stepTimeFinalise,
            const std::string & name,
            bool updateInStepTime ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.31 genVariableAllocation()

```
MemAlloc CodeGenerator::CUDA::Backend::genVariableAllocation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.32 genVariableDefinition()

```
void CodeGenerator::CUDA::Backend::genVariableDefinition (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.33 genVariableFree()

```
void CodeGenerator::CUDA::Backend::genVariableFree (
            CodeStream & os,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.34 genVariableImplementation()

```
void CodeGenerator::CUDA::Backend::genVariableImplementation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.35 genVariableInit()

```
void CodeGenerator::CUDA::Backend::genVariableInit (
            CodeStream & os,
            VarLocation loc,
            size_t count,
            const std::string & indexVarName,
            const Substitutions & kernelSubs,
            Handler handler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.36 genVariablePull()

```
void CodeGenerator::CUDA::Backend::genVariablePull (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.37 genVariablePush()

```
void CodeGenerator::CUDA::Backend::genVariablePush (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            bool autoInitialized,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.1.2.38 getChosenCUDADevice()

```
const cudaDeviceProp& CodeGenerator::CUDA::Backend::getChosenCUDADevice ( ) const  [inline]
```

19.1.2.39 getChosenDeviceID()

```
int CodeGenerator::CUDA::Backend::getChosenDeviceID ( ) const  [inline]
```

19.1.2.40 getDeviceMemoryBytes()

```
virtual size_t CodeGenerator::CUDA::Backend::getDeviceMemoryBytes ( ) const  [inline], [override],
[virtual]
```

How many bytes of memory does 'device' have.

Implements CodeGenerator::BackendBase.

19.1.2.41 getNumPostsynapticUpdateThreads()

```
size_t CodeGenerator::CUDA::Backend::getNumPostsynapticUpdateThreads (
             const SynapseGroupInternal & sg )  [static]
```

19.1.2.42 getNumPresynapticUpdateThreads()

```
size_t CodeGenerator::CUDA::Backend::getNumPresynapticUpdateThreads (
             const SynapseGroupInternal & sg )  [static]
```

19.1.2.43 getNumSynapseDynamicsThreads()

```
size_t CodeGenerator::CUDA::Backend::getNumSynapseDynamicsThreads (
             const SynapseGroupInternal & sg )  [static]
```

19.1.2.44 getNVCCFlags()

```
std::string CodeGenerator::CUDA::Backend::getNVCCFlags ( ) const
```

19.1.2.45 getRuntimeVersion()

```
int CodeGenerator::CUDA::Backend::getRuntimeVersion ( ) const  [inline]
```

19.1.2.46 getVarPrefix()

```
virtual std::string CodeGenerator::CUDA::Backend::getVarPrefix ( ) const  [inline], [override],
[virtual]
```

When backends require separate 'device' and 'host' versions of variables, they are identified with a prefix. This function returns this prefix so it can be used in otherwise platform-independent code.

Reimplemented from CodeGenerator::BackendBase.

19.1.2.47 isGlobalRNGRequired()

```
bool CodeGenerator::CUDA::Backend::isGlobalRNGRequired (
             const ModelSpecInternal & model ) const  [override], [virtual]
```

Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?

Implements CodeGenerator::BackendBase.

19.1.2.48 isPostsynapticRemapRequired()

`virtual bool CodeGenerator::CUDA::Backend::isPostsynapticRemapRequired ( ) const [inline], [override], [virtual]`

Implements CodeGenerator::BackendBase.

19.1.2.49 isSynRemapRequired()

`virtual bool CodeGenerator::CUDA::Backend::isSynRemapRequired ( ) const [inline], [override], [virtual]`

Implements CodeGenerator::BackendBase.

19.1.3 Member Data Documentation

19.1.3.1 KernelNames

`const char * CodeGenerator::CUDA::Backend::KernelNames [static]`

Initial value:

```
= {
    "updateNeuronsKernel",
    "updatePresynapticKernel",
    "updatePostsynapticKernel",
    "updateSynapseDynamicsKernel",
    "initializeKernel",
    "initializeSparseKernel",
    "preNeuronResetKernel",
    "preSynapseResetKernel"}
```

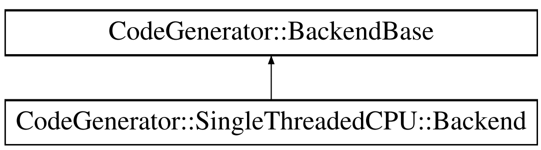
The documentation for this class was generated from the following files:

- cuda/backend.h
- cuda/backend.cc

19.2 CodeGenerator::SingleThreadedCPU::Backend Class Reference

```
#include <backend.h>
```

Inheritance diagram for CodeGenerator::SingleThreadedCPU::Backend:



Public Member Functions

- Backend (int localHostID, const std::string &scalarType, const Preferences &preferences)

- virtual void genNeuronUpdate (CodeStream &os, const ModelSpecInternal &model, NeuronGroupSimHandler simHandler, NeuronGroupHandler wuVarUpdateHandler) const override

  Generate platform-specific function to update the state of all neurons.
- virtual void genSynapseUpdate (CodeStream &os, const ModelSpecInternal &model, SynapseGroupHandler wumThreshHandler, SynapseGroupHandler wumSimHandler, SynapseGroupHandler wumEventHandler, SynapseGroupHandler postLearnHandler, SynapseGroupHandler synapseDynamicsHandler) const override

  Generate platform-specific function to update the state of all synapses.
- virtual void genInit (CodeStream &os, const ModelSpecInternal &model, NeuronGroupHandler localNGHandler, NeuronGroupHandler remoteNGHandler, SynapseGroupHandler sgDenseInitHandler, SynapseGroupHandler sgSparseConnectHandler, SynapseGroupHandler sgSparseInitHandler) const override
- virtual void genDefinitionsPreamble (CodeStream &os) const override

  Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.
- virtual void genDefinitionsInternalPreamble (CodeStream &os) const override

  Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.
- virtual void genRunnerPreamble (CodeStream &os) const override
- virtual void genAllocateMemPreamble (CodeStream &os, const ModelSpecInternal &model) const override
- virtual void genStepTimeFinalisePreamble (CodeStream &os, const ModelSpecInternal &model) const override

  After all timestep logic is complete.
- virtual void genVariableDefinition (CodeStream &definitions, CodeStream &definitionsInternal, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genVariableImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual MemAlloc genVariableAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const override
- virtual void genVariableFree (CodeStream &os, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamDefinition (CodeStream &definitions, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamPush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genExtraGlobalParamPull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const override
- virtual void genPopVariableInit (CodeStream &os, VarLocation loc, const Substitutions &kernelSubs, Handler handler) const override
- virtual void genVariableInit (CodeStream &os, VarLocation loc, size_t count, const std::string &indexVarName, const Substitutions &kernelSubs, Handler handler) const override
- virtual void genSynapseVariableRowInit (CodeStream &os, VarLocation loc, const SynapseGroupInternal &sg, const Substitutions &kernelSubs, Handler handler) const override
- virtual void genCurrentTrueSpikePush (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentTrueSpikePull (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentSpikeLikeEventPush (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genCurrentSpikeLikeEventPull (CodeStream &os, const NeuronGroupInternal &ng) const override
- virtual void genVariablePush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, bool autoInitialized, size_t count) const override
- virtual void genVariablePull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const override
- virtual MemAlloc genGlobalRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const ModelSpecInternal &model) const override

- virtual MemAlloc genPopulationRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const std::string &name, size_t count) const override
- virtual void genTimer (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, CodeStream &stepTimeFinalise, const std::string &name, bool updateInStepTime) const override
- virtual void genMakefilePreamble (std::ostream &os) const override

    This function can be used to generate a preamble for the GNU makefile used to build.
- virtual void genMakefileLinkRule (std::ostream &os) const override
- virtual void genMakefileCompileRule (std::ostream &os) const override
- virtual void genMSBuildConfigProperties (std::ostream &os) const override
- virtual void genMSBuildImportProps (std::ostream &os) const override
- virtual void genMSBuildItemDefinitions (std::ostream &os) const override
- virtual void genMSBuildCompileModule (const std::string &moduleName, std::ostream &os) const override
- virtual void genMSBuildImportTarget (std::ostream &os) const override
- virtual std::string getVarPrefix () const override
- virtual bool isGlobalRNGRequired (const ModelSpecInternal &model) const override

    Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?
- virtual bool isSynRemapRequired () const override
- virtual bool isPostsynapticRemapRequired () const override
- virtual size_t getDeviceMemoryBytes () const override

    How many bytes of memory does 'device' have.

Additional Inherited Members

19.2.1 Constructor & Destructor Documentation

19.2.1.1 Backend()

```
CodeGenerator::SingleThreadedCPU::Backend::Backend (
            int localHostID,
            const std::string & scalarType,
            const Preferences & preferences ) [inline]
```

19.2.2 Member Function Documentation

19.2.2.1 genAllocateMemPreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genAllocateMemPreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const [override], [virtual]
```

Allocate memory is the first function in GeNN generated code called by usercode and it should only ever be called once. Therefore it's a good place for any global initialisation. This function generates a 'preamble' to this function.

Implements CodeGenerator::BackendBase.

19.2.2.2 genCurrentSpikeLikeEventPull()

```
void CodeGenerator::SingleThreadedCPU::Backend::genCurrentSpikeLikeEventPull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.3 genCurrentSpikeLikeEventPush()

```
void CodeGenerator::SingleThreadedCPU::Backend::genCurrentSpikeLikeEventPush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.4 genCurrentTrueSpikePull()

```
void CodeGenerator::SingleThreadedCPU::Backend::genCurrentTrueSpikePull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.5 genCurrentTrueSpikePush()

```
void CodeGenerator::SingleThreadedCPU::Backend::genCurrentTrueSpikePush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.6 genDefinitionsInternalPreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genDefinitionsInternalPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.

This will only be included by the platform-specific compiler used to build this backend so can include platform-specific types or headers

Implements CodeGenerator::BackendBase.

19.2.2.7 genDefinitionsPreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genDefinitionsPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.

This will be included from a standard C++ compiler so shouldn't include any platform-specific types or headers

Implements CodeGenerator::BackendBase.

19.2.2.8 genExtraGlobalParamAllocation()

```
void CodeGenerator::SingleThreadedCPU::Backend::genExtraGlobalParamAllocation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.9 genExtraGlobalParamDefinition()

```
void CodeGenerator::SingleThreadedCPU::Backend::genExtraGlobalParamDefinition (
            CodeStream & definitions,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.10 genExtraGlobalParamImplementation()

```
void CodeGenerator::SingleThreadedCPU::Backend::genExtraGlobalParamImplementation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.11 genExtraGlobalParamPull()

```
void CodeGenerator::SingleThreadedCPU::Backend::genExtraGlobalParamPull (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.12 genExtraGlobalParamPush()

```
void CodeGenerator::SingleThreadedCPU::Backend::genExtraGlobalParamPush (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.13 genGlobalRNG()

```
MemAlloc CodeGenerator::SingleThreadedCPU::Backend::genGlobalRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
```

```
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const ModelSpecInternal & model ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.14 genInit()

```
void CodeGenerator::SingleThreadedCPU::Backend::genInit (
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupHandler localNGHandler,
            NeuronGroupHandler remoteNGHandler,
            SynapseGroupHandler sgDenseInitHandler,
            SynapseGroupHandler sgSparseConnectHandler,
            SynapseGroupHandler sgSparseInitHandler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.15 genMakefileCompileRule()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMakefileCompileRule (
            std::ostream & os ) const  [override], [virtual]
```

The GNU make build system uses 'pattern rules' (`https://www.gnu.org/software/make/manual/html_node/Pattern-Intro.html`) to build backend modules into objects. This function should generate a GNU make pattern rule capable of building each module (i.e. compiling .cc file $< into .o file $@).

Implements CodeGenerator::BackendBase.

19.2.2.16 genMakefileLinkRule()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMakefileLinkRule (
            std::ostream & os ) const  [override], [virtual]
```

The GNU make build system will populate a variable called with a list of objects to link. This function should generate a GNU make rule to build these objects into a shared library.

Implements CodeGenerator::BackendBase.

19.2.2.17 genMakefilePreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMakefilePreamble (
            std::ostream & os ) const  [override], [virtual]
```

This function can be used to generate a preamble for the GNU makefile used to build.

Implements CodeGenerator::BackendBase.

19.2.2.18 genMSBuildCompileModule()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMSBuildCompileModule (
            const std::string & moduleName,
            std::ostream & os ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.19 genMSBuildConfigProperties()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMSBuildConfigProperties (
            std::ostream & os ) const [override], [virtual]
```

In MSBuild, 'properties' are used to configure global project settings e.g. whether the MSBuild project builds a static or dynamic library This function can be used to add additional XML properties to this section.

see https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-properties for more information.

Implements CodeGenerator::BackendBase.

19.2.2.20 genMSBuildImportProps()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMSBuildImportProps (
            std::ostream & os ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.21 genMSBuildImportTarget()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMSBuildImportTarget (
            std::ostream & os ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.22 genMSBuildItemDefinitions()

```
void CodeGenerator::SingleThreadedCPU::Backend::genMSBuildItemDefinitions (
            std::ostream & os ) const [override], [virtual]
```

In MSBuild, the 'item definitions' are used to override the default properties of 'items' such as `<ClCompile>` or `<Link>`. This function should generate XML to correctly configure the 'items' required to build the generated code, taking into account etc.

see https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-items#item-definitions for more information.

Implements CodeGenerator::BackendBase.

19.2.2.23 genNeuronUpdate()

```
void CodeGenerator::SingleThreadedCPU::Backend::genNeuronUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupSimHandler simHandler,
            NeuronGroupHandler wuVarUpdateHandler ) const [override], [virtual]
```

Generate platform-specific function to update the state of all neurons.

Parameters

os	CodeStream to write function to
model	model to generate code for
simHandler	callback to write platform-independent code to update an individual NeuronGroup

Parameters

wuVarUpdateHandler	callback to write platform-independent code to update pre and postsynaptic weight update model variables when neuron spikes

Implements CodeGenerator::BackendBase.

19.2.2.24 genPopulationRNG()

```
MemAlloc CodeGenerator::SingleThreadedCPU::Backend::genPopulationRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const std::string & name,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.25 genPopVariableInit()

```
void CodeGenerator::SingleThreadedCPU::Backend::genPopVariableInit (
            CodeStream & os,
            VarLocation loc,
            const Substitutions & kernelSubs,
            Handler handler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.26 genRunnerPreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genRunnerPreamble (
            CodeStream & os ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.27 genStepTimeFinalisePreamble()

```
void CodeGenerator::SingleThreadedCPU::Backend::genStepTimeFinalisePreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const  [override], [virtual]
```

After all timestep logic is complete.

Implements CodeGenerator::BackendBase.

19.2.2.28 genSynapseUpdate()

```
void CodeGenerator::SingleThreadedCPU::Backend::genSynapseUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            SynapseGroupHandler wumThreshHandler,
            SynapseGroupHandler wumSimHandler,
```

```
            SynapseGroupHandler wumEventHandler,
            SynapseGroupHandler postLearnHandler,
            SynapseGroupHandler synapseDynamicsHandler ) const [override], [virtual]
```

Generate platform-specific function to update the state of all synapses.

Parameters

os	CodeStream to write function to
model	model to generate code for
wumThreshHandler	callback to write platform-independent code to update an individual NeuronGroup
wumSimHandler	callback to write platform-independent code to process presynaptic spikes. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
wumEventHandler	callback to write platform-independent code to process presynaptic spike-like events. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
postLearnHandler	callback to write platform-independent code to process postsynaptic spikes. "id_pre", "id_post" and "id_syn" variables will be provided to callback via Substitutions.
synapseDynamicsHandler	callback to write platform-independent code to update time-driven synapse dynamics. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.

Implements CodeGenerator::BackendBase.

19.2.2.29 genSynapseVariableRowInit()

```
void CodeGenerator::SingleThreadedCPU::Backend::genSynapseVariableRowInit (
            CodeStream & os,
            VarLocation loc,
            const SynapseGroupInternal & sg,
            const Substitutions & kernelSubs,
            Handler handler ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.30 genTimer()

```
void CodeGenerator::SingleThreadedCPU::Backend::genTimer (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            CodeStream & stepTimeFinalise,
            const std::string & name,
            bool updateInStepTime ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.31 genVariableAllocation()

```
MemAlloc CodeGenerator::SingleThreadedCPU::Backend::genVariableAllocation (
```

```
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            size_t count ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.32 genVariableDefinition()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariableDefinition (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.33 genVariableFree()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariableFree (
            CodeStream & os,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.34 genVariableImplementation()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariableImplementation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.35 genVariableInit()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariableInit (
            CodeStream & os,
            VarLocation loc,
            size_t count,
            const std::string & indexVarName,
            const Substitutions & kernelSubs,
            Handler handler ) const  [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.36 genVariablePull()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariablePull (
            CodeStream & os,
            const std::string & type,
```

```
const std::string & name,
VarLocation loc,
size_t count ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.37 genVariablePush()

```
void CodeGenerator::SingleThreadedCPU::Backend::genVariablePush (
    CodeStream & os,
    const std::string & type,
    const std::string & name,
    VarLocation loc,
    bool autoInitialized,
    size_t count ) const [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.38 getDeviceMemoryBytes()

```
virtual size_t CodeGenerator::SingleThreadedCPU::Backend::getDeviceMemoryBytes ( ) const [inline],
[override], [virtual]
```

How many bytes of memory does 'device' have.

Implements CodeGenerator::BackendBase.

19.2.2.39 getVarPrefix()

```
virtual std::string CodeGenerator::SingleThreadedCPU::Backend::getVarPrefix ( ) const [inline],
[override], [virtual]
```

When backends require separate 'device' and 'host' versions of variables, they are identified with a prefix. This function returns this prefix so it can be used in otherwise platform-independent code.

Reimplemented from CodeGenerator::BackendBase.

19.2.2.40 isGlobalRNGRequired()

```
bool CodeGenerator::SingleThreadedCPU::Backend::isGlobalRNGRequired (
    const ModelSpecInternal & model ) const [override], [virtual]
```

Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?

Implements CodeGenerator::BackendBase.

19.2.2.41 isPostsynapticRemapRequired()

```
virtual bool CodeGenerator::SingleThreadedCPU::Backend::isPostsynapticRemapRequired ( ) const
[inline], [override], [virtual]
```

Implements CodeGenerator::BackendBase.

19.2.2.42 isSynRemapRequired()

```
virtual bool CodeGenerator::SingleThreadedCPU::Backend::isSynRemapRequired ( ) const [inline],
[override], [virtual]
```

Implements CodeGenerator::BackendBase.

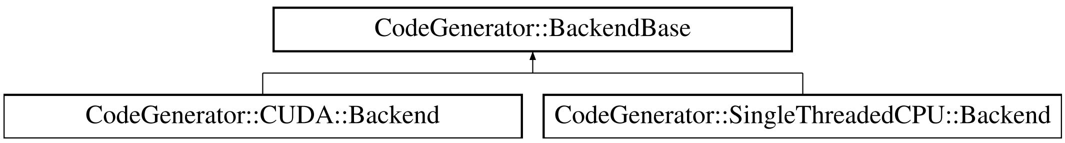
The documentation for this class was generated from the following files:

- single_threaded_cpu/backend.h
- single_threaded_cpu/backend.cc

19.3 CodeGenerator::BackendBase Class Reference

```
#include <backendBase.h>
```

Inheritance diagram for CodeGenerator::BackendBase:



Public Types

- typedef std::function< void(CodeStream &, Substitutions &)> Handler
- template<typename T >
  using GroupHandler = std::function< void(CodeStream &, const T &, Substitutions &)>
- typedef GroupHandler< NeuronGroupInternal > NeuronGroupHandler

  Standard callback type which provides a CodeStream to write platform-independent code for the specified NeuronGroup to.
- typedef GroupHandler< SynapseGroupInternal > SynapseGroupHandler

  Standard callback type which provides a CodeStream to write platform-independent code for the specified SynapseGroup to.
- typedef std::function< void(CodeStream &, const NeuronGroupInternal &, Substitutions &, NeuronGroupHandler, NeuronGroupHandler)> NeuronGroupSimHandler

  Callback function type for generation neuron group simulation code.

Public Member Functions

- BackendBase (int localHostID, const std::string &scalarType)
- virtual ∼BackendBase ()
- virtual void genNeuronUpdate (CodeStream &os, const ModelSpecInternal &model, NeuronGroupSimHandler simHandler, NeuronGroupHandler wuVarUpdateHandler) const =0

  Generate platform-specific function to update the state of all neurons.
- virtual void genSynapseUpdate (CodeStream &os, const ModelSpecInternal &model, SynapseGroupHandler wumThreshHandler, SynapseGroupHandler wumSimHandler, SynapseGroupHandler wumEventHandler, SynapseGroupHandler postLearnHandler, SynapseGroupHandler synapseDynamicsHandler) const =0

  Generate platform-specific function to update the state of all synapses.
- virtual void genInit (CodeStream &os, const ModelSpecInternal &model, NeuronGroupHandler localNGHandler, NeuronGroupHandler remoteNGHandler, SynapseGroupHandler sgDenseInitHandler, SynapseGroupHandler sgSparseConnectHandler, SynapseGroupHandler sgSparseInitHandler) const =0
- virtual void genDefinitionsPreamble (CodeStream &os) const =0

Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.

- virtual void genDefinitionsInternalPreamble (CodeStream &os) const =0

  Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.

- virtual void genRunnerPreamble (CodeStream &os) const =0
- virtual void genAllocateMemPreamble (CodeStream &os, const ModelSpecInternal &model) const =0
- virtual void genStepTimeFinalisePreamble (CodeStream &os, const ModelSpecInternal &model) const =0

  After all timestep logic is complete.

- virtual void genVariableDefinition (CodeStream &definitions, CodeStream &definitionsInternal, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genVariableImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual MemAlloc genVariableAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const =0
- virtual void genVariableFree (CodeStream &os, const std::string &name, VarLocation loc) const =0
- virtual void genExtraGlobalParamDefinition (CodeStream &definitions, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genExtraGlobalParamImplementation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genExtraGlobalParamAllocation (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genExtraGlobalParamPush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genExtraGlobalParamPull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc) const =0
- virtual void genPopVariableInit (CodeStream &os, VarLocation loc, const Substitutions &kernelSubs, Handler handler) const =0
- virtual void genVariableInit (CodeStream &os, VarLocation loc, size_t count, const std::string &indexVarName, const Substitutions &kernelSubs, Handler handler) const =0
- virtual void genSynapseVariableRowInit (CodeStream &os, VarLocation loc, const SynapseGroupInternal &sg, const Substitutions &kernelSubs, Handler handler) const =0
- virtual void genVariablePush (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, bool autoInitialized, size_t count) const =0
- virtual void genVariablePull (CodeStream &os, const std::string &type, const std::string &name, VarLocation loc, size_t count) const =0
- virtual void genCurrentTrueSpikePush (CodeStream &os, const NeuronGroupInternal &ng) const =0
- virtual void genCurrentTrueSpikePull (CodeStream &os, const NeuronGroupInternal &ng) const =0
- virtual void genCurrentSpikeLikeEventPush (CodeStream &os, const NeuronGroupInternal &ng) const =0
- virtual void genCurrentSpikeLikeEventPull (CodeStream &os, const NeuronGroupInternal &ng) const =0
- virtual MemAlloc genGlobalRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const ModelSpecInternal &model) const =0
- virtual MemAlloc genPopulationRNG (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const std::string &name, size_t count) const =0
- virtual void genTimer (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, CodeStream &stepTimeFinalise, const std::string &name, bool updateInStepTime) const =0
- virtual void genMakefilePreamble (std::ostream &os) const =0

  This function can be used to generate a preamble for the GNU makefile used to build.

- virtual void genMakefileLinkRule (std::ostream &os) const =0
- virtual void genMakefileCompileRule (std::ostream &os) const =0
- virtual void genMSBuildConfigProperties (std::ostream &os) const =0
- virtual void genMSBuildImportProps (std::ostream &os) const =0
- virtual void genMSBuildItemDefinitions (std::ostream &os) const =0

- virtual void genMSBuildCompileModule (const std::string &moduleName, std::ostream &os) const =0
- virtual void genMSBuildImportTarget (std::ostream &os) const =0
- virtual std::string getVarPrefix () const
- virtual bool isGlobalRNGRequired (const ModelSpecInternal &model) const =0

    Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?
- virtual bool isSynRemapRequired () const =0
- virtual bool isPostsynapticRemapRequired () const =0
- virtual size_t getDeviceMemoryBytes () const =0

    How many bytes of memory does 'device' have.
- void genVariablePushPull (CodeStream &push, CodeStream &pull, const std::string &type, const std::string &name, VarLocation loc, bool autoInitialized, size_t count) const

    Helper function to generate matching push and pull functions for a variable.
- MemAlloc genArray (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, CodeStream &allocations, CodeStream &free, const std::string &type, const std::string &name, VarLocation loc, size_t count) const

    Helper function to generate matching definition, declaration, allocation and free code for an array.
- void genScalar (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, const std::string &type, const std::string &name, VarLocation loc) const

    Helper function to generate matching definition and declaration code for a scalar variable.
- int getLocalHostID () const

    Gets ID of local host backend is building code for.

Protected Member Functions

- void addType (const std::string &type, size_t size)
- size_t getSize (const std::string &type) const

19.3.1 Member Typedef Documentation

19.3.1.1 GroupHandler

```
template<typename T >
using CodeGenerator::BackendBase::GroupHandler = std::function <void(CodeStream &, const T &,
Substitutions&)>
```

19.3.1.2 Handler

```
typedef std::function<void(CodeStream &, Substitutions&)> CodeGenerator::BackendBase::Handler
```

19.3.1.3 NeuronGroupHandler

```
typedef GroupHandler<NeuronGroupInternal> CodeGenerator::BackendBase::NeuronGroupHandler
```

Standard callback type which provides a CodeStream to write platform-independent code for the specified Neuron↩ Group to.

19.3.1.4 NeuronGroupSimHandler

```
typedef std::function<void(CodeStream &, const NeuronGroupInternal &, Substitutions&, Neuron←↩
GroupHandler, NeuronGroupHandler)> CodeGenerator::BackendBase::NeuronGroupSimHandler
```

Callback function type for generation neuron group simulation code.

Provides additional callbacks to insert code to emit spikes

19.3.1.5 SynapseGroupHandler

```
typedef GroupHandler<SynapseGroupInternal> CodeGenerator::BackendBase::SynapseGroupHandler
```

Standard callback type which provides a CodeStream to write platform-independent code for the specified SynapseGroup to.

19.3.2 Constructor & Destructor Documentation

19.3.2.1 BackendBase()

```
CodeGenerator::BackendBase::BackendBase (
            int localHostID,
            const std::string & scalarType )
```

19.3.2.2 ∼BackendBase()

```
virtual CodeGenerator::BackendBase::~BackendBase ( ) [inline], [virtual]
```

19.3.3 Member Function Documentation

19.3.3.1 addType()

```
void CodeGenerator::BackendBase::addType (
            const std::string & type,
            size_t size ) [inline], [protected]
```

19.3.3.2 genAllocateMemPreamble()

```
virtual void CodeGenerator::BackendBase::genAllocateMemPreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const [pure virtual]
```

Allocate memory is the first function in GeNN generated code called by usercode and it should only ever be called once. Therefore it's a good place for any global initialisation. This function generates a 'preamble' to this function.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.3 genArray()

```
MemAlloc CodeGenerator::BackendBase::genArray (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
```

```
            CodeStream & allocations,
            CodeStream & free,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            size_t count ) const [inline]
```

Helper function to generate matching definition, declaration, allocation and free code for an array.

19.3.3.4 genCurrentSpikeLikeEventPull()

```
virtual void CodeGenerator::BackendBase::genCurrentSpikeLikeEventPull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.5 genCurrentSpikeLikeEventPush()

```
virtual void CodeGenerator::BackendBase::genCurrentSpikeLikeEventPush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.6 genCurrentTrueSpikePull()

```
virtual void CodeGenerator::BackendBase::genCurrentTrueSpikePull (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.7 genCurrentTrueSpikePush()

```
virtual void CodeGenerator::BackendBase::genCurrentTrueSpikePush (
            CodeStream & os,
            const NeuronGroupInternal & ng ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.8 genDefinitionsInternalPreamble()

```
virtual void CodeGenerator::BackendBase::genDefinitionsInternalPreamble (
            CodeStream & os ) const [pure virtual]
```

Definitions internal is the internal header file for the generated code. This function generates a 'preamble' to this header file.

This will only be included by the platform-specific compiler used to build this backend so can include platform-specific types or headers

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.9 genDefinitionsPreamble()

```
virtual void CodeGenerator::BackendBase::genDefinitionsPreamble (
            CodeStream & os ) const  [pure virtual]
```

Definitions is the usercode-facing header file for the generated code. This function generates a 'preamble' to this header file.

This will be included from a standard C++ compiler so shouldn't include any platform-specific types or headers

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.10 genExtraGlobalParamAllocation()

```
virtual void CodeGenerator::BackendBase::genExtraGlobalParamAllocation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.11 genExtraGlobalParamDefinition()

```
virtual void CodeGenerator::BackendBase::genExtraGlobalParamDefinition (
            CodeStream & definitions,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.12 genExtraGlobalParamImplementation()

```
virtual void CodeGenerator::BackendBase::genExtraGlobalParamImplementation (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.13 genExtraGlobalParamPull()

```
virtual void CodeGenerator::BackendBase::genExtraGlobalParamPull (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.14 genExtraGlobalParamPush()

```
virtual void CodeGenerator::BackendBase::genExtraGlobalParamPush (
            CodeStream & os,
```

```
const std::string & type,
const std::string & name,
VarLocation loc ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.15 genGlobalRNG()

```
virtual MemAlloc CodeGenerator::BackendBase::genGlobalRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const ModelSpecInternal & model ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.16 genInit()

```
virtual void CodeGenerator::BackendBase::genInit (
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupHandler localNGHandler,
            NeuronGroupHandler remoteNGHandler,
            SynapseGroupHandler sgDenseInitHandler,
            SynapseGroupHandler sgSparseConnectHandler,
            SynapseGroupHandler sgSparseInitHandler ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.17 genMakefileCompileRule()

```
virtual void CodeGenerator::BackendBase::genMakefileCompileRule (
            std::ostream & os ) const [pure virtual]
```

The GNU make build system uses 'pattern rules' (https://www.gnu.org/software/make/manual/html↩_node/Pattern-Intro.html) to build backend modules into objects. This function should generate a GNU make pattern rule capable of building each module (i.e. compiling .cc file $< into .o file $@).

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.18 genMakefileLinkRule()

```
virtual void CodeGenerator::BackendBase::genMakefileLinkRule (
            std::ostream & os ) const [pure virtual]
```

The GNU make build system will populate a variable called with a list of objects to link. This function should generate a GNU make rule to build these objects into a shared library.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.19 genMakefilePreamble()

```
virtual void CodeGenerator::BackendBase::genMakefilePreamble (
            std::ostream & os ) const [pure virtual]
```

This function can be used to generate a preamble for the GNU makefile used to build.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.20 genMSBuildCompileModule()

```
virtual void CodeGenerator::BackendBase::genMSBuildCompileModule (
            const std::string & moduleName,
            std::ostream & os ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.21 genMSBuildConfigProperties()

```
virtual void CodeGenerator::BackendBase::genMSBuildConfigProperties (
            std::ostream & os ) const [pure virtual]
```

In MSBuild, 'properties' are used to configure global project settings e.g. whether the MSBuild project builds a static or dynamic library This function can be used to add additional XML properties to this section.

see `https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-properties` for more information.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.22 genMSBuildImportProps()

```
virtual void CodeGenerator::BackendBase::genMSBuildImportProps (
            std::ostream & os ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.23 genMSBuildImportTarget()

```
virtual void CodeGenerator::BackendBase::genMSBuildImportTarget (
            std::ostream & os ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.24 genMSBuildItemDefinitions()

```
virtual void CodeGenerator::BackendBase::genMSBuildItemDefinitions (
            std::ostream & os ) const [pure virtual]
```

In MSBuild, the 'item definitions' are used to override the default properties of 'items' such as `<ClCompile>` or `<Link>`. This function should generate XML to correctly configure the 'items' required to build the generated code, taking into account etc.

see `https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-items#item-definitions` for more information.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.25 genNeuronUpdate()

```
virtual void CodeGenerator::BackendBase::genNeuronUpdate (
```

```
            CodeStream & os,
            const ModelSpecInternal & model,
            NeuronGroupSimHandler simHandler,
            NeuronGroupHandler wuVarUpdateHandler ) const  [pure virtual]
```

Generate platform-specific function to update the state of all neurons.

Parameters

os	CodeStream to write function to
model	model to generate code for
simHandler	callback to write platform-independent code to update an individual NeuronGroup
wuVarUpdateHandler	callback to write platform-independent code to update pre and postsynaptic weight update model variables when neuron spikes

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.26 genPopulationRNG()

```
virtual MemAlloc CodeGenerator::BackendBase::genPopulationRNG (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            CodeStream & allocations,
            CodeStream & free,
            const std::string & name,
            size_t count ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.27 genPopVariableInit()

```
virtual void CodeGenerator::BackendBase::genPopVariableInit (
            CodeStream & os,
            VarLocation loc,
            const Substitutions & kernelSubs,
            Handler handler ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.28 genRunnerPreamble()

```
virtual void CodeGenerator::BackendBase::genRunnerPreamble (
            CodeStream & os ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.29 genScalar()

```
void CodeGenerator::BackendBase::genScalar (
            CodeStream & definitions,
            CodeStream & definitionsInternal,
            CodeStream & runner,
            const std::string & type,
```

```
            const std::string & name,
            VarLocation loc ) const  [inline]
```

Helper function to generate matching definition and declaration code for a scalar variable.

19.3.3.30 genStepTimeFinalisePreamble()

```
virtual void CodeGenerator::BackendBase::genStepTimeFinalisePreamble (
            CodeStream & os,
            const ModelSpecInternal & model ) const  [pure virtual]
```

After all timestep logic is complete.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.31 genSynapseUpdate()

```
virtual void CodeGenerator::BackendBase::genSynapseUpdate (
            CodeStream & os,
            const ModelSpecInternal & model,
            SynapseGroupHandler wumThreshHandler,
            SynapseGroupHandler wumSimHandler,
            SynapseGroupHandler wumEventHandler,
            SynapseGroupHandler postLearnHandler,
            SynapseGroupHandler synapseDynamicsHandler ) const  [pure virtual]
```

Generate platform-specific function to update the state of all synapses.

Parameters

os	CodeStream to write function to
model	model to generate code for
wumThreshHandler	callback to write platform-independent code to update an individual NeuronGroup
wumSimHandler	callback to write platform-independent code to process presynaptic spikes. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
wumEventHandler	callback to write platform-independent code to process presynaptic spike-like events. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.
postLearnHandler	callback to write platform-independent code to process postsynaptic spikes. "id_pre", "id_post" and "id_syn" variables will be provided to callback via Substitutions.
synapseDynamicsHandler	callback to write platform-independent code to update time-driven synapse dynamics. "id_pre", "id_post" and "id_syn" variables; and either "addToInSynDelay" or "addToInSyn" function will be provided to callback via Substitutions.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.32 genSynapseVariableRowInit()

```
virtual void CodeGenerator::BackendBase::genSynapseVariableRowInit (
            CodeStream & os,
            VarLocation loc,
            const SynapseGroupInternal & sg,
```

```
const Substitutions & kernelSubs,
Handler handler ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.33 genTimer()

```
virtual void CodeGenerator::BackendBase::genTimer (
        CodeStream & definitions,
        CodeStream & definitionsInternal,
        CodeStream & runner,
        CodeStream & allocations,
        CodeStream & free,
        CodeStream & stepTimeFinalise,
        const std::string & name,
        bool updateInStepTime ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.34 genVariableAllocation()

```
virtual MemAlloc CodeGenerator::BackendBase::genVariableAllocation (
        CodeStream & os,
        const std::string & type,
        const std::string & name,
        VarLocation loc,
        size_t count ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.35 genVariableDefinition()

```
virtual void CodeGenerator::BackendBase::genVariableDefinition (
        CodeStream & definitions,
        CodeStream & definitionsInternal,
        const std::string & type,
        const std::string & name,
        VarLocation loc ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.36 genVariableFree()

```
virtual void CodeGenerator::BackendBase::genVariableFree (
        CodeStream & os,
        const std::string & name,
        VarLocation loc ) const [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.37 genVariableImplementation()

```
virtual void CodeGenerator::BackendBase::genVariableImplementation (
        CodeStream & os,
        const std::string & type,
```

```
            const std::string & name,
            VarLocation loc ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.38 genVariableInit()

```
virtual void CodeGenerator::BackendBase::genVariableInit (
            CodeStream & os,
            VarLocation loc,
            size_t count,
            const std::string & indexVarName,
            const Substitutions & kernelSubs,
            Handler handler ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.39 genVariablePull()

```
virtual void CodeGenerator::BackendBase::genVariablePull (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            size_t count ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.40 genVariablePush()

```
virtual void CodeGenerator::BackendBase::genVariablePush (
            CodeStream & os,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            bool autoInitialized,
            size_t count ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.41 genVariablePushPull()

```
void CodeGenerator::BackendBase::genVariablePushPull (
            CodeStream & push,
            CodeStream & pull,
            const std::string & type,
            const std::string & name,
            VarLocation loc,
            bool autoInitialized,
            size_t count ) const  [inline]
```

Helper function to generate matching push and pull functions for a variable.

19.3.3.42 getDeviceMemoryBytes()

```
virtual size_t CodeGenerator::BackendBase::getDeviceMemoryBytes ( ) const  [pure virtual]
```

How many bytes of memory does 'device' have.

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.43 getLocalHostID()

```
int CodeGenerator::BackendBase::getLocalHostID ( ) const  [inline]
```

Gets ID of local host backend is building code for.

19.3.3.44 getSize()

```
size_t CodeGenerator::BackendBase::getSize (
            const std::string & type ) const  [protected]
```

19.3.3.45 getVarPrefix()

```
virtual std::string CodeGenerator::BackendBase::getVarPrefix ( ) const  [inline], [virtual]
```

When backends require separate 'device' and 'host' versions of variables, they are identified with a prefix. This function returns this prefix so it can be used in otherwise platform-independent code.

Reimplemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.46 isGlobalRNGRequired()

```
virtual bool CodeGenerator::BackendBase::isGlobalRNGRequired (
            const ModelSpecInternal & model ) const  [pure virtual]
```

Different backends use different RNGs for different things. Does this one require a global RNG for the specified model?

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.47 isPostsynapticRemapRequired()

```
virtual bool CodeGenerator::BackendBase::isPostsynapticRemapRequired ( ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

19.3.3.48 isSynRemapRequired()

```
virtual bool CodeGenerator::BackendBase::isSynRemapRequired ( ) const  [pure virtual]
```

Implemented in CodeGenerator::CUDA::Backend, and CodeGenerator::SingleThreadedCPU::Backend.

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

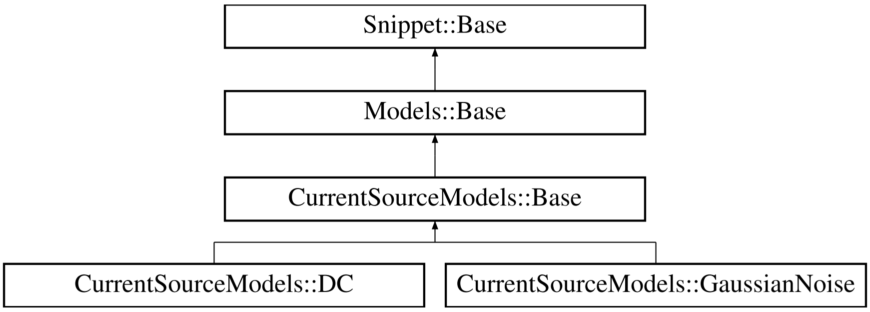
- backendBase.h
- backendBase.cc

19.4 CurrentSourceModels::Base Class Reference

Base class for all current source models.

`#include <currentSourceModels.h>`

Inheritance diagram for CurrentSourceModels::Base:



Public Member Functions

- virtual std::string getInjectionCode () const

    Gets the code that defines current injected each timestep.

Additional Inherited Members

19.4.1 Detailed Description

Base class for all current source models.

19.4.2 Member Function Documentation

19.4.2.1 getInjectionCode()

`virtual std::string CurrentSourceModels::Base::getInjectionCode ( ) const [inline], [virtual]`

Gets the code that defines current injected each timestep.

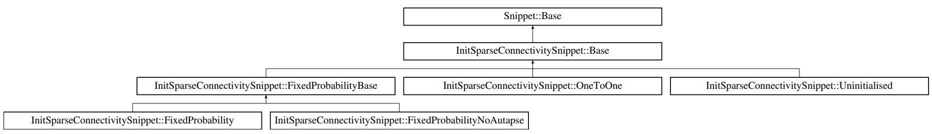
The documentation for this class was generated from the following file:

- currentSourceModels.h

19.5 InitSparseConnectivitySnippet::Base Class Reference

`#include <initSparseConnectivitySnippet.h>`

Inheritance diagram for InitSparseConnectivitySnippet::Base:



Public Types

- typedef std::function< unsigned int(unsigned int, unsigned int, const std::vector< double > &)> CalcMax←
LengthFunc

Public Member Functions

- virtual std::string getRowBuildCode () const
- virtual ParamValVec getRowBuildStateVars () const
- virtual CalcMaxLengthFunc getCalcMaxRowLengthFunc () const

  Get function to calculate the maximum row length of this connector based on the parameters and the size of the pre and postsynaptic population.
- virtual CalcMaxLengthFunc getCalcMaxColLengthFunc () const

  Get function to calculate the maximum column length of this connector based on the parameters and the size of the pre and postsynaptic population.
- virtual VarVec getExtraGlobalParams () const
- size_t getExtraGlobalParamIndex (const std::string &paramName) const

  Find the index of a named extra global parameter.

Additional Inherited Members

19.5.1 Member Typedef Documentation

19.5.1.1 CalcMaxLengthFunc

```
typedef std::function<unsigned int(unsigned int, unsigned int, const std::vector<double> &)>
InitSparseConnectivitySnippet::Base::CalcMaxLengthFunc
```

19.5.2 Member Function Documentation

19.5.2.1 getCalcMaxColLengthFunc()

```
virtual CalcMaxLengthFunc InitSparseConnectivitySnippet::Base::getCalcMaxColLengthFunc ( )
const  [inline], [virtual]
```

Get function to calculate the maximum column length of this connector based on the parameters and the size of the pre and postsynaptic population.

19.5.2.2 getCalcMaxRowLengthFunc()

```
virtual CalcMaxLengthFunc InitSparseConnectivitySnippet::Base::getCalcMaxRowLengthFunc ( )
const  [inline], [virtual]
```

Get function to calculate the maximum row length of this connector based on the parameters and the size of the pre and postsynaptic population.

19.5.2.3 getExtraGlobalParamIndex()

```
size_t InitSparseConnectivitySnippet::Base::getExtraGlobalParamIndex (
            const std::string & paramName ) const  [inline]
```

Find the index of a named extra global parameter.

19.5.2.4 getExtraGlobalParams()

```
virtual VarVec InitSparseConnectivitySnippet::Base::getExtraGlobalParams ( ) const [inline],
[virtual]
```

Gets names and types (as strings) of additional per-population parameters for the connection initialisation snippet

19.5.2.5 getRowBuildCode()

```
virtual std::string InitSparseConnectivitySnippet::Base::getRowBuildCode ( ) const [inline],
[virtual]
```

Reimplemented in InitSparseConnectivitySnippet::FixedProbabilityBase.

19.5.2.6 getRowBuildStateVars()

```
virtual ParamValVec InitSparseConnectivitySnippet::Base::getRowBuildStateVars ( ) const [inline],
[virtual]
```

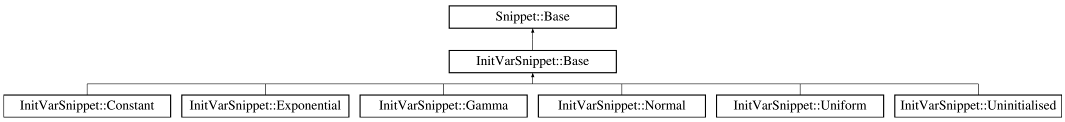
The documentation for this class was generated from the following file:

- initSparseConnectivitySnippet.h

19.6 InitVarSnippet::Base Class Reference

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Base:



Public Member Functions

- virtual std::string getCode () const

Additional Inherited Members

19.6.1 Member Function Documentation

19.6.1.1 getCode()

```
virtual std::string InitVarSnippet::Base::getCode ( ) const [inline], [virtual]
```

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

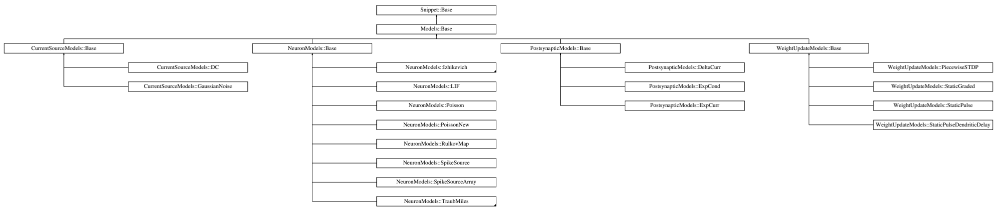
- initVarSnippet.h

19.7 Models::Base Class Reference

Base class for all models - in addition to the parameters snippets have, models can have state variables.

`#include <models.h>`

Inheritance diagram for Models::Base:



Public Member Functions

- virtual VarVec getVars () const

    Gets names and types (as strings) of model variables.
- virtual VarVec getExtraGlobalParams () const
- size_t getVarIndex (const std::string &varName) const

    Find the index of a named variable.
- size_t getExtraGlobalParamIndex (const std::string &paramName) const

    Find the index of a named extra global parameter.

Additional Inherited Members

19.7.1 Detailed Description

Base class for all models - in addition to the parameters snippets have, models can have state variables.

19.7.2 Member Function Documentation

19.7.2.1 getExtraGlobalParamIndex()

```
size_t Models::Base::getExtraGlobalParamIndex (
            const std::string & paramName ) const  [inline]
```

Find the index of a named extra global parameter.

19.7.2.2 getExtraGlobalParams()

```
virtual VarVec Models::Base::getExtraGlobalParams ( ) const  [inline], [virtual]
```

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

Reimplemented in NeuronModels::Poisson, and NeuronModels::SpikeSourceArray.

19.7.2.3 getVarIndex()

```
size_t Models::Base::getVarIndex (
            const std::string & varName ) const  [inline]
```

Find the index of a named variable.

19.7.2.4 getVars()

```
virtual VarVec Models::Base::getVars ( ) const [inline], [virtual]
```

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

Reimplemented in NeuronModels::TraubMiles, NeuronModels::PoissonNew, NeuronModels::Poisson, WeightUpdateModels::PiecewiseSTDP, NeuronModels::SpikeSourceArray, NeuronModels::LIF, WeightUpdateModels::StaticGraded, NeuronModels::IzhikevichVariable, WeightUpdateModels::StaticPulseDendriticDelay, NeuronModels::Izhikevich, WeightUpdateModels::StaticPulse, and NeuronModels::RulkovMap.

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

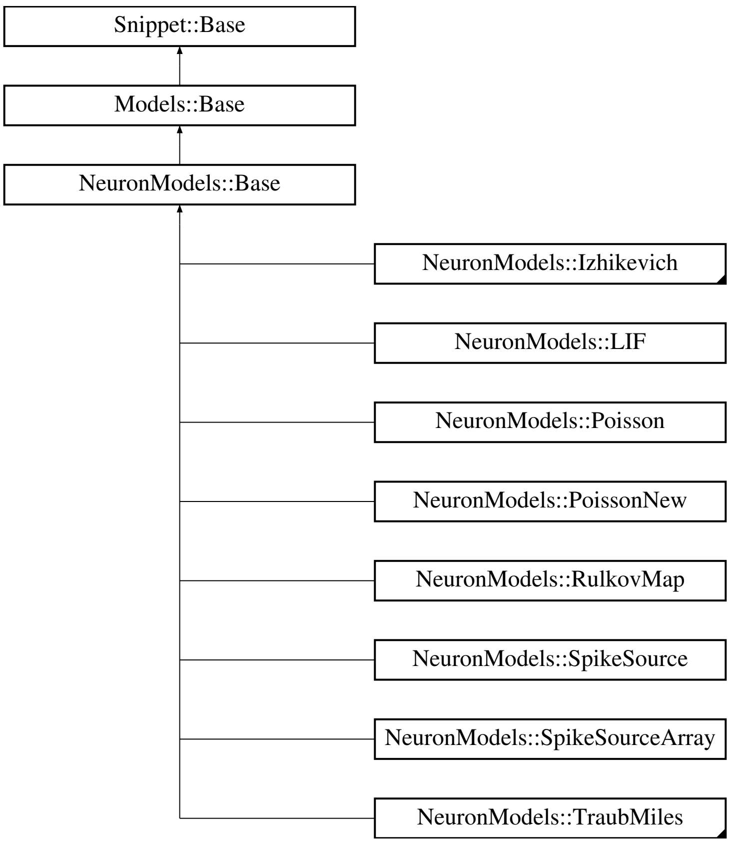
- models.h

19.8 NeuronModels::Base Class Reference

Base class for all neuron models.

`#include <neuronModels.h>`

Inheritance diagram for NeuronModels::Base:



Public Member Functions

- virtual std::string getSimCode () const

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const

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

- virtual std::string getResetCode () const

  Gets code that defines the reset action taken after a spike occurred. This can be empty.
- virtual std::string getSupportCode () const

  Gets support code to be made available within the neuron kernel/funcion.
- virtual Models::Base::ParamValVec getAdditionalInputVars () const
- virtual bool isAutoRefractoryRequired () const

  Does this model require auto-refractory logic?

Additional Inherited Members

19.8.1 Detailed Description

Base class for all neuron models.

19.8.2 Member Function Documentation

19.8.2.1 getAdditionalInputVars()

`virtual Models::Base::ParamValVec NeuronModels::Base::getAdditionalInputVars ( ) const [inline], [virtual]`

Gets names, types (as strings) and initial values of local variables into which the 'apply input code' of (potentially) multiple postsynaptic input models can apply input

19.8.2.2 getResetCode()

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

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

Reimplemented in NeuronModels::SpikeSourceArray, and NeuronModels::LIF.

19.8.2.3 getSimCode()

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

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

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

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMilesAlt, NeuronModels::TraubMiles↵ Fast, NeuronModels::TraubMiles, NeuronModels::PoissonNew, NeuronModels::Poisson, NeuronModels::Spike↵ SourceArray, NeuronModels::LIF, NeuronModels::Izhikevich, and NeuronModels::RulkovMap.

19.8.2.4 getSupportCode()

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

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

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

19.8.2.5 getThresholdConditionCode()

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

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

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

Reimplemented in NeuronModels::TraubMiles, NeuronModels::PoissonNew, NeuronModels::Poisson, NeuronModels::SpikeSourceArray, NeuronModels::SpikeSource, NeuronModels::LIF, NeuronModels::Izhikevich, and NeuronModels::RulkovMap.

19.8.2.6 isAutoRefractoryRequired()

`virtual bool NeuronModels::Base::isAutoRefractoryRequired ( ) const [inline], [virtual]`

Does this model require auto-refractory logic?

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

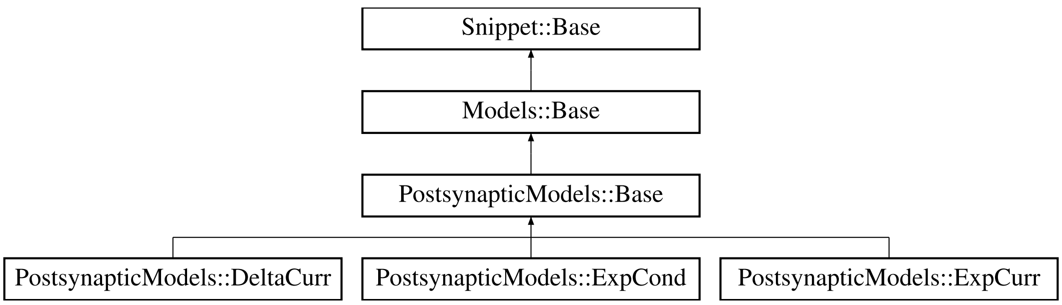
- neuronModels.h

19.9 PostsynapticModels::Base Class Reference

Base class for all postsynaptic models.

`#include <postsynapticModels.h>`

Inheritance diagram for PostsynapticModels::Base:



Public Member Functions

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

Additional Inherited Members

19.9.1 Detailed Description

Base class for all postsynaptic models.

19.9.2 Member Function Documentation

19.9.2.1 getApplyInputCode()

`virtual std::string PostsynapticModels::Base::getApplyInputCode ( ) const [inline], [virtual]`

Reimplemented in PostsynapticModels::DeltaCurr, PostsynapticModels::ExpCond, and PostsynapticModels::Exp↩ Curr.

19.9.2.2 getDecayCode()

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

Reimplemented in PostsynapticModels::ExpCond, and PostsynapticModels::ExpCurr.

19.9.2.3 getSupportCode()

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

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

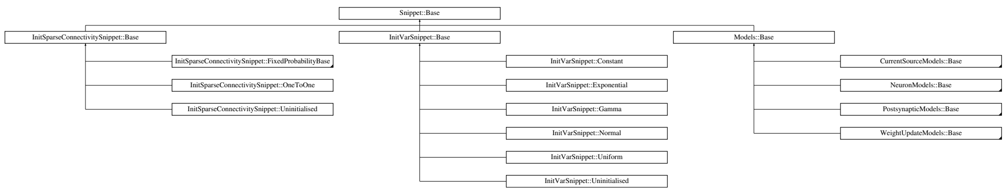
- postsynapticModels.h

19.10 Snippet::Base Class Reference

Base class for all code snippets.

`#include <snippet.h>`

Inheritance diagram for Snippet::Base:



Classes

- struct DerivedParam
  A derived parameter has a name and a function for obtaining its value.
- struct ParamVal
- struct Var
  A variable has a name and a type.

Public Types

- typedef std::vector< std::string > StringVec
- typedef std::vector< Var > VarVec
- typedef std::vector< ParamVal > ParamValVec
- typedef std::vector< DerivedParam > DerivedParamVec

Public Member Functions

- virtual ~Base ()

- virtual StringVec getParamNames () const

    Gets names of of (independent) model parameters.
- virtual DerivedParamVec getDerivedParams () const

Static Protected Member Functions

- static size_t getVarVecIndex (const std::string &varName, const VarVec &vars)

19.10.1 Detailed Description

Base class for all code snippets.

19.10.2 Member Typedef Documentation

19.10.2.1 DerivedParamVec

```
typedef std::vector<DerivedParam> Snippet::Base::DerivedParamVec
```

19.10.2.2 ParamValVec

```
typedef std::vector<ParamVal> Snippet::Base::ParamValVec
```

19.10.2.3 StringVec

```
typedef std::vector<std::string> Snippet::Base::StringVec
```

19.10.2.4 VarVec

```
typedef std::vector<Var> Snippet::Base::VarVec
```

19.10.3 Constructor & Destructor Documentation

19.10.3.1 ∼Base()

```
virtual Snippet::Base::~Base ( ) [inline], [virtual]
```

19.10.4 Member Function Documentation

19.10.4.1 getDerivedParams()

```
virtual DerivedParamVec Snippet::Base::getDerivedParams ( ) const [inline], [virtual]
```

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

Reimplemented in NeuronModels::PoissonNew, WeightUpdateModels::PiecewiseSTDP, NeuronModels::LIF, NeuronModels::RulkovMap, InitSparseConnectivitySnippet::FixedProbabilityBase, PostsynapticModels::ExpCond, and PostsynapticModels::ExpCurr.

19.10.4.2 getParamNames()

```
virtual StringVec Snippet::Base::getParamNames ( ) const  [inline], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMiles, NeuronModels::PoissonNew, NeuronModels::Poisson, WeightUpdateModels::PiecewiseSTDP, NeuronModels::LIF, WeightUpdateModels::↵ StaticGraded, NeuronModels::IzhikevichVariable, NeuronModels::Izhikevich, InitVarSnippet::Gamma, InitSparse↵ ConnectivitySnippet::FixedProbabilityBase, NeuronModels::RulkovMap, InitVarSnippet::Exponential, InitVar↵ Snippet::Normal, InitVarSnippet::Uniform, PostsynapticModels::ExpCond, CurrentSourceModels::GaussianNoise, InitVarSnippet::Constant, CurrentSourceModels::DC, and PostsynapticModels::ExpCurr.

19.10.4.3 getVarVecIndex()

```
static size_t Snippet::Base::getVarVecIndex (
            const std::string & varName,
            const VarVec & vars )  [inline], [static], [protected]
```

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

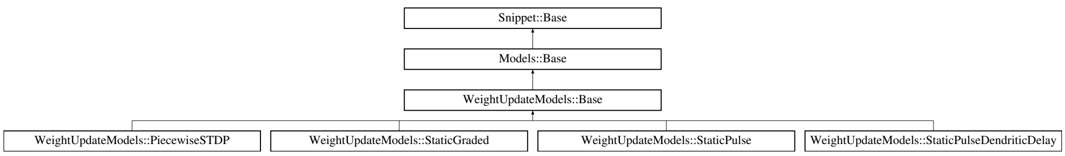
- snippet.h

19.11 WeightUpdateModels::Base Class Reference

Base class for all weight update models.

`#include <weightUpdateModels.h>`

Inheritance diagram for WeightUpdateModels::Base:



Public Member Functions

- virtual std::string getSimCode () const

    Gets simulation code run when 'true' spikes are received.
- virtual std::string getEventCode () const

    Gets code run when events (all the instances where event threshold condition is met) are received.
- virtual std::string getLearnPostCode () const

    Gets code to include in the learnSynapsesPost kernel/function.
- virtual std::string getSynapseDynamicsCode () const

    Gets code for synapse dynamics which are independent of spike detection.
- virtual std::string getEventThresholdConditionCode () const

    Gets codes to test for events.
- virtual std::string getSimSupportCode () const

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

- virtual std::string getLearnPostSupportCode () const

    Gets support code to be made available within learnSynapsesPost kernel/function.
- virtual std::string getSynapseDynamicsSuppportCode () const

    Gets support code to be made available within the synapse dynamics kernel/function.
- virtual std::string getPreSpikeCode () const
- virtual std::string getPostSpikeCode () const
- virtual VarVec getPreVars () const
- virtual VarVec getPostVars () const
- virtual bool isPreSpikeTimeRequired () const

    Whether presynaptic spike times are needed or not.
- virtual bool isPostSpikeTimeRequired () const

    Whether postsynaptic spike times are needed or not.
- size_t getPreVarIndex (const std::string &varName) const

    Find the index of a named presynaptic variable.
- size_t getPostVarIndex (const std::string &varName) const

    Find the index of a named postsynaptic variable.

Additional Inherited Members

19.11.1 Detailed Description

Base class for all weight update models.

19.11.2 Member Function Documentation

19.11.2.1 getEventCode()

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

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

Reimplemented in WeightUpdateModels::StaticGraded.

19.11.2.2 getEventThresholdConditionCode()

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

Gets codes to test for events.

Reimplemented in WeightUpdateModels::StaticGraded.

19.11.2.3 getLearnPostCode()

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

Gets code to include in the learnSynapsesPost kernel/function.

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

Reimplemented in WeightUpdateModels::PiecewiseSTDP.

19.11.2.4 getLearnPostSupportCode()

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

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

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

19.11.2.5 getPostSpikeCode()

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

Gets code to be run once per spiking postsynaptic neuron before learn post code is run on synapses

This is typically for the code to update postsynaptic variables. Presynaptic and synapse variables are not accesible from within this code

19.11.2.6 getPostVarIndex()

```
size_t WeightUpdateModels::Base::getPostVarIndex (
            const std::string & varName ) const [inline]
```

Find the index of a named postsynaptic variable.

19.11.2.7 getPostVars()

```
virtual VarVec WeightUpdateModels::Base::getPostVars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of state variables that are common across all synapses going to the same postsynaptic neuron

19.11.2.8 getPreSpikeCode()

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

Gets code to be run once per spiking presynaptic neuron before sim code is run on synapses

This is typically for the code to update presynaptic variables. Postsynaptic and synapse variables are not accesible from within this code

19.11.2.9 getPreVarIndex()

```
size_t WeightUpdateModels::Base::getPreVarIndex (
            const std::string & varName ) const [inline]
```

Find the index of a named presynaptic variable.

19.11.2.10 getPreVars()

```
virtual VarVec WeightUpdateModels::Base::getPreVars ( ) const [inline], [virtual]
```

Gets names and types (as strings) of state variables that are common across all synapses coming from the same presynaptic neuron

19.11.2.11 getSimCode()

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

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

Reimplemented in WeightUpdateModels::PiecewiseSTDP, WeightUpdateModels::StaticPulseDendriticDelay, and WeightUpdateModels::StaticPulse.

19.11.2.12 getSimSupportCode()

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

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

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

19.11.2.13 getSynapseDynamicsCode()

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

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

19.11.2.14 getSynapseDynamicsSuppportCode()

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

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

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

19.11.2.15 isPostSpikeTimeRequired()

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

Whether postsynaptic spike times are needed or not.

Reimplemented in WeightUpdateModels::PiecewiseSTDP.

19.11.2.16 isPreSpikeTimeRequired()

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

Whether presynaptic spike times are needed or not.

Reimplemented in WeightUpdateModels::PiecewiseSTDP.

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

- weightUpdateModels.h

19.12 CodeGenerator::CodeStream::CB Struct Reference

A close bracket marker.

```
#include <codeStream.h>
```

Public Member Functions

- CB (unsigned int level)

Public Attributes

- const unsigned int Level

19.12.1 Detailed Description

A close bracket marker.

Write to code stream `os` using:

```
os << CB(16);
```

19.12.2 Constructor & Destructor Documentation

19.12.2.1 CB()

```
CodeGenerator::CodeStream::CB::CB (
            unsigned int level )  [inline]
```

19.12.3 Member Data Documentation

19.12.3.1 Level

```
const unsigned int CodeGenerator::CodeStream::CB::Level
```

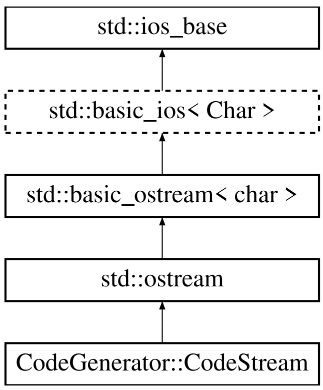
The documentation for this struct was generated from the following file:

- codeStream.h

19.13 CodeGenerator::CodeStream Class Reference

`#include <codeStream.h>`

Inheritance diagram for CodeGenerator::CodeStream:



Classes

- struct CB

A close bracket marker.
- struct OB

An open bracket marker.
- class Scope

Public Member Functions

- CodeStream ()
- CodeStream (std::ostream &stream)
- void setSink (std::ostream &stream)

Friends

- GENN_EXPORT friend std::ostream & operator<< (std::ostream &s, const OB &ob)
- GENN_EXPORT friend std::ostream & operator<< (std::ostream &s, const CB &cb)

19.13.1 Constructor & Destructor Documentation

19.13.1.1 CodeStream() [1/2]

```
CodeGenerator::CodeStream::CodeStream ( )  [inline]
```

19.13.1.2 CodeStream() [2/2]

```
CodeGenerator::CodeStream::CodeStream (
            std::ostream & stream )  [inline]
```

19.13.2 Member Function Documentation

19.13.2.1 setSink()

```
void CodeGenerator::CodeStream::setSink (
            std::ostream & stream )  [inline]
```

19.13.3 Friends And Related Function Documentation

19.13.3.1 operator<< [1/2]

```
GENN_EXPORT friend std::ostream& operator<< (
            std::ostream & s,
            const OB & ob )  [friend]
```

19.13.3.2 operator<< [2/2]

```
GENN_EXPORT friend std::ostream& operator<< (
            std::ostream & s,
            const CB & cb )  [friend]
```

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

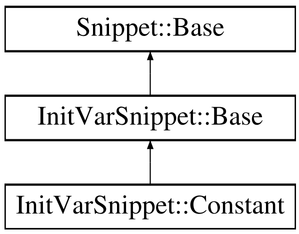
- codeStream.h

19.14 InitVarSnippet::Constant Class Reference

Initialises variable to a constant value.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Constant:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Constant, 1)
- SET_CODE ("$(value) = $(constant);")
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Additional Inherited Members

19.14.1 Detailed Description

Initialises variable to a constant value.

This snippet takes 1 parameter:

- `value` - The value to intialise the variable to

Note

This snippet type is seldom used directly - Models::VarInit has an implicit constructor that, internally, creates one of these snippets

19.14.2 Member Function Documentation

19.14.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Constant::DECLARE_SNIPPET (
            InitVarSnippet::Constant ,
            1 )
```

19.14.2.2 getParamNames()

```
virtual StringVec InitVarSnippet::Constant::getParamNames ( ) const  [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.14.2.3 SET_CODE()

```
InitVarSnippet::Constant::SET_CODE ( )
```

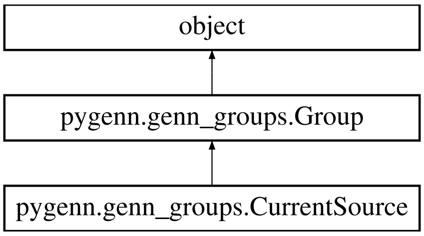
The documentation for this class was generated from the following file:

- initVarSnippet.h

19.15 pygenn.genn_groups.CurrentSource Class Reference

Class representing a current injection into a group of neurons.

Inheritance diagram for pygenn.genn_groups.CurrentSource:



Public Member Functions

- def __init__ (self, name)

    Init CurrentSource.
- def size (self)

    Number of neuron in the injected population.
- def size (self, _)
- def set_current_source_model (self, model, param_space, var_space)

    Set curront source model, its parameters and initial variables.
- def add_to (self, nn_model, pop)

    Inject this CurrentSource into population and add it to the GeNN NNmodel.
- def add_extra_global_param (self, param_name, param_values)

    Add extra global parameter.
- def load (self, slm, scalar)
- def reinitialise (self, slm, scalar)

    Reinitialise current source.

Public Attributes

- current_source_model
- target_pop
- pop

19.15.1 Detailed Description

Class representing a current injection into a group of neurons.

19.15.2 Constructor & Destructor Documentation

19.15.2.1 __init__()

```
def pygenn.genn_groups.CurrentSource.__init__ (
              self,
              name )
```

Init CurrentSource.

Parameters

name	string name of the current source

19.15.3 Member Function Documentation

19.15.3.1 add_extra_global_param()

```
def pygenn.genn_groups.CurrentSource.add_extra_global_param (
              self,
              param_name,
              param_values )
```

Add extra global parameter.

Parameters

param_name	string with the name of the extra global parameter
param_values	iterable or a single value

19.15.3.2 add_to()

```
def pygenn.genn_groups.CurrentSource.add_to (
              self,
              nn_model,
              pop )
```

Inject this CurrentSource into population and add it to the GeNN NNmodel.

Parameters

pop	instance of NeuronGroup into which this CurrentSource should be injected
nn_model	GeNN NNmodel

19.15.3.3 load()

```
def pygenn.genn_groups.CurrentSource.load (
              self,
              slm,
              scalar )
```

19.15.3.4 reinitialise()

```
def pygenn.genn_groups.CurrentSource.reinitialise (
              self,
              slm,
              scalar )
```

Reinitialise current source.

Parameters

slm	SharedLibraryModel instance for acccessing variables
scalar	String specifying "scalar" type

19.15.3.5 set_current_source_model()

```
def pygenn.genn_groups.CurrentSource.set_current_source_model (
              self,
              model,
              param_space,
              var_space )
```

Set curront source model, its parameters and initial variables.

Parameters

model	type as string of intance of the model
param_space	dict with model parameters
var_space	dict with model variables

19.15.3.6 size() [1/2]

```
def pygenn.genn_groups.CurrentSource.size (
              self )
```

Number of neuron in the injected population.

19.15.3.7 size() [2/2]

```
def pygenn.genn_groups.CurrentSource.size (
              self,
              _ )
```

19.15.4 Member Data Documentation

19.15.4.1 current_source_model

```
pygenn.genn_groups.CurrentSource.current_source_model
```

19.15.4.2 pop

```
pygenn.genn_groups.CurrentSource.pop
```

19.15.4.3 target_pop

```
pygenn.genn_groups.CurrentSource.target_pop
```

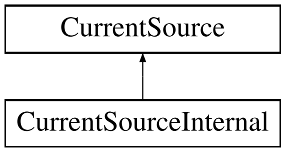
The documentation for this class was generated from the following file:

- genn_groups.py

19.16 CurrentSource Class Reference

```
#include <currentSource.h>
```

Inheritance diagram for CurrentSource:



Public Member Functions

- CurrentSource (const CurrentSource &)=delete
- CurrentSource ()=delete
- void setVarLocation (const std::string &varName, VarLocation loc)

    Set location of current source state variable.
- void setExtraGlobalParamLocation (const std::string &paramName, VarLocation loc)

    Set location of extra global parameter.
- const std::string & getName () const
- const CurrentSourceModels::Base ∗ getCurrentSourceModel () const

    Gets the current source model used by this group.
- const std::vector< double > & getParams () const
- const std::vector< Models::VarInit > & getVarInitialisers () const
- VarLocation getVarLocation (const std::string &varName) const

    Get variable location for current source model state variable.
- VarLocation getVarLocation (size_t index) const

    Get variable location for current source model state variable.
- VarLocation getExtraGlobalParamLocation (const std::string &paramName) const

    Get location of neuron model extra global parameter by name.
- VarLocation getExtraGlobalParamLocation (size_t index) const

    Get location of neuron model extra global parameter by omdex.

Protected Member Functions

- CurrentSource (const std::string &name, const CurrentSourceModels::Base *currentSourceModel, const std::vector< double > &params, const std::vector< Models::VarInit > &varInitialisers, VarLocation default←VarLocation, VarLocation defaultExtraGlobalParamLocation)
- void initDerivedParams (double dt)
- const std::vector< double > & getDerivedParams () const
- bool isSimRNGRequired () const

    Does this current source require an RNG to simulate.
- bool isInitRNGRequired () const

    Does this current source group require an RNG for it's init code.

19.16.1 Constructor & Destructor Documentation

19.16.1.1 CurrentSource() [1/3]

```
CurrentSource::CurrentSource (
            const CurrentSource & )  [delete]
```

19.16.1.2 CurrentSource() [2/3]

```
CurrentSource::CurrentSource ( )  [delete]
```

19.16.1.3 CurrentSource() [3/3]

```
CurrentSource::CurrentSource (
            const std::string & name,
            const CurrentSourceModels::Base * currentSourceModel,
            const std::vector< double > & params,
            const std::vector< Models::VarInit > & varInitialisers,
            VarLocation defaultVarLocation,
            VarLocation defaultExtraGlobalParamLocation )  [inline], [protected]
```

19.16.2 Member Function Documentation

19.16.2.1 getCurrentSourceModel()

```
const CurrentSourceModels::Base* CurrentSource::getCurrentSourceModel ( ) const  [inline]
```

Gets the current source model used by this group.

19.16.2.2 getDerivedParams()

```
const std::vector<double>& CurrentSource::getDerivedParams ( ) const  [inline], [protected]
```

19.16.2.3 getExtraGlobalParamLocation() [1/2]

```
VarLocation CurrentSource::getExtraGlobalParamLocation (
            const std::string & paramName ) const
```

Get location of neuron model extra global parameter by name.

This is only used by extra global parameters which are pointers

19.16.2.4 getExtraGlobalParamLocation() [2/2]

```
VarLocation CurrentSource::getExtraGlobalParamLocation (
            size_t index ) const  [inline]
```

Get location of neuron model extra global parameter by omdex.

This is only used by extra global parameters which are pointers

19.16.2.5 getName()

```
const std::string& CurrentSource::getName ( ) const  [inline]
```

19.16.2.6 getParams()

```
const std::vector<double>& CurrentSource::getParams ( ) const  [inline]
```

19.16.2.7 getVarInitialisers()

```
const std::vector<Models::VarInit>& CurrentSource::getVarInitialisers ( ) const  [inline]
```

19.16.2.8 getVarLocation() [1/2]

```
VarLocation CurrentSource::getVarLocation (
            const std::string & varName ) const
```

Get variable location for current source model state variable.

19.16.2.9 getVarLocation() [2/2]

```
VarLocation CurrentSource::getVarLocation (
            size_t index ) const  [inline]
```

Get variable location for current source model state variable.

19.16.2.10 initDerivedParams()

```
void CurrentSource::initDerivedParams (
            double dt )  [protected]
```

19.16.2.11 isInitRNGRequired()

```
bool CurrentSource::isInitRNGRequired ( ) const  [protected]
```

Does this current source group require an RNG for it's init code.

19.16.2.12 isSimRNGRequired()

```
bool CurrentSource::isSimRNGRequired ( ) const  [protected]
```

Does this current source require an RNG to simulate.

19.16.2.13 setExtraGlobalParamLocation()

```
void CurrentSource::setExtraGlobalParamLocation (
            const std::string & paramName,
            VarLocation loc )
```

Set location of extra global parameter.

This is ignored for simulations on hardware with a single memory space and only applies to extra global parameters which are pointers.

19.16.2.14 setVarLocation()

```
void CurrentSource::setVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set location of current source state variable.

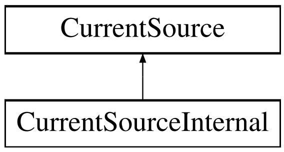
The documentation for this class was generated from the following files:

- currentSource.h
- currentSource.cc

19.17 CurrentSourceInternal Class Reference

```
#include <currentSourceInternal.h>
```

Inheritance diagram for CurrentSourceInternal:



Public Member Functions

- CurrentSourceInternal (const std::string &name, const CurrentSourceModels::Base *currentSourceModel, const std::vector< double > &params, const std::vector< Models::VarInit > &varInitialisers, VarLocation defaultVarLocation, VarLocation defaultExtraGlobalParamLocation)

Additional Inherited Members

19.17.1 Constructor & Destructor Documentation

19.17.1.1 CurrentSourceInternal()

```
CurrentSourceInternal::CurrentSourceInternal (
            const std::string & name,
            const CurrentSourceModels::Base * currentSourceModel,
            const std::vector< double > & params,
            const std::vector< Models::VarInit > & varInitialisers,
```

VarLocation *defaultVarLocation*,
VarLocation *defaultExtraGlobalParamLocation*) [inline]

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

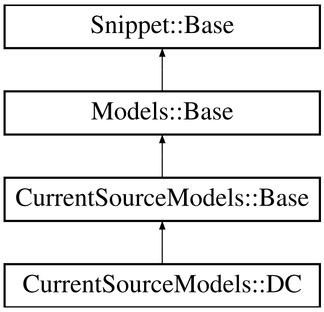
- currentSourceInternal.h

19.18 CurrentSourceModels::DC Class Reference

DC source.

`#include <currentSourceModels.h>`

Inheritance diagram for CurrentSourceModels::DC:



Public Types

- typedef Snippet::ValueBase< 1 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- SET_INJECTION_CODE ("$(injectCurrent, $(amp));\)
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Static Public Member Functions

- static const DC ∗ getInstance ()

Additional Inherited Members

19.18.1 Detailed Description

DC source.

It has a single parameter:

- `amp` - amplitude of the current [nA]

19.18.2 Member Typedef Documentation

19.18.2.1 ParamValues

```
typedef Snippet::ValueBase< 1 > CurrentSourceModels::DC::ParamValues
```

19.18.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> CurrentSourceModels::DC::PostVarValues
```

19.18.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> CurrentSourceModels::DC::PreVarValues
```

19.18.2.4 VarValues

```
typedef Models::VarInitContainerBase< 0 > CurrentSourceModels::DC::VarValues
```

19.18.3 Member Function Documentation

19.18.3.1 getInstance()

```
static const DC* CurrentSourceModels::DC::getInstance ( )  [inline], [static]
```

19.18.3.2 getParamNames()

```
virtual StringVec CurrentSourceModels::DC::getParamNames ( ) const  [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.18.3.3 SET_INJECTION_CODE()

```
CurrentSourceModels::DC::SET_INJECTION_CODE (
            "$(injectCurrent, $(amp));\  )
```

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

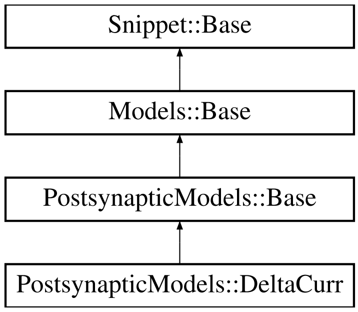
- currentSourceModels.h

19.19 PostsynapticModels::DeltaCurr Class Reference

Simple delta current synapse.

```
#include <postsynapticModels.h>
```

Inheritance diagram for PostsynapticModels::DeltaCurr:



Public Types

- typedef Snippet::ValueBase< 0 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getApplyInputCode () const override

Static Public Member Functions

- static const DeltaCurr * getInstance ()

Additional Inherited Members

19.19.1 Detailed Description

Simple delta current synapse.

Synaptic input provides a direct inject of instantaneous current

19.19.2 Member Typedef Documentation

19.19.2.1 ParamValues

`typedef Snippet::ValueBase< 0 > PostsynapticModels::DeltaCurr::ParamValues`

19.19.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> PostsynapticModels::DeltaCurr::PostVarValues`

19.19.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> PostsynapticModels::DeltaCurr::PreVarValues`

19.19.2.4 VarValues

`typedef Models::VarInitContainerBase< 0 > PostsynapticModels::DeltaCurr::VarValues`

19.19.3 Member Function Documentation

19.19.3.1 getApplyInputCode()

`virtual std::string PostsynapticModels::DeltaCurr::getApplyInputCode ( ) const [inline], [override], [virtual]`

Reimplemented from PostsynapticModels::Base.

19.19.3.2 getInstance()

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

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

- postsynapticModels.h

19.20 Snippet::Base::DerivedParam Struct Reference

A derived parameter has a name and a function for obtaining its value.

`#include <snippet.h>`

Public Attributes

- std::string name
- std::function< double(const std::vector< double > &, double)> func

19.20.1 Detailed Description

A derived parameter has a name and a function for obtaining its value.

19.20.2 Member Data Documentation

19.20.2.1 func

`std::function<double(const std::vector<double> &, double)> Snippet::Base::DerivedParam::func`

19.20.2.2 name

`std::string Snippet::Base::DerivedParam::name`

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

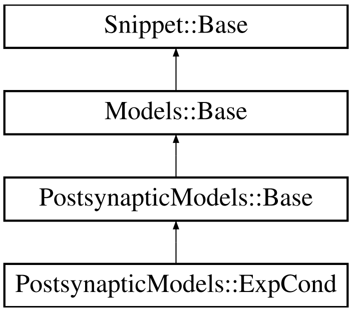
- snippet.h

19.21 PostsynapticModels::ExpCond Class Reference

Exponential decay with synaptic input treated as a conductance value.

`#include <postsynapticModels.h>`

Inheritance diagram for PostsynapticModels::ExpCond:



Public Types

- typedef Snippet::ValueBase< 2 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getDecayCode () const override
- virtual std::string getApplyInputCode () const override
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.
- virtual DerivedParamVec getDerivedParams () const override

Static Public Member Functions

- static const ExpCond * getInstance ()

Additional Inherited Members

19.21.1 Detailed Description

Exponential decay with synaptic input treated as a conductance value.

This model has no variables and two parameters:

- `tau` : Decay time constant
- `E` : Reversal potential

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

19.21.2 Member Typedef Documentation

19.21.2.1 ParamValues

```
typedef Snippet::ValueBase< 2 > PostsynapticModels::ExpCond::ParamValues
```

19.21.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> PostsynapticModels::ExpCond::PostVarValues
```

19.21.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> PostsynapticModels::ExpCond::PreVarValues
```

19.21.2.4 VarValues

```
typedef Models::VarInitContainerBase< 0 > PostsynapticModels::ExpCond::VarValues
```

19.21.3 Member Function Documentation

19.21.3.1 getApplyInputCode()

```
virtual std::string PostsynapticModels::ExpCond::getApplyInputCode ( ) const [inline], [override],
[virtual]
```

Reimplemented from PostsynapticModels::Base.

19.21.3.2 getDecayCode()

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

Reimplemented from PostsynapticModels::Base.

19.21.3.3 getDerivedParams()

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

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

Reimplemented from Snippet::Base.

19.21.3.4 getInstance()

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

19.21.3.5 getParamNames()

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

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

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

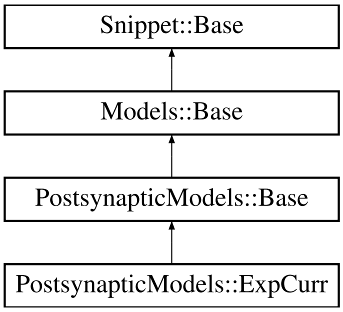
- postsynapticModels.h

19.22 PostsynapticModels::ExpCurr Class Reference

Exponential decay with synaptic input treated as a current value.

```
#include <postsynapticModels.h>
```

Inheritance diagram for PostsynapticModels::ExpCurr:



Public Types

- typedef Snippet::ValueBase< 1 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getDecayCode () const override
- virtual std::string getApplyInputCode () const override
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual DerivedParamVec getDerivedParams () const override

Static Public Member Functions

- static const ExpCurr ∗ getInstance ()

Additional Inherited Members

19.22.1 Detailed Description

Exponential decay with synaptic input treated as a current value.

19.22.2 Member Typedef Documentation

19.22.2.1 ParamValues

`typedef Snippet::ValueBase< 1 > PostsynapticModels::ExpCurr::ParamValues`

19.22.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> PostsynapticModels::ExpCurr::PostVarValues`

19.22.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> PostsynapticModels::ExpCurr::PreVarValues`

19.22.2.4 VarValues

`typedef Models::VarInitContainerBase< 0 > PostsynapticModels::ExpCurr::VarValues`

19.22.3 Member Function Documentation

19.22.3.1 getApplyInputCode()

`virtual std::string PostsynapticModels::ExpCurr::getApplyInputCode ( ) const [inline], [override], [virtual]`

Reimplemented from PostsynapticModels::Base.

19.22.3.2 getDecayCode()

`virtual std::string PostsynapticModels::ExpCurr::getDecayCode ( ) const [inline], [override], [virtual]`

Reimplemented from PostsynapticModels::Base.

19.22.3.3 getDerivedParams()

`virtual DerivedParamVec PostsynapticModels::ExpCurr::getDerivedParams ( ) const [inline], [override], [virtual]`

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

Reimplemented from Snippet::Base.

19.22.3.4 getInstance()

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

19.22.3.5 getParamNames()

`virtual StringVec PostsynapticModels::ExpCurr::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

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

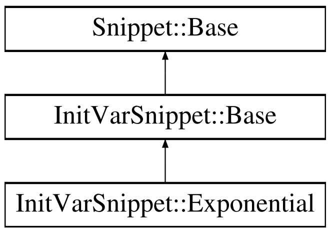
- postsynapticModels.h

19.23 InitVarSnippet::Exponential Class Reference

Initialises variable by sampling from the exponential distribution.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Exponential:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Exponential, 1)
- SET_CODE ("$(value) = $(lambda) ∗ $(gennrand_exponential);")
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Additional Inherited Members

19.23.1 Detailed Description

Initialises variable by sampling from the exponential distribution.

This snippet takes 1 parameter:

- `lambda` - mean event rate (events per unit time/distance)

19.23.2 Member Function Documentation

19.23.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Exponential::DECLARE_SNIPPET (
            InitVarSnippet::Exponential ,
            1 )
```

19.23.2.2 getParamNames()

```
virtual StringVec InitVarSnippet::Exponential::getParamNames ( ) const [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.23.2.3 SET_CODE()

```
InitVarSnippet::Exponential::SET_CODE ( )
```

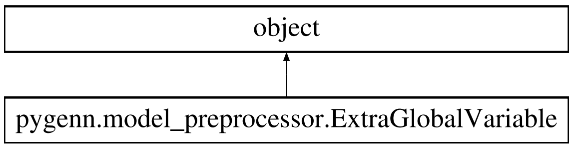
The documentation for this class was generated from the following file:

- initVarSnippet.h

19.24 pygenn.model_preprocessor.ExtraGlobalVariable Class Reference

Class holding information about GeNN extra global pointer variable.

Inheritance diagram for pygenn.model_preprocessor.ExtraGlobalVariable:



Public Member Functions

- def __init__ (self, variable_name, variable_type, values=None)

  Init Variable.
- def set_values (self, values)

  Set Variable's values.

Public Attributes

- name
- type
- view
- values

19.24.1 Detailed Description

Class holding information about GeNN extra global pointer variable.

19.24.2 Constructor & Destructor Documentation

19.24.2.1 __init__()

```
def pygenn.model_preprocessor.ExtraGlobalVariable.__init__ (
              self,
              variable_name,
              variable_type,
              values = None )
```

Init Variable.

Parameters

variable_name	string name of the variable
variable_type	string type of the variable
values	iterable

19.24.3 Member Function Documentation

19.24.3.1 set_values()

```
def pygenn.model_preprocessor.ExtraGlobalVariable.set_values (
              self,
              values )
```

Set Variable's values.

Parameters

values	iterable, single value or VarInit instance

19.24.4 Member Data Documentation

19.24.4.1 name

```
pygenn.model_preprocessor.ExtraGlobalVariable.name
```

19.24.4.2 type

```
pygenn.model_preprocessor.ExtraGlobalVariable.type
```

19.24.4.3 values

```
pygenn.model_preprocessor.ExtraGlobalVariable.values
```

19.24.4.4 view

```
pygenn.model_preprocessor.ExtraGlobalVariable.view
```

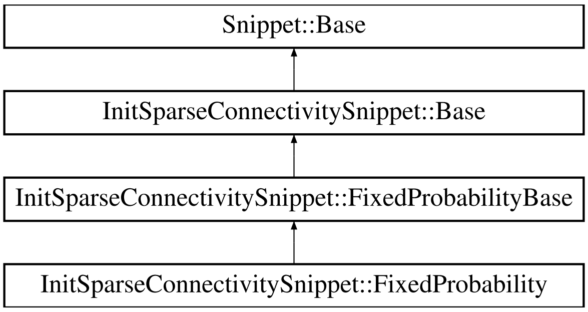
The documentation for this class was generated from the following file:

- model_preprocessor.py

19.25 InitSparseConnectivitySnippet::FixedProbability Class Reference

```
#include <initSparseConnectivitySnippet.h>
```

Inheritance diagram for InitSparseConnectivitySnippet::FixedProbability:



Public Member Functions

- DECLARE_SNIPPET (InitSparseConnectivitySnippet::FixedProbability, 1)
- SET_ROW_BUILD_CODE ("const scalar u = $(gennrand_uniform);\ "prevJ+=(1+(int)(log(u) ∗$(probLog↩ Recip)));\" "if(prevJ< $(num_post)) {\" " $(addSynapse, prevJ);\" "}\" "else {\" " $(endRow);\" "}\")

Additional Inherited Members

19.25.1 Detailed Description

Initialises connectivity with a fixed probability of a synapse existing between a pair of pre and postsynaptic neurons.

Whether a synapse exists between a pair of pre and a postsynaptic neurons can be modelled using a Bernoulli distribution. While this COULD br sampling directly by repeatedly drawing from the uniform distribution, this is inefficient. Instead we sample from the gemetric distribution which describes "the probability distribution of the number of Bernoulli trials needed to get one success" – essentially the distribution of the 'gaps' between synapses. We do this using the "inversion method" described by Devroye (1986) – essentially inverting the CDF of the equivalent continuous distribution (in this case the exponential distribution)

19.25.2 Member Function Documentation

19.25.2.1 DECLARE_SNIPPET()

```
InitSparseConnectivitySnippet::FixedProbability::DECLARE_SNIPPET (
            InitSparseConnectivitySnippet::FixedProbability ,
            1 )
```

19.25.2.2 SET_ROW_BUILD_CODE()

```
InitSparseConnectivitySnippet::FixedProbability::SET_ROW_BUILD_CODE ( )
```

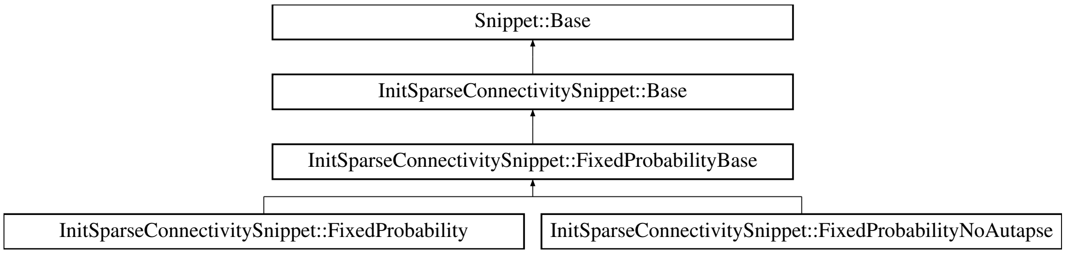
The documentation for this class was generated from the following file:

- initSparseConnectivitySnippet.h

19.26 InitSparseConnectivitySnippet::FixedProbabilityBase Class Reference

```
#include <initSparseConnectivitySnippet.h>
```

Inheritance diagram for InitSparseConnectivitySnippet::FixedProbabilityBase:



Public Member Functions

- virtual std::string getRowBuildCode () const override=0
- SET_ROW_BUILD_STATE_VARS ({{"prevJ", "int", -1}})
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual DerivedParamVec getDerivedParams () const override
- SET_CALC_MAX_ROW_LENGTH_FUNC ([](unsigned int numPre, unsigned int numPost, const std::vector< double > &pars) { const double quantile=pow(0.9999, 1.0/(double) numPre);return binomialInverseCDF(quantile, numPost, pars[0]);})
- SET_CALC_MAX_COL_LENGTH_FUNC ([](unsigned int numPre, unsigned int numPost, const std::vector< double > &pars) { const double quantile=pow(0.9999, 1.0/(double) numPost);return binomialInverseCDF(quantile, numPre, pars[0]);})

Additional Inherited Members

19.26.1 Detailed Description

Base class for snippets which initialise connectivity with a fixed probability of a synapse existing between a pair of pre and postsynaptic neurons.

19.26.2 Member Function Documentation

19.26.2.1 getDerivedParams()

```
virtual DerivedParamVec InitSparseConnectivitySnippet::FixedProbabilityBase::getDerivedParams
( ) const  [inline], [override], [virtual]
```

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

Reimplemented from Snippet::Base.

19.26.2.2 getParamNames()

```
virtual StringVec InitSparseConnectivitySnippet::FixedProbabilityBase::getParamNames ( ) const
[inline], [override], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.26.2.3 getRowBuildCode()

```
virtual std::string InitSparseConnectivitySnippet::FixedProbabilityBase::getRowBuildCode ( )
const [override], [pure virtual]
```

Reimplemented from InitSparseConnectivitySnippet::Base.

19.26.2.4 SET_CALC_MAX_COL_LENGTH_FUNC()

```
InitSparseConnectivitySnippet::FixedProbabilityBase::SET_CALC_MAX_COL_LENGTH_FUNC (
             [] (unsigned int numPre, unsigned int numPost, const std::vector< double > &pars)
{ const double quantile=pow(0.9999, 1.0/(double) numPost);return binomialInverseCDF(quantile,
numPre, pars[0]);}  )
```

19.26.2.5 SET_CALC_MAX_ROW_LENGTH_FUNC()

```
InitSparseConnectivitySnippet::FixedProbabilityBase::SET_CALC_MAX_ROW_LENGTH_FUNC (
             [] (unsigned int numPre, unsigned int numPost, const std::vector< double > &pars)
{ const double quantile=pow(0.9999, 1.0/(double) numPre);return binomialInverseCDF(quantile,
numPost, pars[0]);}  )
```

19.26.2.6 SET_ROW_BUILD_STATE_VARS()

```
InitSparseConnectivitySnippet::FixedProbabilityBase::SET_ROW_BUILD_STATE_VARS (
             {{"prevJ", "int", -1}}  )
```

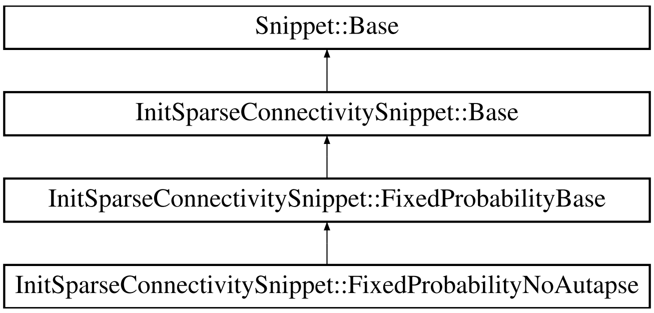
The documentation for this class was generated from the following file:

- initSparseConnectivitySnippet.h

19.27 InitSparseConnectivitySnippet::FixedProbabilityNoAutapse Class Reference

```
#include <initSparseConnectivitySnippet.h>
```

Inheritance diagram for InitSparseConnectivitySnippet::FixedProbabilityNoAutapse:



Public Member Functions

- DECLARE_SNIPPET (InitSparseConnectivitySnippet::FixedProbabilityNoAutapse, 1)
- SET_ROW_BUILD_CODE ("int nextJ;\ "do {\" " const scalar u=$(gennrand_uniform);\" " nextJ=prevJ+(1+(int)(log(u) ∗$(probLogRecip)));\" "} while(nextJ==$(id_pre));\" "prevJ=nextJ;\" "if(prevJ< $(num_post)) {\" " $(add←Synapse, prevJ);\" "}\" "else {\" " $(endRow);\" "}\")

Additional Inherited Members

19.27.1 Detailed Description

Initialises connectivity with a fixed probability of a synapse existing between a pair of pre and postsynaptic neurons. This version ensures there are no autapses - connections between neurons with the same id so should be used for recurrent connections.

Whether a synapse exists between a pair of pre and a postsynaptic neurons can be modelled using a Bernoulli distribution. While this COULD br sampling directly by repeatedly drawing from the uniform distribution, this is inefficient. Instead we sample from the gemetric distribution which describes "the probability distribution of the number of Bernoulli trials needed to get one success" – essentially the distribution of the 'gaps' between synapses. We do this using the "inversion method" described by Devroye (1986) – essentially inverting the CDF of the equivalent continuous distribution (in this case the exponential distribution)

19.27.2 Member Function Documentation

19.27.2.1 DECLARE_SNIPPET()

```
InitSparseConnectivitySnippet::FixedProbabilityNoAutapse::DECLARE_SNIPPET (
            InitSparseConnectivitySnippet::FixedProbabilityNoAutapse ,
            1 )
```

19.27.2.2 SET_ROW_BUILD_CODE()

```
InitSparseConnectivitySnippet::FixedProbabilityNoAutapse::SET_ROW_BUILD_CODE ( )
```

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

- initSparseConnectivitySnippet.h

19.28 CodeGenerator::FunctionTemplate Struct Reference

```
#include <codeGenUtils.h>
```

Public Member Functions

- FunctionTemplate operator= (const FunctionTemplate &o)

Public Attributes

- const std::string genericName

  Generic name used to refer to function in user code.
- const unsigned int numArguments

  Number of function arguments.
- const std::string doublePrecisionTemplate

  The function template (for use with functionSubstitute) used when model uses double precision.
- const std::string singlePrecisionTemplate

  The function template (for use with functionSubstitute) used when model uses single precision.

19.28.1 Detailed Description

Immutable structure for specifying how to implement a generic function e.g. gennrand_uniform

NOTE for the sake of easy initialisation first two parameters of GenericFunction are repeated (C++17 fixes)

19.28.2 Member Function Documentation

19.28.2.1 operator=()

```
FunctionTemplate CodeGenerator::FunctionTemplate::operator= (
            const FunctionTemplate & o ) [inline]
```

19.28.3 Member Data Documentation

19.28.3.1 doublePrecisionTemplate

```
const std::string CodeGenerator::FunctionTemplate::doublePrecisionTemplate
```

The function template (for use with functionSubstitute) used when model uses double precision.

19.28.3.2 genericName

```
const std::string CodeGenerator::FunctionTemplate::genericName
```

Generic name used to refer to function in user code.

19.28.3.3 numArguments

```
const unsigned int CodeGenerator::FunctionTemplate::numArguments
```

Number of function arguments.

19.28.3.4 singlePrecisionTemplate

```
const std::string CodeGenerator::FunctionTemplate::singlePrecisionTemplate
```

The function template (for use with functionSubstitute) used when model uses single precision.

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

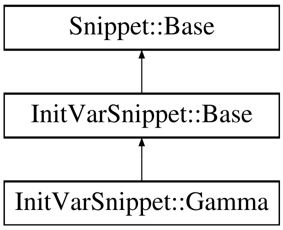
- codeGenUtils.h

19.29 InitVarSnippet::Gamma Class Reference

Initialises variable by sampling from the exponential distribution.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Gamma:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Gamma, 2)
- SET_CODE ("$(value) = $(b) ∗ $(gennrand_gamma, $(a));")
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.

Additional Inherited Members

19.29.1 Detailed Description

Initialises variable by sampling from the exponential distribution.

This snippet takes 1 parameter:

- `lambda` - mean event rate (events per unit time/distance)

19.29.2 Member Function Documentation

19.29.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Gamma::DECLARE_SNIPPET (
            InitVarSnippet::Gamma ,
            2 )
```

19.29.2.2 getParamNames()

```
virtual StringVec InitVarSnippet::Gamma::getParamNames ( ) const  [inline], [override], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.29.2.3 SET_CODE()

```
InitVarSnippet::Gamma::SET_CODE ( )
```

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

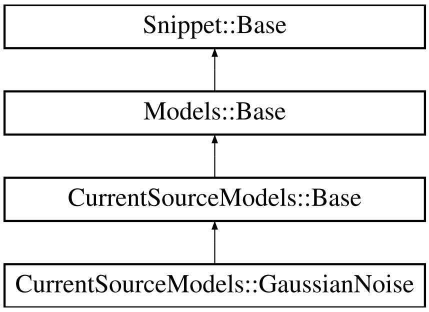
- initVarSnippet.h

19.30 CurrentSourceModels::GaussianNoise Class Reference

Noisy current source with noise drawn from normal distribution.

```
#include <currentSourceModels.h>
```

Inheritance diagram for CurrentSourceModels::GaussianNoise:



Public Types

- typedef Snippet::ValueBase< 2 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- SET_INJECTION_CODE ("$(injectCurrent, $(mean) + $(gennrand_normal) * $(sd));\)
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.

Static Public Member Functions

- static const GaussianNoise * getInstance ()

Additional Inherited Members

19.30.1 Detailed Description

Noisy current source with noise drawn from normal distribution.

It has 2 parameters:

- `mean` - mean of the normal distribution [nA]
- `sd` - standard deviation of the normal distribution [nA]

19.30.2 Member Typedef Documentation

19.30.2.1 ParamValues

`typedef Snippet::ValueBase< 2 > CurrentSourceModels::GaussianNoise::ParamValues`

19.30.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> CurrentSourceModels::GaussianNoise::PostVarValues
```

19.30.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> CurrentSourceModels::GaussianNoise::PreVarValues
```

19.30.2.4 VarValues

```
typedef Models::VarInitContainerBase< 0 > CurrentSourceModels::GaussianNoise::VarValues
```

19.30.3 Member Function Documentation

19.30.3.1 getInstance()

```
static const GaussianNoise* CurrentSourceModels::GaussianNoise::getInstance ( )  [inline],
[static]
```

19.30.3.2 getParamNames()

```
virtual StringVec CurrentSourceModels::GaussianNoise::getParamNames ( ) const  [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.30.3.3 SET_INJECTION_CODE()

```
CurrentSourceModels::GaussianNoise::SET_INJECTION_CODE (
            "$(injectCurrent, $(mean) + $(gennrand_normal) * $(sd));\  )
```

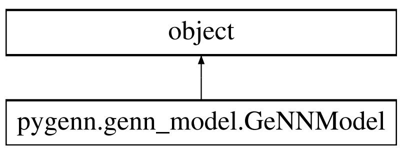
The documentation for this class was generated from the following file:

- currentSourceModels.h

19.31 pygenn.genn_model.GeNNModel Class Reference

GeNNModel class This class helps to define, build and run a GeNN model from python.

Inheritance diagram for pygenn.genn_model.GeNNModel:



Public Member Functions

- def __init__ (self, precision=None, model_name="GeNNModel", enable_debug=False, backend=None)

  Init GeNNModel.
- def use_backend (self)
- def use_backend (self, backend)
- def default_var_location (self)

  Default variable location - defines where state variables are initialised.
- def default_var_location (self, location)
- def default_sparse_connectivity_location (location)

  Default sparse connectivity mode - where connectivity is initialised.
- def default_sparse_connectivity_location (self, location)
- def model_name (self)

  Name of the model.
- def model_name (self, model_name)
- def t (self)

  Simulation time in ms.
- def t (self, t)
- def timestep (self)

  Simulation time step.
- def timestep (self, timestep)
- def dT (self)

  Step size.
- def dT (self, dt)
- def add_neuron_population (self, pop_name, num_neurons, neuron, param_space, var_space)

  Add a neuron population to the GeNN model.
- def add_synapse_population (self, pop_name, matrix_type, delay_steps, source, target, w_update_model, wu_param_space, wu_var_space, wu_pre_var_space, wu_post_var_space, postsyn_model, ps_param_↩ space, ps_var_space, connectivity_initialiser=None)

  Add a synapse population to the GeNN model.
- def add_current_source (self, cs_name, current_source_model, pop_name, param_space, var_space)

  Add a current source to the GeNN model.
- def build (self, path_to_model="./")

  Finalize and build a GeNN model.
- def load (self)

  import the model as shared library and initialize it
- def reinitialise (self)

  reinitialise model to its original state without re-loading
- def step_time (self)
- def pull_state_from_device (self, pop_name)

  Pull state from the device for a given population.
- def pull_spikes_from_device (self, pop_name)

  Pull spikes from the device for a given population.
- def pull_current_spikes_from_device (self, pop_name)

  Pull spikes from the device for a given population.
- def pull_connectivity_from_device (self, pop_name)

  Pull connectivity from the device for a given population.
- def pull_var_from_device (self, pop_name, var_name)

  Pull variable from the device for a given population.
- def push_state_to_device (self, pop_name)

  Push state to the device for a given population.
- def push_spikes_to_device (self, pop_name)

Push spikes to the device for a given population.
- def push_current_spikes_to_device (self, pop_name)

  Push current spikes to the device for a given population.
- def push_connectivity_to_device (self, pop_name)

  Push connectivity to the device for a given population.
- def push_var_to_device (self, pop_name, var_name)

  Push variable to the device for a given population.
- def end (self)

  Free memory.

Public Attributes

- use_backend
- default_var_location
- model_name
- neuron_populations
- synapse_populations
- current_sources
- dT
- T

19.31.1 Detailed Description

GeNNModel class This class helps to define, build and run a GeNN model from python.

19.31.2 Constructor & Destructor Documentation

19.31.2.1 __init__()

```
def pygenn.genn_model.GeNNModel.__init__ (
              self,
              precision = None,
              model_name = "GeNNModel",
              enable_debug = False,
              backend = None )
```

Init GeNNModel.

Parameters

precision	string precision as string ("float", "double" or "long double"). defaults to float.
model_name	string name of the model. Defaults to "GeNNModel".
enable_debug	boolean enable debug mode. Disabled by default.
backend	string specifying name of backend module to use Defaults to None to pick 'best' backend for your system

19.31.3 Member Function Documentation

19.31.3.1 add_current_source()

```
def pygenn.genn_model.GeNNModel.add_current_source (
              self,
              cs_name,
              current_source_model,
              pop_name,
              param_space,
              var_space )
```

Add a current source to the GeNN model.

Parameters

cs_name	name of the new current source
current_source_model	type of the CurrentSourceModels class as string or instance of CurrentSourceModels class derived from `pygenn.genn_wrapper.CurrentSourceModels.Custom` (see also pygenn.genn_model.create_custom_current_source_class)
pop_name	name of the population into which the current source should be injected
param_space	dict with param values for the CurrentSourceModels class
var_space	dict with initial variable values for the CurrentSourceModels class

19.31.3.2 add_neuron_population()

```
def pygenn.genn_model.GeNNModel.add_neuron_population (
              self,
              pop_name,
              num_neurons,
              neuron,
              param_space,
              var_space )
```

Add a neuron population to the GeNN model.

Parameters

pop_name	name of the new population
num_neurons	number of neurons in the new population
neuron	type of the NeuronModels class as string or instance of neuron class derived from `pygenn.genn_wrapper.NeuronModels.Custom` (see also pygenn.genn_model.create_custom_neuron_class)
param_space	dict with param values for the NeuronModels class
var_space	dict with initial variable values for the NeuronModels class

19.31.3.3 add_synapse_population()

```
def pygenn.genn_model.GeNNModel.add_synapse_population (
              self,
              pop_name,
              matrix_type,
              delay_steps,
              source,
```

```
                target,
                w_update_model,
                wu_param_space,
                wu_var_space,
                wu_pre_var_space,
                wu_post_var_space,
                postsyn_model,
                ps_param_space,
                ps_var_space,
                connectivity_initialiser = None )
```

Add a synapse population to the GeNN model.

Parameters

pop_name	name of the new population
matrix_type	type of the matrix as string
delay_steps	delay in number of steps
source	source neuron group
target	target neuron group
w_update_model	type of the WeightUpdateModels class as string or instance of weight update model class derived from `pygenn.genn_wrapper.WeightUpdateModels.Custom` (see also pygenn.genn_model.create_custom_weight_update_class)
wu_param_space	dict with param values for the WeightUpdateModels class
wu_var_space	dict with initial values for WeightUpdateModels state variables
wu_pre_var_space	dict with initial values for WeightUpdateModels presynaptic variables
wu_post_var_space	dict with initial values for WeightUpdateModels postsynaptic variables
postsyn_model	type of the PostsynapticModels class as string or instance of postsynaptic model class derived from `pygenn.genn_wrapper.PostsynapticModels.Custom` (see also pygenn.genn_model.create_custom_postsynaptic_class)
ps_param_space	dict with param values for the PostsynapticModels class
ps_var_space	dict with initial variable values for the PostsynapticModels class
connectivity_initialiser	InitSparseConnectivitySnippet::Init for connectivity

19.31.3.4 build()

```
def pygenn.genn_model.GeNNModel.build (
                self,
                path_to_model = "./" )
```

Finalize and build a GeNN model.

Parameters

path_to_model	path where to place the generated model code. Defaults to the local directory.

19.31.3.5 default_sparse_connectivity_location() [1/2]

```
def pygenn.genn_model.GeNNModel.default_sparse_connectivity_location (
                location )
```

Default sparse connectivity mode - where connectivity is initialised.

19.31.3.6 default_sparse_connectivity_location() [2/2]

```
def pygenn.genn_model.GeNNModel.default_sparse_connectivity_location (
              self,
              location )
```

19.31.3.7 default_var_location() [1/2]

```
def pygenn.genn_model.GeNNModel.default_var_location (
              self )
```

Default variable location - defines where state variables are initialised.

19.31.3.8 default_var_location() [2/2]

```
def pygenn.genn_model.GeNNModel.default_var_location (
              self,
              location )
```

19.31.3.9 dT() [1/2]

```
def pygenn.genn_model.GeNNModel.dT (
              self )
```

Step size.

19.31.3.10 dT() [2/2]

```
def pygenn.genn_model.GeNNModel.dT (
              self,
              dt )
```

19.31.3.11 end()

```
def pygenn.genn_model.GeNNModel.end (
              self )
```

Free memory.

19.31.3.12 load()

```
def pygenn.genn_model.GeNNModel.load (
              self )
```

import the model as shared library and initialize it

19.31.3.13 model_name() [1/2]

```
def pygenn.genn_model.GeNNModel.model_name (
```

```
self )
```

Name of the model.

19.31.3.14 model_name() [2/2]

```
def pygenn.genn_model.GeNNModel.model_name (
              self,
              model_name )
```

19.31.3.15 pull_connectivity_from_device()

```
def pygenn.genn_model.GeNNModel.pull_connectivity_from_device (
              self,
              pop_name )
```

Pull connectivity from the device for a given population.

19.31.3.16 pull_current_spikes_from_device()

```
def pygenn.genn_model.GeNNModel.pull_current_spikes_from_device (
              self,
              pop_name )
```

Pull spikes from the device for a given population.

19.31.3.17 pull_spikes_from_device()

```
def pygenn.genn_model.GeNNModel.pull_spikes_from_device (
              self,
              pop_name )
```

Pull spikes from the device for a given population.

19.31.3.18 pull_state_from_device()

```
def pygenn.genn_model.GeNNModel.pull_state_from_device (
              self,
              pop_name )
```

Pull state from the device for a given population.

19.31.3.19 pull_var_from_device()

```
def pygenn.genn_model.GeNNModel.pull_var_from_device (
              self,
              pop_name,
              var_name )
```

Pull variable from the device for a given population.

19.31.3.20 push_connectivity_to_device()

```
def pygenn.genn_model.GeNNModel.push_connectivity_to_device (
              self,
              pop_name )
```

Push connectivity to the device for a given population.

19.31.3.21 push_current_spikes_to_device()

```
def pygenn.genn_model.GeNNModel.push_current_spikes_to_device (
              self,
              pop_name )
```

Push current spikes to the device for a given population.

19.31.3.22 push_spikes_to_device()

```
def pygenn.genn_model.GeNNModel.push_spikes_to_device (
              self,
              pop_name )
```

Push spikes to the device for a given population.

19.31.3.23 push_state_to_device()

```
def pygenn.genn_model.GeNNModel.push_state_to_device (
              self,
              pop_name )
```

Push state to the device for a given population.

19.31.3.24 push_var_to_device()

```
def pygenn.genn_model.GeNNModel.push_var_to_device (
              self,
              pop_name,
              var_name )
```

Push variable to the device for a given population.

19.31.3.25 reinitialise()

```
def pygenn.genn_model.GeNNModel.reinitialise (
              self )
```

reinitialise model to its original state without re-loading

19.31.3.26 step_time()

```
def pygenn.genn_model.GeNNModel.step_time (
              self )
```

19.31.3.27 t() [1/2]

```
def pygenn.genn_model.GeNNModel.t (
              self )
```

Simulation time in ms.

19.31.3.28 t() [2/2]

```
def pygenn.genn_model.GeNNModel.t (
              self,
              t )
```

19.31.3.29 timestep() [1/2]

```
def pygenn.genn_model.GeNNModel.timestep (
              self )
```

Simulation time step.

19.31.3.30 timestep() [2/2]

```
def pygenn.genn_model.GeNNModel.timestep (
              self,
              timestep )
```

19.31.3.31 use_backend() [1/2]

```
def pygenn.genn_model.GeNNModel.use_backend (
              self )
```

19.31.3.32 use_backend() [2/2]

```
def pygenn.genn_model.GeNNModel.use_backend (
              self,
              backend )
```

19.31.4 Member Data Documentation

19.31.4.1 current_sources

```
pygenn.genn_model.GeNNModel.current_sources
```

19.31.4.2 default_var_location

```
pygenn.genn_model.GeNNModel.default_var_location
```

19.31.4.3 dT

`pygenn.genn_model.GeNNModel.dT`

19.31.4.4 model_name

`pygenn.genn_model.GeNNModel.model_name`

19.31.4.5 neuron_populations

`pygenn.genn_model.GeNNModel.neuron_populations`

19.31.4.6 synapse_populations

`pygenn.genn_model.GeNNModel.synapse_populations`

19.31.4.7 T

`pygenn.genn_model.GeNNModel.T`

19.31.4.8 use_backend

`pygenn.genn_model.GeNNModel.use_backend`

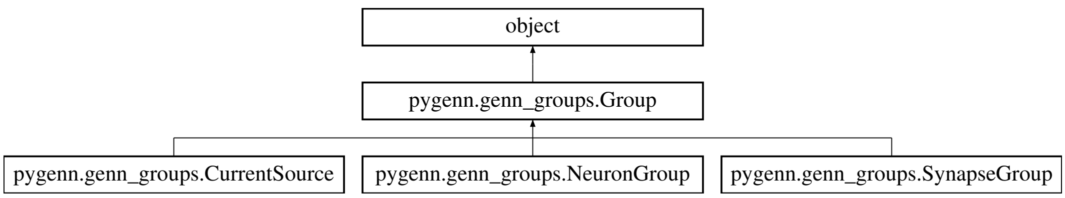
The documentation for this class was generated from the following file:

- genn_model.py

19.32 pygenn.genn_groups.Group Class Reference

Parent class of NeuronGroup, SynapseGroup and CurrentSource.

Inheritance diagram for pygenn.genn_groups.Group:



Public Member Functions

- def __init__ (self, name)
  Init Group.
- def set_var (self, var_name, values)
  Set values for a Variable.

Public Attributes

- name

- vars
- extra_global_params

19.32.1 Detailed Description

Parent class of NeuronGroup, SynapseGroup and CurrentSource.

19.32.2 Constructor & Destructor Documentation

19.32.2.1 __init__()

```
def pygenn.genn_groups.Group.__init__ (
            self,
            name )
```

Init Group.

Parameters

name	string name of the Group

19.32.3 Member Function Documentation

19.32.3.1 set_var()

```
def pygenn.genn_groups.Group.set_var (
            self,
            var_name,
            values )
```

Set values for a Variable.

Parameters

var_name	string with the name of the variable
values	iterable or a single value

19.32.4 Member Data Documentation

19.32.4.1 extra_global_params

```
pygenn.genn_groups.Group.extra_global_params
```

19.32.4.2 name

```
pygenn.genn_groups.Group.name
```

19.32.4.3 vars

```
pygenn.genn_groups.Group.vars
```

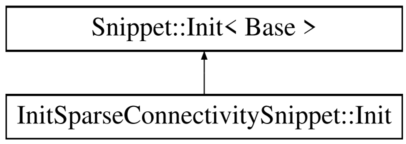
The documentation for this class was generated from the following file:

- genn_groups.py

19.33 InitSparseConnectivitySnippet::Init Class Reference

```
#include <initSparseConnectivitySnippet.h>
```

Inheritance diagram for InitSparseConnectivitySnippet::Init:



Public Member Functions

- Init (const Base ∗snippet, const std::vector< double > &params)

19.33.1 Constructor & Destructor Documentation

19.33.1.1 Init()

```
InitSparseConnectivitySnippet::Init::Init (
            const Base ∗ snippet,
            const std::vector< double > & params ) [inline]
```

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

- initSparseConnectivitySnippet.h

19.34 Snippet::Init< SnippetBase > Class Template Reference

```
#include <snippet.h>
```

Public Member Functions

- Init (const SnippetBase ∗snippet, const std::vector< double > &params)
- const SnippetBase ∗ getSnippet () const
- const std::vector< double > & getParams () const
- const std::vector< double > & getDerivedParams () const
- void initDerivedParams (double dt)

19.34.1 Detailed Description

template<typename SnippetBase>
class Snippet::Init< SnippetBase >

Class used to bind together everything required to utilize a snippet

1. A pointer to a variable initialisation snippet
2. The parameters required to control the variable initialisation snippet

19.34.2 Constructor & Destructor Documentation

19.34.2.1 Init()

```
template<typename SnippetBase>
Snippet::Init< SnippetBase >::Init (
            const SnippetBase * snippet,
            const std::vector< double > & params )  [inline]
```

19.34.3 Member Function Documentation

19.34.3.1 getDerivedParams()

```
template<typename SnippetBase>
const std::vector<double>& Snippet::Init< SnippetBase >::getDerivedParams ( ) const  [inline]
```

19.34.3.2 getParams()

```
template<typename SnippetBase>
const std::vector<double>& Snippet::Init< SnippetBase >::getParams ( ) const  [inline]
```

19.34.3.3 getSnippet()

```
template<typename SnippetBase>
const SnippetBase* Snippet::Init< SnippetBase >::getSnippet ( ) const  [inline]
```

19.34.3.4 initDerivedParams()

```
template<typename SnippetBase>
void Snippet::Init< SnippetBase >::initDerivedParams (
            double dt )  [inline]
```

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

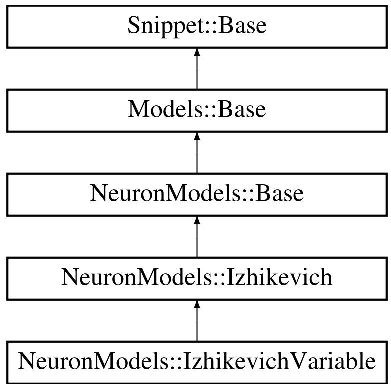
- snippet.h

19.35 NeuronModels::Izhikevich Class Reference

Izhikevich neuron with fixed parameters [1].

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::Izhikevich:



Public Types

- typedef Snippet::ValueBase< 4 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

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

Static Public Member Functions

- static const NeuronModels::Izhikevich * getInstance ()

Additional Inherited Members

19.35.1 Detailed Description

Izhikevich neuron with fixed parameters [1].

It is usually described as

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

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

Variables are:

- V - Membrane potential
- U - Membrane recovery variable

Parameters are:

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

19.35.2 Member Typedef Documentation

19.35.2.1 ParamValues

`typedef Snippet::ValueBase< 4 > NeuronModels::Izhikevich::ParamValues`

19.35.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::Izhikevich::PostVarValues`

19.35.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::Izhikevich::PreVarValues`

19.35.2.4 VarValues

`typedef Models::VarInitContainerBase< 2 > NeuronModels::Izhikevich::VarValues`

19.35.3 Member Function Documentation

19.35.3.1 getInstance()

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

19.35.3.2 getParamNames()

`virtual StringVec NeuronModels::Izhikevich::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

Reimplemented in NeuronModels::IzhikevichVariable.

19.35.3.3 getSimCode()

```
virtual std::string NeuronModels::Izhikevich::getSimCode ( ) const [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.35.3.4 getThresholdConditionCode()

```
virtual std::string NeuronModels::Izhikevich::getThresholdConditionCode ( ) const [inline],
[override], [virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.35.3.5 getVars()

```
virtual VarVec NeuronModels::Izhikevich::getVars ( ) const [inline], [override], [virtual]
```

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

Reimplemented from Models::Base.

Reimplemented in NeuronModels::IzhikevichVariable.

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

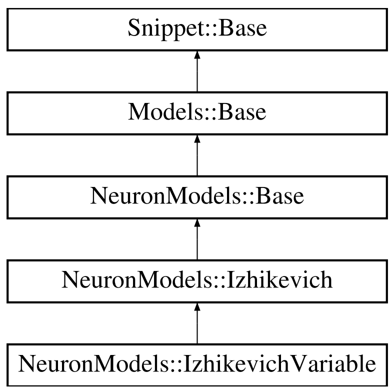
- neuronModels.h

19.36 NeuronModels::IzhikevichVariable Class Reference

Izhikevich neuron with variable parameters [1].

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::IzhikevichVariable:



Public Types

- typedef Snippet::ValueBase< 0 > ParamValues
- typedef Models::VarInitContainerBase< 6 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

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

Static Public Member Functions

- static const NeuronModels::IzhikevichVariable * getInstance ()

Additional Inherited Members

19.36.1 Detailed Description

Izhikevich neuron with variable parameters [1].

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

Accordingly, the model has the Variables:

- `V` - Membrane potential
- `U` - Membrane recovery variable
- `a` - time scale of U
- `b` - sensitivity of U
- `c` - after-spike reset value of V
- `d` - after-spike reset value of U

and no parameters.

19.36.2 Member Typedef Documentation

19.36.2.1 ParamValues

```
typedef Snippet::ValueBase< 0 > NeuronModels::IzhikevichVariable::ParamValues
```

19.36.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::IzhikevichVariable::PostVarValues
```

19.36.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::IzhikevichVariable::PreVarValues
```

19.36.2.4 VarValues

```
typedef Models::VarInitContainerBase< 6 > NeuronModels::IzhikevichVariable::VarValues
```

19.36.3 Member Function Documentation

19.36.3.1 getInstance()

```
static const NeuronModels::IzhikevichVariable* NeuronModels::IzhikevichVariable::getInstance (
) [inline], [static]
```

19.36.3.2 getParamNames()

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

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::Izhikevich.

19.36.3.3 getVars()

```
virtual VarVec NeuronModels::IzhikevichVariable::getVars ( ) const [inline], [override],
[virtual]
```

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

Reimplemented from NeuronModels::Izhikevich.

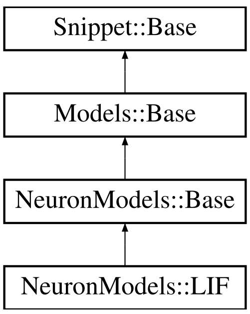
The documentation for this class was generated from the following file:

- neuronModels.h

19.37 NeuronModels::LIF Class Reference

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::LIF:



Public Types

- typedef Snippet::ValueBase< 7 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual std::string getResetCode () const override

  Gets code that defines the reset action taken after a spike occurred. This can be empty.
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual DerivedParamVec getDerivedParams () const override
- virtual VarVec getVars () const override

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

Static Public Member Functions

- static const LIF * getInstance ()

Additional Inherited Members

19.37.1 Member Typedef Documentation

19.37.1.1 ParamValues

```
typedef Snippet::ValueBase< 7 > NeuronModels::LIF::ParamValues
```

19.37.1.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::LIF::PostVarValues
```

19.37.1.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::LIF::PreVarValues
```

19.37.1.4 VarValues

```
typedef Models::VarInitContainerBase< 2 > NeuronModels::LIF::VarValues
```

19.37.2 Member Function Documentation

19.37.2.1 getDerivedParams()

`virtual DerivedParamVec NeuronModels::LIF::getDerivedParams ( ) const [inline], [override], [virtual]`

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

Reimplemented from Snippet::Base.

19.37.2.2 getInstance()

`static const LIF* NeuronModels::LIF::getInstance ( ) [inline], [static]`

19.37.2.3 getParamNames()

`virtual StringVec NeuronModels::LIF::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.37.2.4 getResetCode()

`virtual std::string NeuronModels::LIF::getResetCode ( ) const [inline], [override], [virtual]`

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

Reimplemented from NeuronModels::Base.

19.37.2.5 getSimCode()

`virtual std::string NeuronModels::LIF::getSimCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.37.2.6 getThresholdConditionCode()

`virtual std::string NeuronModels::LIF::getThresholdConditionCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.37.2.7 getVars()

`virtual VarVec NeuronModels::LIF::getVars ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

19.37.2.8 SET_NEEDS_AUTO_REFRACTORY()

```
NeuronModels::LIF::SET_NEEDS_AUTO_REFRACTORY (
            false )
```

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

- neuronModels.h

19.38 CodeGenerator::MemAlloc Class Reference

```
#include <backendBase.h>
```

Public Member Functions

- size_t getHostBytes () const
- size_t getDeviceBytes () const
- size_t getZeroCopyBytes () const
- size_t getHostMBytes () const
- size_t getDeviceMBytes () const
- size_t getZeroCopyMBytes () const
- MemAlloc & operator+= (const MemAlloc &rhs)

Static Public Member Functions

- static MemAlloc zero ()
- static MemAlloc host (size_t hostBytes)
- static MemAlloc device (size_t deviceBytes)
- static MemAlloc zeroCopy (size_t zeroCopyBytes)

19.38.1 Member Function Documentation

19.38.1.1 device()

```
static MemAlloc CodeGenerator::MemAlloc::device (
            size_t deviceBytes )  [inline], [static]
```

19.38.1.2 getDeviceBytes()

```
size_t CodeGenerator::MemAlloc::getDeviceBytes ( ) const  [inline]
```

19.38.1.3 getDeviceMBytes()

```
size_t CodeGenerator::MemAlloc::getDeviceMBytes ( ) const  [inline]
```

19.38.1.4 getHostBytes()

size_t CodeGenerator::MemAlloc::getHostBytes ( ) const [inline]

19.38.1.5 getHostMBytes()

size_t CodeGenerator::MemAlloc::getHostMBytes ( ) const [inline]

19.38.1.6 getZeroCopyBytes()

size_t CodeGenerator::MemAlloc::getZeroCopyBytes ( ) const [inline]

19.38.1.7 getZeroCopyMBytes()

size_t CodeGenerator::MemAlloc::getZeroCopyMBytes ( ) const [inline]

19.38.1.8 host()

static MemAlloc CodeGenerator::MemAlloc::host (
size_t *hostBytes*) [inline], [static]

19.38.1.9 operator+=()

MemAlloc& CodeGenerator::MemAlloc::operator+= (
const MemAlloc & *rhs*) [inline]

19.38.1.10 zero()

static MemAlloc CodeGenerator::MemAlloc::zero ( ) [inline], [static]

19.38.1.11 zeroCopy()

static MemAlloc CodeGenerator::MemAlloc::zeroCopy (
size_t *zeroCopyBytes*) [inline], [static]

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

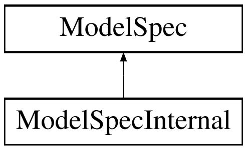
- backendBase.h

19.39 ModelSpec Class Reference

Object used for specifying a neuronal network model.

#include <modelSpec.h>

Inheritance diagram for ModelSpec:



Public Types

- typedef std::map< std::string, NeuronGroupInternal >::value_type NeuronGroupValueType
- typedef std::map< std::string, SynapseGroupInternal >::value_type SynapseGroupValueType

Public Member Functions

- ModelSpec ()
- ModelSpec (const ModelSpec &)=delete
- ModelSpec & operator= (const ModelSpec &)=delete
- ∼ModelSpec ()
- void setName (const std::string &name)

  Method to set the neuronal network model name.
- void setPrecision (FloatType)

  Set numerical precision for floating point.
- void setTimePrecision (TimePrecision timePrecision)

  Set numerical precision for time.
- void setDT (double dt)

  Set the integration step size of the model.
- void setTiming (bool timingEnabled)

  Set whether timers and timing commands are to be included.
- void setSeed (unsigned int rngSeed)

  Set the random seed (disables automatic seeding if argument not 0).
- void setDefaultVarLocation (VarLocation loc)

  What is the default location for model state variables?
- void setDefaultExtraGlobalParamLocation (VarLocation loc)

  What is the default location for model extra global parameters?
- void setDefaultSparseConnectivityLocation (VarLocation loc)

  What is the default location for sparse synaptic connectivity?
- void setMergePostsynapticModels (bool merge)

  Should compatible postsynaptic models and dendritic delay buffers be merged?
- const std::string & getName () const

  Gets the name of the neuronal network model.
- const std::string & getPrecision () const

  Gets the floating point numerical precision.
- std::string getTimePrecision () const

  Gets the floating point numerical precision used to represent time.
- double getDT () const

  Gets the model integration step size.
- unsigned int getSeed () const

  Get the random seed.
- bool isTimingEnabled () const

  Are timers and timing commands enabled.
- unsigned int getNumLocalNeurons () const

  How many neurons are simulated locally in this model.

- unsigned int getNumRemoteNeurons () const

  How many neurons are simulated remotely in this model.
- unsigned int getNumNeurons () const

  How many neurons make up the entire model.
- NeuronGroup ∗ findNeuronGroup (const std::string &name)

  Find a neuron group by name.
- template<typename NeuronModel >
  NeuronGroup ∗ addNeuronPopulation (const std::string &name, unsigned int size, const NeuronModel ∗model, const typename NeuronModel::ParamValues &paramValues, const typename NeuronModel::VarValues &varInitialisers, int hostID=0)

  Adds a new neuron group to the model using a neuron model managed by the user.
- template<typename NeuronModel >
  NeuronGroup ∗ addNeuronPopulation (const std::string &name, unsigned int size, const typename NeuronModel::ParamValues &paramValues, const typename NeuronModel::VarValues &varInitialisers, int hostID=0)

  Adds a new neuron group to the model using a singleton neuron model created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.
- SynapseGroup ∗ findSynapseGroup (const std::string &name)

  Find a synapse group by name.
- template<typename WeightUpdateModel , typename PostsynapticModel >
  SynapseGroup ∗ addSynapsePopulation (const std::string &name, SynapseMatrixType mtype, unsigned int delaySteps, const std::string &src, const std::string &trg, const WeightUpdateModel ∗wum, const typename WeightUpdateModel::ParamValues &weightParamValues, const typename WeightUpdateModel::VarValues &weightVarInitialisers, const typename WeightUpdateModel::PreVarValues &weightPreVarInitialisers, const typename WeightUpdateModel::PostVarValues &weightPostVarInitialisers, const PostsynapticModel ∗psm, const typename PostsynapticModel::ParamValues &postsynapticParamValues, const typename PostsynapticModel::VarValues &postsynapticVarInitialisers, const InitSparseConnectivitySnippet::Init &connectivityInitialiser=uninitialisedConnectivity())

  Adds a synapse population to the model using weight update and postsynaptic models managed by the user.
- template<typename WeightUpdateModel , typename PostsynapticModel >
  SynapseGroup ∗ addSynapsePopulation (const std::string &name, SynapseMatrixType mtype, unsigned int delaySteps, const std::string &src, const std::string &trg, const typename WeightUpdateModel::ParamValues &weightParamValues, const typename WeightUpdateModel::VarValues &weightVarInitialisers, const typename PostsynapticModel::ParamValues &postsynapticParamValues, const typename PostsynapticModel::VarValues &postsynapticVarInitialisers, const InitSparseConnectivitySnippet::Init &connectivityInitialiser=uninitialisedConnectivity())

  Adds a synapse population to the model using singleton weight update and postsynaptic models created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.
- template<typename WeightUpdateModel , typename PostsynapticModel >
  SynapseGroup ∗ addSynapsePopulation (const std::string &name, SynapseMatrixType mtype, unsigned int delaySteps, const std::string &src, const std::string &trg, const typename WeightUpdateModel::ParamValues &weightParamValues, const typename WeightUpdateModel::VarValues &weightVarInitialisers, const typename WeightUpdateModel::PreVarValues &weightPreVarInitialisers, const typename WeightUpdateModel::PostVarValues &weightPostVarInitialisers, const typename PostsynapticModel::ParamValues &postsynapticParamValues, const typename PostsynapticModel::VarValues &postsynapticVarInitialisers, const InitSparseConnectivitySnippet::Init &connectivityInitialiser=uninitialisedConnectivity())

  Adds a synapse population to the model using singleton weight update and postsynaptic models created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.
- CurrentSource ∗ findCurrentSource (const std::string &name)

  Find a current source by name.
- template<typename CurrentSourceModel >
  CurrentSource ∗ addCurrentSource (const std::string &currentSourceName, const CurrentSourceModel ∗model, const std::string &targetNeuronGroupName, const typename CurrentSourceModel::ParamValues &paramValues, const typename CurrentSourceModel::VarValues &varInitialisers)

  Adds a new current source to the model using a current source model managed by the user.

- template<typename CurrentSourceModel >
CurrentSource * addCurrentSource (const std::string &currentSourceName, const std::string &target←NeuronGroupName, const typename CurrentSourceModel::ParamValues &paramValues, const typename CurrentSourceModel::VarValues &varInitialisers)

    Adds a new current source to the model using a singleton current source model created using standard DECLARE←_MODEL and IMPLEMENT_MODEL macros.

Protected Member Functions

- void finalize ()

    Finalise model.
- std::string scalarExpr (double) const

    Get the string literal that should be used to represent a value in the model's floating-point type.
- bool zeroCopyInUse () const

    Are any variables in any populations in this model using zero-copy memory?
- const std::map< std::string, NeuronGroupInternal > & getLocalNeuronGroups () const

    Get std::map containing local named NeuronGroup objects in model.
- const std::map< std::string, NeuronGroupInternal > & getRemoteNeuronGroups () const

    Get std::map containing remote named NeuronGroup objects in model.
- const std::map< std::string, SynapseGroupInternal > & getLocalSynapseGroups () const

    Get std::map containing local named SynapseGroup objects in model.
- const std::map< std::string, SynapseGroupInternal > & getRemoteSynapseGroups () const

    Get std::map containing remote named SynapseGroup objects in model.
- const std::map< std::string, CurrentSourceInternal > & getLocalCurrentSources () const

    Get std::map containing local named CurrentSource objects in model.
- const std::map< std::string, CurrentSourceInternal > & getRemoteCurrentSources () const

    Get std::map containing remote named CurrentSource objects in model.

19.39.1 Detailed Description

Object used for specifying a neuronal network model.

19.39.2 Member Typedef Documentation

19.39.2.1 NeuronGroupValueType

```
typedef std::map<std::string, NeuronGroupInternal>::value_type ModelSpec::NeuronGroupValueType
```

19.39.2.2 SynapseGroupValueType

```
typedef std::map<std::string, SynapseGroupInternal>::value_type ModelSpec::SynapseGroupValue←
Type
```

19.39.3 Constructor & Destructor Documentation

19.39.3.1 ModelSpec() [1/2]

```
ModelSpec::ModelSpec ( )
```

19.39.3.2 ModelSpec() [2/2]

```
ModelSpec::ModelSpec (
            const ModelSpec & )  [delete]
```

19.39.3.3 ∼ModelSpec()

```
ModelSpec::∼ModelSpec ( )
```

19.39.4 Member Function Documentation

19.39.4.1 addCurrentSource() [1/2]

```
template<typename CurrentSourceModel >
CurrentSource* ModelSpec::addCurrentSource (
            const std::string & currentSourceName,
            const CurrentSourceModel * model,
            const std::string & targetNeuronGroupName,
            const typename CurrentSourceModel::ParamValues & paramValues,
            const typename CurrentSourceModel::VarValues & varInitialisers )  [inline]
```

Adds a new current source to the model using a current source model managed by the user.

Template Parameters

CurrentSourceModel	type of current source model (derived from CurrentSourceModels::Base).

Parameters

currentSourceName	string containing unique name of current source.
model	current source model to use for current source.
targetNeuronGroupName	string name of the target neuron group
paramValues	parameters for model wrapped in CurrentSourceModel::ParamValues object.
varInitialisers	state variable initialiser snippets and parameters wrapped in CurrentSource::VarValues object.

Returns

pointer to newly created CurrentSource

19.39.4.2 addCurrentSource() [2/2]

```
template<typename CurrentSourceModel >
CurrentSource* ModelSpec::addCurrentSource (
            const std::string & currentSourceName,
            const std::string & targetNeuronGroupName,
```

```
            const typename CurrentSourceModel::ParamValues & paramValues,
            const typename CurrentSourceModel::VarValues & varInitialisers )  [inline]
```

Adds a new current source to the model using a singleton current source model created using standard DECLAR↩ E_MODEL and IMPLEMENT_MODEL macros.

Template Parameters

CurrentSourceModel	type of neuron model (derived from CurrentSourceModel::Base).

Parameters

currentSourceName	string containing unique name of current source.
targetNeuronGroupName	string name of the target neuron group
paramValues	parameters for model wrapped in CurrentSourceModel::ParamValues object.
varInitialisers	state variable initialiser snippets and parameters wrapped in CurrentSourceModel::VarValues object.

Returns

pointer to newly created CurrentSource

19.39.4.3 addNeuronPopulation() [1/2]

```
template<typename NeuronModel >
NeuronGroup* ModelSpec::addNeuronPopulation (
            const std::string & name,
            unsigned int size,
            const NeuronModel * model,
            const typename NeuronModel::ParamValues & paramValues,
            const typename NeuronModel::VarValues & varInitialisers,
            int hostID = 0 )  [inline]
```

Adds a new neuron group to the model using a neuron model managed by the user.

Template Parameters

NeuronModel	type of neuron model (derived from NeuronModels::Base).

Parameters

name	string containing unique name of neuron population.
size	integer specifying how many neurons are in the population.
model	neuron model to use for neuron group.
paramValues	parameters for model wrapped in NeuronModel::ParamValues object.
varInitialisers	state variable initialiser snippets and parameters wrapped in NeuronModel::VarValues object.
hostID	if using MPI, the ID of the node to simulate this population on.

Returns

pointer to newly created NeuronGroup

19.39.4.4 addNeuronPopulation() [2/2]

```
template<typename NeuronModel >
NeuronGroup* ModelSpec::addNeuronPopulation (
             const std::string & name,
             unsigned int size,
             const typename NeuronModel::ParamValues & paramValues,
             const typename NeuronModel::VarValues & varInitialisers,
             int hostID = 0 )  [inline]
```

Adds a new neuron group to the model using a singleton neuron model created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.

Template Parameters

NeuronModel	type of neuron model (derived from NeuronModels::Base).

Parameters

name	string containing unique name of neuron population.
size	integer specifying how many neurons are in the population.
paramValues	parameters for model wrapped in NeuronModel::ParamValues object.
varInitialisers	state variable initialiser snippets and parameters wrapped in NeuronModel::VarValues object.
hostID	if using MPI, the ID of the node to simulate this population on.

Returns

pointer to newly created NeuronGroup

19.39.4.5 addSynapsePopulation() [1/3]

```
template<typename WeightUpdateModel , typename PostsynapticModel >
SynapseGroup* ModelSpec::addSynapsePopulation (
             const std::string & name,
             SynapseMatrixType mtype,
             unsigned int delaySteps,
             const std::string & src,
             const std::string & trg,
             const WeightUpdateModel * wum,
             const typename WeightUpdateModel::ParamValues & weightParamValues,
             const typename WeightUpdateModel::VarValues & weightVarInitialisers,
             const typename WeightUpdateModel::PreVarValues & weightPreVarInitialisers,
             const typename WeightUpdateModel::PostVarValues & weightPostVarInitialisers,
             const PostsynapticModel * psm,
             const typename PostsynapticModel::ParamValues & postsynapticParamValues,
             const typename PostsynapticModel::VarValues & postsynapticVarInitialisers,
             const InitSparseConnectivitySnippet::Init & connectivityInitialiser = uninitialised↩
Connectivity() )  [inline]
```

Adds a synapse population to the model using weight update and postsynaptic models managed by the user.

Template Parameters

WeightUpdateModel	type of weight update model (derived from WeightUpdateModels::Base).
PostsynapticModel	type of postsynaptic model (derived from PostsynapticModels::Base).

Parameters

name	string containing unique name of neuron population.
mtype	how the synaptic matrix associated with this synapse population should be represented.
delaySteps	integer specifying number of timesteps delay this synaptic connection should incur (or NO_DELAY for none)
src	string specifying name of presynaptic (source) population
trg	string specifying name of postsynaptic (target) population
wum	weight update model to use for synapse group.
weightParamValues	parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.
weightVarInitialisers	weight update model state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
weightPreVarInitialisers	weight update model presynaptic state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
weightPostVarInitialisers	weight update model postsynaptic state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
psm	postsynaptic model to use for synapse group.
postsynapticParamValues	parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.
postsynapticVarInitialisers	postsynaptic model state variable initialiser snippets and parameters wrapped in NeuronModel::VarValues object.
connectivityInitialiser	sparse connectivity initialisation snippet used to initialise connectivity for SynapseMatrixConnectivity::SPARSE or SynapseMatrixConnectivity::BITMASK. Typically wrapped with it's parameters using `initConnectivity` function

Returns

pointer to newly created SynapseGroup

19.39.4.6 addSynapsePopulation() [2/3]

```
template<typename WeightUpdateModel , typename PostsynapticModel >
SynapseGroup* ModelSpec::addSynapsePopulation (
            const std::string & name,
            SynapseMatrixType mtype,
            unsigned int delaySteps,
            const std::string & src,
            const std::string & trg,
            const typename WeightUpdateModel::ParamValues & weightParamValues,
            const typename WeightUpdateModel::VarValues & weightVarInitialisers,
            const typename PostsynapticModel::ParamValues & postsynapticParamValues,
            const typename PostsynapticModel::VarValues & postsynapticVarInitialisers,
            const InitSparseConnectivitySnippet::Init & connectivityInitialiser = uninitialised↵
Connectivity() )  [inline]
```

Adds a synapse population to the model using singleton weight update and postsynaptic models created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.

Template Parameters

WeightUpdateModel	type of weight update model (derived from WeightUpdateModels::Base).
PostsynapticModel	type of postsynaptic model (derived from PostsynapticModels::Base).

Parameters

name	string containing unique name of neuron population.
mtype	how the synaptic matrix associated with this synapse population should be represented.
delaySteps	integer specifying number of timesteps delay this synaptic connection should incur (or NO_DELAY for none)
src	string specifying name of presynaptic (source) population
trg	string specifying name of postsynaptic (target) population
weightParamValues	parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.
weightVarInitialisers	weight update model state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
postsynapticParamValues	parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.
postsynapticVarInitialisers	postsynaptic model state variable initialiser snippets and parameters wrapped in NeuronModel::VarValues object.
connectivityInitialiser	sparse connectivity initialisation snippet used to initialise connectivity for SynapseMatrixConnectivity::SPARSE or SynapseMatrixConnectivity::BITMASK. Typically wrapped with it's parameters using `initConnectivity` function

Returns

pointer to newly created SynapseGroup

19.39.4.7 addSynapsePopulation() [3/3]

```
template<typename WeightUpdateModel , typename PostsynapticModel >
SynapseGroup* ModelSpec::addSynapsePopulation (
            const std::string & name,
            SynapseMatrixType mtype,
            unsigned int delaySteps,
            const std::string & src,
            const std::string & trg,
            const typename WeightUpdateModel::ParamValues & weightParamValues,
            const typename WeightUpdateModel::VarValues & weightVarInitialisers,
            const typename WeightUpdateModel::PreVarValues & weightPreVarInitialisers,
            const typename WeightUpdateModel::PostVarValues & weightPostVarInitialisers,
            const typename PostsynapticModel::ParamValues & postsynapticParamValues,
            const typename PostsynapticModel::VarValues & postsynapticVarInitialisers,
            const InitSparseConnectivitySnippet::Init & connectivityInitialiser = uninitialised←
Connectivity() ) [inline]
```

Adds a synapse population to the model using singleton weight update and postsynaptic models created using standard DECLARE_MODEL and IMPLEMENT_MODEL macros.

Template Parameters

WeightUpdateModel	type of weight update model (derived from WeightUpdateModels::Base).
PostsynapticModel	type of postsynaptic model (derived from PostsynapticModels::Base).

Parameters

name	string containing unique name of neuron population.
mtype	how the synaptic matrix associated with this synapse population should be represented.
delaySteps	integer specifying number of timesteps delay this synaptic connection should incur (or NO_DELAY for none)
src	string specifying name of presynaptic (source) population
trg	string specifying name of postsynaptic (target) population
weightParamValues	parameters for weight update model wrapped in WeightUpdateModel::ParamValues object.
weightVarInitialisers	weight update model per-synapse state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
weightPreVarInitialisers	weight update model presynaptic state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
weightPostVarInitialisers	weight update model postsynaptic state variable initialiser snippets and parameters wrapped in WeightUpdateModel::VarValues object.
postsynapticParamValues	parameters for postsynaptic model wrapped in PostsynapticModel::ParamValues object.
postsynapticVarInitialisers	postsynaptic model state variable initialiser snippets and parameters wrapped in NeuronModel::VarValues object.
connectivityInitialiser	sparse connectivity initialisation snippet used to initialise connectivity for SynapseMatrixConnectivity::SPARSE or SynapseMatrixConnectivity::BITMASK. Typically wrapped with it's parameters using `initConnectivity` function

Returns

pointer to newly created SynapseGroup

19.39.4.8 finalize()

```
void ModelSpec::finalize ( )  [protected]
```

Finalise model.

19.39.4.9 findCurrentSource()

```
CurrentSource * ModelSpec::findCurrentSource (
            const std::string & name )
```

Find a current source by name.

This function attempts to find an existing current source.

19.39.4.10 findNeuronGroup()

```
NeuronGroup* ModelSpec::findNeuronGroup (
            const std::string & name )  [inline]
```

Find a neuron group by name.

19.39.4.11 findSynapseGroup()

```
SynapseGroup * ModelSpec::findSynapseGroup (
            const std::string & name )
```

Find a synapse group by name.

19.39.4.12 getDT()

```
double ModelSpec::getDT ( ) const  [inline]
```

Gets the model integration step size.

19.39.4.13 getLocalCurrentSources()

```
const std::map<std::string, CurrentSourceInternal>& ModelSpec::getLocalCurrentSources ( )
const  [inline], [protected]
```

Get std::map containing local named CurrentSource objects in model.

19.39.4.14 getLocalNeuronGroups()

```
const std::map<std::string, NeuronGroupInternal>& ModelSpec::getLocalNeuronGroups ( ) const
[inline], [protected]
```

Get std::map containing local named NeuronGroup objects in model.

19.39.4.15 getLocalSynapseGroups()

```
const std::map<std::string, SynapseGroupInternal>& ModelSpec::getLocalSynapseGroups ( ) const
[inline], [protected]
```

Get std::map containing local named SynapseGroup objects in model.

19.39.4.16 getName()

```
const std::string& ModelSpec::getName ( ) const  [inline]
```

Gets the name of the neuronal network model.

19.39.4.17 getNumLocalNeurons()

```
unsigned int ModelSpec::getNumLocalNeurons ( ) const
```

How many neurons are simulated locally in this model.

19.39.4.18 getNumNeurons()

```
unsigned int ModelSpec::getNumNeurons ( ) const  [inline]
```

How many neurons make up the entire model.

19.39.4.19 getNumRemoteNeurons()

`unsigned int ModelSpec::getNumRemoteNeurons ( ) const`

How many neurons are simulated remotely in this model.

19.39.4.20 getPrecision()

`const std::string& ModelSpec::getPrecision ( ) const [inline]`

Gets the floating point numerical precision.

19.39.4.21 getRemoteCurrentSources()

`const std::map<std::string, CurrentSourceInternal>& ModelSpec::getRemoteCurrentSources ( ) const [inline], [protected]`

Get std::map containing remote named CurrentSource objects in model.

19.39.4.22 getRemoteNeuronGroups()

`const std::map<std::string, NeuronGroupInternal>& ModelSpec::getRemoteNeuronGroups ( ) const [inline], [protected]`

Get std::map containing remote named NeuronGroup objects in model.

19.39.4.23 getRemoteSynapseGroups()

`const std::map<std::string, SynapseGroupInternal>& ModelSpec::getRemoteSynapseGroups ( ) const [inline], [protected]`

Get std::map containing remote named SynapseGroup objects in model.

19.39.4.24 getSeed()

`unsigned int ModelSpec::getSeed ( ) const [inline]`

Get the random seed.

19.39.4.25 getTimePrecision()

`std::string ModelSpec::getTimePrecision ( ) const`

Gets the floating point numerical precision used to represent time.

19.39.4.26 isTimingEnabled()

`bool ModelSpec::isTimingEnabled ( ) const [inline]`

Are timers and timing commands enabled.

19.39.4.27 operator=()

```
ModelSpec& ModelSpec::operator= (
            const ModelSpec & ) [delete]
```

19.39.4.28 scalarExpr()

```
std::string ModelSpec::scalarExpr (
            double val ) const [protected]
```

Get the string literal that should be used to represent a value in the model's floating-point type.

19.39.4.29 setDefaultExtraGlobalParamLocation()

```
void ModelSpec::setDefaultExtraGlobalParamLocation (
            VarLocation loc ) [inline]
```

What is the default location for model extra global parameters?

Historically, this was just left up to the user to handle

19.39.4.30 setDefaultSparseConnectivityLocation()

```
void ModelSpec::setDefaultSparseConnectivityLocation (
            VarLocation loc ) [inline]
```

What is the default location for sparse synaptic connectivity?

Historically, everything was allocated on both the host AND device

19.39.4.31 setDefaultVarLocation()

```
void ModelSpec::setDefaultVarLocation (
            VarLocation loc ) [inline]
```

What is the default location for model state variables?

Historically, everything was allocated on both the host AND device

19.39.4.32 setDT()

```
void ModelSpec::setDT (
            double dt ) [inline]
```

Set the integration step size of the model.

19.39.4.33 setMergePostsynapticModels()

```
void ModelSpec::setMergePostsynapticModels (
            bool merge ) [inline]
```

Should compatible postsynaptic models and dendritic delay buffers be merged?

This can significantly reduce the cost of updating neuron population but means that per-synapse group inSyn arrays can not be retrieved

19.39.4.34 setName()

```
void ModelSpec::setName (
            const std::string & name ) [inline]
```

Method to set the neuronal network model name.

19.39.4.35 setPrecision()

```
void ModelSpec::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.

19.39.4.36 setSeed()

```
void ModelSpec::setSeed (
            unsigned int rngSeed )  [inline]
```

Set the random seed (disables automatic seeding if argument not 0).

19.39.4.37 setTimePrecision()

```
void ModelSpec::setTimePrecision (
            TimePrecision timePrecision )  [inline]
```

Set numerical precision for time.

19.39.4.38 setTiming()

```
void ModelSpec::setTiming (
            bool timingEnabled )  [inline]
```

Set whether timers and timing commands are to be included.

19.39.4.39 zeroCopyInUse()

```
bool ModelSpec::zeroCopyInUse ( ) const  [protected]
```

Are any variables in any populations in this model using zero-copy memory?

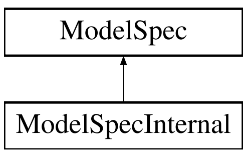
The documentation for this class was generated from the following files:

- modelSpec.h
- modelSpec.cc

19.40 ModelSpecInternal Class Reference

```
#include <modelSpecInternal.h>
```

Inheritance diagram for ModelSpecInternal:



Additional Inherited Members

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

- modelSpecInternal.h

19.41 CodeGenerator::NameIterCtx< Container > Struct Template Reference

```
#include <codeGenUtils.h>
```

Public Types

- typedef StructNameConstIter< typename Container::const_iterator > NameIter

Public Member Functions

- NameIterCtx (const Container &c)

Public Attributes

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

19.41.1 Member Typedef Documentation

19.41.1.1 NameIter

```
template<typename Container >
typedef StructNameConstIter<typename Container::const_iterator> CodeGenerator::NameIterCtx<
Container >::NameIter
```

19.41.2 Constructor & Destructor Documentation

19.41.2.1 NameIterCtx()

```
template<typename Container >
CodeGenerator::NameIterCtx< Container >::NameIterCtx (
            const Container & c )  [inline]
```

19.41.3 Member Data Documentation

19.41.3.1 container

```
template<typename Container >
const Container CodeGenerator::NameIterCtx< Container >::container
```

19.41.3.2 nameBegin

```
template<typename Container >
const NameIter CodeGenerator::NameIterCtx< Container >::nameBegin
```

19.41.3.3 nameEnd

```
template<typename Container >
const NameIter CodeGenerator::NameIterCtx< Container >::nameEnd
```

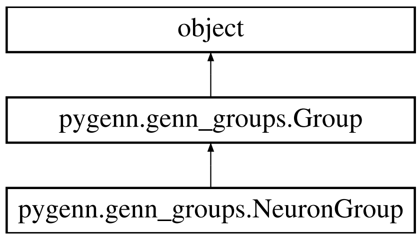
The documentation for this struct was generated from the following file:

- codeGenUtils.h

19.42 pygenn.genn_groups.NeuronGroup Class Reference

Class representing a group of neurons.

Inheritance diagram for pygenn.genn_groups.NeuronGroup:



Public Member Functions

- def __init__ (self, name)

  Init NeuronGroup.
- def current_spikes (self)

  Current spikes from GeNN.
- def delay_slots (self)

  Maximum delay steps needed for this group.
- def size (self)
- def set_neuron (self, model, param_space, var_space)

  Set neuron, its parameters and initial variables.
- def add_to (self, model_spec, num_neurons)

  Add this NeuronGroup to a model.
- def add_extra_global_param (self, param_name, param_values)

  Add extra global parameter.
- def load (self, slm, scalar)

  Loads neuron group.
- def reinitialise (self, slm, scalar)

  Reinitialise neuron group.

Public Attributes

- neuron
- spikes

- spike_count
- spike_que_ptr
- is_spike_source_array
- type
- pop

19.42.1 Detailed Description

Class representing a group of neurons.

19.42.2 Constructor & Destructor Documentation

19.42.2.1 __init__()

```
def pygenn.genn_groups.NeuronGroup.__init__ (
            self,
            name )
```

Init NeuronGroup.

Parameters

name	string name of the group

19.42.3 Member Function Documentation

19.42.3.1 add_extra_global_param()

```
def pygenn.genn_groups.NeuronGroup.add_extra_global_param (
            self,
            param_name,
            param_values )
```

Add extra global parameter.

Parameters

param_name	string with the name of the extra global parameter
param_values	iterable or a single value

19.42.3.2 add_to()

```
def pygenn.genn_groups.NeuronGroup.add_to (
            self,
            model_spec,
            num_neurons )
```

Add this NeuronGroup to a model.

Parameters

model_spec	`pygenn.genn_model.GeNNModel` to add to
num_neurons	int number of neurons

19.42.3.3 current_spikes()

```
def pygenn.genn_groups.NeuronGroup.current_spikes (
              self )
```

Current spikes from GeNN.

19.42.3.4 delay_slots()

```
def pygenn.genn_groups.NeuronGroup.delay_slots (
              self )
```

Maximum delay steps needed for this group.

19.42.3.5 load()

```
def pygenn.genn_groups.NeuronGroup.load (
              self,
              slm,
              scalar )
```

Loads neuron group.

Parameters

slm	SharedLibraryModel instance for acccessing variables
scalar	String specifying "scalar" type

19.42.3.6 reinitialise()

```
def pygenn.genn_groups.NeuronGroup.reinitialise (
              self,
              slm,
              scalar )
```

Reinitialise neuron group.

Parameters

slm	SharedLibraryModel instance for acccessing variables
scalar	String specifying "scalar" type

19.42.3.7 set_neuron()

```
def pygenn.genn_groups.NeuronGroup.set_neuron (
```

```
                self,
                model,
                param_space,
                var_space )
```

Set neuron, its parameters and initial variables.

Parameters

model	type as string of intance of the model
param_space	dict with model parameters
var_space	dict with model variables

19.42.3.8 size()

```
def pygenn.genn_groups.NeuronGroup.size (
                self )
```

19.42.4 Member Data Documentation

19.42.4.1 is_spike_source_array

```
pygenn.genn_groups.NeuronGroup.is_spike_source_array
```

19.42.4.2 neuron

```
pygenn.genn_groups.NeuronGroup.neuron
```

19.42.4.3 pop

```
pygenn.genn_groups.NeuronGroup.pop
```

19.42.4.4 spike_count

```
pygenn.genn_groups.NeuronGroup.spike_count
```

19.42.4.5 spike_que_ptr

```
pygenn.genn_groups.NeuronGroup.spike_que_ptr
```

19.42.4.6 spikes

```
pygenn.genn_groups.NeuronGroup.spikes
```

19.42.4.7 type

`pygenn.genn_groups.NeuronGroup.type`

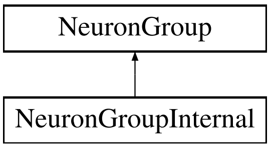
The documentation for this class was generated from the following file:

- genn_groups.py

19.43 NeuronGroup Class Reference

```
#include <neuronGroup.h>
```

Inheritance diagram for NeuronGroup:



Public Member Functions

- NeuronGroup (const NeuronGroup &)=delete
- NeuronGroup ()=delete
- void setSpikeLocation (VarLocation loc)

    Set location of this neuron group's output spikes.
- void setSpikeEventLocation (VarLocation loc)

    Set location of this neuron group's output spike events.
- void setSpikeTimeLocation (VarLocation loc)

    Set location of this neuron group's output spike times.
- void setVarLocation (const std::string &varName, VarLocation loc)

    Set variable location of neuron model state variable.
- void setExtraGlobalParamLocation (const std::string &paramName, VarLocation loc)

    Set location of neuron model extra global parameter.
- const std::string & getName () const
- unsigned int getNumNeurons () const

    Gets number of neurons in group.
- const NeuronModels::Base * getNeuronModel () const

    Gets the neuron model used by this group.
- const std::vector< double > & getParams () const
- const std::vector< Models::VarInit > & getVarInitialisers () const
- int getClusterHostID () const
- bool isSpikeTimeRequired () const
- bool isTrueSpikeRequired () const
- bool isSpikeEventRequired () const
- unsigned int getNumDelaySlots () const
- bool isDelayRequired () const
- bool isZeroCopyEnabled () const
- VarLocation getSpikeLocation () const

    Get location of this neuron group's output spikes.
- VarLocation getSpikeEventLocation () const

    Get location of this neuron group's output spike events.
- VarLocation getSpikeTimeLocation () const

Get location of this neuron group's output spike times.
- VarLocation getVarLocation (const std::string &varName) const

  Get location of neuron model state variable by name.
- VarLocation getVarLocation (size_t index) const

  Get location of neuron model state variable by index.
- VarLocation getExtraGlobalParamLocation (const std::string &paramName) const

  Get location of neuron model extra global parameter by name.
- VarLocation getExtraGlobalParamLocation (size_t index) const

  Get location of neuron model extra global parameter by omdex.
- bool isSimRNGRequired () const

  Does this neuron group require an RNG to simulate?
- bool isInitRNGRequired () const

  Does this neuron group require an RNG for it's init code?
- bool hasOutputToHost (int targetHostID) const

  Does this neuron group have outgoing connections specified host id?

Protected Member Functions

- NeuronGroup (const std::string &name, int numNeurons, const NeuronModels::Base ∗neuronModel, const std::vector< double > &params, const std::vector< Models::VarInit > &varInitialisers, VarLocation default↵ VarLocation, VarLocation defaultExtraGlobalParamLocation, int hostID)
- void checkNumDelaySlots (unsigned int requiredDelay)

  Checks delay slots currently provided by the neuron group against a required delay and extends if required.
- void updatePreVarQueues (const std::string &code)

  Update which presynaptic variables require queues based on piece of code.
- void updatePostVarQueues (const std::string &code)

  Update which postsynaptic variables require queues based on piece of code.
- void addSpkEventCondition (const std::string &code, const std::string &supportCodeNamespace)
- void addInSyn (SynapseGroupInternal ∗synapseGroup)
- void addOutSyn (SynapseGroupInternal ∗synapseGroup)
- void initDerivedParams (double dt)
- void mergeIncomingPSM (bool merge)

  Merge incoming postsynaptic models.
- void injectCurrent (CurrentSourceInternal ∗source)

  add input current source
- const std::vector< SynapseGroupInternal ∗ > & getInSyn () const

  Gets pointers to all synapse groups which provide input to this neuron group.
- const std::vector< std::pair< SynapseGroupInternal ∗, std::vector< SynapseGroupInternal ∗ > > > & get↵ MergedInSyn () const
- const std::vector< SynapseGroupInternal ∗ > & getOutSyn () const

  Gets pointers to all synapse groups emanating from this neuron group.
- const std::vector< CurrentSourceInternal ∗ > & getCurrentSources () const

  Gets pointers to all current sources which provide input to this neuron group.
- const std::vector< double > & getDerivedParams () const
- const std::set< std::pair< std::string, std::string > > & getSpikeEventCondition () const
- bool isParamRequiredBySpikeEventCondition (const std::string &pnamefull) const

  Do any of the spike event conditions tested by this neuron require specified parameter?
- std::string getCurrentQueueOffset (const std::string &devPrefix) const

  Get the expression to calculate the queue offset for accessing state of variables this timestep.
- std::string getPrevQueueOffset (const std::string &devPrefix) const

  Get the expression to calculate the queue offset for accessing state of variables in previous timestep.
- bool isVarQueueRequired (const std::string &var) const
- bool isVarQueueRequired (size_t index) const

19.43.1 Constructor & Destructor Documentation

19.43.1.1 NeuronGroup() [1/3]

```
NeuronGroup::NeuronGroup (
            const NeuronGroup & )  [delete]
```

19.43.1.2 NeuronGroup() [2/3]

```
NeuronGroup::NeuronGroup ( )  [delete]
```

19.43.1.3 NeuronGroup() [3/3]

```
NeuronGroup::NeuronGroup (
            const std::string & name,
            int numNeurons,
            const NeuronModels::Base * neuronModel,
            const std::vector< double > & params,
            const std::vector< Models::VarInit > & varInitialisers,
            VarLocation defaultVarLocation,
            VarLocation defaultExtraGlobalParamLocation,
            int hostID )  [inline], [protected]
```

19.43.2 Member Function Documentation

19.43.2.1 addInSyn()

```
void NeuronGroup::addInSyn (
            SynapseGroupInternal * synapseGroup )  [inline], [protected]
```

19.43.2.2 addOutSyn()

```
void NeuronGroup::addOutSyn (
            SynapseGroupInternal * synapseGroup )  [inline], [protected]
```

19.43.2.3 addSpkEventCondition()

```
void NeuronGroup::addSpkEventCondition (
            const std::string & code,
            const std::string & supportCodeNamespace )  [protected]
```

19.43.2.4 checkNumDelaySlots()

```
void NeuronGroup::checkNumDelaySlots (
            unsigned int requiredDelay )  [protected]
```

Checks delay slots currently provided by the neuron group against a required delay and extends if required.

19.43.2.5 getClusterHostID()

`int NeuronGroup::getClusterHostID ( ) const [inline]`

19.43.2.6 getCurrentQueueOffset()

```
std::string NeuronGroup::getCurrentQueueOffset (
            const std::string & devPrefix ) const [protected]
```

Get the expression to calculate the queue offset for accessing state of variables this timestep.

19.43.2.7 getCurrentSources()

`const std::vector<CurrentSourceInternal*>& NeuronGroup::getCurrentSources ( ) const [inline], [protected]`

Gets pointers to all current sources which provide input to this neuron group.

19.43.2.8 getDerivedParams()

`const std::vector<double>& NeuronGroup::getDerivedParams ( ) const [inline], [protected]`

19.43.2.9 getExtraGlobalParamLocation() [1/2]

```
VarLocation NeuronGroup::getExtraGlobalParamLocation (
            const std::string & paramName ) const
```

Get location of neuron model extra global parameter by name.

This is only used by extra global parameters which are pointers

19.43.2.10 getExtraGlobalParamLocation() [2/2]

```
VarLocation NeuronGroup::getExtraGlobalParamLocation (
            size_t index ) const [inline]
```

Get location of neuron model extra global parameter by omdex.

This is only used by extra global parameters which are pointers

19.43.2.11 getInSyn()

`const std::vector<SynapseGroupInternal*>& NeuronGroup::getInSyn ( ) const [inline], [protected]`

Gets pointers to all synapse groups which provide input to this neuron group.

19.43.2.12 getMergedInSyn()

`const std::vector<std::pair<SynapseGroupInternal*, std::vector<SynapseGroupInternal*> > >& NeuronGroup::getMergedInSyn ( ) const [inline], [protected]`

19.43.2.13 getName()

`const std::string& NeuronGroup::getName ( ) const [inline]`

19.43.2.14 getNeuronModel()

```
const NeuronModels::Base* NeuronGroup::getNeuronModel ( ) const  [inline]
```

Gets the neuron model used by this group.

19.43.2.15 getNumDelaySlots()

```
unsigned int NeuronGroup::getNumDelaySlots ( ) const  [inline]
```

19.43.2.16 getNumNeurons()

```
unsigned int NeuronGroup::getNumNeurons ( ) const  [inline]
```

Gets number of neurons in group.

19.43.2.17 getOutSyn()

```
const std::vector<SynapseGroupInternal*>& NeuronGroup::getOutSyn ( ) const  [inline], [protected]
```

Gets pointers to all synapse groups emanating from this neuron group.

19.43.2.18 getParams()

```
const std::vector<double>& NeuronGroup::getParams ( ) const  [inline]
```

19.43.2.19 getPrevQueueOffset()

```
std::string NeuronGroup::getPrevQueueOffset (
            const std::string & devPrefix ) const  [protected]
```

Get the expression to calculate the queue offset for accessing state of variables in previous timestep.

19.43.2.20 getSpikeEventCondition()

```
const std::set<std::pair<std::string, std::string> >& NeuronGroup::getSpikeEventCondition ( )
const  [inline], [protected]
```

19.43.2.21 getSpikeEventLocation()

```
VarLocation NeuronGroup::getSpikeEventLocation ( ) const  [inline]
```

Get location of this neuron group's output spike events.

19.43.2.22 getSpikeLocation()

```
VarLocation NeuronGroup::getSpikeLocation ( ) const  [inline]
```

Get location of this neuron group's output spikes.

19.43.2.23 getSpikeTimeLocation()

```
VarLocation NeuronGroup::getSpikeTimeLocation ( ) const [inline]
```

Get location of this neuron group's output spike times.

19.43.2.24 getVarInitialisers()

```
const std::vector<Models::VarInit>& NeuronGroup::getVarInitialisers ( ) const [inline]
```

19.43.2.25 getVarLocation() [1/2]

```
VarLocation NeuronGroup::getVarLocation (
            const std::string & varName ) const
```

Get location of neuron model state variable by name.

19.43.2.26 getVarLocation() [2/2]

```
VarLocation NeuronGroup::getVarLocation (
            size_t index ) const [inline]
```

Get location of neuron model state variable by index.

19.43.2.27 hasOutputToHost()

```
bool NeuronGroup::hasOutputToHost (
            int targetHostID ) const
```

Does this neuron group have outgoing connections specified host id?

19.43.2.28 initDerivedParams()

```
void NeuronGroup::initDerivedParams (
            double dt ) [protected]
```

19.43.2.29 injectCurrent()

```
void NeuronGroup::injectCurrent (
            CurrentSourceInternal * source ) [protected]
```

add input current source

19.43.2.30 isDelayRequired()

```
bool NeuronGroup::isDelayRequired ( ) const [inline]
```

19.43.2.31 isInitRNGRequired()

```
bool NeuronGroup::isInitRNGRequired ( ) const
```

Does this neuron group require an RNG for it's init code?

19.43.2.32 isParamRequiredBySpikeEventCondition()

```
bool NeuronGroup::isParamRequiredBySpikeEventCondition (
            const std::string & pnamefull ) const  [protected]
```

Do any of the spike event conditions tested by this neuron require specified parameter?

19.43.2.33 isSimRNGRequired()

```
bool NeuronGroup::isSimRNGRequired ( ) const
```

Does this neuron group require an RNG to simulate?

19.43.2.34 isSpikeEventRequired()

```
bool NeuronGroup::isSpikeEventRequired ( ) const
```

19.43.2.35 isSpikeTimeRequired()

```
bool NeuronGroup::isSpikeTimeRequired ( ) const
```

19.43.2.36 isTrueSpikeRequired()

```
bool NeuronGroup::isTrueSpikeRequired ( ) const
```

19.43.2.37 isVarQueueRequired() [1/2]

```
bool NeuronGroup::isVarQueueRequired (
            const std::string & var ) const  [protected]
```

19.43.2.38 isVarQueueRequired() [2/2]

```
bool NeuronGroup::isVarQueueRequired (
            size_t index ) const  [inline], [protected]
```

19.43.2.39 isZeroCopyEnabled()

```
bool NeuronGroup::isZeroCopyEnabled ( ) const
```

19.43.2.40 mergeIncomingPSM()

```
void NeuronGroup::mergeIncomingPSM (
            bool merge )  [protected]
```

Merge incoming postsynaptic models.

19.43.2.41 setExtraGlobalParamLocation()

```
void NeuronGroup::setExtraGlobalParamLocation (
            const std::string & paramName,
            VarLocation loc )
```

Set location of neuron model extra global parameter.

This is ignored for simulations on hardware with a single memory space and only applies to extra global parameters which are pointers.

19.43.2.42 setSpikeEventLocation()

```
void NeuronGroup::setSpikeEventLocation (
            VarLocation loc )  [inline]
```

Set location of this neuron group's output spike events.

This is ignored for simulations on hardware with a single memory space

19.43.2.43 setSpikeLocation()

```
void NeuronGroup::setSpikeLocation (
            VarLocation loc )  [inline]
```

Set location of this neuron group's output spikes.

This is ignored for simulations on hardware with a single memory space

19.43.2.44 setSpikeTimeLocation()

```
void NeuronGroup::setSpikeTimeLocation (
            VarLocation loc )  [inline]
```

Set location of this neuron group's output spike times.

This is ignored for simulations on hardware with a single memory space

19.43.2.45 setVarLocation()

```
void NeuronGroup::setVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set variable location of neuron model state variable.

This is ignored for simulations on hardware with a single memory space

19.43.2.46 updatePostVarQueues()

```
void NeuronGroup::updatePostVarQueues (
            const std::string & code )  [protected]
```

Update which postsynaptic variables require queues based on piece of code.

19.43.2.47 updatePreVarQueues()

```
void NeuronGroup::updatePreVarQueues (
            const std::string & code )  [protected]
```

Update which presynaptic variables require queues based on piece of code.

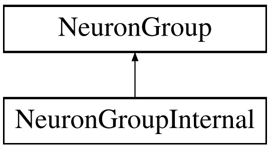
The documentation for this class was generated from the following files:

- neuronGroup.h
- neuronGroup.cc

19.44 NeuronGroupInternal Class Reference

```
#include <neuronGroupInternal.h>
```

Inheritance diagram for NeuronGroupInternal:



Public Member Functions

- NeuronGroupInternal (const std::string &name, int numNeurons, const NeuronModels::Base *neuronModel, const std::vector< double > &params, const std::vector< Models::VarInit > &varInitialisers, VarLocation defaultVarLocation, VarLocation defaultExtraGlobalParamLocation, int hostID)

Additional Inherited Members

19.44.1 Constructor & Destructor Documentation

19.44.1.1 NeuronGroupInternal()

```
NeuronGroupInternal::NeuronGroupInternal (
            const std::string & name,
            int numNeurons,
            const NeuronModels::Base * neuronModel,
            const std::vector< double > & params,
            const std::vector< Models::VarInit > & varInitialisers,
            VarLocation defaultVarLocation,
            VarLocation defaultExtraGlobalParamLocation,
            int hostID )  [inline]
```

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

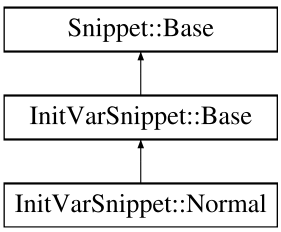
- neuronGroupInternal.h

19.45 InitVarSnippet::Normal Class Reference

Initialises variable by sampling from the normal distribution.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Normal:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Normal, 2)
- SET_CODE ("$(value) = $(mean) + ($(gennrand_normal) * $(sd));")
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Additional Inherited Members

19.45.1 Detailed Description

Initialises variable by sampling from the normal distribution.

This snippet takes 2 parameters:

- `mean` - The mean
- `sd` - The standard distribution

19.45.2 Member Function Documentation

19.45.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Normal::DECLARE_SNIPPET (
            InitVarSnippet::Normal ,
            2 )
```

19.45.2.2 getParamNames()

```
virtual StringVec InitVarSnippet::Normal::getParamNames ( ) const [inline], [override], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.45.2.3 SET_CODE()

```
InitVarSnippet::Normal::SET_CODE ( )
```

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

- initVarSnippet.h

19.46 CodeGenerator::CodeStream::OB Struct Reference

An open bracket marker.

`#include <codeStream.h>`

Public Member Functions

- OB (unsigned int level)

Public Attributes

- const unsigned int Level

19.46.1 Detailed Description

An open bracket marker.

Write to code stream `os` using:

```
os << OB(16);
```

19.46.2 Constructor & Destructor Documentation

19.46.2.1 OB()

```
CodeGenerator::CodeStream::OB::OB (
            unsigned int level )  [inline]
```

19.46.3 Member Data Documentation

19.46.3.1 Level

```
const unsigned int CodeGenerator::CodeStream::OB::Level
```

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

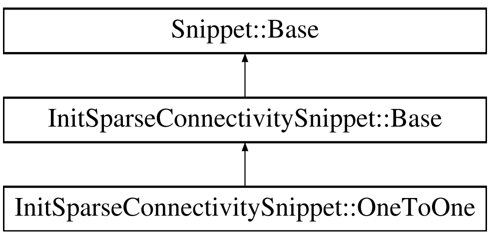
- codeStream.h

19.47 InitSparseConnectivitySnippet::OneToOne Class Reference

Initialises connectivity to a 'one-to-one' diagonal matrix.

`#include <initSparseConnectivitySnippet.h>`

Inheritance diagram for InitSparseConnectivitySnippet::OneToOne:



Public Member Functions

- DECLARE_SNIPPET (InitSparseConnectivitySnippet::OneToOne, 0)
- SET_ROW_BUILD_CODE ("$(addSynapse, $(id_pre));\ "$(endRow);\")
- SET_MAX_ROW_LENGTH (1)
- SET_MAX_COL_LENGTH (1)

Additional Inherited Members

19.47.1 Detailed Description

Initialises connectivity to a 'one-to-one' diagonal matrix.

19.47.2 Member Function Documentation

19.47.2.1 DECLARE_SNIPPET()

```
InitSparseConnectivitySnippet::OneToOne::DECLARE_SNIPPET (
            InitSparseConnectivitySnippet::OneToOne ,
            0 )
```

19.47.2.2 SET_MAX_COL_LENGTH()

```
InitSparseConnectivitySnippet::OneToOne::SET_MAX_COL_LENGTH (
            1 )
```

19.47.2.3 SET_MAX_ROW_LENGTH()

```
InitSparseConnectivitySnippet::OneToOne::SET_MAX_ROW_LENGTH (
            1 )
```

19.47.2.4 SET_ROW_BUILD_CODE()

```
InitSparseConnectivitySnippet::OneToOne::SET_ROW_BUILD_CODE (
            "$(addSynapse, $(id_pre));\$(endRow);\ )
```

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

- initSparseConnectivitySnippet.h

19.48 Snippet::Base::ParamVal Struct Reference

```
#include <snippet.h>
```

Public Attributes

- std::string name
- std::string type
- double value

19.48.1 Member Data Documentation

19.48.1.1 name

```
std::string Snippet::Base::ParamVal::name
```

19.48.1.2 type

`std::string Snippet::Base::ParamVal::type`

19.48.1.3 value

`double Snippet::Base::ParamVal::value`

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

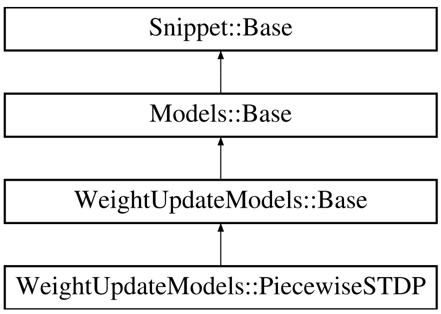
- snippet.h

19.49 WeightUpdateModels::PiecewiseSTDP Class Reference

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

`#include <weightUpdateModels.h>`

Inheritance diagram for WeightUpdateModels::PiecewiseSTDP:



Public Member Functions

- DECLARE_WEIGHT_UPDATE_MODEL (PiecewiseSTDP, 10, 2, 0, 0)
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

  Gets names and types (as strings) of model variables.
- virtual std::string getSimCode () const override

  Gets simulation code run when 'true' spikes are received.
- virtual std::string getLearnPostCode () const override

  Gets code to include in the learnSynapsesPost kernel/function.
- virtual DerivedParamVec getDerivedParams () const override
- virtual bool isPreSpikeTimeRequired () const override

  Whether presynaptic spike times are needed or not.
- virtual bool isPostSpikeTimeRequired () const override

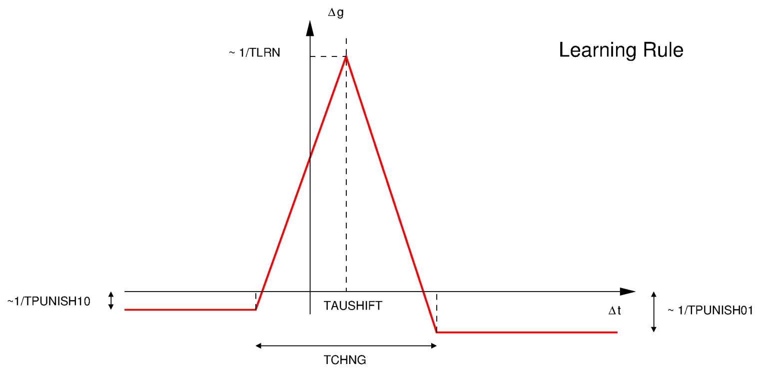
  Whether postsynaptic spike times are needed or not.

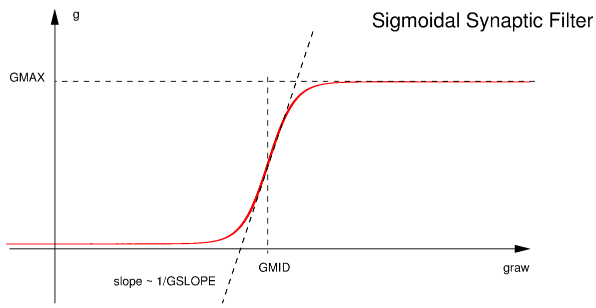
Additional Inherited Members

19.49.1 Detailed Description

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

The STDP window is defined as a piecewise function:





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

Note

The STDP curve implies that unpaired pre- and post-synaptic spikes incur a negative increment in `gRaw` (and hence in `g`).
The time of the last spike in each neuron, "sTXX", where XX is the name of a neuron population is (somewhat arbitrarily) initialised to -10.0 ms. If neurons never spike, these spike times are used.
It is the raw synaptic conductance `gRaw` that is subject to the STDP rule. The resulting synaptic conductance is a sigmoid filter of `gRaw`. This implies that `g` is initialised but not `gRaw`, the synapse will revert to the value that corresponds to `gRaw`.

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

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

Note

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

The model has 2 variables:

- `g:` conductance of `scalar` type

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

Parameters are (compare to the figure above):

- `tLrn`: Time scale of learning changes
- `tChng`: Width of learning window
- `tDecay`: Time scale of synaptic strength decay
- `tPunish10`: Time window of suppression in response to 1/0
- `tPunish01`: Time window of suppression in response to 0/1
- `gMax`: Maximal conductance achievable
- `gMid`: Midpoint of sigmoid g filter curve
- `gSlope`: Slope of sigmoid g filter curve
- `tauShift`: Shift of learning curve
- `gSyn0`: Value of syn conductance g decays to

19.49.2 Member Function Documentation

19.49.2.1 DECLARE_WEIGHT_UPDATE_MODEL()

```
WeightUpdateModels::PiecewiseSTDP::DECLARE_WEIGHT_UPDATE_MODEL (
            PiecewiseSTDP ,
            10 ,
            2 ,
            0 ,
            0  )
```

19.49.2.2 getDerivedParams()

```
virtual DerivedParamVec WeightUpdateModels::PiecewiseSTDP::getDerivedParams ( ) const  [inline],
[override], [virtual]
```

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

Reimplemented from Snippet::Base.

19.49.2.3 getLearnPostCode()

```
virtual std::string WeightUpdateModels::PiecewiseSTDP::getLearnPostCode ( ) const  [inline],
[override], [virtual]
```

Gets code to include in the learnSynapsesPost kernel/function.

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

Reimplemented from WeightUpdateModels::Base.

19.49.2.4 getParamNames()

```
virtual StringVec WeightUpdateModels::PiecewiseSTDP::getParamNames ( ) const  [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.49.2.5 getSimCode()

```
virtual std::string WeightUpdateModels::PiecewiseSTDP::getSimCode ( ) const  [inline], [override],
[virtual]
```

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

Reimplemented from WeightUpdateModels::Base.

19.49.2.6 getVars()

```
virtual VarVec WeightUpdateModels::PiecewiseSTDP::getVars ( ) const  [inline], [override],
[virtual]
```

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

Reimplemented from Models::Base.

19.49.2.7 isPostSpikeTimeRequired()

```
virtual bool WeightUpdateModels::PiecewiseSTDP::isPostSpikeTimeRequired ( ) const  [inline],
[override], [virtual]
```

Whether postsynaptic spike times are needed or not.

Reimplemented from WeightUpdateModels::Base.

19.49.2.8 isPreSpikeTimeRequired()

```
virtual bool WeightUpdateModels::PiecewiseSTDP::isPreSpikeTimeRequired ( ) const  [inline],
[override], [virtual]
```

Whether presynaptic spike times are needed or not.

Reimplemented from WeightUpdateModels::Base.

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

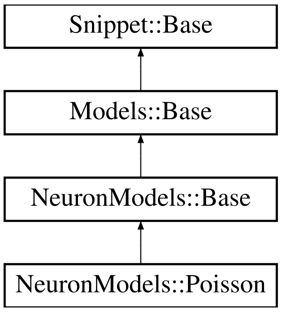
- weightUpdateModels.h

19.50 NeuronModels::Poisson Class Reference

Poisson neurons.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::Poisson:



Public Types

- typedef Snippet::ValueBase< 4 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

  Gets names and types (as strings) of model variables.
- virtual VarVec getExtraGlobalParams () const override

Static Public Member Functions

- static const NeuronModels::Poisson ∗ getInstance ()

Additional Inherited Members

19.50.1 Detailed Description

Poisson neurons.

Poisson neurons have constant membrane potential (`Vrest`) unless they are activated randomly to the `Vspike` value if (t- `SpikeTime`) > `trefract`.

It has 2 variables:

- `V` - Membrane potential (mV)
- `SpikeTime` - Time at which the neuron spiked for the last time (ms)

and 4 parameters:

- `trefract` - Refractory period (ms)
- `tspike` - duration of spike (ms)

- `Vspike` - Membrane potential at spike (mV)
- `Vrest` - Membrane potential at rest (mV)

Note

The initial values array for the `Poisson` type needs two entries for `V`, and `SpikeTime` and the parameter array needs four entries for `therate`, `trefract`, `Vspike` and `Vrest`, *in that order*.
This model 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 λ 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.

19.50.2 Member Typedef Documentation

19.50.2.1 ParamValues

```
typedef Snippet::ValueBase< 4 > NeuronModels::Poisson::ParamValues
```

19.50.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::Poisson::PostVarValues
```

19.50.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::Poisson::PreVarValues
```

19.50.2.4 VarValues

```
typedef Models::VarInitContainerBase< 2 > NeuronModels::Poisson::VarValues
```

19.50.3 Member Function Documentation

19.50.3.1 getExtraGlobalParams()

```
virtual VarVec NeuronModels::Poisson::getExtraGlobalParams ( ) const [inline], [override],
[virtual]
```

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

Reimplemented from Models::Base.

19.50.3.2 getInstance()

```
static const NeuronModels::Poisson* NeuronModels::Poisson::getInstance ( ) [inline], [static]
```

19.50.3.3 getParamNames()

```
virtual StringVec NeuronModels::Poisson::getParamNames ( ) const [inline], [override], [virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.50.3.4 getSimCode()

`virtual std::string NeuronModels::Poisson::getSimCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.50.3.5 getThresholdConditionCode()

`virtual std::string NeuronModels::Poisson::getThresholdConditionCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.50.3.6 getVars()

`virtual VarVec NeuronModels::Poisson::getVars ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

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

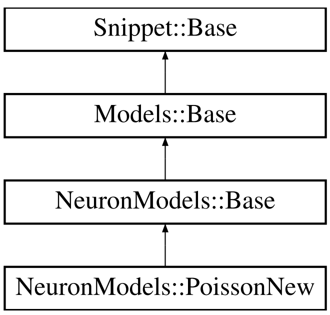
- neuronModels.h

19.51 NeuronModels::PoissonNew Class Reference

Poisson neurons.

`#include <neuronModels.h>`

Inheritance diagram for NeuronModels::PoissonNew:



Public Types

- typedef Snippet::ValueBase< 1 > ParamValues
- typedef Models::VarInitContainerBase< 1 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

  Gets names and types (as strings) of model variables.
- virtual DerivedParamVec getDerivedParams () const override
- SET_NEEDS_AUTO_REFRACTORY (false)

Static Public Member Functions

- static const NeuronModels::PoissonNew * getInstance ()

Additional Inherited Members

19.51.1 Detailed Description

Poisson neurons.

It has 1 state variable:

- `timeStepToSpike` - Number of timesteps to next spike

and 1 parameter:

- `rate` - Mean firing rate (Hz)

Note

Internally this samples from the exponential distribution using the C++ 11 <random> library on the CPU and by transforming the uniform distribution, generated using cuRAND, with a natural log on the GPU.

19.51.2 Member Typedef Documentation

19.51.2.1 ParamValues

```
typedef Snippet::ValueBase< 1 > NeuronModels::PoissonNew::ParamValues
```

19.51.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::PoissonNew::PostVarValues
```

19.51.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::PoissonNew::PreVarValues
```

19.51.2.4 VarValues

```
typedef Models::VarInitContainerBase< 1 > NeuronModels::PoissonNew::VarValues
```

19.51.3 Member Function Documentation

19.51.3.1 getDerivedParams()

```
virtual DerivedParamVec NeuronModels::PoissonNew::getDerivedParams ( ) const [inline], [override],
[virtual]
```

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

Reimplemented from Snippet::Base.

19.51.3.2 getInstance()

```
static const NeuronModels::PoissonNew* NeuronModels::PoissonNew::getInstance ( ) [inline],
[static]
```

19.51.3.3 getParamNames()

```
virtual StringVec NeuronModels::PoissonNew::getParamNames ( ) const [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.51.3.4 getSimCode()

```
virtual std::string NeuronModels::PoissonNew::getSimCode ( ) const [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.51.3.5 getThresholdConditionCode()

```
virtual std::string NeuronModels::PoissonNew::getThresholdConditionCode ( ) const [inline],
[override], [virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.51.3.6 getVars()

```
virtual VarVec NeuronModels::PoissonNew::getVars ( ) const [inline], [override], [virtual]
```

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

Reimplemented from Models::Base.

19.51.3.7 SET_NEEDS_AUTO_REFRACTORY()

```
NeuronModels::PoissonNew::SET_NEEDS_AUTO_REFRACTORY (
            false )
```

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

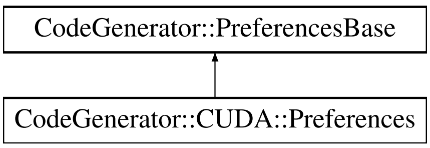
- neuronModels.h

19.52 CodeGenerator::CUDA::Preferences Struct Reference

Preferences for CUDA backend.

```
#include <backend.h>
```

Inheritance diagram for CodeGenerator::CUDA::Preferences:



Public Member Functions

- Preferences ()

Public Attributes

- bool showPtxInfo = false

  Should PTX assembler information be displayed for each CUDA kernel during compilation.
- DeviceSelect deviceSelectMethod = DeviceSelect::OPTIMAL

  How to select GPU device.
- unsigned int manualDeviceID = 0

  If device select method is set to DeviceSelect::MANUAL, id of device to use.
- BlockSizeSelect blockSizeSelectMethod = BlockSizeSelect::OCCUPANCY

  How to select CUDA blocksize.

- KernelBlockSize manualBlockSizes

    If block size select method is set to BlockSizeSelect::MANUAL, block size to use for each kernel.
- std::string userNvccFlags = ""

    NVCC compiler options for all GPU code.

19.52.1 Detailed Description

Preferences for CUDA backend.

19.52.2 Constructor & Destructor Documentation

19.52.2.1 Preferences()

`CodeGenerator::CUDA::Preferences::Preferences ( )  [inline]`

19.52.3 Member Data Documentation

19.52.3.1 blockSizeSelectMethod

`BlockSizeSelect CodeGenerator::CUDA::Preferences::blockSizeSelectMethod = BlockSizeSelect::OC←CUPANCY`

How to select CUDA blocksize.

19.52.3.2 deviceSelectMethod

`DeviceSelect CodeGenerator::CUDA::Preferences::deviceSelectMethod = DeviceSelect::OPTIMAL`

How to select GPU device.

19.52.3.3 manualBlockSizes

`KernelBlockSize CodeGenerator::CUDA::Preferences::manualBlockSizes`

If block size select method is set to BlockSizeSelect::MANUAL, block size to use for each kernel.

19.52.3.4 manualDeviceID

`unsigned int CodeGenerator::CUDA::Preferences::manualDeviceID = 0`

If device select method is set to DeviceSelect::MANUAL, id of device to use.

19.52.3.5 showPtxInfo

`bool CodeGenerator::CUDA::Preferences::showPtxInfo = false`

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

19.52.3.6 userNvccFlags

```
std::string CodeGenerator::CUDA::Preferences::userNvccFlags = ""
```

NVCC compiler options for all GPU code.

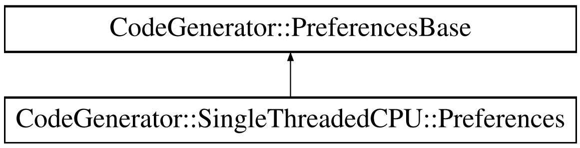
The documentation for this struct was generated from the following file:

- cuda/backend.h

19.53 CodeGenerator::SingleThreadedCPU::Preferences Struct Reference

```
#include <backend.h>
```

Inheritance diagram for CodeGenerator::SingleThreadedCPU::Preferences:



Additional Inherited Members

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

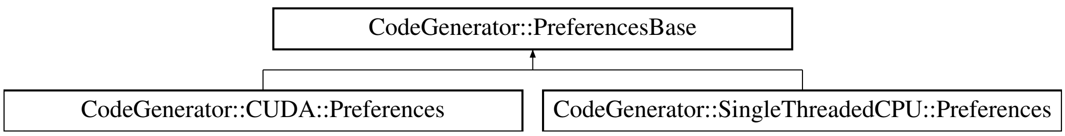
- single_threaded_cpu/backend.h

19.54 CodeGenerator::PreferencesBase Struct Reference

Base class for backend preferences - can be accessed via a global in 'classic' C++ code generator.

```
#include <backendBase.h>
```

Inheritance diagram for CodeGenerator::PreferencesBase:



Public Attributes

- bool optimizeCode = false

  Generate speed-optimized code, potentially at the expense of floating-point accuracy.
- bool debugCode = false

  Generate code with debug symbols.
- std::string userCxxFlagsGNU = ""

  C++ compiler options to be used for building all host side code (used for unix based platforms)
- std::string userNvccFlagsGNU = ""

  NVCC compiler options they may want to use for all GPU code (used for unix based platforms)
- plog::Severity logLevel = plog::info

  Logging level to use for code generation.

19.54.1 Detailed Description

Base class for backend preferences - can be accessed via a global in 'classic' C++ code generator.

19.54.2 Member Data Documentation

19.54.2.1 debugCode

```
bool CodeGenerator::PreferencesBase::debugCode = false
```

Generate code with debug symbols.

19.54.2.2 logLevel

```
plog::Severity CodeGenerator::PreferencesBase::logLevel = plog::info
```

Logging level to use for code generation.

19.54.2.3 optimizeCode

```
bool CodeGenerator::PreferencesBase::optimizeCode = false
```

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

19.54.2.4 userCxxFlagsGNU

```
std::string CodeGenerator::PreferencesBase::userCxxFlagsGNU = ""
```

C++ compiler options to be used for building all host side code (used for unix based platforms)

19.54.2.5 userNvccFlagsGNU

```
std::string CodeGenerator::PreferencesBase::userNvccFlagsGNU = ""
```

NVCC compiler options they may want to use for all GPU code (used for unix based platforms)

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

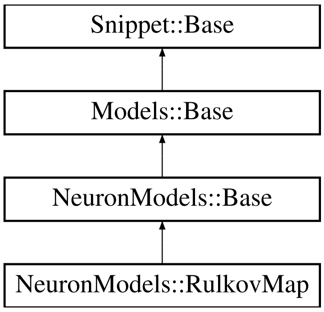
- backendBase.h

19.55 NeuronModels::RulkovMap Class Reference

Rulkov Map neuron.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::RulkovMap:



Public Types

- typedef Snippet::ValueBase< 4 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

    Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

    Gets code which defines the condition for a true spike in the described neuron model.
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

    Gets names and types (as strings) of model variables.
- virtual DerivedParamVec getDerivedParams () const override

Static Public Member Functions

- static const NeuronModels::RulkovMap * getInstance ()

Additional Inherited Members

19.55.1 Detailed Description

Rulkov Map neuron.

The RulkovMap type is a map based neuron model based on [5] but in the 1-dimensional map form used in [4] :

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

Note

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

The `RulkovMap` type has 2 variables:

- `V` - the membrane potential

- preV - the membrane potential at the previous time step

and it has 4 parameters:

- Vspike - determines the amplitude of spikes, typically -60mV
- alpha - determines the shape of the iteration function, typically α= 3
- y - "shift / excitation" parameter, also determines the iteration function,originally, y= -2.468
- beta - roughly speaking equivalent to the input resistance, i.e. it regulates the scale of the input into the neuron, typically β= 2.64 $\mathrm{M}\Omega$.

Note

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

19.55.2 Member Typedef Documentation

19.55.2.1 ParamValues

```
typedef Snippet::ValueBase< 4 > NeuronModels::RulkovMap::ParamValues
```

19.55.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::RulkovMap::PostVarValues
```

19.55.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::RulkovMap::PreVarValues
```

19.55.2.4 VarValues

```
typedef Models::VarInitContainerBase< 2 > NeuronModels::RulkovMap::VarValues
```

19.55.3 Member Function Documentation

19.55.3.1 getDerivedParams()

```
virtual DerivedParamVec NeuronModels::RulkovMap::getDerivedParams ( ) const [inline], [override],
[virtual]
```

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

Reimplemented from Snippet::Base.

19.55.3.2 getInstance()

`static const NeuronModels::RulkovMap* NeuronModels::RulkovMap::getInstance ( ) [inline], [static]`

19.55.3.3 getParamNames()

`virtual StringVec NeuronModels::RulkovMap::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.55.3.4 getSimCode()

`virtual std::string NeuronModels::RulkovMap::getSimCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.55.3.5 getThresholdConditionCode()

`virtual std::string NeuronModels::RulkovMap::getThresholdConditionCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::Base.

19.55.3.6 getVars()

`virtual VarVec NeuronModels::RulkovMap::getVars ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

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

- neuronModels.h

19.56 CodeGenerator::CodeStream::Scope Class Reference

`#include <codeStream.h>`

Public Member Functions

- Scope (CodeStream &codeStream)
- ∼Scope ()

19.56.1 Constructor & Destructor Documentation

19.56.1.1 Scope()

```
CodeGenerator::CodeStream::Scope::Scope (
            CodeStream & codeStream ) [inline]
```

19.56.1.2 ∼Scope()

```
CodeGenerator::CodeStream::Scope::∼Scope ( ) [inline]
```

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

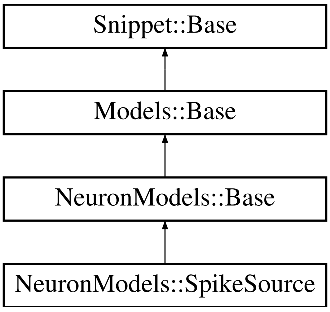
- codeStream.h
- codeStream.cc

19.57 NeuronModels::SpikeSource Class Reference

Empty neuron which allows setting spikes from external sources.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::SpikeSource:



Public Types

- typedef Snippet::ValueBase< 0 > ParamValues
- typedef Models::VarInitContainerBase< 0 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- SET_NEEDS_AUTO_REFRACTORY (false)

Static Public Member Functions

- static const NeuronModels::SpikeSource ∗ getInstance ()

Additional Inherited Members

19.57.1 Detailed Description

Empty neuron which allows setting spikes from external sources.

This model does not contain any update code and can be used to implement the equivalent of a SpikeGenerator↩ Group in Brian or a SpikeSourceArray in PyNN.

19.57.2 Member Typedef Documentation

19.57.2.1 ParamValues

```
typedef Snippet::ValueBase< 0 > NeuronModels::SpikeSource::ParamValues
```

19.57.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::SpikeSource::PostVarValues
```

19.57.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::SpikeSource::PreVarValues
```

19.57.2.4 VarValues

```
typedef Models::VarInitContainerBase< 0 > NeuronModels::SpikeSource::VarValues
```

19.57.3 Member Function Documentation

19.57.3.1 getInstance()

```
static const NeuronModels::SpikeSource* NeuronModels::SpikeSource::getInstance ( ) [inline],
[static]
```

19.57.3.2 getThresholdConditionCode()

```
virtual std::string NeuronModels::SpikeSource::getThresholdConditionCode ( ) const [inline],
[override], [virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.57.3.3 SET_NEEDS_AUTO_REFRACTORY()

```
NeuronModels::SpikeSource::SET_NEEDS_AUTO_REFRACTORY (
            false )
```

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

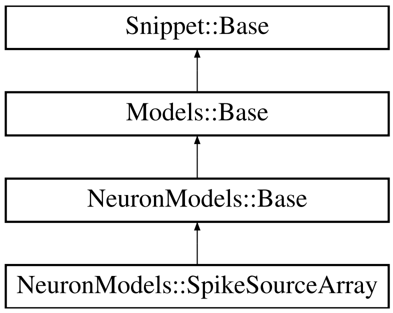
- neuronModels.h

19.58 NeuronModels::SpikeSourceArray Class Reference

Spike source array.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::SpikeSourceArray:



Public Types

- typedef Snippet::ValueBase< 0 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual std::string getResetCode () const override

  Gets code that defines the reset action taken after a spike occurred. This can be empty.
- virtual VarVec getVars () const override

  Gets names and types (as strings) of model variables.
- virtual VarVec getExtraGlobalParams () const override
- SET_NEEDS_AUTO_REFRACTORY (false)

Static Public Member Functions

- static const NeuronModels::SpikeSourceArray * getInstance ()

Additional Inherited Members

19.58.1 Detailed Description

Spike source array.

A neuron which reads spike times from a global spikes array It has 2 variables:

- `startSpike` - Index of the next spike in the global array
- `endSpike` - Index of the spike next to the last in the globel array

and 1 global parameter:

- `spikeTimes` - Array with all spike times

19.58.2 Member Typedef Documentation

19.58.2.1 ParamValues

`typedef Snippet::ValueBase< 0 > NeuronModels::SpikeSourceArray::ParamValues`

19.58.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::SpikeSourceArray::PostVarValues`

19.58.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::SpikeSourceArray::PreVarValues`

19.58.2.4 VarValues

`typedef Models::VarInitContainerBase< 2 > NeuronModels::SpikeSourceArray::VarValues`

19.58.3 Member Function Documentation

19.58.3.1 getExtraGlobalParams()

`virtual VarVec NeuronModels::SpikeSourceArray::getExtraGlobalParams ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

19.58.3.2 getInstance()

`static const NeuronModels::SpikeSourceArray* NeuronModels::SpikeSourceArray::getInstance ( ) [inline], [static]`

19.58.3.3 getResetCode()

`virtual std::string NeuronModels::SpikeSourceArray::getResetCode ( ) const [inline], [override], [virtual]`

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

Reimplemented from NeuronModels::Base.

19.58.3.4 getSimCode()

```
virtual std::string NeuronModels::SpikeSourceArray::getSimCode ( ) const  [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.58.3.5 getThresholdConditionCode()

```
virtual std::string NeuronModels::SpikeSourceArray::getThresholdConditionCode ( ) const  [inline],
[override], [virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.58.3.6 getVars()

```
virtual VarVec NeuronModels::SpikeSourceArray::getVars ( ) const  [inline], [override], [virtual]
```

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

Reimplemented from Models::Base.

19.58.3.7 SET_NEEDS_AUTO_REFRACTORY()

```
NeuronModels::SpikeSourceArray::SET_NEEDS_AUTO_REFRACTORY (
            false  )
```

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

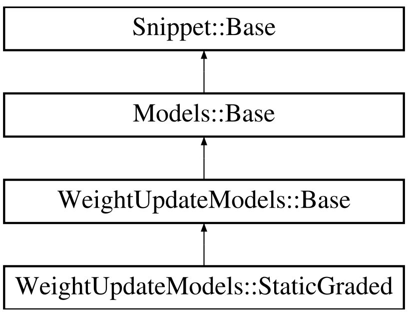
- neuronModels.h

19.59 WeightUpdateModels::StaticGraded Class Reference

Graded-potential, static synapse.

```
#include <weightUpdateModels.h>
```

Inheritance diagram for WeightUpdateModels::StaticGraded:



Public Member Functions

- DECLARE_WEIGHT_UPDATE_MODEL (StaticGraded, 2, 1, 0, 0)
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

    Gets names and types (as strings) of model variables.
- virtual std::string getEventCode () const override

    Gets code run when events (all the instances where event threshold condition is met) are received.
- virtual std::string getEventThresholdConditionCode () const override

    Gets codes to test for events.

Additional Inherited Members

19.59.1 Detailed Description

Graded-potential, static synapse.

In a graded synapse, the conductance is updated gradually with the rule:

$$gSyn = g * tanh((V - E_{pre})/V_{slope}$$

whenever the membrane potential V is larger than the threshold E_{pre}. The model has 1 variable:

- `g:` conductance of `scalar` type

The parameters are:

- `Epre:` Presynaptic threshold potential
- `Vslope:` Activation slope of graded release

`event` code is:

```
$(addToInSyn, $(g)* tanh(($(V_pre)-($(Epre)))*DT*2/$(Vslope)));
```

`event` threshold condition code is:

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

Note

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

19.59.2 Member Function Documentation

19.59.2.1 DECLARE_WEIGHT_UPDATE_MODEL()

```
WeightUpdateModels::StaticGraded::DECLARE_WEIGHT_UPDATE_MODEL (
              StaticGraded ,
              2 ,
              1 ,
              0 ,
              0 )
```

19.59.2.2 getEventCode()

`virtual std::string WeightUpdateModels::StaticGraded::getEventCode ( ) const [inline], [override], [virtual]`

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

Reimplemented from WeightUpdateModels::Base.

19.59.2.3 getEventThresholdConditionCode()

`virtual std::string WeightUpdateModels::StaticGraded::getEventThresholdConditionCode ( ) const [inline], [override], [virtual]`

Gets codes to test for events.

Reimplemented from WeightUpdateModels::Base.

19.59.2.4 getParamNames()

`virtual StringVec WeightUpdateModels::StaticGraded::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.59.2.5 getVars()

`virtual VarVec WeightUpdateModels::StaticGraded::getVars ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

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

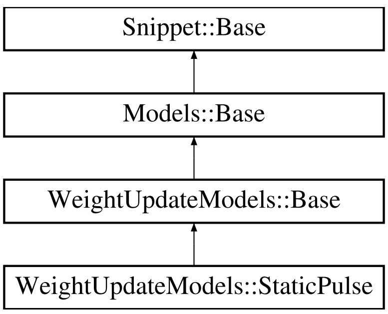
- weightUpdateModels.h

19.60 WeightUpdateModels::StaticPulse Class Reference

Pulse-coupled, static synapse.

```
#include <weightUpdateModels.h>
```

Inheritance diagram for WeightUpdateModels::StaticPulse:



Public Member Functions

- DECLARE_WEIGHT_UPDATE_MODEL (StaticPulse, 0, 1, 0, 0)
- virtual VarVec getVars () const override

  Gets names and types (as strings) of model variables.
- virtual std::string getSimCode () const override

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

Additional Inherited Members

19.60.1 Detailed Description

Pulse-coupled, static synapse.

No learning rule is applied to the synapse and for each pre-synaptic spikes, the synaptic conductances are simply added to the postsynaptic input variable. The model has 1 variable:

- g - conductance of scalar type and no other parameters.

`sim` code is:

```
"$(addToInSyn, $(g));\n"
```

19.60.2 Member Function Documentation

19.60.2.1 DECLARE_WEIGHT_UPDATE_MODEL()

```
WeightUpdateModels::StaticPulse::DECLARE_WEIGHT_UPDATE_MODEL (
            StaticPulse ,
            0 ,
            1 ,
            0 ,
            0 )
```

19.60.2.2 getSimCode()

```
virtual std::string WeightUpdateModels::StaticPulse::getSimCode ( ) const [inline], [override],
[virtual]
```

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

Reimplemented from WeightUpdateModels::Base.

19.60.2.3 getVars()

```
virtual VarVec WeightUpdateModels::StaticPulse::getVars ( ) const [inline], [override], [virtual]
```

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

Reimplemented from Models::Base.

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

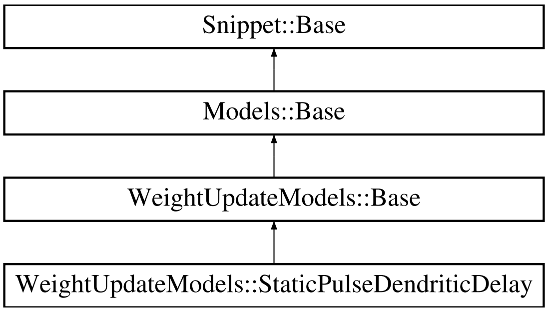
- weightUpdateModels.h

19.61 WeightUpdateModels::StaticPulseDendriticDelay Class Reference

Pulse-coupled, static synapse with heterogenous dendritic delays.

```
#include <weightUpdateModels.h>
```

Inheritance diagram for WeightUpdateModels::StaticPulseDendriticDelay:



Public Types

- typedef Snippet::ValueBase< 0 > ParamValues
- typedef Models::VarInitContainerBase< 2 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual VarVec getVars () const override

    Gets names and types (as strings) of model variables.
- virtual std::string getSimCode () const override

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

Static Public Member Functions

- static const StaticPulseDendriticDelay ∗ getInstance ()

Additional Inherited Members

19.61.1 Detailed Description

Pulse-coupled, static synapse with heterogenous dendritic delays.

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 2 variables:

- g - conductance of scalar type
- d - dendritic delay in timesteps and no other parameters.

`sim` code is:

```
" $(addToInSynDelay, $(g), $(d));\n\
```

19.61.2 Member Typedef Documentation

19.61.2.1 ParamValues

`typedef Snippet::ValueBase< 0 > WeightUpdateModels::StaticPulseDendriticDelay::ParamValues`

19.61.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> WeightUpdateModels::StaticPulseDendriticDelay::PostVarValues`

19.61.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> WeightUpdateModels::StaticPulseDendriticDelay::PreVarValues`

19.61.2.4 VarValues

`typedef Models::VarInitContainerBase< 2 > WeightUpdateModels::StaticPulseDendriticDelay::VarValues`

19.61.3 Member Function Documentation

19.61.3.1 getInstance()

`static const StaticPulseDendriticDelay* WeightUpdateModels::StaticPulseDendriticDelay::getInstance ( ) [inline], [static]`

19.61.3.2 getSimCode()

`virtual std::string WeightUpdateModels::StaticPulseDendriticDelay::getSimCode ( ) const [inline], [override], [virtual]`

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

Reimplemented from WeightUpdateModels::Base.

19.61.3.3 getVars()

`virtual VarVec WeightUpdateModels::StaticPulseDendriticDelay::getVars ( ) const [inline], [override], [virtual]`

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

Reimplemented from Models::Base.

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

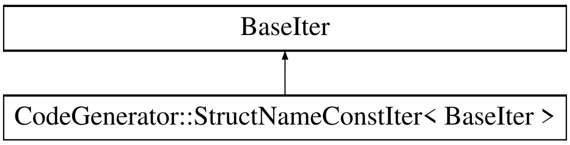
- weightUpdateModels.h

19.62 CodeGenerator::StructNameConstIter< BaseIter > Class Template Reference

Custom iterator for iterating through the containers of structs with 'name' members.

`#include <codeGenUtils.h>`

Inheritance diagram for CodeGenerator::StructNameConstIter< BaseIter >:



Public Member Functions

- StructNameConstIter ()
- StructNameConstIter (BaseIter iter)
- const std::string * operator-> () const
- const std::string & operator* () const

19.62.1 Detailed Description

template<typename BaseIter>
class CodeGenerator::StructNameConstIter< BaseIter >

Custom iterator for iterating through the containers of structs with 'name' members.

19.62.2 Constructor & Destructor Documentation

19.62.2.1 StructNameConstIter() [1/2]

```
template<typename BaseIter >
CodeGenerator::StructNameConstIter< BaseIter >::StructNameConstIter ( )  [inline]
```

19.62.2.2 StructNameConstIter() [2/2]

```
template<typename BaseIter >
CodeGenerator::StructNameConstIter< BaseIter >::StructNameConstIter (
            BaseIter iter )  [inline]
```

19.62.3 Member Function Documentation

19.62.3.1 operator*()

```
template<typename BaseIter >
const std::string& CodeGenerator::StructNameConstIter< BaseIter >::operator* ( ) const  [inline]
```

19.62.3.2 operator->()

```
template<typename BaseIter >
const std::string* CodeGenerator::StructNameConstIter< BaseIter >::operator-> ( ) const [inline]
```

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

- codeGenUtils.h

19.63 CodeGenerator::Substitutions Class Reference

```
#include <substitutions.h>
```

Public Member Functions

- Substitutions (const Substitutions *parent=nullptr)
- Substitutions (const std::vector< FunctionTemplate > &functions, const std::string &ftype)
- void addVarSubstitution (const std::string &source, const std::string &destionation, bool allowOverride=false)
- void addFuncSubstitution (const std::string &source, unsigned int numArguments, const std::string &func↩ Template, bool allowOverride=false)
- bool hasVarSubstitution (const std::string &source) const
- const std::string & getVarSubstitution (const std::string &source) const
- void apply (std::string &code) const
- const std::string operator[] (const std::string &source) const

19.63.1 Constructor & Destructor Documentation

19.63.1.1 Substitutions() [1/2]

```
CodeGenerator::Substitutions::Substitutions (
            const Substitutions * parent = nullptr ) [inline]
```

19.63.1.2 Substitutions() [2/2]

```
CodeGenerator::Substitutions::Substitutions (
            const std::vector< FunctionTemplate > & functions,
            const std::string & ftype ) [inline]
```

19.63.2 Member Function Documentation

19.63.2.1 addFuncSubstitution()

```
void CodeGenerator::Substitutions::addFuncSubstitution (
            const std::string & source,
            unsigned int numArguments,
            const std::string & funcTemplate,
            bool allowOverride = false ) [inline]
```

19.63.2.2 addVarSubstitution()

```
void CodeGenerator::Substitutions::addVarSubstitution (
            const std::string & source,
            const std::string & destionation,
            bool allowOverride = false )  [inline]
```

19.63.2.3 apply()

```
void CodeGenerator::Substitutions::apply (
            std::string & code ) const  [inline]
```

19.63.2.4 getVarSubstitution()

```
const std::string& CodeGenerator::Substitutions::getVarSubstitution (
            const std::string & source ) const  [inline]
```

19.63.2.5 hasVarSubstitution()

```
bool CodeGenerator::Substitutions::hasVarSubstitution (
            const std::string & source ) const  [inline]
```

19.63.2.6 operator[]()

```
const std::string CodeGenerator::Substitutions::operator[] (
            const std::string & source ) const  [inline]
```

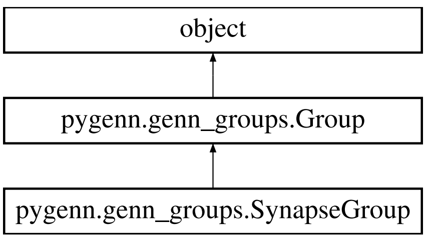
The documentation for this class was generated from the following file:

- substitutions.h

19.64 pygenn.genn_groups.SynapseGroup Class Reference

Class representing synaptic connection between two groups of neurons.

Inheritance diagram for pygenn.genn_groups.SynapseGroup:



Public Member Functions

- def __init__ (self, name)

    Init SynapseGroup.
- def num_synapses (self)

    Number of synapses in group.

- def weight_update_var_size (self)

    Size of each weight update variable.
- def max_row_length (self)
- def set_psm_var (self, var_name, values)

    Set values for a postsynaptic model variable.
- def set_pre_var (self, var_name, values)

    Set values for a presynaptic variable.
- def set_post_var (self, var_name, values)

    Set values for a postsynaptic variable.
- def set_weight_update (self, model, param_space, var_space, pre_var_space, post_var_space)

    Set weight update model, its parameters and initial variables.
- def set_post_syn (self, model, param_space, var_space)

    Set postsynaptic model, its parameters and initial variables.
- def get_var_values (self, var_name)
- def is_connectivity_init_required (self)
- def matrix_type (self)

    Type of the projection matrix.
- def matrix_type (self, matrix_type)
- def is_ragged (self)

    Tests whether synaptic connectivity uses Ragged format.
- def is_bitmask (self)

    Tests whether synaptic connectivity uses Bitmask format.
- def is_dense (self)

    Tests whether synaptic connectivity uses dense format.
- def has_individual_synapse_vars (self)

    Tests whether synaptic connectivity has individual weights.
- def has_individual_postsynaptic_vars (self)

    Tests whether synaptic connectivity has individual postsynaptic model variables.
- def set_sparse_connections (self, pre_indices, post_indices)

    Set ragged format connections between two groups of neurons.
- def get_sparse_pre_inds (self)

    Get presynaptic indices of synapse group connections.
- def get_sparse_post_inds (self)

    Get postsynaptic indices of synapse group connections.
- def set_connected_populations (self, source, target)

    Set two groups of neurons connected by this SynapseGroup.
- def add_to (self, model_spec, delay_steps)

    Add this SynapseGroup to the a model.
- def add_extra_global_param (self, param_name, param_values)

    Add extra global parameter.
- def load (self, slm, scalar)
- def reinitialise (self, slm, scalar)

    Reinitialise synapse group.

Public Attributes

- connections_set
- w_update
- postsyn
- src
- trg
- psm_vars
- pre_vars
- post_vars
- connectivity_initialiser
- synapse_order
- ind
- row_lengths
- pop

19.64.1 Detailed Description

Class representing synaptic connection between two groups of neurons.

19.64.2 Constructor & Destructor Documentation

19.64.2.1 __init__()

```
def pygenn.genn_groups.SynapseGroup.__init__ (
              self,
              name )
```

Init SynapseGroup.

Parameters

name	string name of the group

19.64.3 Member Function Documentation

19.64.3.1 add_extra_global_param()

```
def pygenn.genn_groups.SynapseGroup.add_extra_global_param (
              self,
              param_name,
              param_values )
```

Add extra global parameter.

Parameters

param_name	string with the name of the extra global parameter
param_values	iterable or a single value

19.64.3.2 add_to()

```
def pygenn.genn_groups.SynapseGroup.add_to (
              self,
              model_spec,
              delay_steps )
```

Add this SynapseGroup to the a model.

Parameters

model_spec	`pygenn.genn_model.GeNNModel` to add to
delay_steps	number of axonal delay timesteps to simulate for this synapse group

19.64.3.3 get_sparse_post_inds()

```
def pygenn.genn_groups.SynapseGroup.get_sparse_post_inds (
              self )
```

Get postsynaptic indices of synapse group connections.

Returns

ndarrays of postsynaptic indices

19.64.3.4 get_sparse_pre_inds()

```
def pygenn.genn_groups.SynapseGroup.get_sparse_pre_inds (
              self )
```

Get presynaptic indices of synapse group connections.

Returns

ndarray of presynaptic indices

19.64.3.5 get_var_values()

```
def pygenn.genn_groups.SynapseGroup.get_var_values (
              self,
              var_name )
```

19.64.3.6 has_individual_postsynaptic_vars()

```
def pygenn.genn_groups.SynapseGroup.has_individual_postsynaptic_vars (
              self )
```

Tests whether synaptic connectivity has individual postsynaptic model variables.

19.64.3.7 has_individual_synapse_vars()

```
def pygenn.genn_groups.SynapseGroup.has_individual_synapse_vars (
              self )
```

Tests whether synaptic connectivity has individual weights.

19.64.3.8 is_bitmask()

```
def pygenn.genn_groups.SynapseGroup.is_bitmask (
              self )
```

Tests whether synaptic connectivity uses Bitmask format.

19.64.3.9 is_connectivity_init_required()

```
def pygenn.genn_groups.SynapseGroup.is_connectivity_init_required (
              self )
```

19.64.3.10 is_dense()

```
def pygenn.genn_groups.SynapseGroup.is_dense (
              self )
```

Tests whether synaptic connectivity uses dense format.

19.64.3.11 is_ragged()

```
def pygenn.genn_groups.SynapseGroup.is_ragged (
              self )
```

Tests whether synaptic connectivity uses Ragged format.

19.64.3.12 load()

```
def pygenn.genn_groups.SynapseGroup.load (
              self,
              slm,
              scalar )
```

19.64.3.13 matrix_type() [1/2]

```
def pygenn.genn_groups.SynapseGroup.matrix_type (
              self )
```

Type of the projection matrix.

19.64.3.14 matrix_type() [2/2]

```
def pygenn.genn_groups.SynapseGroup.matrix_type (
              self,
              matrix_type )
```

19.64.3.15 max_row_length()

```
def pygenn.genn_groups.SynapseGroup.max_row_length (
              self )
```

19.64.3.16 num_synapses()

```
def pygenn.genn_groups.SynapseGroup.num_synapses (
              self )
```

Number of synapses in group.

19.64.3.17 reinitialise()

```
def pygenn.genn_groups.SynapseGroup.reinitialise (
              self,
              slm,
              scalar )
```

Reinitialise synapse group.

Parameters

slm	SharedLibraryModel instance for acccessing variables
scalar	String specifying "scalar" type

19.64.3.18 set_connected_populations()

```
def pygenn.genn_groups.SynapseGroup.set_connected_populations (
              self,
              source,
              target )
```

Set two groups of neurons connected by this SynapseGroup.

Parameters

source	string name of the presynaptic neuron group
target	string name of the postsynaptic neuron group

19.64.3.19 set_post_syn()

```
def pygenn.genn_groups.SynapseGroup.set_post_syn (
              self,
              model,
              param_space,
              var_space )
```

Set postsynaptic model, its parameters and initial variables.

Parameters

model	type as string of intance of the model
param_space	dict with model parameters
var_space	dict with model variables

19.64.3.20 set_post_var()

```
def pygenn.genn_groups.SynapseGroup.set_post_var (
              self,
              var_name,
              values )
```

Set values for a postsynaptic variable.

Parameters

var_name	string with the name of the presynaptic variable
values	iterable or a single value

19.64.3.21 set_pre_var()

```
def pygenn.genn_groups.SynapseGroup.set_pre_var (
              self,
              var_name,
              values )
```

Set values for a presynaptic variable.

Parameters

var_name	string with the name of the presynaptic variable
values	iterable or a single value

19.64.3.22 set_psm_var()

```
def pygenn.genn_groups.SynapseGroup.set_psm_var (
              self,
              var_name,
              values )
```

Set values for a postsynaptic model variable.

Parameters

var_name	string with the name of the postsynaptic model variable
values	iterable or a single value

19.64.3.23 set_sparse_connections()

```
def pygenn.genn_groups.SynapseGroup.set_sparse_connections (
              self,
              pre_indices,
              post_indices )
```

Set ragged format connections between two groups of neurons.

Parameters

pre_indices	ndarray of presynaptic indices
post_indices	ndarray of postsynaptic indices

19.64.3.24 set_weight_update()

```
def pygenn.genn_groups.SynapseGroup.set_weight_update (
              self,
              model,
              param_space,
              var_space,
              pre_var_space,
              post_var_space )
```

Set weight update model, its parameters and initial variables.

Parameters

model	type as string of intance of the model
param_space	dict with model parameters
var_space	dict with model variables
pre_var_space	dict with model presynaptic variables
post_var_space	dict with model postsynaptic variables

19.64.3.25 weight_update_var_size()

```
def pygenn.genn_groups.SynapseGroup.weight_update_var_size (
              self )
```

Size of each weight update variable.

19.64.4 Member Data Documentation

19.64.4.1 connections_set

```
pygenn.genn_groups.SynapseGroup.connections_set
```

19.64.4.2 connectivity_initialiser

```
pygenn.genn_groups.SynapseGroup.connectivity_initialiser
```

19.64.4.3 ind

`pygenn.genn_groups.SynapseGroup.ind`

19.64.4.4 pop

`pygenn.genn_groups.SynapseGroup.pop`

19.64.4.5 post_vars

`pygenn.genn_groups.SynapseGroup.post_vars`

19.64.4.6 postsyn

`pygenn.genn_groups.SynapseGroup.postsyn`

19.64.4.7 pre_vars

`pygenn.genn_groups.SynapseGroup.pre_vars`

19.64.4.8 psm_vars

`pygenn.genn_groups.SynapseGroup.psm_vars`

19.64.4.9 row_lengths

`pygenn.genn_groups.SynapseGroup.row_lengths`

19.64.4.10 src

`pygenn.genn_groups.SynapseGroup.src`

19.64.4.11 synapse_order

`pygenn.genn_groups.SynapseGroup.synapse_order`

19.64.4.12 trg

`pygenn.genn_groups.SynapseGroup.trg`

19.64.4.13 w_update

`pygenn.genn_groups.SynapseGroup.w_update`

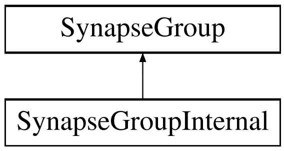
The documentation for this class was generated from the following file:

- genn_groups.py

19.65 SynapseGroup Class Reference

`#include <synapseGroup.h>`

Inheritance diagram for SynapseGroup:



Public Types

- enum SpanType { SpanType::POSTSYNAPTIC, SpanType::PRESYNAPTIC }

Public Member Functions

- SynapseGroup (const SynapseGroup &)=delete
- SynapseGroup ()=delete
- void setWUVarLocation (const std::string &varName, VarLocation loc)

  Set location of weight update model state variable.
- void setWUPreVarLocation (const std::string &varName, VarLocation loc)

  Set location of weight update model presynaptic state variable.
- void setWUPostVarLocation (const std::string &varName, VarLocation loc)

  Set location of weight update model postsynaptic state variable.
- void setWUExtraGlobalParamLocation (const std::string &paramName, VarLocation loc)

  Set location of weight update model extra global parameter.
- void setPSVarLocation (const std::string &varName, VarLocation loc)

  Set location of postsynaptic model state variable.
- void setPSExtraGlobalParamLocation (const std::string &paramName, VarLocation loc)

  Set location of postsynaptic model extra global parameter.
- void setSparseConnectivityExtraGlobalParamLocation (const std::string &paramName, VarLocation loc)

  Set location of sparse connectivity initialiser extra global parameter.
- void setInSynVarLocation (VarLocation loc)

  Set location of variables used to combine input from this synapse group.
- void setSparseConnectivityLocation (VarLocation loc)

  Set variable mode used for sparse connectivity.
- void setDendriticDelayLocation (VarLocation loc)

  Set variable mode used for this synapse group's dendritic delay buffers.
- void setMaxConnections (unsigned int maxConnections)

  Sets the maximum number of target neurons any source neurons can connect to.
- void setMaxSourceConnections (unsigned int maxPostConnections)

  Sets the maximum number of source neurons any target neuron can connect to.
- void setMaxDendriticDelayTimesteps (unsigned int maxDendriticDelay)

  Sets the maximum dendritic delay for synapses in this synapse group.
- void setSpanType (SpanType spanType)

  Set how CUDA implementation is parallelised.
- void setBackPropDelaySteps (unsigned int timesteps)

Sets the number of delay steps used to delay postsynaptic spikes travelling back along dendrites to synapses.

- const std::string & getName () const
- SpanType getSpanType () const
- unsigned int getDelaySteps () const
- unsigned int getBackPropDelaySteps () const
- unsigned int getMaxConnections () const
- unsigned int getMaxSourceConnections () const
- unsigned int getMaxDendriticDelayTimesteps () const
- SynapseMatrixType getMatrixType () const
- VarLocation getInSynLocation () const

  Get variable mode used for variables used to combine input from this synapse group.
- VarLocation getSparseConnectivityLocation () const

  Get variable mode used for sparse connectivity.
- VarLocation getDendriticDelayLocation () const

  Get variable mode used for this synapse group's dendritic delay buffers.
- int getClusterHostID () const
- bool isTrueSpikeRequired () const

  Does synapse group need to handle 'true' spikes.
- bool isSpikeEventRequired () const

  Does synapse group need to handle spike-like events.
- const WeightUpdateModels::Base ∗ getWUModel () const
- const std::vector< double > & getWUParams () const
- const std::vector< Models::VarInit > & getWUVarInitialisers () const
- const std::vector< Models::VarInit > & getWUPreVarInitialisers () const
- const std::vector< Models::VarInit > & getWUPostVarInitialisers () const
- const std::vector< double > getWUConstInitVals () const
- const PostsynapticModels::Base ∗ getPSModel () const
- const std::vector< double > & getPSParams () const
- const std::vector< Models::VarInit > & getPSVarInitialisers () const
- const std::vector< double > getPSConstInitVals () const
- const InitSparseConnectivitySnippet::Init & getConnectivityInitialiser () const
- bool isZeroCopyEnabled () const
- VarLocation getWUVarLocation (const std::string &var) const

  Get location of weight update model per-synapse state variable by name.
- VarLocation getWUVarLocation (size_t index) const

  Get location of weight update model per-synapse state variable by index.
- VarLocation getWUPreVarLocation (const std::string &var) const

  Get location of weight update model presynaptic state variable by name.
- VarLocation getWUPreVarLocation (size_t index) const

  Get location of weight update model presynaptic state variable by index.
- VarLocation getWUPostVarLocation (const std::string &var) const

  Get location of weight update model postsynaptic state variable by name.
- VarLocation getWUPostVarLocation (size_t index) const

  Get location of weight update model postsynaptic state variable by index.
- VarLocation getWUExtraGlobalParamLocation (const std::string &paramName) const

  Get location of weight update model extra global parameter by name.
- VarLocation getWUExtraGlobalParamLocation (size_t index) const

  Get location of weight update model extra global parameter by index.
- VarLocation getPSVarLocation (const std::string &var) const

  Get location of postsynaptic model state variable.
- VarLocation getPSVarLocation (size_t index) const

  Get location of postsynaptic model state variable.

- VarLocation getPSExtraGlobalParamLocation (const std::string &paramName) const

  Get location of postsynaptic model extra global parameter by name.
- VarLocation getPSExtraGlobalParamLocation (size_t index) const

  Get location of postsynaptic model extra global parameter by index.
- VarLocation getSparseConnectivityExtraGlobalParamLocation (const std::string &paramName) const

  Get location of sparse connectivity initialiser extra global parameter by name.
- VarLocation getSparseConnectivityExtraGlobalParamLocation (size_t index) const

  Get location of sparse connectivity initialiser extra global parameter by index.
- bool isDendriticDelayRequired () const

  Does this synapse group require dendritic delay?
- bool isPSInitRNGRequired () const

  Does this synapse group require an RNG for it's postsynaptic init code?
- bool isWUInitRNGRequired () const

  Does this synapse group require an RNG for it's weight update init code?
- bool isWUVarInitRequired () const

  Is var init code required for any variables in this synapse group's weight update model?
- bool isSparseConnectivityInitRequired () const

  Is sparse connectivity initialisation code required for this synapse group?

Protected Member Functions

- SynapseGroup (const std::string name, SynapseMatrixType matrixType, unsigned int delaySteps, const WeightUpdateModels::Base ∗wu, const std::vector< double > &wuParams, const std::vector< Models::↩ VarInit > &wuVarInitialisers, const std::vector< Models::VarInit > &wuPreVarInitialisers, const std::vector< Models::VarInit > &wuPostVarInitialisers, const PostsynapticModels::Base ∗ps, const std::vector< double > &psParams, const std::vector< Models::VarInit > &psVarInitialisers, NeuronGroupInternal ∗srcNeuronGroup, NeuronGroupInternal ∗trgNeuronGroup, const InitSparseConnectivitySnippet::Init &connectivityInitialiser, VarLocation defaultVarLocation, VarLocation defaultExtraGlobalParamLocation, VarLocation defaultSparse↩ ConnectivityLocation)
- NeuronGroupInternal ∗ getSrcNeuronGroup ()
- NeuronGroupInternal ∗ getTrgNeuronGroup ()
- void setEventThresholdReTestRequired (bool req)
- void setPSModelMergeTarget (const std::string &targetName)
- void initDerivedParams (double dt)
- const NeuronGroupInternal ∗ getSrcNeuronGroup () const
- const NeuronGroupInternal ∗ getTrgNeuronGroup () const
- const std::vector< double > & getWUDerivedParams () const
- const std::vector< double > & getPSDerivedParams () const

  Does the event threshold needs to be retested in the synapse kernel?
- bool isEventThresholdReTestRequired () const
- const std::string & getPSModelTargetName () const
- bool isPSModelMerged () const
- std::string getPresynapticAxonalDelaySlot (const std::string &devPrefix) const
- std::string getPostsynapticBackPropDelaySlot (const std::string &devPrefix) const
- std::string getDendriticDelayOffset (const std::string &devPrefix, const std::string &offset="") const

19.65.1 Member Enumeration Documentation

19.65.1.1 SpanType

```
enum SynapseGroup::SpanType [strong]
```

Enumerator

POSTSYNAPTIC	
PRESYNAPTIC	

19.65.2 Constructor & Destructor Documentation

19.65.2.1 SynapseGroup() [1/3]

```
SynapseGroup::SynapseGroup (
            const SynapseGroup & ) [delete]
```

19.65.2.2 SynapseGroup() [2/3]

```
SynapseGroup::SynapseGroup ( ) [delete]
```

19.65.2.3 SynapseGroup() [3/3]

```
SynapseGroup::SynapseGroup (
            const std::string name,
            SynapseMatrixType matrixType,
            unsigned int delaySteps,
            const WeightUpdateModels::Base * wu,
            const std::vector< double > & wuParams,
            const std::vector< Models::VarInit > & wuVarInitialisers,
            const std::vector< Models::VarInit > & wuPreVarInitialisers,
            const std::vector< Models::VarInit > & wuPostVarInitialisers,
            const PostsynapticModels::Base * ps,
            const std::vector< double > & psParams,
            const std::vector< Models::VarInit > & psVarInitialisers,
            NeuronGroupInternal * srcNeuronGroup,
            NeuronGroupInternal * trgNeuronGroup,
            const InitSparseConnectivitySnippet::Init & connectivityInitialiser,
            VarLocation defaultVarLocation,
            VarLocation defaultExtraGlobalParamLocation,
            VarLocation defaultSparseConnectivityLocation ) [protected]
```

19.65.3 Member Function Documentation

19.65.3.1 getBackPropDelaySteps()

```
unsigned int SynapseGroup::getBackPropDelaySteps ( ) const [inline]
```

19.65.3.2 getClusterHostID()

```
int SynapseGroup::getClusterHostID ( ) const
```

19.65.3.3 getConnectivityInitialiser()

```
const InitSparseConnectivitySnippet::Init& SynapseGroup::getConnectivityInitialiser ( ) const
[inline]
```

19.65.3.4 getDelaySteps()

```
unsigned int SynapseGroup::getDelaySteps ( ) const [inline]
```

19.65.3.5 getDendriticDelayLocation()

```
VarLocation SynapseGroup::getDendriticDelayLocation ( ) const [inline]
```

Get variable mode used for this synapse group's dendritic delay buffers.

19.65.3.6 getDendriticDelayOffset()

```
std::string SynapseGroup::getDendriticDelayOffset (
            const std::string & devPrefix,
            const std::string & offset = "" ) const [protected]
```

19.65.3.7 getInSynLocation()

```
VarLocation SynapseGroup::getInSynLocation ( ) const [inline]
```

Get variable mode used for variables used to combine input from this synapse group.

19.65.3.8 getMatrixType()

```
SynapseMatrixType SynapseGroup::getMatrixType ( ) const [inline]
```

19.65.3.9 getMaxConnections()

```
unsigned int SynapseGroup::getMaxConnections ( ) const [inline]
```

19.65.3.10 getMaxDendriticDelayTimesteps()

```
unsigned int SynapseGroup::getMaxDendriticDelayTimesteps ( ) const [inline]
```

19.65.3.11 getMaxSourceConnections()

```
unsigned int SynapseGroup::getMaxSourceConnections ( ) const [inline]
```

19.65.3.12 getName()

```
const std::string& SynapseGroup::getName ( ) const [inline]
```

19.65.3.13 getPostsynapticBackPropDelaySlot()

```
std::string SynapseGroup::getPostsynapticBackPropDelaySlot (
            const std::string & devPrefix ) const [protected]
```

Get the expression to calculate the delay slot for accessing Postsynaptic neuron state variables, taking into account back propagation delay

19.65.3.14 getPresynapticAxonalDelaySlot()

```
std::string SynapseGroup::getPresynapticAxonalDelaySlot (
            const std::string & devPrefix ) const [protected]
```

Get the expression to calculate the delay slot for accessing Presynaptic neuron state variables, taking into account axonal delay

19.65.3.15 getPSConstInitVals()

```
const std::vector< double > SynapseGroup::getPSConstInitVals ( ) const
```

19.65.3.16 getPSDerivedParams()

```
const std::vector<double>& SynapseGroup::getPSDerivedParams ( ) const [inline], [protected]
```

Does the event threshold needs to be retested in the synapse kernel?

19.65.3.17 getPSExtraGlobalParamLocation() [1/2]

```
VarLocation SynapseGroup::getPSExtraGlobalParamLocation (
            const std::string & paramName ) const
```

Get location of postsynaptic model extra global parameter by name.

This is only used by extra global parameters which are pointers

19.65.3.18 getPSExtraGlobalParamLocation() [2/2]

```
VarLocation SynapseGroup::getPSExtraGlobalParamLocation (
            size_t index ) const [inline]
```

Get location of postsynaptic model extra global parameter by index.

This is only used by extra global parameters which are pointers

19.65.3.19 getPSModel()

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

19.65.3.20 getPSModelTargetName()

```
const std::string& SynapseGroup::getPSModelTargetName ( ) const [inline], [protected]
```

19.65.3.21 getPSParams()

```
const std::vector<double>& SynapseGroup::getPSParams ( ) const [inline]
```

19.65.3.22 getPSVarInitialisers()

```
const std::vector<Models::VarInit>& SynapseGroup::getPSVarInitialisers ( ) const [inline]
```

19.65.3.23 getPSVarLocation() [1/2]

```
VarLocation SynapseGroup::getPSVarLocation (
        const std::string & var ) const
```

Get location of postsynaptic model state variable.

19.65.3.24 getPSVarLocation() [2/2]

```
VarLocation SynapseGroup::getPSVarLocation (
        size_t index ) const [inline]
```

Get location of postsynaptic model state variable.

19.65.3.25 getSpanType()

```
SpanType SynapseGroup::getSpanType ( ) const [inline]
```

19.65.3.26 getSparseConnectivityExtraGlobalParamLocation() [1/2]

```
VarLocation SynapseGroup::getSparseConnectivityExtraGlobalParamLocation (
        const std::string & paramName ) const
```

Get location of sparse connectivity initialiser extra global parameter by name.

This is only used by extra global parameters which are pointers

19.65.3.27 getSparseConnectivityExtraGlobalParamLocation() [2/2]

```
VarLocation SynapseGroup::getSparseConnectivityExtraGlobalParamLocation (
        size_t index ) const [inline]
```

Get location of sparse connectivity initialiser extra global parameter by index.

This is only used by extra global parameters which are pointers

19.65.3.28 getSparseConnectivityLocation()

```
VarLocation SynapseGroup::getSparseConnectivityLocation ( ) const [inline]
```

Get variable mode used for sparse connectivity.

19.65.3.29 getSrcNeuronGroup() [1/2]

```
NeuronGroupInternal* SynapseGroup::getSrcNeuronGroup ( ) [inline], [protected]
```

19.65.3.30 getSrcNeuronGroup() [2/2]

```
const NeuronGroupInternal* SynapseGroup::getSrcNeuronGroup ( ) const [inline], [protected]
```

19.65.3.31 getTrgNeuronGroup() [1/2]

`NeuronGroupInternal* SynapseGroup::getTrgNeuronGroup ( ) [inline], [protected]`

19.65.3.32 getTrgNeuronGroup() [2/2]

`const NeuronGroupInternal* SynapseGroup::getTrgNeuronGroup ( ) const [inline], [protected]`

19.65.3.33 getWUConstInitVals()

`const std::vector< double > SynapseGroup::getWUConstInitVals ( ) const`

19.65.3.34 getWUDerivedParams()

`const std::vector<double>& SynapseGroup::getWUDerivedParams ( ) const [inline], [protected]`

19.65.3.35 getWUExtraGlobalParamLocation() [1/2]

```
VarLocation SynapseGroup::getWUExtraGlobalParamLocation (
             const std::string & paramName ) const
```

Get location of weight update model extra global parameter by name.

This is only used by extra global parameters which are pointers

19.65.3.36 getWUExtraGlobalParamLocation() [2/2]

```
VarLocation SynapseGroup::getWUExtraGlobalParamLocation (
             size_t index ) const [inline]
```

Get location of weight update model extra global parameter by index.

This is only used by extra global parameters which are pointers

19.65.3.37 getWUModel()

`const WeightUpdateModels::Base* SynapseGroup::getWUModel ( ) const [inline]`

19.65.3.38 getWUParams()

`const std::vector<double>& SynapseGroup::getWUParams ( ) const [inline]`

19.65.3.39 getWUPostVarInitialisers()

`const std::vector<Models::VarInit>& SynapseGroup::getWUPostVarInitialisers ( ) const [inline]`

19.65.3.40 getWUPostVarLocation() [1/2]

```
VarLocation SynapseGroup::getWUPostVarLocation (
             const std::string & var ) const
```

Get location of weight update model postsynaptic state variable by name.

19.65.3.41 getWUPostVarLocation() [2/2]

```
VarLocation SynapseGroup::getWUPostVarLocation (
            size_t index ) const [inline]
```

Get location of weight update model postsynaptic state variable by index.

19.65.3.42 getWUPreVarInitialisers()

```
const std::vector<Models::VarInit>& SynapseGroup::getWUPreVarInitialisers ( ) const [inline]
```

19.65.3.43 getWUPreVarLocation() [1/2]

```
VarLocation SynapseGroup::getWUPreVarLocation (
            const std::string & var ) const
```

Get location of weight update model presynaptic state variable by name.

19.65.3.44 getWUPreVarLocation() [2/2]

```
VarLocation SynapseGroup::getWUPreVarLocation (
            size_t index ) const [inline]
```

Get location of weight update model presynaptic state variable by index.

19.65.3.45 getWUVarInitialisers()

```
const std::vector<Models::VarInit>& SynapseGroup::getWUVarInitialisers ( ) const [inline]
```

19.65.3.46 getWUVarLocation() [1/2]

```
VarLocation SynapseGroup::getWUVarLocation (
            const std::string & var ) const
```

Get location of weight update model per-synapse state variable by name.

19.65.3.47 getWUVarLocation() [2/2]

```
VarLocation SynapseGroup::getWUVarLocation (
            size_t index ) const [inline]
```

Get location of weight update model per-synapse state variable by index.

19.65.3.48 initDerivedParams()

```
void SynapseGroup::initDerivedParams (
            double dt ) [protected]
```

19.65.3.49 isDendriticDelayRequired()

```
bool SynapseGroup::isDendriticDelayRequired ( ) const
```

Does this synapse group require dendritic delay?

19.65.3.50 isEventThresholdReTestRequired()

`bool SynapseGroup::isEventThresholdReTestRequired ( ) const [inline], [protected]`

This is required when the pre-synaptic neuron population's outgoing synapse groups require different event threshold

19.65.3.51 isPSInitRNGRequired()

`bool SynapseGroup::isPSInitRNGRequired ( ) const`

Does this synapse group require an RNG for it's postsynaptic init code?

19.65.3.52 isPSModelMerged()

`bool SynapseGroup::isPSModelMerged ( ) const [inline], [protected]`

19.65.3.53 isSparseConnectivityInitRequired()

`bool SynapseGroup::isSparseConnectivityInitRequired ( ) const`

Is sparse connectivity initialisation code required for this synapse group?

19.65.3.54 isSpikeEventRequired()

`bool SynapseGroup::isSpikeEventRequired ( ) const`

Does synapse group need to handle spike-like events.

19.65.3.55 isTrueSpikeRequired()

`bool SynapseGroup::isTrueSpikeRequired ( ) const`

Does synapse group need to handle 'true' spikes.

19.65.3.56 isWUInitRNGRequired()

`bool SynapseGroup::isWUInitRNGRequired ( ) const`

Does this synapse group require an RNG for it's weight update init code?

19.65.3.57 isWUVarInitRequired()

`bool SynapseGroup::isWUVarInitRequired ( ) const`

Is var init code required for any variables in this synapse group's weight update model?

19.65.3.58 isZeroCopyEnabled()

`bool SynapseGroup::isZeroCopyEnabled ( ) const`

19.65.3.59 setBackPropDelaySteps()

```
void SynapseGroup::setBackPropDelaySteps (
            unsigned int timesteps )
```

Sets the number of delay steps used to delay postsynaptic spikes travelling back along dendrites to synapses.

19.65.3.60 setDendriticDelayLocation()

```
void SynapseGroup::setDendriticDelayLocation (
            VarLocation loc )  [inline]
```

Set variable mode used for this synapse group's dendritic delay buffers.

19.65.3.61 setEventThresholdReTestRequired()

```
void SynapseGroup::setEventThresholdReTestRequired (
            bool req )  [inline], [protected]
```

19.65.3.62 setInSynVarLocation()

```
void SynapseGroup::setInSynVarLocation (
            VarLocation loc )  [inline]
```

Set location of variables used to combine input from this synapse group.

This is ignored for simulations on hardware with a single memory space

19.65.3.63 setMaxConnections()

```
void SynapseGroup::setMaxConnections (
            unsigned int maxConnections )
```

Sets the maximum number of target neurons any source neurons can connect to.

Use with synaptic matrix types with SynapseMatrixConnectivity::SPARSE to optimise CUDA implementation

19.65.3.64 setMaxDendriticDelayTimesteps()

```
void SynapseGroup::setMaxDendriticDelayTimesteps (
            unsigned int maxDendriticDelay )
```

Sets the maximum dendritic delay for synapses in this synapse group.

19.65.3.65 setMaxSourceConnections()

```
void SynapseGroup::setMaxSourceConnections (
            unsigned int maxPostConnections )
```

Sets the maximum number of source neurons any target neuron can connect to.

Use with synaptic matrix types with SynapseMatrixConnectivity::SPARSE and postsynaptic learning to optimise CUDA implementation

19.65.3.66 setPSExtraGlobalParamLocation()

```
void SynapseGroup::setPSExtraGlobalParamLocation (
```

```
            const std::string & paramName,
            VarLocation loc )
```

Set location of postsynaptic model extra global parameter.

This is ignored for simulations on hardware with a single memory space and only applies to extra global parameters which are pointers.

19.65.3.67 setPSModelMergeTarget()

```
void SynapseGroup::setPSModelMergeTarget (
            const std::string & targetName ) [inline], [protected]
```

19.65.3.68 setPSVarLocation()

```
void SynapseGroup::setPSVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set location of postsynaptic model state variable.

This is ignored for simulations on hardware with a single memory space

19.65.3.69 setSpanType()

```
void SynapseGroup::setSpanType (
            SpanType spanType )
```

Set how CUDA implementation is parallelised.

with a thread per target neuron (default) or a thread per source spike

19.65.3.70 setSparseConnectivityExtraGlobalParamLocation()

```
void SynapseGroup::setSparseConnectivityExtraGlobalParamLocation (
            const std::string & paramName,
            VarLocation loc )
```

Set location of sparse connectivity initialiser extra global parameter.

This is ignored for simulations on hardware with a single memory space and only applies to extra global parameters which are pointers.

19.65.3.71 setSparseConnectivityLocation()

```
void SynapseGroup::setSparseConnectivityLocation (
            VarLocation loc ) [inline]
```

Set variable mode used for sparse connectivity.

This is ignored for simulations on hardware with a single memory space

19.65.3.72 setWUExtraGlobalParamLocation()

```
void SynapseGroup::setWUExtraGlobalParamLocation (
            const std::string & paramName,
            VarLocation loc )
```

Set location of weight update model extra global parameter.

This is ignored for simulations on hardware with a single memory space and only applies to extra global parameters which are pointers.

19.65.3.73 setWUPostVarLocation()

```
void SynapseGroup::setWUPostVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set location of weight update model postsynaptic state variable.

This is ignored for simulations on hardware with a single memory space

19.65.3.74 setWUPreVarLocation()

```
void SynapseGroup::setWUPreVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set location of weight update model presynaptic state variable.

This is ignored for simulations on hardware with a single memory space

19.65.3.75 setWUVarLocation()

```
void SynapseGroup::setWUVarLocation (
            const std::string & varName,
            VarLocation loc )
```

Set location of weight update model state variable.

This is ignored for simulations on hardware with a single memory space

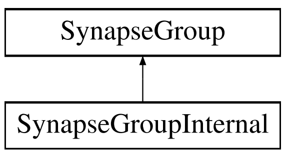
The documentation for this class was generated from the following files:

- synapseGroup.h
- synapseGroup.cc

19.66 SynapseGroupInternal Class Reference

```
#include <synapseGroupInternal.h>
```

Inheritance diagram for SynapseGroupInternal:



Public Member Functions

- SynapseGroupInternal (const std::string name, SynapseMatrixType matrixType, unsigned int delaySteps, const WeightUpdateModels::Base ∗wu, const std::vector< double > &wuParams, const std::vector< Models::VarInit > &wuVarInitialisers, const std::vector< Models::VarInit > &wuPreVarInitialisers, const std::vector< Models::VarInit > &wuPostVarInitialisers, const PostsynapticModels::Base ∗ps, const std::vector< double > &psParams, const std::vector< Models::VarInit > &psVarInitialisers, NeuronGroupInternal ∗srcNeuronGroup, NeuronGroupInternal ∗trgNeuronGroup, const InitSparseConnectivitySnippet::Init &connectivityInitialiser, VarLocation defaultVarLocation, VarLocation defaultExtraGlobalParamLocation, VarLocation defaultSparseConnectivityLocation)

Additional Inherited Members

19.66.1 Constructor & Destructor Documentation

19.66.1.1 SynapseGroupInternal()

```
SynapseGroupInternal::SynapseGroupInternal (
            const std::string name,
            SynapseMatrixType matrixType,
            unsigned int delaySteps,
            const WeightUpdateModels::Base * wu,
            const std::vector< double > & wuParams,
            const std::vector< Models::VarInit > & wuVarInitialisers,
            const std::vector< Models::VarInit > & wuPreVarInitialisers,
            const std::vector< Models::VarInit > & wuPostVarInitialisers,
            const PostsynapticModels::Base * ps,
            const std::vector< double > & psParams,
            const std::vector< Models::VarInit > & psVarInitialisers,
            NeuronGroupInternal * srcNeuronGroup,
            NeuronGroupInternal * trgNeuronGroup,
            const InitSparseConnectivitySnippet::Init & connectivityInitialiser,
            VarLocation defaultVarLocation,
            VarLocation defaultExtraGlobalParamLocation,
            VarLocation defaultSparseConnectivityLocation ) [inline]
```

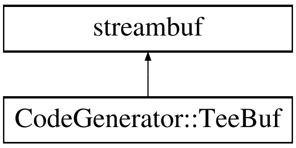
The documentation for this class was generated from the following file:

- synapseGroupInternal.h

19.67 CodeGenerator::TeeBuf Class Reference

```
#include <teeStream.h>
```

Inheritance diagram for CodeGenerator::TeeBuf:



Public Member Functions

- template<typename... T>
  TeeBuf (T &&... streamBufs)

19.67.1 Constructor & Destructor Documentation

19.67.1.1 TeeBuf()

```
template<typename... T>
CodeGenerator::TeeBuf::TeeBuf (
            T &&... streamBufs ) [inline]
```

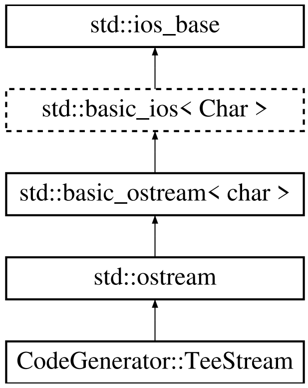
The documentation for this class was generated from the following file:

- teeStream.h

19.68 CodeGenerator::TeeStream Class Reference

```
#include <teeStream.h>
```

Inheritance diagram for CodeGenerator::TeeStream:



Public Member Functions

- template<typename... T>
  TeeStream (T &&... streamBufs)

19.68.1 Constructor & Destructor Documentation

19.68.1.1 TeeStream()

```
template<typename...  T>
CodeGenerator::TeeStream::TeeStream (
            T &&...  streamBufs )  [inline]
```

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

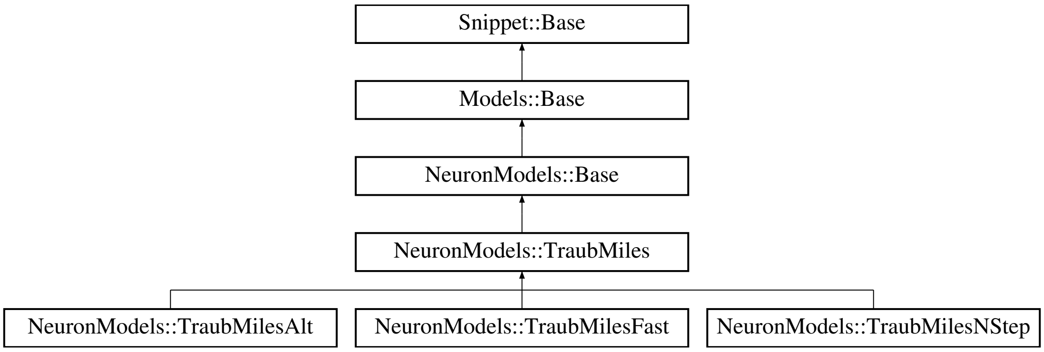
- teeStream.h

19.69 NeuronModels::TraubMiles Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMiles:



Public Types

- typedef Snippet::ValueBase< 7 > ParamValues
- typedef Models::VarInitContainerBase< 4 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

  Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual std::string getThresholdConditionCode () const override

  Gets code which defines the condition for a true spike in the described neuron model.
- virtual StringVec getParamNames () const override

  Gets names of of (independent) model parameters.
- virtual VarVec getVars () const override

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

Static Public Member Functions

- static const NeuronModels::TraubMiles * getInstance ()

Additional Inherited Members

19.69.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

This conductance based model has been taken from [7] and can be described by the equations:

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

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

$$\begin{aligned}
\alpha_n &= 0.032(-50 - V)/\big(\exp((-50 - V)/5) - 1\big) \\
\beta_n &= 0.5\exp((-55 - V)/40) \\
\alpha_m &= 0.32(-52 - V)/\big(\exp((-52 - V)/4) - 1\big)
\end{aligned}$$

$$\begin{aligned}
\beta_m &= 0.28(25+V)/\left(\exp((25+V)/5)-1\right) \\
\alpha_h &= 0.128\exp((-48-V)/18) \\
\beta_h &= 4/\left(\exp((-25-V)/5)+1\right).
\end{aligned}$$

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

It has 4 variables:

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

and 7 parameters:

- `gNa` - Na conductance in 1/(mOhms $*$ cm^2)
- `ENa` - Na equi potential in mV
- `gK` - K conductance in 1/(mOhms $*$ cm^2)
- `EK` - K equi potential in mV
- `gl` - Leak conductance in 1/(mOhms $*$ cm^2)
- `El` - Leak equi potential in mV
- `Cmem` - Membrane capacity density in muF/cm^2

Note

Internally, the ordinary differential equations defining the model are integrated with a linear Euler algorithm and GeNN integrates 25 internal time steps for each neuron for each network time step. I.e., if the network is simulated at `DT= 0.1` ms, then the neurons are integrated with a linear Euler algorithm with `lDT= 0.004` ms. This variant uses IF statements to check for a value at which a singularity would be hit. If so, value calculated by L'Hospital rule is used.

19.69.2 Member Typedef Documentation

19.69.2.1 ParamValues

`typedef Snippet::ValueBase< 7 > NeuronModels::TraubMiles::ParamValues`

19.69.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::TraubMiles::PostVarValues`

19.69.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::TraubMiles::PreVarValues`

19.69.2.4 VarValues

```
typedef Models::VarInitContainerBase< 4 > NeuronModels::TraubMiles::VarValues
```

19.69.3 Member Function Documentation

19.69.3.1 getInstance()

```
static const NeuronModels::TraubMiles* NeuronModels::TraubMiles::getInstance ( ) [inline],
[static]
```

19.69.3.2 getParamNames()

```
virtual StringVec NeuronModels::TraubMiles::getParamNames ( ) const [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

Reimplemented in NeuronModels::TraubMilesNStep.

19.69.3.3 getSimCode()

```
virtual std::string NeuronModels::TraubMiles::getSimCode ( ) const [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::Base.

Reimplemented in NeuronModels::TraubMilesNStep, NeuronModels::TraubMilesAlt, and NeuronModels::TraubMilesFast.

19.69.3.4 getThresholdConditionCode()

```
virtual std::string NeuronModels::TraubMiles::getThresholdConditionCode ( ) const [inline],
[override], [virtual]
```

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

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

Reimplemented from NeuronModels::Base.

19.69.3.5 getVars()

```
virtual VarVec NeuronModels::TraubMiles::getVars ( ) const [inline], [override], [virtual]
```

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

Reimplemented from Models::Base.

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

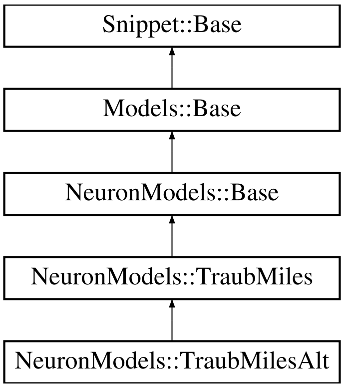
- neuronModels.h

19.70 NeuronModels::TraubMilesAlt Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMilesAlt:



Public Types

- typedef Snippet::ValueBase< 7 > ParamValues
- typedef Models::VarInitContainerBase< 4 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override
  Gets the code that defines the execution of one timestep of integration of the neuron model.

Static Public Member Functions

- static const NeuronModels::TraubMilesAlt ∗ getInstance ()

Additional Inherited Members

19.70.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

Using a workaround to avoid singularity: adding the munimum numerical value of the floating point precision used.

19.70.2 Member Typedef Documentation

19.70.2.1 ParamValues

`typedef Snippet::ValueBase< 7 > NeuronModels::TraubMilesAlt::ParamValues`

19.70.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesAlt::PostVarValues
```

19.70.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesAlt::PreVarValues
```

19.70.2.4 VarValues

```
typedef Models::VarInitContainerBase< 4 > NeuronModels::TraubMilesAlt::VarValues
```

19.70.3 Member Function Documentation

19.70.3.1 getInstance()

```
static const NeuronModels::TraubMilesAlt* NeuronModels::TraubMilesAlt::getInstance ( ) [inline],
[static]
```

19.70.3.2 getSimCode()

```
virtual std::string NeuronModels::TraubMilesAlt::getSimCode ( ) const [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::TraubMiles.

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

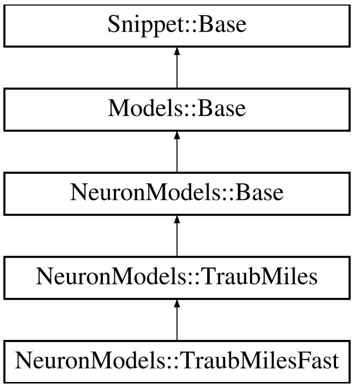
- neuronModels.h

19.71 NeuronModels::TraubMilesFast Class Reference

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

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMilesFast:



Public Types

- typedef Snippet::ValueBase< 7 > ParamValues
- typedef Models::VarInitContainerBase< 4 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

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

Static Public Member Functions

- static const NeuronModels::TraubMilesFast ∗ getInstance ()

Additional Inherited Members

19.71.1 Detailed Description

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

There are singularities in this model, which can be easily hit in float precision

19.71.2 Member Typedef Documentation

19.71.2.1 ParamValues

```
typedef Snippet::ValueBase< 7 > NeuronModels::TraubMilesFast::ParamValues
```

19.71.2.2 PostVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesFast::PostVarValues
```

19.71.2.3 PreVarValues

```
typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesFast::PreVarValues
```

19.71.2.4 VarValues

```
typedef Models::VarInitContainerBase< 4 > NeuronModels::TraubMilesFast::VarValues
```

19.71.3 Member Function Documentation

19.71.3.1 getInstance()

```
static const NeuronModels::TraubMilesFast* NeuronModels::TraubMilesFast::getInstance ( ) [inline],
[static]
```

19.71.3.2 getSimCode()

```
virtual std::string NeuronModels::TraubMilesFast::getSimCode ( ) const [inline], [override],
[virtual]
```

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

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

Reimplemented from NeuronModels::TraubMiles.

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

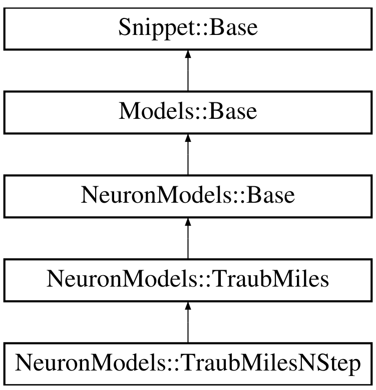
- neuronModels.h

19.72 NeuronModels::TraubMilesNStep Class Reference

Hodgkin-Huxley neurons with Traub & Miles algorithm.

```
#include <neuronModels.h>
```

Inheritance diagram for NeuronModels::TraubMilesNStep:



Public Types

- typedef Snippet::ValueBase< 8 > ParamValues

- typedef Models::VarInitContainerBase< 4 > VarValues
- typedef Models::VarInitContainerBase< 0 > PreVarValues
- typedef Models::VarInitContainerBase< 0 > PostVarValues

Public Member Functions

- virtual std::string getSimCode () const override

    Gets the code that defines the execution of one timestep of integration of the neuron model.
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Static Public Member Functions

- static const NeuronModels::TraubMilesNStep ∗ getInstance ()

Additional Inherited Members

19.72.1 Detailed Description

Hodgkin-Huxley neurons with Traub & Miles algorithm.

Same as standard TraubMiles model but number of inner loops can be set using a parameter

19.72.2 Member Typedef Documentation

19.72.2.1 ParamValues

`typedef Snippet::ValueBase< 8 > NeuronModels::TraubMilesNStep::ParamValues`

19.72.2.2 PostVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesNStep::PostVarValues`

19.72.2.3 PreVarValues

`typedef Models::VarInitContainerBase<0> NeuronModels::TraubMilesNStep::PreVarValues`

19.72.2.4 VarValues

`typedef Models::VarInitContainerBase< 4 > NeuronModels::TraubMilesNStep::VarValues`

19.72.3 Member Function Documentation

19.72.3.1 getInstance()

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

19.72.3.2 getParamNames()

`virtual StringVec NeuronModels::TraubMilesNStep::getParamNames ( ) const [inline], [override], [virtual]`

Gets names of of (independent) model parameters.

Reimplemented from NeuronModels::TraubMiles.

19.72.3.3 getSimCode()

`virtual std::string NeuronModels::TraubMilesNStep::getSimCode ( ) const [inline], [override], [virtual]`

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

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

Reimplemented from NeuronModels::TraubMiles.

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

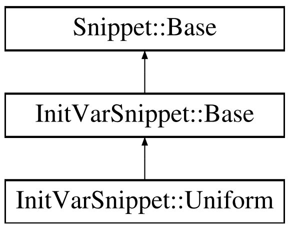
- neuronModels.h

19.73 InitVarSnippet::Uniform Class Reference

Initialises variable by sampling from the uniform distribution.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Uniform:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Uniform, 2)
- SET_CODE ("const scalar scale = $(max) - $(min);\ "$(value)=$(min)+($(gennrand_uniform) ∗scale);")
- virtual StringVec getParamNames () const override

    Gets names of of (independent) model parameters.

Additional Inherited Members

19.73.1 Detailed Description

Initialises variable by sampling from the uniform distribution.

This snippet takes 2 parameters:

- `min` - The minimum value
- `max` - The maximum value

19.73.2 Member Function Documentation

19.73.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Uniform::DECLARE_SNIPPET (
              InitVarSnippet::Uniform ,
              2 )
```

19.73.2.2 getParamNames()

```
virtual StringVec InitVarSnippet::Uniform::getParamNames ( ) const  [inline], [override],
[virtual]
```

Gets names of of (independent) model parameters.

Reimplemented from Snippet::Base.

19.73.2.3 SET_CODE()

```
InitVarSnippet::Uniform::SET_CODE (
              "const scalar scale = $(max) - $(min);\ "$(value)=$(min)+($(gennrand_uniform) *scale);"
)
```

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

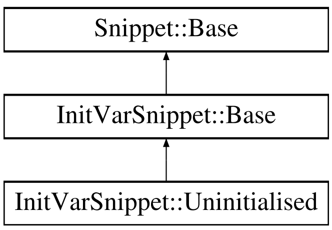
- initVarSnippet.h

19.74 InitVarSnippet::Uninitialised Class Reference

Used to mark variables as uninitialised - no initialisation code will be run.

```
#include <initVarSnippet.h>
```

Inheritance diagram for InitVarSnippet::Uninitialised:



Public Member Functions

- DECLARE_SNIPPET (InitVarSnippet::Uninitialised, 0)

Additional Inherited Members

19.74.1 Detailed Description

Used to mark variables as uninitialised - no initialisation code will be run.

19.74.2 Member Function Documentation

19.74.2.1 DECLARE_SNIPPET()

```
InitVarSnippet::Uninitialised::DECLARE_SNIPPET (
            InitVarSnippet::Uninitialised ,
            0 )
```

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

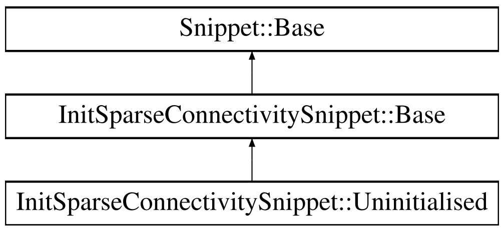
- initVarSnippet.h

19.75 InitSparseConnectivitySnippet::Uninitialised Class Reference

Used to mark connectivity as uninitialised - no initialisation code will be run.

```
#include <initSparseConnectivitySnippet.h>
```

Inheritance diagram for InitSparseConnectivitySnippet::Uninitialised:



Public Member Functions

- DECLARE_SNIPPET (InitSparseConnectivitySnippet::Uninitialised, 0)

Additional Inherited Members

19.75.1 Detailed Description

Used to mark connectivity as uninitialised - no initialisation code will be run.

19.75.2 Member Function Documentation

19.75.2.1 DECLARE_SNIPPET()

```
InitSparseConnectivitySnippet::Uninitialised::DECLARE_SNIPPET (
            InitSparseConnectivitySnippet::Uninitialised ,
            0 )
```

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

- initSparseConnectivitySnippet.h

19.76 Snippet::ValueBase< NumVars > Class Template Reference

```
#include <snippet.h>
```

Public Member Functions

- template<typename... T>
  ValueBase (T &&... vals)
- const std::vector< double > & getValues () const
  Gets values as a vector of doubles.
- double operator[] (size_t pos) const

19.76.1 Constructor & Destructor Documentation

19.76.1.1 ValueBase()

```
template<size_t NumVars>
template<typename...  T>
Snippet::ValueBase< NumVars >::ValueBase (
            T &&...   vals )   [inline]
```

19.76.2 Member Function Documentation

19.76.2.1 getValues()

```
template<size_t NumVars>
const std::vector<double>& Snippet::ValueBase< NumVars >::getValues ( ) const  [inline]
```

Gets values as a vector of doubles.

19.76.2.2 operator[]()

```
template<size_t NumVars>
double Snippet::ValueBase< NumVars >::operator[] (
            size_t pos ) const  [inline]
```

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

- snippet.h

19.77 Snippet::ValueBase< 0 > Class Template Reference

```
#include <snippet.h>
```

Public Member Functions

- template<typename... T>
  ValueBase (T &&... vals)
- std::vector< double > getValues () const
  Gets values as a vector of doubles.

19.77.1 Detailed Description

template<>
class Snippet::ValueBase< 0 >

Template specialisation of ValueBase to avoid compiler warnings in the case when a model requires no parameters or state variables

19.77.2 Constructor & Destructor Documentation

19.77.2.1 ValueBase()

```
template<typename... T>
Snippet::ValueBase< 0 >::ValueBase (
            T &&... vals )  [inline]
```

19.77.3 Member Function Documentation

19.77.3.1 getValues()

```
std::vector<double> Snippet::ValueBase< 0 >::getValues ( ) const  [inline]
```

Gets values as a vector of doubles.

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

- snippet.h

19.78 Snippet::Base::Var Struct Reference

A variable has a name and a type.

```
#include <snippet.h>
```

Public Attributes

- std::string name
- std::string type

19.78.1 Detailed Description

A variable has a name and a type.

19.78.2 Member Data Documentation

19.78.2.1 name

```
std::string Snippet::Base::Var::name
```

19.78.2.2 type

```
std::string Snippet::Base::Var::type
```

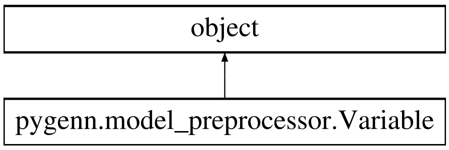
The documentation for this struct was generated from the following file:

- snippet.h

19.79 pygenn.model_preprocessor.Variable Class Reference

Class holding information about GeNN variables.

Inheritance diagram for pygenn.model_preprocessor.Variable:



Public Member Functions

- def __init__ (self, variable_name, variable_type, values=None)

    Init Variable.
- def set_values (self, values)

    Set Variable's values.

Public Attributes

- name
- type
- view
- needs_allocation
- init_required
- init_val
- values

19.79.1 Detailed Description

Class holding information about GeNN variables.

19.79.2 Constructor & Destructor Documentation

19.79.2.1 __init__()

```
def pygenn.model_preprocessor.Variable.__init__ (
                self,
                variable_name,
                variable_type,
                values = None )
```

Init Variable.

Parameters

variable_name	string name of the variable
variable_type	string type of the variable
values	iterable, single value or VarInit instance

19.79.3 Member Function Documentation

19.79.3.1 set_values()

```
def pygenn.model_preprocessor.Variable.set_values (
              self,
              values )
```

Set Variable's values.

Parameters

values	iterable, single value or VarInit instance

19.79.4 Member Data Documentation

19.79.4.1 init_required

```
pygenn.model_preprocessor.Variable.init_required
```

19.79.4.2 init_val

```
pygenn.model_preprocessor.Variable.init_val
```

19.79.4.3 name

```
pygenn.model_preprocessor.Variable.name
```

19.79.4.4 needs_allocation

```
pygenn.model_preprocessor.Variable.needs_allocation
```

19.79.4.5 type

```
pygenn.model_preprocessor.Variable.type
```

19.79.4.6 values

```
pygenn.model_preprocessor.Variable.values
```

19.79.4.7 view

```
pygenn.model_preprocessor.Variable.view
```

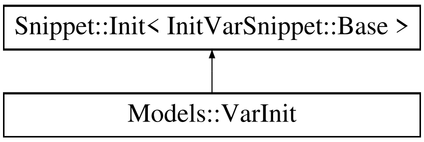
The documentation for this class was generated from the following file:

- model_preprocessor.py

19.80 Models::VarInit Class Reference

```
#include <models.h>
```

Inheritance diagram for Models::VarInit:



Public Member Functions

- VarInit (const InitVarSnippet::Base *snippet, const std::vector< double > &params)
- VarInit (double constant)

19.80.1 Detailed Description

Class used to bind together everything required to initialise a variable:

1. A pointer to a variable initialisation snippet
2. The parameters required to control the variable initialisation snippet

19.80.2 Constructor & Destructor Documentation

19.80.2.1 VarInit() [1/2]

```
Models::VarInit::VarInit (
            const InitVarSnippet::Base * snippet,
            const std::vector< double > & params )  [inline]
```

19.80.2.2 VarInit() [2/2]

```
Models::VarInit::VarInit (
            double constant )  [inline]
```

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

- models.h

19.81 Models::VarInitContainerBase< NumVars > Class Template Reference

```
#include <models.h>
```

Public Member Functions

- template<typename... T>
  VarInitContainerBase (T &&... initialisers)
- const std::vector< VarInit > & getInitialisers () const

  Gets initialisers as a vector of Values.
- const VarInit & operator[] (size_t pos) const

19.81.1 Detailed Description

template<size_t NumVars>
class Models::VarInitContainerBase< NumVars >

Wrapper to ensure at compile time that correct number of value initialisers are used when specifying the values of a model's initial state.

19.81.2 Constructor & Destructor Documentation

19.81.2.1 VarInitContainerBase()

```
template<size_t NumVars>
template<typename... T>
Models::VarInitContainerBase< NumVars >::VarInitContainerBase (
            T &&... initialisers ) [inline]
```

19.81.3 Member Function Documentation

19.81.3.1 getInitialisers()

```
template<size_t NumVars>
const std::vector<VarInit>& Models::VarInitContainerBase< NumVars >::getInitialisers ( )
const [inline]
```

Gets initialisers as a vector of Values.

19.81.3.2 operator[]()

```
template<size_t NumVars>
const VarInit& Models::VarInitContainerBase< NumVars >::operator[] (
            size_t pos ) const [inline]
```

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

- models.h

19.82 Models::VarInitContainerBase< 0 > Class Template Reference

```
#include <models.h>
```

Public Member Functions

- template<typename... T>
  VarInitContainerBase (T &&... initialisers)
- VarInitContainerBase (const Snippet::ValueBase< 0 > &)
- std::vector< VarInit > getInitialisers () const

  Gets initialisers as a vector of Values.

19.82.1 Detailed Description

template<>
class Models::VarInitContainerBase< 0 >

Template specialisation of ValueInitBase to avoid compiler warnings in the case when a model requires no variable initialisers

19.82.2 Constructor & Destructor Documentation

19.82.2.1 VarInitContainerBase() [1/2]

```
template<typename... T>
Models::VarInitContainerBase< 0 >::VarInitContainerBase (
            T &&... initialisers ) [inline]
```

19.82.2.2 VarInitContainerBase() [2/2]

```
Models::VarInitContainerBase< 0 >::VarInitContainerBase (
            const Snippet::ValueBase< 0 > & ) [inline]
```

19.82.3 Member Function Documentation

19.82.3.1 getInitialisers()

```
std::vector<VarInit> Models::VarInitContainerBase< 0 >::getInitialisers ( ) const [inline]
```

Gets initialisers as a vector of Values.

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

- models.h

20 File Documentation

20.1 00_MainPage.dox File Reference

20.2 01_Installation.dox File Reference

20.3 02_Quickstart.dox File Reference

20.4 03_Examples.dox File Reference

20.5 05_SpineML.dox File Reference

20.6 06_Brian2GeNN.dox File Reference

20.7 07_PyGeNN.dox File Reference

20.8 09_ReleaseNotes.dox File Reference

20.9 10_UserManual.dox File Reference

20.10 11_Tutorial.dox File Reference

20.11 12_Tutorial.dox File Reference

20.12 13_UserGuide.dox File Reference

20.13 14_Credits.dox File Reference

20.14 __init__.py File Reference

Namespaces

- pygenn

20.15 backend.cc File Reference

```
#include "backend.h"
#include <algorithm>
#include <plog/Log.h>
#include "gennUtils.h"
#include "modelSpecInternal.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
#include "code_generator/codeGenUtils.h"
#include "utils.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::CUDA

20.16 backend.cc File Reference

```
#include "backend.h"
#include "gennUtils.h"
```

```
#include "modelSpecInternal.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
#include "code_generator/codeGenUtils.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::SingleThreadedCPU

20.17 backend.h File Reference

```
#include <algorithm>
#include <array>
#include <functional>
#include <map>
#include <string>
#include <cuda.h>
#include <cuda_runtime.h>
#include "backendExport.h"
#include "code_generator/backendBase.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
```

Classes

- struct CodeGenerator::CUDA::Preferences

  Preferences for CUDA backend.
- class CodeGenerator::CUDA::Backend

Namespaces

- filesystem
- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::CUDA

Typedefs

- using CodeGenerator::CUDA::KernelBlockSize = std::array< size_t, KernelMax >

  Array of block sizes for each kernel.

Enumerations

- enum CodeGenerator::CUDA::DeviceSelect { CodeGenerator::CUDA::DeviceSelect::OPTIMAL, Code←Generator::CUDA::DeviceSelect::MOST_MEMORY, CodeGenerator::CUDA::DeviceSelect::MANUAL }

  Methods for selecting CUDA device.
- enum CodeGenerator::CUDA::BlockSizeSelect { CodeGenerator::CUDA::BlockSizeSelect::OCCUPANCY, CodeGenerator::CUDA::BlockSizeSelect::MANUAL }

Methods for selecting CUDA kernel block size.

- enum CodeGenerator::CUDA::Kernel {
CodeGenerator::CUDA::KernelNeuronUpdate, CodeGenerator::CUDA::KernelPresynapticUpdate, CodeGenerator::CUDA::KernelPostsynapticUpdate, CodeGenerator::CUDA::KernelSynapseDynamicsUpdate, CodeGenerator::CUDA::KernelInitialize, CodeGenerator::CUDA::KernelInitializeSparse, CodeGenerator::CUDA::KernelPreNeuronReset, CodeGenerator::CUDA::KernelPreSynapseReset, CodeGenerator::CUDA::KernelMax }

  Kernels generated by CUDA backend.

20.18 backend.h File Reference

```
#include <functional>
#include <map>
#include <string>
#include "backendExport.h"
#include "code_generator/backendBase.h"
```

Classes

- struct CodeGenerator::SingleThreadedCPU::Preferences
- class CodeGenerator::SingleThreadedCPU::Backend

Namespaces

- filesystem
- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::SingleThreadedCPU

20.19 backendBase.cc File Reference

```
#include "code_generator/backendBase.h"
#include <plog/Log.h>
#include "gennUtils.h"
```

Macros

- #define TYPE(T) {#T, sizeof(T)}

20.19.1 Macro Definition Documentation

20.19.1.1 TYPE

```
#define TYPE(
            T ) {#T, sizeof(T)}
```

20.20 backendBase.h File Reference

```
#include <functional>
#include <map>
#include <string>
#include <unordered_map>
#include <vector>
#include <plog/Severity.h>
#include "codeStream.h"
#include "gennExport.h"
#include "variableMode.h"
```

Classes

- struct CodeGenerator::PreferencesBase

  Base class for backend preferences - can be accessed via a global in 'classic' C++ code generator.
- class CodeGenerator::MemAlloc
- class CodeGenerator::BackendBase

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

20.21 backendExport.h File Reference

Macros

- #define BACKEND_EXPORT

20.21.1 Macro Definition Documentation

20.21.1.1 BACKEND_EXPORT

```
#define BACKEND_EXPORT
```

20.22 binomial.cc File Reference

```
#include "binomial.h"
#include <stdexcept>
#include <cassert>
#include <cmath>
#include <cstdint>
```

Functions

- unsigned int binomialInverseCDF (double cdf, unsigned int n, double p)

20.22.1 Function Documentation

20.22.1.1 binomialInverseCDF()

```
unsigned int binomialInverseCDF (
            double cdf,
            unsigned int n,
            double p )
```

20.23 binomial.h File Reference

```
#include "gennExport.h"
```

Functions

- GENN_EXPORT unsigned int binomialInverseCDF (double cdf, unsigned int n, double p)

20.23.1 Function Documentation

20.23.1.1 binomialInverseCDF()

```
GENN_EXPORT unsigned int binomialInverseCDF (
            double cdf,
            unsigned int n,
            double p )
```

20.24 codeGenUtils.cc File Reference

```
#include "code_generator/codeGenUtils.h"
#include <regex>
#include <cstring>
#include "modelSpec.h"
```

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Enumerations

- enum MathsFunc

Functions

- void CodeGenerator::substitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting strings in the neuron code strings or other templates.
- bool CodeGenerator::regexVarSubstitute (std::string &s, const std::string &trg, const std::string &rep)

Tool for substituting variable names in the neuron code strings or other templates using regular expressions.

- bool CodeGenerator::regexFuncSubstitute (std::string &s, const std::string &trg, const std::string &rep)

  Tool for substituting function names in the neuron code strings or other templates using regular expressions.
- void CodeGenerator::functionSubstitute (std::string &code, const std::string &funcName, unsigned int num↩ Params, const std::string &replaceFuncTemplate)

  This function substitutes function calls in the form:
- std::string CodeGenerator::ensureFtype (const std::string &oldcode, const std::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 CodeGenerator::checkUnreplacedVariables (const std::string &code, const std::string &codeName)

  This function checks for unknown variable definitions and returns a gennError if any are found.
- void CodeGenerator::preNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroup↩ Internal &sg, const std::string &offset, const std::string &axonalDelayOffset, const std::string &postIdx, const std::string &devPrefix, const std::string &preVarPrefix="", const std::string &preVarSuffix="")

  suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
- void CodeGenerator::postNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroup↩ Internal &sg, const std::string &offset, const std::string &backPropDelayOffset, const std::string &preIdx, const std::string &devPrefix, const std::string &postVarPrefix="", const std::string &postVarSuffix="")

  suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
- void CodeGenerator::neuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &preIdx, const std::string &postIdx, const std::string &devPrefix, double dt, const std↩ ::string &preVarPrefix="", const std::string &preVarSuffix="", const std::string &postVarPrefix="", const std↩ ::string &postVarSuffix="")

  Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

20.24.1 Enumeration Type Documentation

20.24.1.1 MathsFunc

```
enum MathsFunc
```

20.25 codeGenUtils.h File Reference

```
#include <iomanip>
#include <limits>
#include <string>
#include <sstream>
#include <vector>
#include "snippet.h"
#include "variableMode.h"
```

Classes

- struct CodeGenerator::FunctionTemplate
- class CodeGenerator::StructNameConstIter< BaseIter >

  Custom iterator for iterating through the containers of structs with 'name' members.
- struct CodeGenerator::NameIterCtx< Container >

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Typedefs

- typedef NameIterCtx< Snippet::Base::VarVec > CodeGenerator::VarNameIterCtx
- typedef NameIterCtx< Snippet::Base::DerivedParamVec > CodeGenerator::DerivedParamNameIterCtx
- typedef NameIterCtx< Snippet::Base::ParamValVec > CodeGenerator::ParamValIterCtx

Functions

- void CodeGenerator::substitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting strings in the neuron code strings or other templates.
- bool CodeGenerator::regexVarSubstitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting variable names in the neuron code strings or other templates using regular expressions.
- bool CodeGenerator::regexFuncSubstitute (std::string &s, const std::string &trg, const std::string &rep)

    Tool for substituting function names in the neuron code strings or other templates using regular expressions.
- void CodeGenerator::functionSubstitute (std::string &code, const std::string &funcName, unsigned int numParams, const std::string &replaceFuncTemplate)

    This function substitutes function calls in the form:
- template< typename NameIter >
  void CodeGenerator::name_substitutions (std::string &code, const std::string &prefix, NameIter namesBegin, NameIter namesEnd, const std::string &postfix="", const std::string &ext="")

    This function performs a list of name substitutions for variables in code snippets.
- void CodeGenerator::name_substitutions (std::string &code, const std::string &prefix, const std::vector< std::string > &names, const std::string &postfix="", const std::string &ext="")

    This function performs a list of name substitutions for variables in code snippets.
- template< class T , typename std::enable_if< std::is_floating_point< T >::value >::type * = nullptr>
  void CodeGenerator::writePreciseString (std::ostream &os, T value)

    This function writes a floating point value to a stream -setting the precision so no digits are lost.
- template< class T , typename std::enable_if< std::is_floating_point< T >::value >::type * = nullptr>
  std::string CodeGenerator::writePreciseString (T value)

    This function writes a floating point value to a string - setting the precision so no digits are lost.
- template< typename NameIter >
  void CodeGenerator::value_substitutions (std::string &code, NameIter namesBegin, NameIter namesEnd, const std::vector< double > &values, const std::string &ext="")

    This function performs a list of value substitutions for parameters in code snippets.
- void CodeGenerator::value_substitutions (std::string &code, const std::vector< std::string > &names, const std::vector< double > &values, const std::string &ext="")

    This function performs a list of value substitutions for parameters in code snippets.
- std::string CodeGenerator::ensureFtype (const std::string &oldcode, const std::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 CodeGenerator::checkUnreplacedVariables (const std::string &code, const std::string &codeName)

    This function checks for unknown variable definitions and returns a gennError if any are found.
- void CodeGenerator::preNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &offset, const std::string &axonalDelayOffset, const std::string &postIdx, const std::string &devPrefix, const std::string &preVarPrefix="", const std::string &preVarSuffix="")

    suffix to be used for presynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)

- void CodeGenerator::postNeuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &offset, const std::string &backPropDelayOffset, const std::string &preIdx, const std::string &devPrefix, const std::string &postVarPrefix="", const std::string &postVarSuffix="")

    suffix to be used for postsynaptic variable accesses - typically combined with prefix to wrap in function call such as __ldg(&XXX)
- void CodeGenerator::neuronSubstitutionsInSynapticCode (std::string &wCode, const SynapseGroupInternal &sg, const std::string &preIdx, const std::string &postIdx, const std::string &devPrefix, double dt, const std::string &preVarPrefix="", const std::string &preVarSuffix="", const std::string &postVarPrefix="", const std::string &postVarSuffix="")

    Function for performing the code and value substitutions necessary to insert neuron related variables, parameters, and extraGlobal parameters into synaptic code.

20.26 codeStream.cc File Reference

```
#include "code_generator/codeStream.h"
#include <algorithm>
```

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- GENN_EXPORT std::ostream & CodeGenerator::operator<< (std::ostream &s, const CodeStream::OB &ob)
- GENN_EXPORT std::ostream & CodeGenerator::operator<< (std::ostream &s, const CodeStream::CB &cb)

20.27 codeStream.h File Reference

```
#include <ostream>
#include <streambuf>
#include <string>
#include <vector>
#include <plog/Log.h>
#include "gennExport.h"
```

Classes

- class CodeGenerator::CodeStream
- struct CodeGenerator::CodeStream::OB

    An open bracket marker.
- struct CodeGenerator::CodeStream::CB

    A close bracket marker.
- class CodeGenerator::CodeStream::Scope

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- GENN_EXPORT std::ostream & CodeGenerator::operator<< (std::ostream &s, const CodeStream::OB &ob)
- GENN_EXPORT std::ostream & CodeGenerator::operator<< (std::ostream &s, const CodeStream::CB &cb)

20.28 currentSource.cc File Reference

```
#include "currentSource.h"
#include <algorithm>
#include <cmath>
#include "gennUtils.h"
```

20.29 currentSource.h File Reference

```
#include <map>
#include <set>
#include <string>
#include <vector>
#include "currentSourceModels.h"
#include "gennExport.h"
#include "variableMode.h"
```

Classes

- class CurrentSource

20.30 currentSourceInternal.h File Reference

```
#include "currentSource.h"
```

Classes

- class CurrentSourceInternal

20.31 currentSourceModels.cc File Reference

```
#include "currentSourceModels.h"
```

Functions

- IMPLEMENT_MODEL (CurrentSourceModels::DC)
- IMPLEMENT_MODEL (CurrentSourceModels::GaussianNoise)

20.31.1 Function Documentation

20.31.1.1 IMPLEMENT_MODEL() [1/2]

```
IMPLEMENT_MODEL (
            CurrentSourceModels::DC  )
```

20.31.1.2 IMPLEMENT_MODEL() [2/2]

```
IMPLEMENT_MODEL (
            CurrentSourceModels::GaussianNoise  )
```

20.32 currentSourceModels.h File Reference

```
#include <array>
#include <functional>
#include <string>
#include <tuple>
#include <vector>
#include <cmath>
#include "gennExport.h"
#include "models.h"
```

Classes

- class CurrentSourceModels::Base

  Base class for all current source models.
- class CurrentSourceModels::DC

  DC source.
- class CurrentSourceModels::GaussianNoise

  Noisy current source with noise drawn from normal distribution.

Namespaces

- CurrentSourceModels

Macros

- #define SET_INJECTION_CODE(INJECTION_CODE) virtual std::string getInjectionCode() const override{ return INJECTION_CODE; }

20.32.1 Macro Definition Documentation

20.32.1.1 SET_INJECTION_CODE

```
#define SET_INJECTION_CODE(
             INJECTION_CODE ) virtual std::string getInjectionCode() const override{ return
INJECTION_CODE; }
```

20.33 generateAll.cc File Reference

```
#include "code_generator/generateAll.h"
#include <fstream>
#include <string>
#include <vector>
#include <plog/Log.h>
#include "path.h"
#include "code_generator/codeStream.h"
#include "code_generator/generateInit.h"
#include "code_generator/generateMPI.h"
#include "code_generator/generateNeuronUpdate.h"
#include "code_generator/generateSupportCode.h"
#include "code_generator/generateSynapseUpdate.h"
#include "code_generator/generateRunner.h"
```

20.34 generateAll.h File Reference

```
#include <string>
#include <vector>
#include "gennExport.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- filesystem

Functions

- GENN_EXPORT std::vector< std::string > CodeGenerator::generateAll (const ModelSpecInternal &model, const BackendBase &backend, const filesystem::path &outputPath, bool standaloneModules=false)

20.35 generateInit.cc File Reference

```
#include "code_generator/generateInit.h"
#include <string>
#include "models.h"
#include "modelSpecInternal.h"
#include "code_generator/codeGenUtils.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
#include "code_generator/backendBase.h"
```

20.36 generateInit.h File Reference

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void CodeGenerator::generateInit (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

20.37 generateMakefile.cc File Reference

```
#include "code_generator/generateMakefile.h"
#include <string>
#include "modelSpec.h"
#include "code_generator/backendBase.h"
```

20.38 generateMakefile.h File Reference

```
#include <string>
#include <vector>
#include "gennExport.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void GENN_EXPORT CodeGenerator::generateMakefile (std::ostream &os, const BackendBase &backend, const std::vector< std::string > &moduleNames)

20.39 generateMPI.cc File Reference

Contains functions to generate code for running the simulation with MPI. Part of the code generation section.

```
#include "code_generator/generateMPI.h"
#include <fstream>
#include <cstring>
#include "modelSpecInternal.h"
#include "code_generator/backendBase.h"
#include "code_generator/codeStream.h"
```

20.39.1 Detailed Description

Contains functions to generate code for running the simulation with MPI. Part of the code generation section.

20.40 generateMPI.h File Reference

Contains functions to generate code for running the simulation with MPI. Part of the code generation section.

```
#include <string>
#include "gennExport.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void GENN_EXPORT CodeGenerator::generateMPI (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

  A function that generates predominantly MPI infrastructure code.

20.40.1 Detailed Description

Contains functions to generate code for running the simulation with MPI. Part of the code generation section.

20.41 generateMSBuild.cc File Reference

```
#include "code_generator/generateMSBuild.h"
#include <string>
#include "code_generator/backendBase.h"
```

20.42 generateMSBuild.h File Reference

```
#include <string>
#include <vector>
#include "gennExport.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void GENN_EXPORT CodeGenerator::generateMSBuild (std::ostream &os, const BackendBase &backend, const std::string &projectGUID, const std::vector< std::string > &moduleNames)

20.43 generateNeuronUpdate.cc File Reference

```
#include "code_generator/generateNeuronUpdate.h"
#include <iostream>
#include <string>
#include <plog/Log.h>
#include "models.h"
#include "modelSpecInternal.h"
#include "code_generator/codeGenUtils.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
#include "code_generator/backendBase.h"
```

20.44 generateNeuronUpdate.h File Reference

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void CodeGenerator::generateNeuronUpdate (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

20.45 generateRunner.cc File Reference

```
#include "code_generator/generateRunner.h"
#include <sstream>
#include <string>
#include "gennUtils.h"
#include "modelSpecInternal.h"
#include "code_generator/codeGenUtils.h"
#include "code_generator/codeStream.h"
#include "code_generator/teeStream.h"
#include "code_generator/backendBase.h"
```

20.46 generateRunner.h File Reference

```
#include "code_generator/backendBase.h"
```

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- MemAlloc CodeGenerator::generateRunner (CodeStream &definitions, CodeStream &definitionsInternal, CodeStream &runner, const ModelSpecInternal &model, const BackendBase &backend, int localHostID)

20.47 generateSupportCode.cc File Reference

```
#include "code_generator/generateSupportCode.h"
#include <string>
#include "modelSpecInternal.h"
#include "code_generator/codeGenUtils.h"
#include "code_generator/codeStream.h"
```

20.48 generateSupportCode.h File Reference

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void CodeGenerator::generateSupportCode (CodeStream &os, const ModelSpecInternal &model)

20.49 generateSynapseUpdate.cc File Reference

```
#include "code_generator/generateSynapseUpdate.h"
#include <string>
#include "modelSpecInternal.h"
#include "code_generator/codeStream.h"
#include "code_generator/substitutions.h"
#include "code_generator/backendBase.h"
```

20.50 generateSynapseUpdate.h File Reference

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

Functions

- void CodeGenerator::generateSynapseUpdate (CodeStream &os, const ModelSpecInternal &model, const BackendBase &backend, bool standaloneModules)

20.51 generator.cc File Reference

```
#include <fstream>
#include <plog/Log.h>
#include <plog/Appenders/ConsoleAppender.h>
#include "path.h"
#include "modelSpecInternal.h"
#include "code_generator/generateAll.h"
#include "code_generator/generateMakefile.h"
#include "code_generator/generateMSBuild.h"
#include "optimiser.h"
#include <MODEL>
```

Functions

- int main (int argc, char ∗argv[])

Variables

- Preferences GENN_PREFERENCES

20.51.1 Function Documentation

20.51.1.1 main()

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

Parameters

argc	number of arguments; expected to be 2
argv	Arguments; expected to contain the target directory for code generation.

20.51.2 Variable Documentation

20.51.2.1 GENN_PREFERENCES

```
Preferences GENN_PREFERENCES
```

20.52 genn_groups.py File Reference

Classes

- class pygenn.genn_groups.Group

  Parent class of NeuronGroup, SynapseGroup and CurrentSource.
- class pygenn.genn_groups.NeuronGroup

  Class representing a group of neurons.
- class pygenn.genn_groups.SynapseGroup

  Class representing synaptic connection between two groups of neurons.
- class pygenn.genn_groups.CurrentSource

  Class representing a current injection into a group of neurons.

Namespaces

- pygenn.genn_groups

Variables

- pygenn.genn_groups.xrange = range

  GeNNGroups This module provides classes which automatize model checks and parameter convesions for GeNN Groups.

20.53 genn_model.py File Reference

Classes

- class pygenn.genn_model.GeNNModel

    GeNNModel class This class helps to define, build and run a GeNN model from python.

Namespaces

- pygenn.genn_model

Functions

- def pygenn.genn_model.init_var (init_var_snippet, param_space)

    This helper function creates a VarInit object to easily initialise a variable using a snippet.
- def pygenn.genn_model.init_connectivity (init_sparse_connect_snippet, param_space)

    This helper function creates a InitSparseConnectivitySnippet::Init object to easily initialise connectivity using a snippet.
- def pygenn.genn_model.create_custom_neuron_class (class_name, param_names=None, var_name_types=None, derived_params=None, sim_code=None, threshold_condition_code=None, reset_code=None, support_code=None, extra_global_params=None, additional_input_vars=None, is_auto_refractory_required=None, custom_body=None)

    This helper function creates a custom NeuronModel class.
- def pygenn.genn_model.create_custom_postsynaptic_class (class_name, param_names=None, var_name_types=None, derived_params=None, decay_code=None, apply_input_code=None, support_code=None, custom_body=None)

    This helper function creates a custom PostsynapticModel class.
- def pygenn.genn_model.create_custom_weight_update_class (class_name, param_names=None, var_name_types=None, pre_var_name_types=None, post_var_name_types=None, derived_params=None, sim_code=None, event_code=None, learn_post_code=None, synapse_dynamics_code=None, event_threshold_condition_code=None, pre_spike_code=None, post_spike_code=None, sim_support_code=None, learn_post_support_code=None, synapse_dynamics_suppport_code=None, extra_global_params=None, is_pre_spike_time_required=None, is_post_spike_time_required=None, custom_body=None)

    This helper function creates a custom WeightUpdateModel class.
- def pygenn.genn_model.create_custom_current_source_class (class_name, param_names=None, var_name_types=None, derived_params=None, injection_code=None, extra_global_params=None, custom_body=None)

    This helper function creates a custom NeuronModel class.
- def pygenn.genn_model.create_custom_model_class (class_name, base, param_names, var_name_types, derived_params, custom_body)

    This helper function completes a custom model class creation.
- def pygenn.genn_model.create_dpf_class (dp_func)

    Helper function to create derived parameter function class.
- def pygenn.genn_model.create_cmlf_class (cml_func)

    Helper function to create function class for calculating sizes of matrices initialised with sparse connectivity initialisation snippet.
- def pygenn.genn_model.create_custom_init_var_snippet_class (class_name, param_names=None, derived_params=None, var_init_code=None, custom_body=None)

    This helper function creates a custom InitVarSnippet class.
- def pygenn.genn_model.create_custom_sparse_connect_init_snippet_class (class_name, param_names=None, derived_params=None, row_build_code=None, row_build_state_vars=None, calc_max_row_len_func=None, calc_max_col_len_func=None, extra_global_params=None, custom_body=None)

    This helper function creates a custom InitSparseConnectivitySnippet class.

Variables

- pygenn.genn_model.backend_modules = OrderedDict()
- pygenn.genn_model.m = import_module(".genn_wrapper." + b + "Backend", "pygenn")

20.54 gennExport.h File Reference

Macros

- #define GENN_EXPORT

20.54.1 Macro Definition Documentation

20.54.1.1 GENN_EXPORT

```
#define GENN_EXPORT
```

20.55 gennUtils.cc File Reference

```
#include "gennUtils.h"
#include <algorithm>
```

Namespaces

- Utils

Functions

- GENN_EXPORT bool Utils::isRNGRequired (const std::string &code)

  Does the code string contain any functions requiring random number generator.
- GENN_EXPORT bool Utils::isInitRNGRequired (const std::vector< Models::VarInit > &varInitialisers)

  Does the model with the vectors of variable initialisers and modes require an RNG for the specified init location i.e. host or device.
- GENN_EXPORT bool Utils::isTypePointer (const std::string &type)

  Function to determine whether a string containing a type is a pointer.
- GENN_EXPORT std::string Utils::getUnderlyingType (const std::string &type)

  Assuming type is a string containing a pointer type, function to return the underlying type.

20.56 gennUtils.h File Reference

```
#include <string>
#include <vector>
#include "gennExport.h"
#include "models.h"
```

Namespaces

- Utils

Functions

- GENN_EXPORT bool Utils::isRNGRequired (const std::string &code)

  Does the code string contain any functions requiring random number generator.
- GENN_EXPORT bool Utils::isInitRNGRequired (const std::vector< Models::VarInit > &varInitialisers)

  Does the model with the vectors of variable initialisers and modes require an RNG for the specified init location i.e. host or device.
- GENN_EXPORT bool Utils::isTypePointer (const std::string &type)

  Function to determine whether a string containing a type is a pointer.
- GENN_EXPORT std::string Utils::getUnderlyingType (const std::string &type)

  Assuming type is a string containing a pointer type, function to return the underlying type.

20.57 initSparseConnectivitySnippet.cc File Reference

```
#include "initSparseConnectivitySnippet.h"
```

Functions

- IMPLEMENT_SNIPPET (InitSparseConnectivitySnippet::Uninitialised)
- IMPLEMENT_SNIPPET (InitSparseConnectivitySnippet::OneToOne)
- IMPLEMENT_SNIPPET (InitSparseConnectivitySnippet::FixedProbability)
- IMPLEMENT_SNIPPET (InitSparseConnectivitySnippet::FixedProbabilityNoAutapse)

20.57.1 Function Documentation

20.57.1.1 IMPLEMENT_SNIPPET() [1/4]

```
IMPLEMENT_SNIPPET (
            InitSparseConnectivitySnippet::Uninitialised )
```

20.57.1.2 IMPLEMENT_SNIPPET() [2/4]

```
IMPLEMENT_SNIPPET (
            InitSparseConnectivitySnippet::OneToOne )
```

20.57.1.3 IMPLEMENT_SNIPPET() [3/4]

```
IMPLEMENT_SNIPPET (
            InitSparseConnectivitySnippet::FixedProbability )
```

20.57.1.4 IMPLEMENT_SNIPPET() [4/4]

```
IMPLEMENT_SNIPPET (
            InitSparseConnectivitySnippet::FixedProbabilityNoAutapse )
```

20.58 initSparseConnectivitySnippet.h File Reference

```
#include <functional>
#include <vector>
#include <cassert>
#include <cmath>
#include "binomial.h"
#include "snippet.h"
```

Classes

- class InitSparseConnectivitySnippet::Base
- class InitSparseConnectivitySnippet::Init
- class InitSparseConnectivitySnippet::Uninitialised

    Used to mark connectivity as uninitialised - no initialisation code will be run.
- class InitSparseConnectivitySnippet::OneToOne

    Initialises connectivity to a 'one-to-one' diagonal matrix.
- class InitSparseConnectivitySnippet::FixedProbabilityBase
- class InitSparseConnectivitySnippet::FixedProbability
- class InitSparseConnectivitySnippet::FixedProbabilityNoAutapse

Namespaces

- InitSparseConnectivitySnippet

    Base class for all sparse connectivity initialisation snippets.

Macros

- #define SET_ROW_BUILD_CODE(CODE) virtual std::string getRowBuildCode() const override{ return CO↩ DE; }
- #define SET_ROW_BUILD_STATE_VARS(...) virtual ParamValVec getRowBuildStateVars() const override{ return __VA_ARGS__; }
- #define SET_CALC_MAX_ROW_LENGTH_FUNC(FUNC) virtual CalcMaxLengthFunc getCalcMaxRow↩ LengthFunc() const override{ return FUNC; }
- #define SET_CALC_MAX_COL_LENGTH_FUNC(FUNC) virtual CalcMaxLengthFunc getCalcMaxCol↩ LengthFunc() const override{ return FUNC; }
- #define SET_MAX_ROW_LENGTH(MAX_ROW_LENGTH) virtual CalcMaxLengthFunc getCalcMaxRow↩ LengthFunc() const override{ return [](unsigned int, unsigned int, const std::vector<double> &){ return M↩ AX_ROW_LENGTH; }; }
- #define SET_MAX_COL_LENGTH(MAX_COL_LENGTH) virtual CalcMaxLengthFunc getCalcMaxCol↩ LengthFunc() const override{ return [](unsigned int, unsigned int, const std::vector<double> &){ return MAX_COL_LENGTH; }; }
- #define SET_EXTRA_GLOBAL_PARAMS(...) virtual VarVec getExtraGlobalParams() const override{ return __VA_ARGS__; }

20.58.1 Macro Definition Documentation

20.58.1.1 SET_CALC_MAX_COL_LENGTH_FUNC

```
#define SET_CALC_MAX_COL_LENGTH_FUNC(
              FUNC ) virtual CalcMaxLengthFunc getCalcMaxColLengthFunc() const override{ return
FUNC; }
```

20.58.1.2 SET_CALC_MAX_ROW_LENGTH_FUNC

```
#define SET_CALC_MAX_ROW_LENGTH_FUNC(
              FUNC ) virtual CalcMaxLengthFunc getCalcMaxRowLengthFunc() const override{ return
FUNC; }
```

20.58.1.3 SET_EXTRA_GLOBAL_PARAMS

```
#define SET_EXTRA_GLOBAL_PARAMS(
              ...  ) virtual VarVec getExtraGlobalParams() const override{ return __VA_ARGS__;
}
```

20.58.1.4 SET_MAX_COL_LENGTH

```
#define SET_MAX_COL_LENGTH(
              MAX_COL_LENGTH ) virtual CalcMaxLengthFunc getCalcMaxColLengthFunc() const override{
return [](unsigned int, unsigned int, const std::vector<double> &){ return MAX_COL_LENGTH; };
}
```

20.58.1.5 SET_MAX_ROW_LENGTH

```
#define SET_MAX_ROW_LENGTH(
              MAX_ROW_LENGTH ) virtual CalcMaxLengthFunc getCalcMaxRowLengthFunc() const override{
return [](unsigned int, unsigned int, const std::vector<double> &){ return MAX_ROW_LENGTH; };
}
```

20.58.1.6 SET_ROW_BUILD_CODE

```
#define SET_ROW_BUILD_CODE(
              CODE ) virtual std::string getRowBuildCode() const override{ return CODE; }
```

20.58.1.7 SET_ROW_BUILD_STATE_VARS

```
#define SET_ROW_BUILD_STATE_VARS(
              ...  ) virtual ParamValVec getRowBuildStateVars() const override{ return __VA_A↩
RGS__; }
```

20.59 initVarSnippet.cc File Reference

```
#include "initVarSnippet.h"
```

Functions

- IMPLEMENT_SNIPPET (InitVarSnippet::Uninitialised)
- IMPLEMENT_SNIPPET (InitVarSnippet::Constant)
- IMPLEMENT_SNIPPET (InitVarSnippet::Uniform)
- IMPLEMENT_SNIPPET (InitVarSnippet::Normal)
- IMPLEMENT_SNIPPET (InitVarSnippet::Exponential)
- IMPLEMENT_SNIPPET (InitVarSnippet::Gamma)

20.59.1 Function Documentation

20.59.1.1 IMPLEMENT_SNIPPET() [1/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Uninitialised )
```

20.59.1.2 IMPLEMENT_SNIPPET() [2/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Constant )
```

20.59.1.3 IMPLEMENT_SNIPPET() [3/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Uniform )
```

20.59.1.4 IMPLEMENT_SNIPPET() [4/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Normal )
```

20.59.1.5 IMPLEMENT_SNIPPET() [5/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Exponential )
```

20.59.1.6 IMPLEMENT_SNIPPET() [6/6]

```
IMPLEMENT_SNIPPET (
        InitVarSnippet::Gamma )
```

20.60 initVarSnippet.h File Reference

```
#include "snippet.h"
```

Classes

- class InitVarSnippet::Base
- class InitVarSnippet::Uninitialised

  Used to mark variables as uninitialised - no initialisation code will be run.
- class InitVarSnippet::Constant

  Initialises variable to a constant value.
- class InitVarSnippet::Uniform

  Initialises variable by sampling from the uniform distribution.
- class InitVarSnippet::Normal

  Initialises variable by sampling from the normal distribution.
- class InitVarSnippet::Exponential

  Initialises variable by sampling from the exponential distribution.
- class InitVarSnippet::Gamma

  Initialises variable by sampling from the exponential distribution.

Namespaces

- InitVarSnippet

  Base class for all value initialisation snippets.

Macros

- #define SET_CODE(CODE) virtual std::string getCode() const override{ return CODE; }

20.60.1 Macro Definition Documentation

20.60.1.1 SET_CODE

```
#define SET_CODE(
             CODE ) virtual std::string getCode() const override{ return CODE; }
```

20.61 model_preprocessor.py File Reference

Classes

- class pygenn.model_preprocessor.Variable

  Class holding information about GeNN variables.
- class pygenn.model_preprocessor.ExtraGlobalVariable

  Class holding information about GeNN extra global pointer variable.

Namespaces

- pygenn.model_preprocessor

Functions

- def pygenn.model_preprocessor.prepare_model (model, param_space, var_space, pre_var_space=None, post_var_space=None, model_family=None)

    Prepare a model by checking its validity and extracting information about variables and parameters.
- def pygenn.model_preprocessor.prepare_snippet (snippet, param_space, snippet_family)

    Prepare a snippet by checking its validity and extracting information about parameters.
- def pygenn.model_preprocessor.is_model_valid (model, model_family)

    Check whether the model is valid, i.e is native or derived from model_family.Custom.
- def pygenn.model_preprocessor.param_space_to_vals (model, param_space)

    Convert a param_space dict to ParamValues.
- def pygenn.model_preprocessor.param_space_to_val_vec (model, param_space)

    Convert a param_space dict to a std::vector< double >
- def pygenn.model_preprocessor.var_space_to_vals (model, var_space)

    Convert a var_space dict to VarValues.
- def pygenn.model_preprocessor.pre_var_space_to_vals (model, var_space)

    Convert a var_space dict to PreVarValues.
- def pygenn.model_preprocessor.post_var_space_to_vals (model, var_space)

    Convert a var_space dict to PostVarValues.

Variables

- dictionary pygenn.model_preprocessor.genn_to_numpy_types

20.62 models.h File Reference

```
#include <string>
#include <vector>
#include "snippet.h"
#include "initVarSnippet.h"
```

Classes

- class Models::VarInit
- class Models::VarInitContainerBase< NumVars >
- class Models::VarInitContainerBase< 0 >
- class Models::Base

    Base class for all models - in addition to the parameters snippets have, models can have state variables.

Namespaces

- Models

Macros

- #define DECLARE_MODEL(TYPE, NUM_PARAMS, NUM_VARS)
- #define IMPLEMENT_MODEL(TYPE) IMPLEMENT_SNIPPET(TYPE)
- #define SET_VARS(...) virtual VarVec getVars() const override{ return __VA_ARGS__; }
- #define SET_EXTRA_GLOBAL_PARAMS(...) virtual VarVec getExtraGlobalParams() const override{ return __VA_ARGS__; }

20.62.1 Macro Definition Documentation

20.62.1.1 DECLARE_MODEL

```
#define DECLARE_MODEL(
              TYPE,
              NUM_PARAMS,
              NUM_VARS )
```

Value:

```
DECLARE_SNIPPET(TYPE, NUM_PARAMS);                            \
    typedef Models::VarInitContainerBase<NUM_VARS> VarValues;
      \
    typedef Models::VarInitContainerBase<0> PreVarValues;            \
    typedef Models::VarInitContainerBase<0> PostVarValues
```

20.62.1.2 IMPLEMENT_MODEL

```
#define IMPLEMENT_MODEL(
              TYPE ) IMPLEMENT_SNIPPET(TYPE)
```

20.62.1.3 SET_EXTRA_GLOBAL_PARAMS

```
#define SET_EXTRA_GLOBAL_PARAMS(
              ... ) virtual VarVec getExtraGlobalParams() const override{ return __VA_ARGS__;
}
```

20.62.1.4 SET_VARS

```
#define SET_VARS(
              ... ) virtual VarVec getVars() const override{ return __VA_ARGS__; }
```

20.63 modelSpec.cc File Reference

```
#include <algorithm>
#include <numeric>
#include <typeinfo>
#include <cstdio>
#include <cmath>
#include <cassert>
#include "modelSpec.h"
#include "code_generator/codeGenUtils.h"
```

20.64 modelSpec.h File Reference

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of ModelSpec for defining a neuronal network model. Part of the code generation and generated code sections.

```
#include <map>
#include <set>
```

```
#include <string>
#include <vector>
#include "gennExport.h"
#include "neuronGroupInternal.h"
#include "synapseGroupInternal.h"
#include "currentSourceInternal.h"
```

Classes

- class ModelSpec

  Object used for specifying a neuronal network model.

Macros

- #define NO_DELAY 0

  Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

Typedefs

- typedef ModelSpec NNmodel

Enumerations

- enum FloatType { , GENN_LONG_DOUBLE }

  Floating point precision to use for models.
- enum TimePrecision { TimePrecision::DEFAULT, TimePrecision::FLOAT, TimePrecision::DOUBLE }

  Precision to use for variables which store time.

Functions

- template< typename S >
  Models::VarInit initVar (const typename S::ParamValues &params)

  Initialise a variable using an initialisation snippet.
- template< typename S >
  std::enable_if< std::is_same< typename S::ParamValues, Snippet::ValueBase< 0 > >::value, Models::VarInit >::type initVar ()

  Initialise a variable using an initialisation snippet with no parameters.
- Models::VarInit uninitialisedVar ()

  Mark a variable as uninitialised.
- template< typename S >
  InitSparseConnectivitySnippet::Init initConnectivity (const typename S::ParamValues &params)

  Initialise connectivity using a sparse connectivity snippet.
- template< typename S >
  std::enable_if< std::is_same< typename S::ParamValues, Snippet::ValueBase< 0 > >::value, InitSparseConnectivitySnippet::Init >::type initConnectivity ()

  Initialise connectivity using a sparse connectivity snippet with no parameters.
- InitSparseConnectivitySnippet::Init uninitialisedConnectivity ()

  Mark a synapse group's sparse connectivity as uninitialised.

20.64.1 Detailed Description

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of ModelSpec for defining a neuronal network model. Part of the code generation and generated code sections.

20.64.2 Macro Definition Documentation

20.64.2.1 NO_DELAY

```
#define NO_DELAY 0
```

Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

20.64.3 Typedef Documentation

20.64.3.1 NNmodel

```
typedef ModelSpec NNmodel
```

20.64.4 Enumeration Type Documentation

20.64.4.1 FloatType

```
enum FloatType
```

Floating point precision to use for models.

Enumerator

GENN_LONG_DOUBLE	

20.64.4.2 TimePrecision

```
enum TimePrecision [strong]
```

Precision to use for variables which store time.

Enumerator

DEFAULT	Time uses default model precision.
FLOAT	Time uses single precision - not suitable for long simulations.
DOUBLE	Time uses double precision - may reduce performance.

20.64.5 Function Documentation

20.64.5.1 initConnectivity() [1/2]

```
template<typename S >
InitSparseConnectivitySnippet::Init initConnectivity (
            const typename S::ParamValues & params ) [inline]
```

Initialise connectivity using a sparse connectivity snippet.

Template Parameters

S	type of sparse connectivitiy initialisation snippet (derived from InitSparseConnectivitySnippet::Base).

Parameters

params	parameters for snippet wrapped in S::ParamValues object.

Returns

InitSparseConnectivitySnippet::Init object for passing to `ModelSpec::addSynapsePopulation`

20.64.5.2 initConnectivity() [2/2]

```
template<typename S >
std::enable_if<std::is_same<typename S::ParamValues, Snippet::ValueBase<0> >::value, Init←
SparseConnectivitySnippet::Init>::type initConnectivity ( ) [inline]
```

Initialise connectivity using a sparse connectivity snippet with no parameters.

Template Parameters

S	type of sparse connectivitiy initialisation snippet (derived from InitSparseConnectivitySnippet::Base).

Returns

InitSparseConnectivitySnippet::Init object for passing to `ModelSpec::addSynapsePopulation`

20.64.5.3 initVar() [1/2]

```
template<typename S >
Models::VarInit initVar (
            const typename S::ParamValues & params ) [inline]
```

Initialise a variable using an initialisation snippet.

Template Parameters

S	type of variable initialisation snippet (derived from InitVarSnippet::Base).

Parameters

params	parameters for snippet wrapped in S::ParamValues object.

Returns

Models::VarInit object for use within model's VarValues

20.64.5.4 initVar() [2/2]

```
template<typename S >
std::enable_if<std::is_same<typename S::ParamValues, Snippet::ValueBase<0> >::value, Models↩
::VarInit>::type initVar ( ) [inline]
```

Initialise a variable using an initialisation snippet with no parameters.

Template Parameters

S	type of variable initialisation snippet (derived from InitVarSnippet::Base).

Returns

Models::VarInit object for use within model's VarValues

20.64.5.5 uninitialisedConnectivity()

```
InitSparseConnectivitySnippet::Init uninitialisedConnectivity ( ) [inline]
```

Mark a synapse group's sparse connectivity as uninitialised.

This means that the backend will not generate any automatic initialization code, but will instead copy the connectivity from host to device during `initializeSparse` function (and, if necessary generate any additional data structures it requires)

20.64.5.6 uninitialisedVar()

```
Models::VarInit uninitialisedVar ( ) [inline]
```

Mark a variable as uninitialised.

This means that the backend will not generate any automatic initialization code, but will instead copy the variable from host to device during `initializeSparse` function

20.65 modelSpecInternal.h File Reference

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

Classes

- class ModelSpecInternal

20.66 neuronGroup.cc File Reference

```
#include "neuronGroup.h"
#include <algorithm>
#include <cmath>
#include "currentSourceInternal.h"
#include "neuronGroupInternal.h"
```

```
#include "synapseGroupInternal.h"
#include "gennUtils.h"
```

20.67 neuronGroup.h File Reference

```
#include <map>
#include <set>
#include <string>
#include <vector>
#include "gennExport.h"
#include "neuronModels.h"
#include "variableMode.h"
```

Classes

- class NeuronGroup

20.68 neuronGroupInternal.h File Reference

```
#include "neuronGroup.h"
```

Classes

- class NeuronGroupInternal

20.69 neuronModels.cc File Reference

```
#include "neuronModels.h"
```

Functions

- IMPLEMENT_MODEL (NeuronModels::RulkovMap)
- IMPLEMENT_MODEL (NeuronModels::Izhikevich)
- IMPLEMENT_MODEL (NeuronModels::IzhikevichVariable)
- IMPLEMENT_MODEL (NeuronModels::LIF)
- IMPLEMENT_MODEL (NeuronModels::SpikeSource)
- IMPLEMENT_MODEL (NeuronModels::SpikeSourceArray)
- IMPLEMENT_MODEL (NeuronModels::Poisson)
- IMPLEMENT_MODEL (NeuronModels::PoissonNew)
- IMPLEMENT_MODEL (NeuronModels::TraubMiles)
- IMPLEMENT_MODEL (NeuronModels::TraubMilesFast)
- IMPLEMENT_MODEL (NeuronModels::TraubMilesAlt)
- IMPLEMENT_MODEL (NeuronModels::TraubMilesNStep)

20.69.1 Function Documentation

20.69.1.1 IMPLEMENT_MODEL() [1/12]

```
IMPLEMENT_MODEL (
        NeuronModels::RulkovMap )
```

20.69.1.2 IMPLEMENT_MODEL() [2/12]

```
IMPLEMENT_MODEL (
        NeuronModels::Izhikevich )
```

20.69.1.3 IMPLEMENT_MODEL() [3/12]

```
IMPLEMENT_MODEL (
        NeuronModels::IzhikevichVariable )
```

20.69.1.4 IMPLEMENT_MODEL() [4/12]

```
IMPLEMENT_MODEL (
        NeuronModels::LIF )
```

20.69.1.5 IMPLEMENT_MODEL() [5/12]

```
IMPLEMENT_MODEL (
        NeuronModels::SpikeSource )
```

20.69.1.6 IMPLEMENT_MODEL() [6/12]

```
IMPLEMENT_MODEL (
        NeuronModels::SpikeSourceArray )
```

20.69.1.7 IMPLEMENT_MODEL() [7/12]

```
IMPLEMENT_MODEL (
        NeuronModels::Poisson )
```

20.69.1.8 IMPLEMENT_MODEL() [8/12]

```
IMPLEMENT_MODEL (
        NeuronModels::PoissonNew )
```

20.69.1.9 IMPLEMENT_MODEL() [9/12]

```
IMPLEMENT_MODEL (
        NeuronModels::TraubMiles )
```

20.69.1.10 IMPLEMENT_MODEL() [10/12]

```
IMPLEMENT_MODEL (
```

```
            NeuronModels::TraubMilesFast  )
```

20.69.1.11 IMPLEMENT_MODEL() [11/12]

```
IMPLEMENT_MODEL (
            NeuronModels::TraubMilesAlt  )
```

20.69.1.12 IMPLEMENT_MODEL() [12/12]

```
IMPLEMENT_MODEL (
            NeuronModels::TraubMilesNStep  )
```

20.70 neuronModels.h File Reference

```
#include <array>
#include <functional>
#include <string>
#include <tuple>
#include <vector>
#include <cmath>
#include "models.h"
```

Classes

- class NeuronModels::Base

    Base class for all neuron models.
- class NeuronModels::RulkovMap

    Rulkov Map neuron.
- class NeuronModels::Izhikevich

    Izhikevich neuron with fixed parameters [1].
- class NeuronModels::IzhikevichVariable

    Izhikevich neuron with variable parameters [1].
- class NeuronModels::LIF
- class NeuronModels::SpikeSource

    Empty neuron which allows setting spikes from external sources.
- class NeuronModels::SpikeSourceArray

    Spike source array.
- class NeuronModels::Poisson

    Poisson neurons.
- class NeuronModels::PoissonNew

    Poisson neurons.
- class NeuronModels::TraubMiles

    Hodgkin-Huxley neurons with Traub & Miles algorithm.
- class NeuronModels::TraubMilesFast

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

    Hodgkin-Huxley neurons with Traub & Miles algorithm.
- class NeuronModels::TraubMilesNStep

    Hodgkin-Huxley neurons with Traub & Miles algorithm.

Namespaces

- NeuronModels

Macros

- #define SET_SIM_CODE(SIM_CODE) virtual std::string getSimCode() const override{ return SIM_CODE; }
- #define SET_THRESHOLD_CONDITION_CODE(THRESHOLD_CONDITION_CODE) virtual std::string getThresholdConditionCode() const override{ return THRESHOLD_CONDITION_CODE; }
- #define SET_RESET_CODE(RESET_CODE) virtual std::string getResetCode() const override{ return RESET_CODE; }
- #define SET_SUPPORT_CODE(SUPPORT_CODE) virtual std::string getSupportCode() const override{ return SUPPORT_CODE; }
- #define SET_ADDITIONAL_INPUT_VARS(...) virtual ParamValVec getAdditionalInputVars() const override{ return __VA_ARGS__; }
- #define SET_NEEDS_AUTO_REFRACTORY(AUTO_REFRACTORY_REQUIRED) virtual bool isAutoRefractoryRequired() const override{ return AUTO_REFRACTORY_REQUIRED; }

20.70.1 Macro Definition Documentation

20.70.1.1 SET_ADDITIONAL_INPUT_VARS

```
#define SET_ADDITIONAL_INPUT_VARS(
              ... ) virtual ParamValVec getAdditionalInputVars() const override{ return __VA_ARGS__; }
```

20.70.1.2 SET_NEEDS_AUTO_REFRACTORY

```
#define SET_NEEDS_AUTO_REFRACTORY(
              AUTO_REFRACTORY_REQUIRED ) virtual bool isAutoRefractoryRequired() const override{
return AUTO_REFRACTORY_REQUIRED; }
```

20.70.1.3 SET_RESET_CODE

```
#define SET_RESET_CODE(
              RESET_CODE ) virtual std::string getResetCode() const override{ return RESET_CODE; }
```

20.70.1.4 SET_SIM_CODE

```
#define SET_SIM_CODE(
              SIM_CODE ) virtual std::string getSimCode() const override{ return SIM_CODE; }
```

20.70.1.5 SET_SUPPORT_CODE

```
#define SET_SUPPORT_CODE(
              SUPPORT_CODE ) virtual std::string getSupportCode() const override{ return SUPPORT_CODE; }
```

20.70.1.6 SET_THRESHOLD_CONDITION_CODE

```
#define SET_THRESHOLD_CONDITION_CODE(
            THRESHOLD_CONDITION_CODE ) virtual std::string getThresholdConditionCode() const
override{ return THRESHOLD_CONDITION_CODE; }
```

20.71 optimiser.cc File Reference

```
#include "optimiser.h"
#include <algorithm>
#include <iostream>
#include <map>
#include <numeric>
#include <cstdlib>
#include <cuda.h>
#include <cuda_runtime.h>
#include <plog/Log.h>
#include "path.h"
#include "modelSpecInternal.h"
#include "code_generator/generateAll.h"
#include "utils.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::CUDA
- CodeGenerator::CUDA::Optimiser

Functions

- BACKEND_EXPORT Backend CodeGenerator::CUDA::Optimiser::createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

20.72 optimiser.cc File Reference

```
#include "optimiser.h"
#include "modelSpecInternal.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::SingleThreadedCPU
- CodeGenerator::SingleThreadedCPU::Optimiser

Functions

- BACKEND_EXPORT Backend CodeGenerator::SingleThreadedCPU::Optimiser::createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

20.73 optimiser.h File Reference

```
#include "backendExport.h"
#include "backend.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::CUDA
- CodeGenerator::CUDA::Optimiser

Functions

- BACKEND_EXPORT Backend CodeGenerator::CUDA::Optimiser::createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

20.74 optimiser.h File Reference

```
#include "backendExport.h"
#include "backend.h"
```

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::SingleThreadedCPU
- CodeGenerator::SingleThreadedCPU::Optimiser

Functions

- BACKEND_EXPORT Backend CodeGenerator::SingleThreadedCPU::Optimiser::createBackend (const ModelSpecInternal &model, const filesystem::path &outputPath, int localHostID, const Preferences &preferences)

20.75 postsynapticModels.cc File Reference

```
#include "postsynapticModels.h"
```

Functions

- IMPLEMENT_MODEL (PostsynapticModels::ExpCurr)
- IMPLEMENT_MODEL (PostsynapticModels::ExpCond)
- IMPLEMENT_MODEL (PostsynapticModels::DeltaCurr)

20.75.1 Function Documentation

20.75.1.1 IMPLEMENT_MODEL() [1/3]

```
IMPLEMENT_MODEL (
            PostsynapticModels::ExpCurr  )
```

20.75.1.2 IMPLEMENT_MODEL() [2/3]

```
IMPLEMENT_MODEL (
            PostsynapticModels::ExpCond  )
```

20.75.1.3 IMPLEMENT_MODEL() [3/3]

```
IMPLEMENT_MODEL (
            PostsynapticModels::DeltaCurr  )
```

20.76 postsynapticModels.h File Reference

```
#include <cmath>
#include "models.h"
```

Classes

- class PostsynapticModels::Base

  Base class for all postsynaptic models.
- class PostsynapticModels::ExpCurr

  Exponential decay with synaptic input treated as a current value.
- class PostsynapticModels::ExpCond

  Exponential decay with synaptic input treated as a conductance value.
- class PostsynapticModels::DeltaCurr

  Simple delta current synapse.

Namespaces

- PostsynapticModels

Macros

- #define SET_DECAY_CODE(DECAY_CODE) virtual std::string getDecayCode() const override{ return DE↩ CAY_CODE; }
- #define SET_CURRENT_CONVERTER_CODE(CURRENT_CONVERTER_CODE) virtual std::string get↩ ApplyInputCode() const override{ return "$(Isyn) += " CURRENT_CONVERTER_CODE ";"; }
- #define SET_APPLY_INPUT_CODE(APPLY_INPUT_CODE) virtual std::string getApplyInputCode() const override{ return APPLY_INPUT_CODE; }
- #define SET_SUPPORT_CODE(SUPPORT_CODE) virtual std::string getSupportCode() const override{ return SUPPORT_CODE; }

20.76.1 Macro Definition Documentation

20.76.1.1 SET_APPLY_INPUT_CODE

```
#define SET_APPLY_INPUT_CODE(
              APPLY_INPUT_CODE ) virtual std::string getApplyInputCode() const override{ return
APPLY_INPUT_CODE; }
```

20.76.1.2 SET_CURRENT_CONVERTER_CODE

```
#define SET_CURRENT_CONVERTER_CODE(
              CURRENT_CONVERTER_CODE ) virtual std::string getApplyInputCode() const override{
return "$(Isyn) += " CURRENT_CONVERTER_CODE ";"; }
```

20.76.1.3 SET_DECAY_CODE

```
#define SET_DECAY_CODE(
              DECAY_CODE ) virtual std::string getDecayCode() const override{ return DECAY_CO↩
DE; }
```

20.76.1.4 SET_SUPPORT_CODE

```
#define SET_SUPPORT_CODE(
              SUPPORT_CODE ) virtual std::string getSupportCode() const override{ return SUPP↩
ORT_CODE; }
```

20.77 snippet.h File Reference

```
#include <algorithm>
#include <functional>
#include <string>
#include <vector>
#include <cassert>
#include "gennExport.h"
```

Classes

- class Snippet::ValueBase< NumVars >
- class Snippet::ValueBase< 0 >
- class Snippet::Base

  Base class for all code snippets.
- struct Snippet::Base::Var

  A variable has a name and a type.
- struct Snippet::Base::ParamVal
- struct Snippet::Base::DerivedParam

  A derived parameter has a name and a function for obtaining its value.
- class Snippet::Init< SnippetBase >

Namespaces

- Snippet

Macros

- #define DECLARE_SNIPPET(TYPE, NUM_PARAMS)
- #define IMPLEMENT_SNIPPET(TYPE) TYPE *TYPE::s_Instance = NULL
- #define SET_PARAM_NAMES(...) virtual StringVec getParamNames() const override{ return __VA_ARGS↩ __; }
- #define SET_DERIVED_PARAMS(...) virtual DerivedParamVec getDerivedParams() const override{ return __VA_ARGS__; }

20.77.1 Macro Definition Documentation

20.77.1.1 DECLARE_SNIPPET

```
#define DECLARE_SNIPPET(
                TYPE,
                NUM_PARAMS )
```

Value:

```
private:                                                        \
    GENN_EXPORT static TYPE *s_Instance;                        \
public:                                                         \
    static const TYPE *getInstance()                            \
    {                                                           \
        if(s_Instance == NULL)                                  \
        {                                                       \
            s_Instance = new TYPE;                              \
        }                                                       \
        return s_Instance;                                      \
    }                                                           \
    typedef Snippet::ValueBase<NUM_PARAMS> ParamValues          \
```

20.77.1.2 IMPLEMENT_SNIPPET

```
#define IMPLEMENT_SNIPPET(
                TYPE ) TYPE *TYPE::s_Instance = NULL
```

20.77.1.3 SET_DERIVED_PARAMS

```
#define SET_DERIVED_PARAMS(
                ... ) virtual DerivedParamVec getDerivedParams() const override{ return __VA_A↩
RGS__; }
```

20.77.1.4 SET_PARAM_NAMES

```
#define SET_PARAM_NAMES(
                ... ) virtual StringVec getParamNames() const override{ return __VA_ARGS__; }
```

20.78 substitutions.h File Reference

```
#include <map>
#include <stdexcept>
#include <string>
#include <cassert>
#include "codeGenUtils.h"
```

Classes

- class CodeGenerator::Substitutions

Namespaces

- CodeGenerator

    Helper class for generating code - automatically inserts brackets, indents etc.

20.79 synapseGroup.cc File Reference

```
#include "synapseGroup.h"
#include <algorithm>
#include <cmath>
#include <iostream>
#include "neuronGroupInternal.h"
#include "gennUtils.h"
```

20.80 synapseGroup.h File Reference

```
#include <map>
#include <set>
#include <string>
#include <vector>
#include "gennExport.h"
#include "initSparseConnectivitySnippet.h"
#include "postsynapticModels.h"
#include "weightUpdateModels.h"
#include "synapseMatrixType.h"
#include "variableMode.h"
```

Classes

- class SynapseGroup

20.81 synapseGroupInternal.h File Reference

```
#include "synapseGroup.h"
```

Classes

- class SynapseGroupInternal

20.82 synapseMatrixType.h File Reference

Enumerations

- enum SynapseMatrixConnectivity : unsigned int { SynapseMatrixConnectivity::DENSE = (1 << 0), SynapseMatrixConnectivity::BITMASK = (1 << 1), SynapseMatrixConnectivity::SPARSE = (1 << 2) }

< *Flags defining differnet types of synaptic matrix connectivity*

- enum SynapseMatrixWeight : unsigned int { SynapseMatrixWeight::GLOBAL = (1 << 5), SynapseMatrixWeight::INDIVIDUAL = (1 << 6), SynapseMatrixWeight::INDIVIDUAL_PSM = (1 << 7) }
- enum SynapseMatrixType : unsigned int {
SynapseMatrixType::DENSE_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::DENSE) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL), SynapseMatrixType::DENSE_GLOBALG_INDIVIDUAL_PSM = static_cast<unsigned int>(SynapseMatrixConnectivity::DENSE) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL_PSM), SynapseMatrixType::DENSE_INDIVIDUALG = static_cast<unsigned int>(SynapseMatrixConnectivity::DENSE) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL_PSM), SynapseMatrixType::BITMASK_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::BITMASK) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL),
SynapseMatrixType::BITMASK_GLOBALG_INDIVIDUAL_PSM = static_cast<unsigned int>(SynapseMatrixConnectivity::BITMASK) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL_PSM), SynapseMatrixType::SPARSE_GLOBALG = static_cast<unsigned int>(SynapseMatrixConnectivity::SPARSE) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL), SynapseMatrixType::SPARSE_GLOBALG_INDIVIDUAL_PSM = static_cast<unsigned int>(SynapseMatrixConnectivity::SPARSE) | static_cast<unsigned int>(SynapseMatrixWeight::GLOBAL) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL_PSM), SynapseMatrixType::SPARSE_INDIVIDUALG = static_cast<unsigned int>(SynapseMatrixConnectivity::SPARSE) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL) | static_cast<unsigned int>(SynapseMatrixWeight::INDIVIDUAL_PSM) }

Functions

- bool operator& (SynapseMatrixType type, SynapseMatrixConnectivity connType)
- bool operator& (SynapseMatrixType type, SynapseMatrixWeight weightType)

20.82.1 Enumeration Type Documentation

20.82.1.1 SynapseMatrixConnectivity

```
enum SynapseMatrixConnectivity : unsigned int [strong]
```

< Flags defining differnet types of synaptic matrix connectivity

Enumerator

DENSE	
BITMASK	
SPARSE	

20.82.1.2 SynapseMatrixType

```
enum SynapseMatrixType : unsigned int [strong]
```

Enumerator

DENSE_GLOBALG	
DENSE_GLOBALG_INDIVIDUAL_PSM	
DENSE_INDIVIDUALG	
BITMASK_GLOBALG	

Enumerator

BITMASK_GLOBALG_INDIVIDUAL_PSM	
SPARSE_GLOBALG	
SPARSE_GLOBALG_INDIVIDUAL_PSM	
SPARSE_INDIVIDUALG	

20.82.1.3 SynapseMatrixWeight

```
enum SynapseMatrixWeight : unsigned int [strong]
```

Enumerator

GLOBAL	
INDIVIDUAL	
INDIVIDUAL_PSM	

20.82.2 Function Documentation

20.82.2.1 operator&() [1/2]

```
bool operator & (
            SynapseMatrixType type,
            SynapseMatrixConnectivity connType ) [inline]
```

20.82.2.2 operator&() [2/2]

```
bool operator & (
            SynapseMatrixType type,
            SynapseMatrixWeight weightType ) [inline]
```

20.83 teeStream.h File Reference

```
#include <ostream>
#include <streambuf>
#include <vector>
```

Classes

- class CodeGenerator::TeeBuf
- class CodeGenerator::TeeStream

Namespaces

- CodeGenerator

  Helper class for generating code - automatically inserts brackets, indents etc.

20.84 utils.h File Reference

```
#include <iostream>
#include <plog/Log.h>
```

Namespaces

- CodeGenerator
  Helper class for generating code - automatically inserts brackets, indents etc.
- CodeGenerator::CUDA
- CodeGenerator::CUDA::Utils

Macros

- #define CHECK_CU_ERRORS(call) call
- #define CHECK_CUDA_ERRORS(call)

Functions

- size_t CodeGenerator::CUDA::Utils::ceilDivide (size_t numerator, size_t denominator)
- size_t CodeGenerator::CUDA::Utils::padSize (size_t size, size_t blockSize)

20.84.1 Macro Definition Documentation

20.84.1.1 CHECK_CU_ERRORS

```
#define CHECK_CU_ERRORS(
                call ) call
```

20.84.1.2 CHECK_CUDA_ERRORS

```
#define CHECK_CUDA_ERRORS(
                call )
```

Value:

```
{                      \
    cudaError_t error = call;
                       \
    if (error != cudaSuccess) {
                       \
        LOGE << __FILE__ << ": " <<  __LINE__ << ": cuda runtime error " << error << ": " <<
      cudaGetErrorString(error); \
        exit(EXIT_FAILURE);
                       \
    }
                       \
}
```

20.85 variableMode.h File Reference

```
#include <cstdint>
```

Enumerations

- enum VarLocation : uint8_t {
  VarLocation::HOST = (1 $<<$ 0), VarLocation::DEVICE = (1 $<<$ 1), VarLocation::ZERO_COPY = (1 $<<$ 2),
  VarLocation::HOST_DEVICE = HOST | DEVICE,
  VarLocation::HOST_DEVICE_ZERO_COPY = HOST | DEVICE | ZERO_COPY }

  $<$ *Flags defining which memory space variables should be allocated in*

Functions

- bool operator& (VarLocation locA, VarLocation locB)

20.85.1 Enumeration Type Documentation

20.85.1.1 VarLocation

```
enum VarLocation : uint8_t [strong]
```

$<$ Flags defining which memory space variables should be allocated in

Enumerator

HOST	
DEVICE	
ZERO_COPY	
HOST_DEVICE	
HOST_DEVICE_ZERO_COPY	

20.85.2 Function Documentation

20.85.2.1 operator&()

```
bool operator & (
            VarLocation locA,
            VarLocation locB ) [inline]
```

20.86 weightUpdateModels.cc File Reference

```
#include "weightUpdateModels.h"
```

Functions

- IMPLEMENT_MODEL (WeightUpdateModels::StaticPulse)
- IMPLEMENT_MODEL (WeightUpdateModels::StaticPulseDendriticDelay)
- IMPLEMENT_MODEL (WeightUpdateModels::StaticGraded)
- IMPLEMENT_MODEL (WeightUpdateModels::PiecewiseSTDP)

20.86.1 Function Documentation

20.86.1.1 IMPLEMENT_MODEL() [1/4]

```
IMPLEMENT_MODEL (
            WeightUpdateModels::StaticPulse  )
```

20.86.1.2 IMPLEMENT_MODEL() [2/4]

```
IMPLEMENT_MODEL (
            WeightUpdateModels::StaticPulseDendriticDelay  )
```

20.86.1.3 IMPLEMENT_MODEL() [3/4]

```
IMPLEMENT_MODEL (
            WeightUpdateModels::StaticGraded  )
```

20.86.1.4 IMPLEMENT_MODEL() [4/4]

```
IMPLEMENT_MODEL (
            WeightUpdateModels::PiecewiseSTDP  )
```

20.87 weightUpdateModels.h File Reference

```
#include "models.h"
```

Classes

- class WeightUpdateModels::Base

    Base class for all weight update models.
- class WeightUpdateModels::StaticPulse

    Pulse-coupled, static synapse.
- class WeightUpdateModels::StaticPulseDendriticDelay

    Pulse-coupled, static synapse with heterogenous dendritic delays.
- class WeightUpdateModels::StaticGraded

    Graded-potential, static synapse.
- class WeightUpdateModels::PiecewiseSTDP

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

Namespaces

- WeightUpdateModels

Macros

- #define DECLARE_WEIGHT_UPDATE_MODEL(TYPE, NUM_PARAMS, NUM_VARS, NUM_PRE_VARS, NUM_POST_VARS)
- #define SET_SIM_CODE(SIM_CODE) virtual std::string getSimCode() const override{ return SIM_CODE; }

- #define SET_EVENT_CODE(EVENT_CODE) virtual std::string getEventCode() const override{ return EV↩ ENT_CODE; }
- #define SET_LEARN_POST_CODE(LEARN_POST_CODE) virtual std::string getLearnPostCode() const override{ return LEARN_POST_CODE; }
- #define SET_SYNAPSE_DYNAMICS_CODE(SYNAPSE_DYNAMICS_CODE) virtual std::string get↩ SynapseDynamicsCode() const override{ return SYNAPSE_DYNAMICS_CODE; }
- #define SET_EVENT_THRESHOLD_CONDITION_CODE(EVENT_THRESHOLD_CONDITION_CODE) virtual std::string getEventThresholdConditionCode() const override{ return EVENT_THRESHOLD_CONDITI↩ ON_CODE; }
- #define SET_SIM_SUPPORT_CODE(SIM_SUPPORT_CODE) virtual std::string getSimSupportCode() const override{ return SIM_SUPPORT_CODE; }
- #define SET_LEARN_POST_SUPPORT_CODE(LEARN_POST_SUPPORT_CODE) virtual std::string get↩ LearnPostSupportCode() const override{ return LEARN_POST_SUPPORT_CODE; }
- #define SET_SYNAPSE_DYNAMICS_SUPPORT_CODE(SYNAPSE_DYNAMICS_SUPPORT_CODE) virtual std::string getSynapseDynamicsSuppportCode() const override{ return SYNAPSE_DYNAMICS_SUP↩ PORT_CODE; }
- #define SET_PRE_SPIKE_CODE(PRE_SPIKE_CODE) virtual std::string getPreSpikeCode() const override{ return PRE_SPIKE_CODE; }
- #define SET_POST_SPIKE_CODE(POST_SPIKE_CODE) virtual std::string getPostSpikeCode() const override{ return POST_SPIKE_CODE; }
- #define SET_PRE_VARS(...) virtual VarVec getPreVars() const override{ return __VA_ARGS__; }
- #define SET_POST_VARS(...) virtual VarVec getPostVars() const override{ return __VA_ARGS__; }
- #define SET_NEEDS_PRE_SPIKE_TIME(PRE_SPIKE_TIME_REQUIRED) virtual bool isPreSpikeTime↩ Required() const override{ return PRE_SPIKE_TIME_REQUIRED; }
- #define SET_NEEDS_POST_SPIKE_TIME(POST_SPIKE_TIME_REQUIRED) virtual bool isPostSpike↩ TimeRequired() const override{ return POST_SPIKE_TIME_REQUIRED; }

20.87.1 Macro Definition Documentation

20.87.1.1 DECLARE_WEIGHT_UPDATE_MODEL

```
#define DECLARE_WEIGHT_UPDATE_MODEL(
                TYPE,
                NUM_PARAMS,
                NUM_VARS,
                NUM_PRE_VARS,
                NUM_POST_VARS )
```

Value:

```
DECLARE_SNIPPET(TYPE, NUM_PARAMS);
          \
    typedef Models::VarInitContainerBase<NUM_VARS> VarValues;
                                        \
    typedef Models::VarInitContainerBase<NUM_PRE_VARS>
      PreVarValues;                                   \
    typedef Models::VarInitContainerBase<NUM_POST_VARS>
      PostVarValues
```

20.87.1.2 SET_EVENT_CODE

```
#define SET_EVENT_CODE(
                EVENT_CODE ) virtual std::string getEventCode() const override{ return EVENT_CO↩
DE; }
```

20.87.1.3 SET_EVENT_THRESHOLD_CONDITION_CODE

```
#define SET_EVENT_THRESHOLD_CONDITION_CODE(
            EVENT_THRESHOLD_CONDITION_CODE ) virtual std::string getEventThresholdCondition←
Code() const override{ return EVENT_THRESHOLD_CONDITION_CODE; }
```

20.87.1.4 SET_LEARN_POST_CODE

```
#define SET_LEARN_POST_CODE(
            LEARN_POST_CODE ) virtual std::string getLearnPostCode() const override{ return
LEARN_POST_CODE; }
```

20.87.1.5 SET_LEARN_POST_SUPPORT_CODE

```
#define SET_LEARN_POST_SUPPORT_CODE(
            LEARN_POST_SUPPORT_CODE ) virtual std::string getLearnPostSupportCode() const
override{ return LEARN_POST_SUPPORT_CODE; }
```

20.87.1.6 SET_NEEDS_POST_SPIKE_TIME

```
#define SET_NEEDS_POST_SPIKE_TIME(
            POST_SPIKE_TIME_REQUIRED ) virtual bool isPostSpikeTimeRequired() const override{
return POST_SPIKE_TIME_REQUIRED; }
```

20.87.1.7 SET_NEEDS_PRE_SPIKE_TIME

```
#define SET_NEEDS_PRE_SPIKE_TIME(
            PRE_SPIKE_TIME_REQUIRED ) virtual bool isPreSpikeTimeRequired() const override{
return PRE_SPIKE_TIME_REQUIRED; }
```

20.87.1.8 SET_POST_SPIKE_CODE

```
#define SET_POST_SPIKE_CODE(
            POST_SPIKE_CODE ) virtual std::string getPostSpikeCode() const override{ return
POST_SPIKE_CODE; }
```

20.87.1.9 SET_POST_VARS

```
#define SET_POST_VARS(
            ... ) virtual VarVec getPostVars() const override{ return __VA_ARGS__; }
```

20.87.1.10 SET_PRE_SPIKE_CODE

```
#define SET_PRE_SPIKE_CODE(
            PRE_SPIKE_CODE ) virtual std::string getPreSpikeCode() const override{ return P←
RE_SPIKE_CODE; }
```

20.87.1.11 SET_PRE_VARS

```
#define SET_PRE_VARS(
             ... ) virtual VarVec getPreVars() const override{ return __VA_ARGS__; }
```

20.87.1.12 SET_SIM_CODE

```
#define SET_SIM_CODE(
             SIM_CODE ) virtual std::string getSimCode() const override{ return SIM_CODE; }
```

20.87.1.13 SET_SIM_SUPPORT_CODE

```
#define SET_SIM_SUPPORT_CODE(
             SIM_SUPPORT_CODE ) virtual std::string getSimSupportCode() const override{ return
SIM_SUPPORT_CODE; }
```

20.87.1.14 SET_SYNAPSE_DYNAMICS_CODE

```
#define SET_SYNAPSE_DYNAMICS_CODE(
             SYNAPSE_DYNAMICS_CODE ) virtual std::string getSynapseDynamicsCode() const override{
return SYNAPSE_DYNAMICS_CODE; }
```

20.87.1.15 SET_SYNAPSE_DYNAMICS_SUPPORT_CODE

```
#define SET_SYNAPSE_DYNAMICS_SUPPORT_CODE(
             SYNAPSE_DYNAMICS_SUPPORT_CODE ) virtual std::string getSynapseDynamicsSuppport←
Code() const override{ return SYNAPSE_DYNAMICS_SUPPORT_CODE; }
```

References

[1] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. 7, 8, 9, 60, 77, 180, 181, 183, 184, 321

[2] Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98:459–478, 2008. 32

[3] 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. 32

[4] 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. 11, 233

[5] Nikolai F Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65(4):041922, 2002. 233

[6] Marcel Stimberg, Dan F. M. Goodman, and Thomas Nowotny. Brian2genn: a system for accelerating a large variety of spiking neural networks with graphics hardware. *bioRxiv*, 2018. 14

[7] R. D. Traub and R. Miles. *Neural Networks of the Hippocampus*. Cambridge University Press, New York, 1991. 11, 12, 272

Index