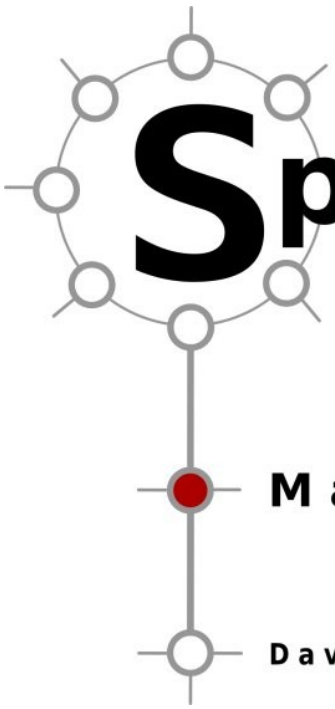


SpikeStream 0.1

Manual

David Gamez

Acknowledgments

Many thanks to Owen Holland for feedback, support and advice about SpikeStream. The interface between SIMNOS and SpikeStream was developed in collaboration with Richard Newcombe, who designed the spike conversion methods. Thanks also to Renzo De Nardi and Hugo Gravato Marques in the Machine Consciousness lab at Essex and to everyone at the 2006 Telluride workshop. This work was funded by the Engineering and Physical Science Research Council Adventure Fund (GR/S47946/01).

Contents

1. Introduction

1.1 Overview

SpikeStream is a simulator that has been tested on medium sized networks of up to 100,000 spiking neurons. It works in a modular distributed manner and can run in parallel across an arbitrary number of machines. SpikeStream exchanges spikes with external devices over a network and comes ready to work with the SIMNOS virtual humanoid robot (see section 9.4). More information about the architecture of SpikeStream can be found in Gamez (2007). This manual covers the installation of SpikeStream and use of its key features.

I have tried to make Linux installation as painless as possible using four scripts that set the necessary variables, build SpikeStream, install SpikeStream and create the databases. However, these depend on third party software and a database, and so a certain amount of work is required to get the whole system running. For other operating systems a virtual machine distribution has been prepared, which is covered in Section 2.8.

SpikeStream is a complex piece of software with many useful features and it is stable enough to run experiments. However, it is still at an early stage of development and subject to a number of bugs and limitations. Occasionally it will crash, but most of the time you will not lose data because all changes are immediately stored in the database and restarting most often solves the problem. If you let me know about any undocumented bugs and limitations, I will do my best to solve them and any offers of help with SpikeStream are extremely welcome. If there is enough interest, I will turn it into a collaborative open source project.

This manual is targeted at the user of SpikeStream who wants to use the simulation functions and may want to extend the Neuron or Synapse classes to create their own neural and synapse models. I have tried to make the information in this manual as accurate as possible, but time limitations have preventing me from checking it as thoroughly as I would have liked. Documentation of the source code is available in the doc folder of the distribution and at http://spikestream.sourceforge.net.

1.2 Support

Feel free to get in touch if you have any problems building and running SpikeStream. You can reach me at david@davidgamez.eu or on +44 (0) 7790 803 368. I have also set up a mailing list for SpikeStream at spikestream-user@lists.sourceforge.net.

2. Installation

2.1 Overview

Before installing SpikeStream it is recommended that you read the paper covering its architecture and operation (Gamez, 2007). Sections 2.2-2.7 give full instructions for installing SpikeStream on Linux and other UNIX based systems. If you just want to try SpikeStream out or use it on Windows or OS X, it is available pre-installed on a SUSE 10.2 virtual machine, which can be run using the VMware Player (see Section 2.8).

2.2 System Requirements for Linux Installation

2.2.1 Operating System

SpikeStream has been written and tested on SUSE 10.0 and SUSE 10.2. SpikeStream Simulation and SpikeStream Archiver have also been tested on Debian 3.1. A few tweaks and hacks may be required to get it working on other Linux and UNIX operating systems. It should be possible to get SpikeStream running on Cygwin under Windows, but I have not attempted this yet.

2.2.2 Hardware

SpikeStream can run on a single machine or across a cluster. On the main workstation, hardware graphics acceleration will speed up the visualization of large networks. A megabit network is useful if you want to run SpikeStream across several machines.

2.3 Dependencies

SpikeStream depends on a number of other libraries, which must be installed first. Some of these are only needed on the main workstation to compile and run SpikeStream Application. Others are needed by all modules.

2.3.1 Google Sparse Hash

Fast and efficient dense and sparse hash maps developed by Google. Available at http://goog-sparsehash.sourceforge.net/.
Install on all machines.

2.3.2 MySQL Database and Development Libraries

May form part of your Linux distribution. Otherwise available at www.mysql.org. You need the development parts of MySQL as well as the server.
The development libraries need to be installed on all machines. The server only needs to be on the machine(s) that are hosting the databases.

2.3.3 MySQL++

C++ wrapper for MySQL. Available at: http://tangentsoft.net/mysql++/ .
Install on all machines.

2.3.4 Qt

Provides a graphical user interface and many useful functions. Likely to come with your distribution of Linux.
IMPORTANT NOTE: *SpikeStream only compiles and runs using Qt version 3.*.*.* It will not compile using Qt 4.*.*. If 4.*.* is your default version of Qt, you need to install Qt 3.*.* in a separate location to compile SpikeStream. In SUSE 10.2 the default Qt version is 4, but Qt 3 is also installed and you can make Qt 3 the default by adding the Qt 3 directory to the start of your path in your .bashrc file using: `export PATH=$QTDIR/bin:$PATH.` You can also directly invoke this version of qmake on the command line by using **`$QTDIR/bin/qmake`** instead of **`qmake`** when you generate the makefiles.
Qt is only needed on the main workstation.

2.3.5 PVM (Parallel Virtual Machine)

Used for distributed message passing and spawning of remote processes. Included with some Linux distributions, otherwise install manually. Available at: http://www.netlib.org/pvm3/index.html.
IMPORTANT NOTES:

1. The build of PVM may break with recent versions of gcc. If it breaks with the error:

   ```
   ...  src/global.h: 321: error: array type has incomplete element type
   ...  src/global.h: 323: error: array type has incomplete element type
   ```

   Replacing PVM_ROOT/src/global.h with global.h from the 'extras' folder of the SpikeStream distribution should fix the problem.

2. It can be useful to give user level accounts permission to write to $PVM_ROOT/bin/LINUX. This makes it easier when you have to manually install spikestreamarchiver and spikestreamsimulation, which have to be installed in this directory to be launched by pvm.
3. On SUSE 10.2 (and perhaps elsewhere) you may get an error along the lines of : ”netoutput() sendto: Invalid argument” when adding a second host in pvm. This can be fixed by adding an entry to your hosts file along the lines of:
   `[machine ip address] [machine name]`
   for example: `192.168.1.22  desktopmachine`

If you install pvm yourself, don't forget to create a link to PVM_ROOT/lib/pvm from your bin folder so that it can be run from anywhere. You may also want to install xpvm, which can be very helpful for debugging processes and messages when things go wrong. Getting pvm to run successfully across several machines can be tricky and is beyond the scope of this manual.
Install on all machines.

2.3.6 Qwt

Graph drawing libraries available at: http://qwt.sourceforge.net/.
Only needed on the main workstation.

2.4 Build and Installation Using Scripts

This section covers the installation of SpikeStream using scripts that set the variables, build the modules and install the libraries. These are the quickest and easiest way to install SpikeStream on Linux. If anything goes wrong with these scripts, section 2.5 covers manual installation of the individual modules.

2.4.1 Unpack Distribution

When you have downloaded SpikeStream, you need to unpack it using the command:

```
tar -xzvf spikestream-0.1.tar.gz
```

This will extract it to a directory called spikestream-0.1. This will be the root directory for building and running the application, so move this directory to its final location before moving on to the next step.

2.4.2 Set SPIKESTREAM_ROOT

SpikeStream depends on a shell variable called SPIKESTREAM_ROOT, which is *essential* for building and running the application. This variable should be set to the root of the spikestream-0.1 directory. The best place to set this is in your .bashrc file by adding, for example:

```
export SPIKESTREAM_ROOT=/home/davidg/spikestream-0.1
```

This needs to be done on all machines that you build and run SpikeStream on and you need to make sure that the remote shell invoked by pvm (which may be different from your default bash shell) also has SPIKESTREAM_ROOT set correctly.

2.4.3 Set Build Variables

To keep everything as simple as possible, the locations of the libraries needed for building SpikeStream are set by the SetSpikeStreamVariables script, which can be found in the scripts folder of the distribution. Open this script up and check that the library and include locations match those on your system:

```
#Location of MySQL
export MYSQL_INCLUDE=/usr/include/mysql
# Location of MySQL++
export MYSQLPP_INCLUDE=/usr/local/include/mysql++
# Location of Qwt files. Not needed for simulation builds
export QWT_ROOT=/usr/local/qwt
# Location of Google hash map include files.
export GOOGLE_INCLUDE=/usr/local/include/google
```

When you are installing SpikeStream across several machines, the Qt and Qwt libraries are only needed on the machine running SpikeStream Application. In this case, run the script with the option "-s".

SpikeStream cannot be built unless these variables have been set correctly for the type of build. When you have checked the locations, save the script and try running it from the scripts folder using:

`./SetSpikeStreamVariables` (Main workstation)

`./SetSpikeStreamVariables -s` (Other machines used in the simulation)

If it exits without errors, you can move on to the next stage of the installation. If you get errors setting the variables, make sure that all of the required libraries are in the places set by the script and SPIKESTREAM_ROOT and PVM_ROOT are set correctly.

2.4.4 Run Build Script

SpikeStream comes with a build script that compiles all of the modules and copies the ones that are installed in the SPIKESTREAM_ROOT directory to their correct locations. This is not guaranteed to work on every occasion, but can speed up the installation process considerably. If you do have problems running this script it is worth taking a look inside it for the list of commands that are needed to build and install the parts of the application. To run this script, change to the scripts folder and type:

`./BuildSpikeStream` (Main workstation)

`./BuildSpikeStream -s` (Other machines used in the simulation)

If all goes well you should end up with the following output on the main workstation:

```
------------------------------------------------------------------------
---------------            Build Results             ----------------
------------------------------------------------------------------------
SpikeStreamApplication: Built ok.
SpikeStreamSimulation: Built and installed ok.
SpikeStreamArchiver: Built and installed ok.
STDP1 Neuron: Built ok.
STDP1 Synapse Built ok.
SpikeStream built successfully.
```

If one of the libraries or applications does not build, you will have to track down the error by looking at the configure and make output and either re-run the build script or install the missing component(s) individually. Instructions for installing each of the components individually are given in section 2.5.

2.4.5 Install SpikeStream

This script installs spikestreamsimulation and spikestreamarchiver in the $PVM_ROOT/bin/LINUX directory, which often requires root privileges. Some neuron and synapse libraries also need to be installed as root to enable dynamic linking and the install script creates symbolic links between one of the default library locations on your system and the neuron

and synapse libraries in $SPIKESTREAM_ROOT/lib. The use of symbolic links is suggested because it is anticipated that you will be recompiling the neuron and synapse libraries to implement your own learning algorithms and the use of symbolic links saves you the trouble of installing them as root each time you do this. If you are planning to use only the supplied neuron and synapse classes, then copies of these can be placed in the specified locations. More information about this can be found in section 12.3.

IMPORTANT NOTE: You should only install links to these libraries as root if you are the sole user of SpikeStream on the system. Otherwise you may end up dynamically loading another user's libraries!

To run the install script, get a root shell, make sure that SPIKESTREAM_ROOT is defined in the root shell ("`echo $SPIKESTREAM_ROOT`" should return the correct location) and run:

```
$SPIKESTREAM_ROOT/scripts/InstallSpikeStream
```

If everything has worked up to this point you can move on to set up the databases, as described in section 3. If the build has broken for some reason, take a look at some of the common build and installation problems covered in section 2.7. Instructions for manually building each component are given in the next section.

2.5 Manual Installation Procedure

Once your have unpacked the distribution (section 2.4.1) and set the SPIKESTREAM_ROOT variable (section 2.4.2), you are ready to manually build and install the SpikeStream components. You should only install SpikeStream this way if you have run into some problems with the build and installation scripts.

2.5.1 SpikeStream Library

This contains classes that are common to many parts of the system and should be compiled first.

- Check the locations in the SetSpikeStreamVariables script and run it using "**`. ./SetSpikeStreamVariables`**" (don't miss the *second* dot before the slash!).
- Change to directory $(SPIKESTREAM_ROOT)/spikestreamlibrary/
- Run the command: **`./configure –libdir=$SPIKESTREAM_ROOT/lib`**
- Type **`make`**
- If everything goes ok, type **`make install`**. There should be a file called libspikestream.a

in the $SPIKESTREAM_ROOT/lib directory.

2.5.2 SpikeStream Application

This is the graphical application for editing neuron groups and launching simulations and only needs to be built on the main workstation. It is a Qt project, so installation is a little different from the other parts of the system.

- Check your version of Qt is correct by typing **`qmake –version`**. The output should contain the version of Qt that qmake is using, for example (Qt 3.3.7). If your version is greater than 3 you need to install Qt 3 on your system and make sure that qmake uses this version of Qt. See section 2.3.4 for more on this.
- Check the locations and debug flags in the SetSpikeStreamVariables script and run it using: **`. ./SetSpikeStreamVariables`** (don't miss the *second* dot before the slash!).
- Change to the $SPIKESTREAM_ROOT/spikestreamapplication directory and use qmake to create the makefiles: **`qmake spikestreamapplication.pro`**
- Type **`make`**
- If everything goes ok, there should be a program called spikestreamapplication in the $SPIKESTREAM_ROOT/spikestreamapplication/bin directory.
- If you want, create a symbolic link to $SPIKESTREAM_ROOT/bin or your local bin directory using: **`ln -s $SPIKESTREAM_ROOT/spikestreamapplication/bin /spikestreamapplication $SPIKESTREAM_ROOT/bin/spikestream`**.

You can try to run it, but it will not work properly until the database has been configured – see section 3.

2.5.3 SpikeStream Simulation

This is the program that runs to simulate a neuron group. It is launched using pvm, so has to be installed in the $PVM_ROOT/bin/LINUX directory on every machine that you want to run a simulation on. If you are running SpikeStream across several different Linux versions, this program will have to be recompiled for each architecture.

- Check the locations and debug flags in the SetSpikeStreamVariables script and run it using: **`. ./SetSpikeStreamVariables`** (don't miss the *second* dot before the slash!).
- Change to the $SPIKESTREAM_ROOT/spikestreamsimulation directory.
- run the command: **`./configure --bindir=$PVM_ROOT/bin/LINUX --`**

libdir=$SPIKESTREAM_ROOT/lib

- Type **make**
- If all goes well type **make install**. You will need to have write permission to the $PVM_ROOT/bin/LINUX directory or change to superuser for this step.
- If everything goes ok, there should be a file called libspikestreamsimulation.a in the $SPIKESTREAM_ROOT/lib directory and an executable file called spikestreamsimulation in the $PVM_ROOT/bin/LINUX directory.

2.5.4 SpikeStream Archiver

This program stores firing patterns in the database. It is launched using pvm, so has to be in the $PVM_ROOT/bin/LINUX directory of every machine that you want to run a simulation on. If you are running SpikeStream across several different Linux versions, this program will have to be recompiled for each architecture.

- Check the locations and debug flags in the SetSpikeStreamVariables script and run it using **. ./SetSpikeStreamVariables** (don't miss the *second* dot before the slash!).
- Change to the $SPIKESTREAM_ROOT/spikestreamarchiver directory.
- run the command: **./configure --bindir=$PVM_ROOT/bin/LINUX**
- Type **make**
- If all goes well type **make install**. You will need to have write permission to the $PVM_ROOT/bin/LINUX directory or change to superuser for this step.
- If everything goes ok, there should be an executable file called spikestreamarchiver in the $PVM_ROOT/bin/LINUX directory.

2.5.5 Neuron and Synapse Classes

Neuron and Synapse classes are stored as libraries that are dynamically loaded at runtime and the name of each library should be added to NeuronTypes or SynapseTypes in the database. Some neuron and synapse libraries may need to call methods on each other and so they need to be placed in the $SPIKESTREAM_ROOT/lib directory to enable cross linking. Copies also need to be placed in /user/local/lib to enable dynamic loading. Chapter 12 gives detailed information about adding your own neuron and synapse classes to SpikeStream. Installation instructions are given for STDP1Synapse here, which should be followed for each of the neuron and synapse libraries.

- Check the order in which the neuron and synapse classes need to be built. Some neuron and synapse classes depend on each other so the build order may be important. For example,

STDP1Synapse must be built before STDP1Neuron.

- The neuron and synapse classes depend on the spikestreamsimulation library, so make sure that this is installed correctly before commencing installation.
- Check the locations and debug flags in the SetSpikeStreamVariables script and run it using: **`. ./SetSpikeStreamVariables`** (don't miss the *second* dot before the slash!).
- Change to the $SPIKESTREAM_ROOT/STDP1Synapse directory.
- run the command: **`./configure`**
- Type **`make`**
- If all goes well copy the libstdp1synapse.so library to $SPIKESTREAM_ROOT/lib directory.
- Log in as root and change to your system's library location: **`cd /usr/local/lib`**
- Create a link from your system's library location to the neuron library: **`ln -s -f ${SPIKESTREAM_ROOT}/lib/libstdp1synapse.so libstdp1synapse.so.1`**
- Add the information about the neuron class that you have installed to the database – see section 12.4.

2.6 Cleaning Up and Uninstalling SpikeStream

2.6.1 CleanSpikeStream Script

SpikeStream can cleaned up using the CleanSpikeStream script. This removes all of the files in SPIKESTREAM_ROOT created by the build script and runs make clean in each of the directories. It also removes the "Makefile" files created by qmake in the spikestreamapplication directory. The clean script does not remove spikestreamsimulation, spikestreamarchiver or the symbolic links to libstdp1neuron.so and libstdp1synapse.so that are created by the InstallSpikeStream script. You need to run the uninstall script to delete these components of SpikeStream.

2.6.2 UninstallSpikeStream Script

This script uninstalls spikestreamsimulation, spikestreamarchiver and deletes the symbolic links to the neuron and synapse libraries. Use this when you want to remove all SpikeStream files from the system except for those at SPIKESTREAM_ROOT.

IMPORTANT NOTE: *This script must be run as root.*

2.7 Common Build and Installation Problems

Some common build and installation problems are as follows.

1. When building SpikeStream application you are likely to get the warning "has virtual functions but non-virtual destructor". This is a known issue, which should be ignored. See http://lists.trolltech.com/qt-interest/2005-10/msg00342.html.
2. You may get some strange Qt errors that break the build, such as:

```
In file included from NetworkDataXmlHandler.h:27,
                 from ArchiveManager.h:28,
                 from ArchiveManager.cpp:24:
NetworkMonitor.h:33:17: error: qgl.h: No such file or directory
In file included from ArchiveManager.h:28,
                 from ArchiveManager.cpp:24:
NetworkDataXmlHandler.h:30:18: error: qxml.h: No such file or directory
In file included from SpikeStreamMainWindow.h:28,
                 from ArchiveManager.cpp:28:
NetworkViewer.h:33:20: error: qaccel.h: No such file or directory
In file included from SpikeStreamMainWindow.h:29,
                 from ArchiveManager.cpp:28:
NetworkViewerProperties.h:38:20: error: qtable.h: No such file or
directory
In file included from MonitorArea.h:28,
                 from SimulationWidget.h:29,
                 from SpikeStreamMainWindow.h:31,
                 from ArchiveManager.cpp:28:
MonitorWindow.h:32:25: error: qdockwindow.h: No such file or directory
In file included from SimulationWidget.h:29,
                 from SpikeStreamMainWindow.h:31,
                 from ArchiveManager.cpp:28:
MonitorArea.h:37:23: error: qdockarea.h: No such file or directory
In file included from SpikeStreamMainWindow.h:34,
                 from ArchiveManager.cpp:28:
LayerWidget.h:32:24: error: qpopupmenu.h: No such file or directory
```

   These are almost certainly caused by compiling with the wrong Qt version. Check the Qt version by using "qmake –version**".** *If the Qt version is 4.*.*, it will not work! You must build SpikeStream Application using Qt 3.*.*.* When you have sorted out the correct version of Qt (see section 2.3.4) you need to remove the "Makefile" files from spikestreamappliction and spikestreamapplication/src before running the build script again.

This can be done manually or by invoking the CleanSpikeStream script, which will do it for you. A future version of SpikeStream will compatible with Qt 4.

3. Double check that all the libraries are installed in the places specified in the SetSpikeStreamVariables script. If, during manual installation, you run this script without a dot and space before it, then the variables will not be set.
4. Double check that SPIKESTREAM_ROOT and PVM_ROOT are set correctly for your system. Both are crucial to a successful build. A common problem when running SpikeStream across several machines is that the default shell invoked by pvm is different from the one that has SPIKESTREAM_ROOT and PVM_ROOT set.
5. The error: "**cp: cannot create regular file `/home/davidg/lib/pvm3/bin/LINUX/spikestreamarchiver': Permission denied**" is caused because you do not have permission to access the directory where pvm is installed. Change to root before running the installation script again or give all users write access to this directory. If you lack superuser access you may need to create a local pvm installation.
6. A *build* problem related to permissions may occur if you copy the spikestream-0.1.tar.gz file as root and then unpack and build it. This can cause errors building STDP1 Neuron and STDP1 Synapse, which gcc attributes to inadequate permission to access the file .lib. To solve this problem, set yourself as the user of spikesream-0.1.tar.gz and set its group to users before unpacking it.
7. If the SpikeStream Application GUI looks like it was built in the 1970's and does not share the look and feel of other KDE applications on your machine, rebooting may solve the problem. Otherwise check that you are not compiling against an old version of Qt (before 3.*.*).
8. If you have database problems launching SpikeStream across several machines, make sure that the database configuration is not set to 'localhost' – put the ip address in spikestream.config instead.

If you cannot find a solution to your problem, see section 1.2 for further support.

2.8 Virtual Machine Installation

2.8.1 Overview

SpikeStream is also available pre-installed on a SUSE 10.2 virtual machine. This is a much bulkier distribution (around 4GB) that enables it to be run on a variety of operating systems with the minimum of installation difficulties. The disadvantages of this are the size, a slightly reduced

running speed and the fact that you have to boot up the virtual machine every time that you want to run SpikeStream (although SpikeStream can be restarted any number of times once the virtual machine has booted up). This manual only covers the basics and the VMware documentation should be consulted for full instructions about installing the VMware Player and running virtual machines.

2.8.2 Virtual Machine Files

The virtual machine files are available on DVD (drop me an email if you would like to receive a copy) or for download at: http://csres82.essex.ac.uk/~daogam/.

2.8.3 Install VMware Player

Download and install the free VMware Player from: http://www.vmware.com. If you want to use SpikeStream with SIMNOS (see section 9.4) you will need to configure the networking between the SUSE virtual machine and the host operating system so that you can ping each operating system from the other and access the Devices database on the host operating system from SUSE. This is not necessary if you are not using SIMNOS. Support with installation of VMware Player and its networking can be found in the VMware documentation and forums.

2.8.4 Run Virtual Machine

Once SUSE 10.2 is running in your VMware Player, click the SpikeStream icon on the SUSE desktop to start SpikeStream. Some of the devices that were present on the machine that was used to create the virtual machine may not be available on your system – the DVD drive at location E: and the floppy drive, for example. If you want to correct these problems or change the configuration of the virtual machine, you will have to purchase a copy of VMware Workstation, since the free VMware Player does not allow you to edit the virtual machine.

IMPORTANT NOTE: To reduce the size of the virtual machine distribution, the virtual hard drive has been kept as small as possible. There is only about 500MB free space on the drive, so take care not to over fill it or you may not be able to boot the virtual machine.

3. Databases

3.1 Introduction

SpikeStream depends on a number of databases, which can be distributed across different machines. The parameters for these databases are set in the $SPIKESTREAM_ROOT/spikestream.config file. This file is only used on the main workstation since the database parameters are passed to SpikeStream Simulation and SpikeStream Archiver as command line parameters. The SpikeStream databases are as follows:

- *NeuralNetwork*. Stores neurons, synapses and the connections between them. Different types of neuron and synapse classes are also stored here, along with parameters and the amount of noise injected into each of the neuron groups.
- *NeuralArchive*. Stores patterns of spikes or firing neurons that are recorded by the user during a simulation run.
- *Patterns*. Stores patterns that can be applied by the user to a layer during a simulation run. More information about patterns is given in Chapter 10.
- *Devices*. Lists the devices that are available for SpikeStream to connect to. Also breaks the device layer down into receptors and groups of receptors known as components. See Chapter 9 for more about SpikeStream and external devices.

More detailed information about the structure and purpose of these databases can be found in the SQL files in $SPIKESTREAM_ROOT/databases, which are used to create and populate the databases. When running SpikeStream with SIMNOS, SIMNOS sets up and updates the Devices and SIMNOSSpikeReceptors tables in the *Devices* database, and the host, username and password of the *Devices* database needs to be coordinated with SIMNOS. This manual assumes that all four databases will be set up using the same host, username and password.

3.2 Setting up MySQL

3.2.1 Introduction

Before SpikeStream can run, the correct databases need to be created and their user, host and password information entered in the $SPIKESTREAM_ROOT/spikestream.config file. This only needs to be done on the main workstation since the database parameters are passed to SpikeStream Simulation and SpikeStream Archiver as command line parameters. You can go straight on to section 3.4 if you already have a MySQL server and an account set up that you want to use with

SpikeStream. Details about setting up and running MySQL can be found in many places and there is extensive MySQL documentation online. Only the basics are given here.

3.2.2 Start MySQL Server

When you have installed MySQL (see section 2.3.2), test to see if it is running using: **`ps -el | grep mysql`**. This should return a line containing “mysqld” as one of the running processes. If this is not listed, use chkconfig to enable the service. As superuser type: **`chkconfig –list mysql`**, which should tell you if mysql is enabled or not. If it is not enabled for your current run level, type: **`chkconfig mysql on`** and make sure that it is enabled.

Even when mysql is enabled, the daemon may not have started. To start the daemon go to /etc/init.d/ and log in as root. Then run the mysql command by typing: **`./mysql start`**, which should start up the daemon. Check that it has started, then you are ready to set up the accounts.

3.2.3 Set Maximum Number of Connections

Each layer is handled by SpikeStream using a separate pvm process, which may have several connections to the database. If you are going to be using a large number of layers it is a good idea to increase the number of allowed connections to the database, which is set by default to 100. You can view the maximum number of connections using:

```
SHOW VARIABLES LIKE 'max_connections';
```

and change the maximum number of connections using, for example:

```
SET GLOBAL max_connections=150;
```

3.2.4 Configure Firewall

You need to to allow external access to MySQL if you are running SpikeStream across several machines and your system's firewall may need to be changed to allow this. In SUSE this can be done by adding MySQL to the firewall configuration using YAST. If you are communicating with SIMNOS on Windows you will also need to open ports for each device, in addition to the Devices database (if this is on the Windows machine).

3.3 Create Accounts

3.3.1 Root Account

Log in as root using **`mysql -u root`**

Display the current accounts: **`SELECT user, host, password FROM mysql.user;`**

Set a password for root: **`SET password=PASSWORD("secretpassword")`**

Get rid of unnecessary users: **`DELETE FROM mysql.user WHERE user != "root";`**

Get rid of logins from outside machine: **`DELETE FROM mysql.user WHERE host != "localhost";`**

3.3.2 SpikeStream Account

Create accounts with the user SpikeStream and the password 'myPassword' that can access the database on localhost or a subnetwork:

```
GRANT ALL ON *.* TO SpikeStream@localhost IDENTIFIED BY "myPassword";

GRANT ALL ON *.* TO SpikeStream@'192.168.1.0/255.255.255.0' IDENTIFIED
BY "myPassword";
```

If these have been created successfully it should be possible to log into the database locally or from another machine on the same network using:

`mysql -uSpikeStream -pmyPassword` (local login with password "myPassword")

`mysql -uSpikeStream -pmyPassword -h192.168.1.9 (`remote login with mysql hosted on 192.168.1.9 and password "myPassword")

You can create a different account for each database or put the databases on different machines. As long as the privileges are set up correctly it should work fine. The details for each database need to be added into the spikestream.config file on the main workstation.

3.4 Create Databases and Tables

3.4.1 Create Database Script

Once you have configured the account(s), you can use a SpikeStream script to set up the databases. Open up the script in a text editor and change the user, host and password information to match that which you set in section 3.3.2. When this information has been set correctly run it using:

```
$SPIKESTREAM_ROOT/scripts/CreateSpikeStreamDatabases
```

IMPORTANT NOTE: This script will overwrite the contents of all SpikeStream databases that are already on the system. It can also be used at a later point to reset all of the databases.

3.4.2 Manual Database Creation

Four SQL files are used to create the databases. These can be found at $SPIKESTREAM_ROOT/database:

- *NeuralNetwork.sql*
- *NeuralArchive.sql*
- *Patterns.sql*
- *Devices.sql*

Another four SQL files are used to add neuron types, synapse types, probe types and devices to the databases that have been created.

- *AddNeuronTypes.sql*
- *AddSynapseTypes.sql*
- *AddProbeTypes.sql*
- *AddDevices.sql*

Finally, each neuron and synapse type needs an entry in the NeuronTypes and SynapseTypes tables indicating the location of their parameter table and the location of their class library. See the CreateSpikeStreamDatabases script for the commands needed to load these sql files individually into the database.

IMPORTANT NOTE: The NeuralNetwork SQL sets up the database so that neuron ids start at 10, rather than 0. It is essential for the operation of the system that neuron ids 0-10 remain unused. These ids are generated each time a neuron is added to the system and I am not certain what happens when the automatically generated ids wrap around back to the beginning. It is worth keeping an eye on this and periodically re-initialise the database if necessary.

4. Running SpikeStream

4.1 Configuration

Open up the $SPIKESTREAM_ROOT/spikestream.config file and make sure that the database information is set correctly for the four databases. This only needs to be done on the main workstation since the database parameters are passed to SpikeStream Simulation and SpikeStream Archiver as command line parameters. I recommend leaving the database name untouched. You may also want to set the default location for saving and loading files. Once the config file has been saved you can start SpikeStream Application using the symbolic link spikestream in the SPIKESTREAM_ROOT/bin directory.

4.2 PVM

On a single machine SpikeStream will launch pvm and run without problems. If you want to run SpikeStream across several machines, you will need to start pvm and add the other machines as hosts before starting a simulation using SpikeStream. The SpikeStream Application can be running whilst you are doing this as long as a simulation is not initialised.

Getting pvm to work across several machines depends on being able to remotely invoke commands on the other machines using rsh (it can also be configured using ssh, but this probably incurs a significant performance penalty). Many Linux clusters are already set up for this, but configuring it from scratch on a new distribution can be a tricky process since rsh is usually disabled by default for security reasons. Finding the right place to set PVM_ROOT and SPIKESTREAM_ROOT so that they is available when pvm is remotely invoked can also cause problems. When pvm has been correctly configured you should be able to start it and add the remote host using the commands:

`pvm` (should return the prompt: “pvm>”)

```
pvm>add newHostName
```

If this works typing **`conf`** should list the new virtual machine configuration. Once the virtual machine has been configured you can use SpikeStream to launch hosts across multiple machines.

4.3 Monitoring and Debugging Information

Some of the monitoring and debugging information that is available when running SpikeStream is as follows:

- The command line output of SpikeStream generally gives more information than is explicitly displayed in error messages. You will need to launch SpikeStream from the command line (rather than a desktop shortcut) to see this information.
- xpvm enables the monitoring of messages sent between the different processes.
- Output of processes started with pvm (all the simulation and archiving tasks) is routed to /tmp/pvml.1000. It can also be picked up using the task output feature of xpvm, although this can cause crashes when there is a large amount of output.
- Most SpikeStream modules have a file called Debug.h, which enables different types of debugging information to be displayed. The relevant part will have to be recompiled for this to take effect.
- pvm has a command line interface that lets you see what processes are running and kill them if necessary. Type pvm and then "help" to find out more about the available commands and look at the online documentation for pvm.

4.4 Common Problems Running SpikeStream

A number of problems can arise when running SpikeStream:

- You will occasionally get an error message "FAILURE TO UPDATE DATABASE WITH TASKID", even when everything is set up correctly between SpikeStream and its databases. This is a bug that has not been sorted out. Restarting the simulation usually fixes the problem.
- When you have built SpikeStream and try clicking on spikestreamapplication with the mouse you may get an error message informing you that SPIKESTREAM_ROOT is not defined and SpikeStream will exit. If SpikeStream runs ok when you type **`./spikestream`** in the SPIKESREAM_ROOT/bin directory, this problem can be solved by logging out of your user account and logging in again. If SpikeStream does not run from the command line either then you need to make sure that SPIKESREAM_ROOT is defined in the appropriate file for your shell (probably .bashrc). See section 2.4.2 for more on this.
- Sometimes you will get errors along the lines of "mksocs() connect Connection Refused". This is probably due to a problem with pvm. If this happens, it is most likely due to some old files left over in /tmp from a previous simulation run that crashed. The best solution is to wait 30 seconds until SpikeStream times out, when it will ask you if you want to run the CleanPVM script. Run this script and the problem should go away. Persistent problems can often be solved by deleting all pvm related files from /tmp. The CleanPVM script can also be separately invoked to reset pvm and delete unused files from /tmp.

- SpikeStream will fail to connect with databases and devices on other machines if the firewalls on both machines are not set correctly.
- Simulations will not start if the dynamic neuron and synapse libraries cannot be found by the operating system (see section 12.3). This may generate the message “libstdp1neuron.so: cannot open shared object file” or “libstdp1neuron.so: cannot open shared object file”, which can be caused by omitting to run the install script as part of the installation process. It can also be caused by copying a library across from another machine, instead of recompiling it for your system.
- Simulations will not start if pvm is not installed properly. You can check that pvm is working correctly by typing **`pvm`**, which should return the pvm command prompt: **`pvm>.`**
- Loading a saved database occasionally creates problems when you have added or removed a neuron or synapse type since the saved database contains tables with the old information. Similar problems can occur with the Devices database. If SpikeStream generates parameter errors or crashes after loading a database containing different neuron or synapse types, restarting it usually resolves the problem., which is caused by a bug in the parameter dialogs.
- If you have problems adding additional hosts to pvm make sure that you have rsh installed on your system, which may have been left off the default install for security reasons. You will also need to add the main workstation to your list of allowed hosts in .rhosts on the remote machines so that pvm can invoke commands on them without being prompted for the password. Use the IP address if you are working on a local network since the name of the machine may not be resolved (this will have to be set up each time the machines boot if using DHCP).
- With more recent versions of qwt you may get the error “libqwt.so.5: cannot open shared object file: No such file or directory”. This linking error arises because the operating system cannot find the qwt library that spikestream was compiled against. One way of solving this problem is to create a symbolic link that points to the appropriate libraries. To solve the qwt problem change to /usr/lib in super user mode, and type

  ```
  ln -s /usr/local/qwt-5.0.2/lib/libqwt.so.5 libqwt.so.5
  ```

  The details of this solution will change depending on the version of qwt that you are using.
- A similar problem can arise with mysqlpp libraries, which can be solved in a similar way by changing to /usr/lib in super user mode and typing:

  ```
  ln -s /usr/local/lib/libmysqlpp.so.2 libmysqlpp.so.2
  ```

Again, the specific paths and library will change depending on the versions that you are using. Linking problems can also be solved by adding the appropriate locations to the LD_LIBRARY_PATH system variable, which is probably the best bet if you do not have root access to the system.

4.5 Error Messages

When SpikeStream Application detects an error it generally displays an error message. When this error only affects the function currently being performed, SpikeStream will not exit, but you will probably want to restart SpikeStream (if possible after sorting the problem out). For example, if you get a database related error when loading a simulation, try to resolve the problem and then restart the simulation. When the error is likely to corrupt the database or make future work impossible, SpikeStream will immediately exit.

When simulation and archive tasks detect an error they will not exit immediately, but enter an error state in which they only respond to exit messages. This is to enable the simulation manager to do an explicit clean up after the end of the simulation without needing to restart and clean pvm. If you get an error message from a task, destroy the simulation, determine the cause of the error if possible and then restart the simulation. Let me know about any persistent problems and I will try to resolve them.

4.6 Known Bugs and Missing Functionality

Known bugs and limitations in SpikeStream 0.1 are as follows:

- The probe feature is still under development and has not been fully implemented.
- Rotation of layers for patterns and devices is missing. Although it may be possible to connect a layer with width 10 and length 25 up to a device or pattern with width 25 and length 10, the simulation will not work. You are advised to only connect up layers to patterns or devices with identical width and height.
- The ability to set and change the neuron spacing is not well tested. It should work, but is best left at the default of 1.
- The simulation will only run for 2^{32} time steps, which is around 1000 hours at 1 ms per timestep. After this, the simulation clock will overflow with unknown consequences.
- On later versions of Qt 3, the Network Monitor goes black when resized beyond a certain point. The firing patterns have been made dark red so that they can still be seen, but I have

not found a better work around for this problem, which will probably disappear when SpikeStream is rewritten for Qt4.

- There is a limit to the maximum number of network monitors that can be open at once. This is currently 100, which is set using the variable MAX_NUMBER_MONITOR_WINDOWS in SPIKESTREAM_ROOT/include/GlobalVariables.h.
- Off center on surround connections are not implemented in the present version of SpikeStream.
- Make defaults button is not implemented on most of the parameter dialogs.
- Neuron and synapse types can only be changed when SpikeStream Application is not running. If SpikeStream Application is running when they are changed, it is likely to crash, but this will not affect the data in the database and restarting solves the problem.
- The exchange of spikes between SIMNOS and SpikeStream (see section 9.4) is still at the early stages. This feature does work, but expect a certain amount of sweat and hassle to get everything working.
- The “Load Defaults” button is not implemented in the neuron or synapse parameters dialogs and the “Make Defaults” button has not been implemented in the edit neuron or synapse parameters dialogs.
- The canceling of operations is not well handled at present and may generate an error message when canceling the loading of a simulation. A future version of SpikeStream will address this problem by using separate threads to handle heavy operations.
- The recording of network patterns is buggy and currently runs without synchronization to the spikesreamsimulation tasks. This occasionally results in the dropping of recorded time steps, particularly at the beginning or end of the simulation run. You may also get an error: “ArchiveWidget: MYSQL QUERY EXCEPTION MySQL server has gone away", which can be resolved by restarting SpikeStream. These problems will be sorted out in a later version of SpikeStream, which will tightly synchronize spikesreamarchiver with the simulation tasks.

5. Creating Neural Networks

5.1 The Editor Tab

The creation and editing of neural networks is carried out on the Editor tab (see Figure 5.1). The top table shows information about the current neuron groups and the bottom table contains information about the connections between neuron groups.

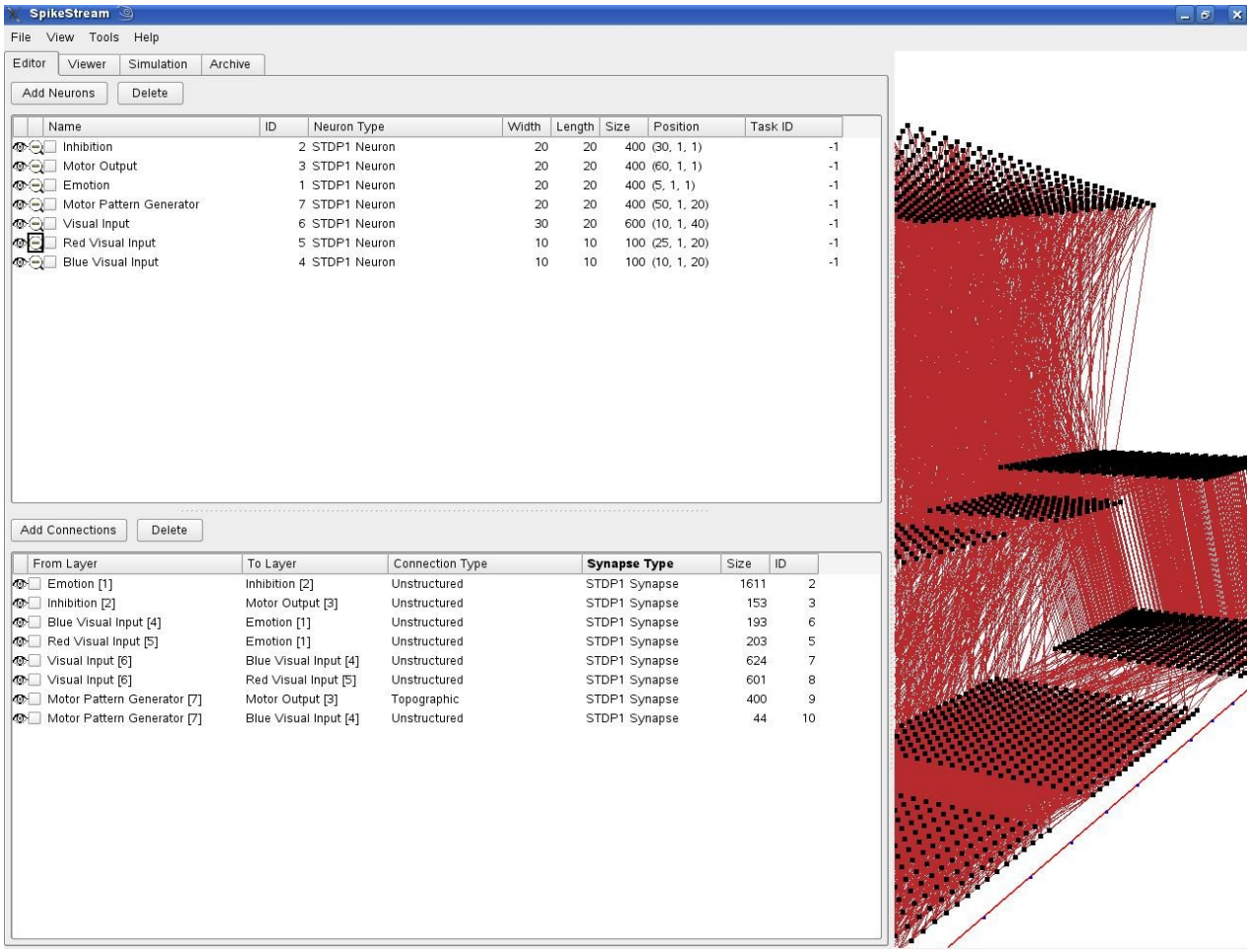


Figure 5.1. Editor Tab

5.1.1 Neuron Group Table

The top half of the Editor tab contains the neuron group table which displays neuron group information in the database. The start of each row has an eye and a magnifier symbol. Clicking on the eye symbol hides or shows a neuron group and you can click on the column header to hide or show all neuron groups. A single click on the magnifying glass zooms in to the side of the appropriate neuron group. Click on it again and you are taken to the top of the appropriate neuron group. A third click returns you to a wide view of the entire network.

5.1.2 Connection Group Table

The bottom half of the Editor tab is taken up with the connection group table, which displays information about the connection groups in the database. At the left of each row is an eye symbol that can be used to show or hide the connection groups and you can click on the table header to view or hide all connection groups and to check or uncheck all of the tick boxes. Viewing of connection groups is disabled by default and very large connection groups will only be loaded when you attempt to view them, which may lead to a short delay whilst this is carried out. Virtual connections can never be viewed and are coloured light grey. Clicking on the blue "View" button in the connection group table shows the parameters that were used to create the connection group.

5.2 Adding Neuron Groups

Clicking on the "Add Neurons" button above the neuron group table displays the Neuron Group Properties Dialog, shown in Figure 5.2.

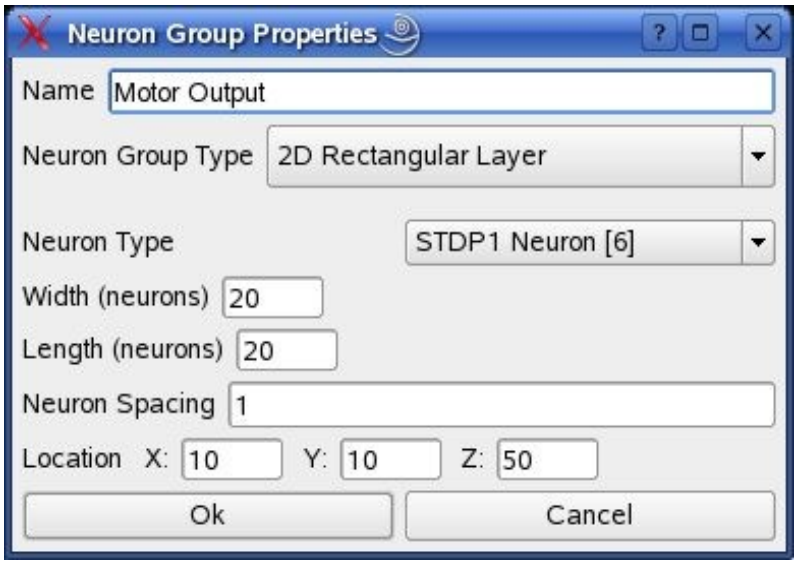


Figure 5.2. Neuron Group Properties Dialog

This dialog allows you to set the following information about the layer.

- **Name**. The name of the new neuron group
- **Neuron Group Type**. This combo box has three options.
    - *2D Rectangular Layer.* Creates a standard 2D layer 1 neuron thick.
    - *3D Rectangular Layer. C*reates a 3D layer. This is not fully implemented yet.
    - *SIMNOS Component*. Uses information from the Devices database to create a layer that connects to a sub-part of an input layer - see section 9.4.

- **Neuron Type**. A list of the neuron classes in the NeuronTypes table.
- **Width**. The width of the neuron group in neurons.
- **Length**. The length of the neuron group in neurons.
- **Neuron Spacing**. Allows you to change the spacing between the neurons. *WARNING: This feature has not been fully tested and it is recommended to leave it at 1.*
- **Location**. The location of the bottom left corner of the neuron group when seen from above. Make sure that your selected location does not clash with an existing layer.

5.3 Editing Neuron Groups

Some of the properties of a neuron group can be changed at a later point in time by right clicking on the neuron group in the neuron group table and selecting "Edit Neuron Group Properties" from the popup menu.

5.4 Deleting Neuron Groups

Check the neuron groups that you want to delete and click on the "Delete" button. A dialog will popup to confirm your decision. Clicking "Ok" will permanently delete the neuron group from the database.

IMPORTANT NOTE: There is no undo function in SpikeStream and no method of reversing this step. Future work on SpikeStream may look into using the MySQL rollback feature to undo transactions.

5.5 Adding Connection Groups

SpikeStream comes with a number of predefined connection patterns. Once you are familiar with SpikeStream you are likely to start creating your own connection patterns by directly editing the database (see section 5.7). To use the built in connection patterns, start by clicking on "Add Connections". This launches the Connection Properties Dialog shown in Figure 5.3.

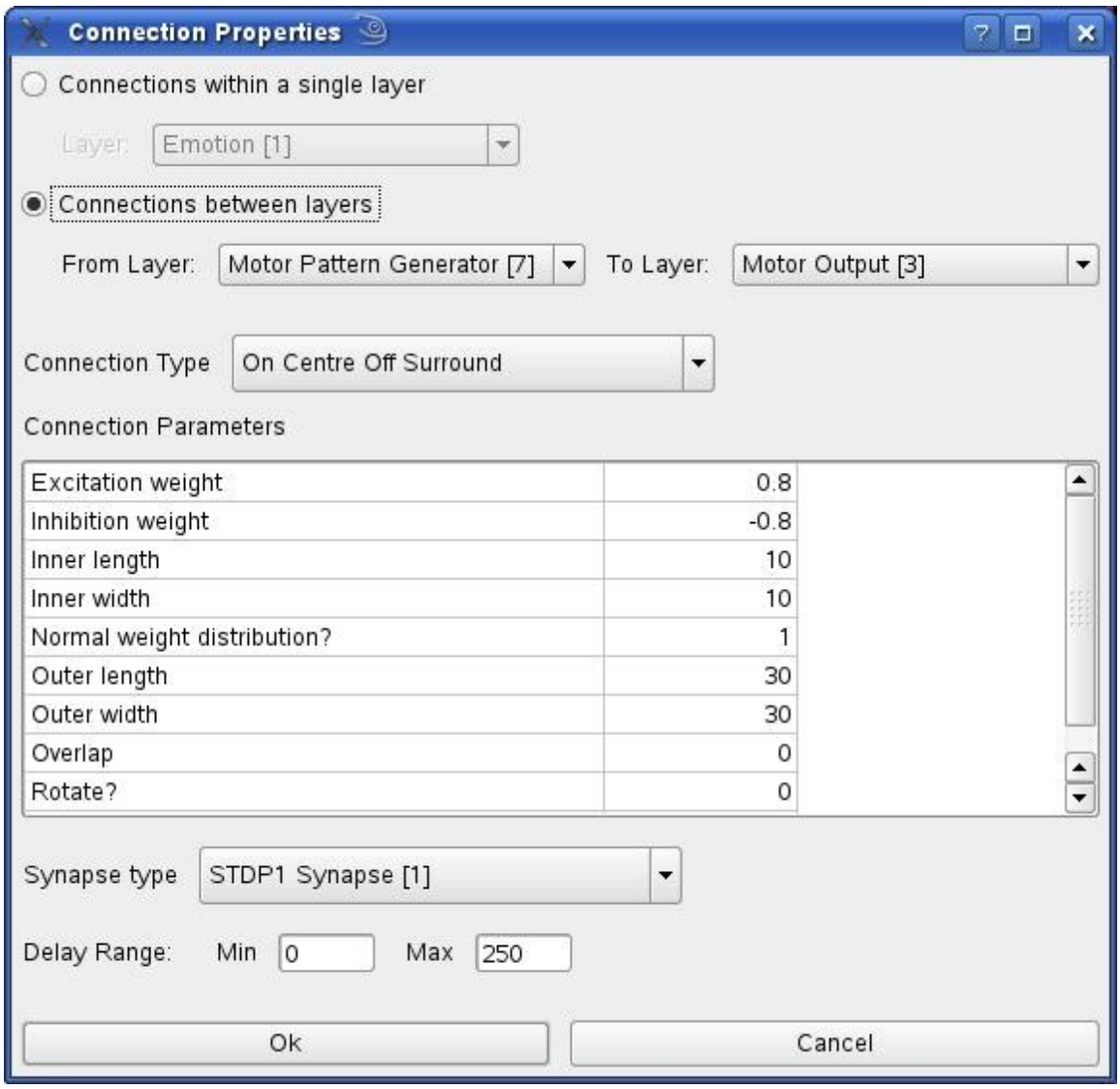


Figure 5.3. Connection Properties Dialog

The properties that can be set in this dialog are as follows:

- **Connections within a single layer/ between layers**. These radio buttons select between inter and intra layer connections. Different types of connection are available for each.
- **From layer**. The starting layer for the connection.
- **To layer**. The layer that the connection is made to.
- **Connection Type**. Several different connection types are available in the current version of SpikeStream.
    - **Simple Cortex**. Neurons are connected with short range excitatory connections and long range inhibitory connections. The parameters for this type of connection are given in Table 5.1.

Parameter	Description
Excitation connection probability	The number of neurons connected to within the excitation radius. Set to greater than 1 to increase the connection density; set to less than 1 to reduce the connection density.
Excitation radius	Select neurons within this radius for the neuron to connect to.
Excitation weight	The weight of excitation connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inhibition connection density	The proportion of neurons connected to within the inhibition radius. Set to greater than 1 to increase the connection density; set to less than 1 to reduce the connection density.
Inhibition radius	Neurons within this radius, but outside of the excitation radius minus the overlap are selected for inhibitory connections.
Inhibition weight	The weight of inhibitory connections +/- the weight range.
Normal weight distribution	Randomness in the weight is selected using a normal distribution. 1 switches normal distribution on; 0 switches it off.
Overlap	Overlap between the inhibitory and excitatory connections
Weight range	The amount by which the weights can vary randomly.

Table 5.1. Simple cortex connection parameters

- **Unstructured excitatory (inter) and Unstructured excitatory (intra).** Unstructured connections in which each neuron makes all excitatory or all inhibitory connections. The parameters for this type of connection are given in Table 5.2.

Parameter	Description
Excitation connection prob	The probability of an excitatory neuron connecting to another excitatory neuron. This parameter can vary between 0 and 1.0.
Excitation weight	The weight of excitation connections +/- the weight range. Weights can range from -1.0 to 1.0.
Excitation weight range	The range of the excitation weight.
Excitation percentage	The percentage of excitatory neurons. Ranges from 0-100.
Inhibition connection prob	The probability of an inhibitory neuron connecting to another inhibitory neuron. This parameter can vary between 0 and 1.0.
Inhibition weight	The weight of inhibitory connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inhibition weight range	The range of the inhibitory weights.

Table 5.2. Unstructured excitatory (inter) and Unstructured excitatory (intra) parameters

- **On Center Off Surround**. Rectangular connection with an excitatory centre and inhibitory surround. The to layer must be smaller than the from layer for this type of connection to work. The parameters for this type of connection are given in Table 5.3. *WARNING: Some of these parameters are not fully tested.*

Parameter	**Description**
Excitation weight	The weight of excitation connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inhibition weight	The weight of inhibitory connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inner length	The length of the central excitatory connection area.
Inner width	The width of the central excitatory connection area.
Outer length	The length of the inhibitory connection area.
Outer width	The width of the inhibitory connection area.
Overlap	Overlap between the excitatory and inhibitory connection areas.
Rotate	One layer may be rotated relative to the other one.
Weight range	The amount by which the weights can vary randomly.

T**able 5.3**. On center off surround connection parameters

- **Off Center On Surround**. Similar to on center off surround connections. Note that the to layer must be smaller than the from layer for this type of connection to work. The parameters for this type of connection are given in Table 5.4. *IMPORTANT NOTE*: *Not implemented at present.*

Parameter	**Description**
Excitation weight	The weight of excitation connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inhibition weight	The weight of inhibitory connections +/- the weight range. Weights can range from -1.0 to 1.0.
Inner length	The length of the central inhibitory connection area.
Inner width	The width of the central inhibitory connection area.
Outer length	The length of the excitatory connection area.
Outer width	The width of the excitatory connection area.
Overlap	Overlap between the excitatory and inhibitory connection areas.
Rotate	One layer may be rotated relative to the other one.
Weight range	The amount by which the weights can vary randomly.

Table 5.4 Off center on surround connection parameters

- **Unstructured**. Each neuron in the from layer is connected to a random number of neurons in the to layer. The parameters for this type of connection are given in Table 5.5.

Parameter	Description
Average weight	The weight of connections +/- the weight range. Weights can range from -1.0 to 1.0.
Connection density	The proportion of neurons connected to. This parameter can vary between 0 and 1.0.
Weight range	The amount by which the weights can vary randomly.

Table 5.5. Unstructured connection parameters

- **Virtual**. In order to run the simulation, each neuron group needs to be connected to at least one other neuron group. When there are no functional connections, virtual connections need to be created between neuron groups so that they can be synchronized in the simulation. *NOTE: The simulation may also create temporary virtual connections to enable synchronization between the layers. The creation and destruction of these does not require any intervention by the user.*
- **Topographic**. This works between two layers of exactly the same width and length and creates topographic connections between the layers. The parameters for topographic connections are given in Table 5.6.

Parameter	Description
Average weight	The weight of connections +/- the weight range. Weights can range from -1.0 to 1.0.
Overlap	When layers of different size are topographically connected there can be an overlap between each set of connections.
Rotate	One layer can be rotated relative to the other.
Weight range	The amount by which the weights can vary randomly.

Table 5.6. Topographic connection parameters

- **Synapse Type**. Selects one of the currently selected synapse classes for the connection.
- **Delay Range**. Sets the range of delay *expressed in timesteps*. The absolute value of the delay for each connection is the update time per timestep multiplied by the number of timesteps delay.

5.6 Deleting Connection Groups

Select the connection groups that you want to delete and press the “Delete” button above the connections table. Press “Ok” to confirm deletion and the connection groups will be removed from the database.

IMPORTANT NOTE: There is no undo function in SpikeStream and no method of reversing this step. Future work on SpikeStream may look into using the MySQL rollback feature to undo transactions.

5.7 Other Ways to Create Neuron and Connection Groups

The preset ways of creating and editing neuron and connection groups in SpikeStream Application are hard coded and can only be changed by modifying SpikeStream. However, it is reasonably easy to write your own programs or scripts to add new neurons or connection patterns to the SpikeStream database. The following limitations apply when doing this:

- Any pair of neurons can only have a single connection between them.
- Each neuron group can only have one connection of each type between it. Thus, there can be several connection groups of different types between two layers, but not two connection groups of the same type.
- SpikeStream can visualise neuron groups of any shape, but it is currently unable to connect patterns or devices to non-rectangular neuron groups, or to provide live monitoring of non-rectangular neuron groups.

6. Viewing Neural Networks

6.1 Viewer Tab

The Network Viewer (see Figure 6.1) enables networks to be viewed in three dimensions. This three dimensional window is permanently on the right hand side of the screen and its size can be adjusted by grabbing the dividing bar. The Network Viewer tab has controls that enable you to view different aspects of the connections and set the rendering properties.

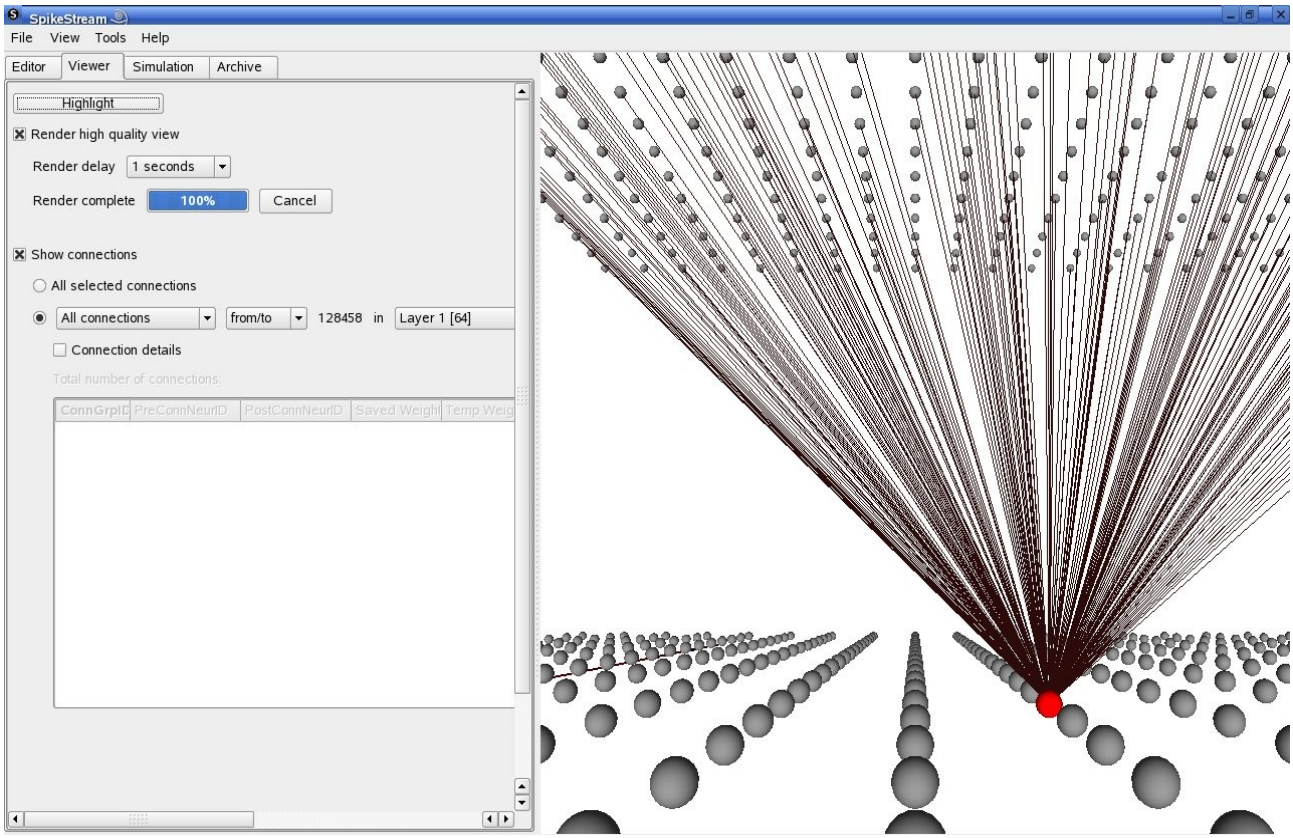


Figure 6.1. Network Viewer tab (left) and Network Viewer (right)

The controls available in the Network Viewer Tab are covered in the next few sections.

6.1.1 Highlight

Clicking on the highlight button launches the Highlight Dialog shown in Figure 6.2. Type or paste in a list of comma separated neuron IDs that you want to highlight and click on “Add Highlight” to highlight them. The colour can be changed by clicking on the colour field. Multiple groups of neurons can be highlighted in different colours.

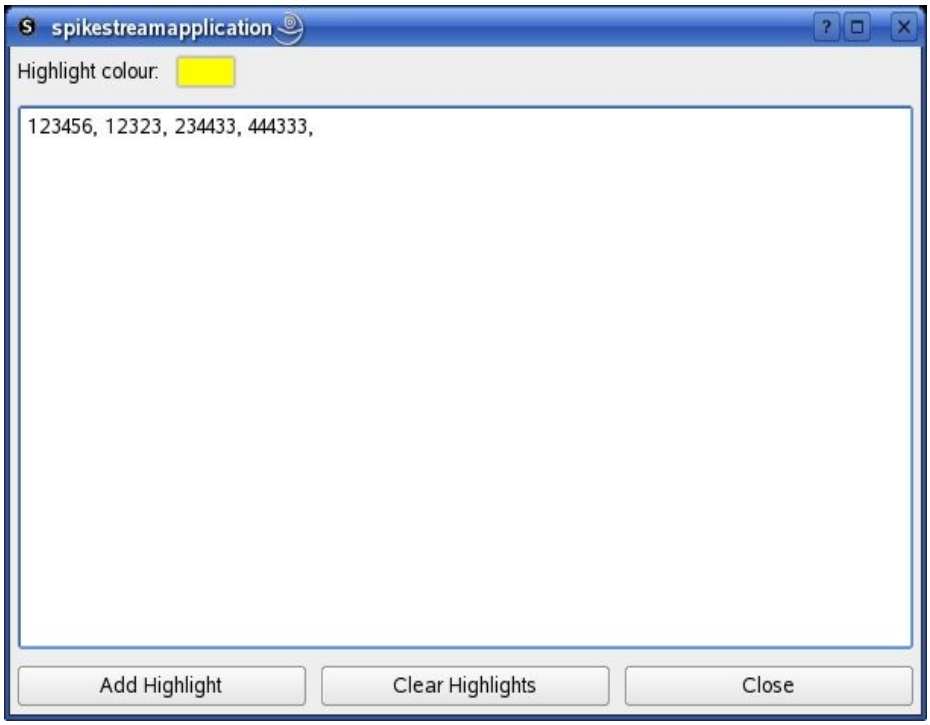


Figure 6.2 Highlight Dialog

6.1.2 Render Settings

Normally neurons are drawn using simple vertices, which considerably speeds up the rendering time. However, if you want a more attractive view, you can check this box to draw neurons as grey spheres. The render delay sets the time between the last navigation event in the Network Viewer and the start of the render.

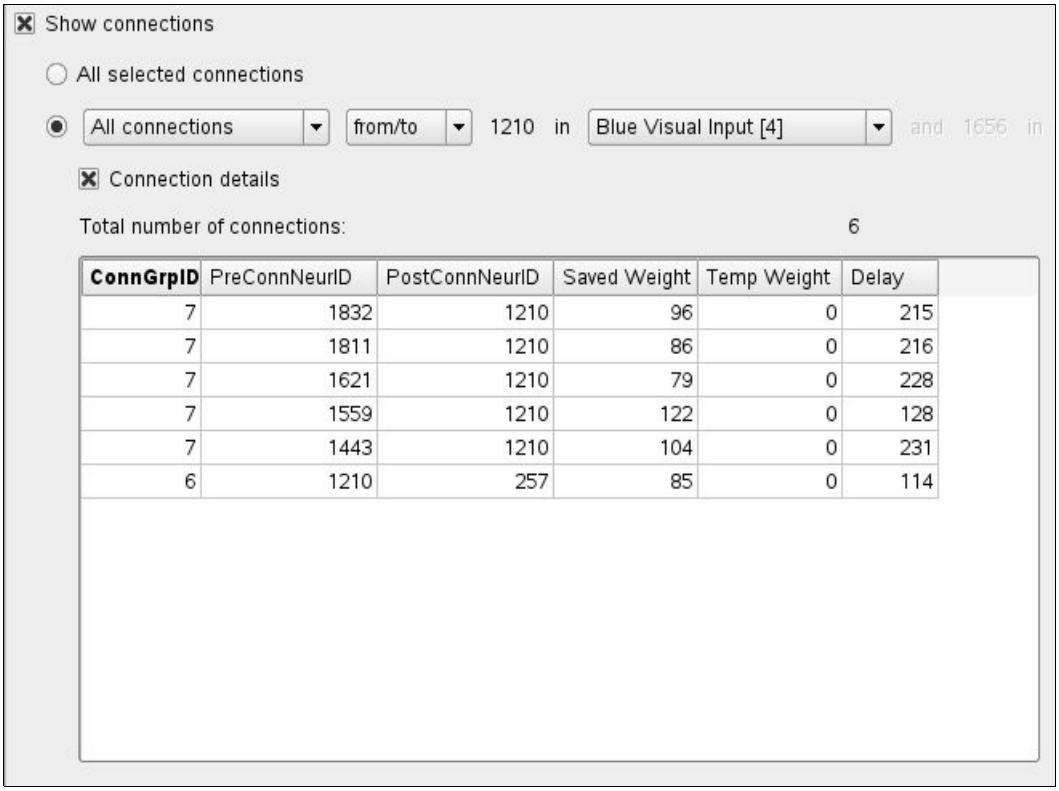
6.1.3 Connection Settings

When the Show Connections check box is selected the Network Viewer displays all of the connections that are set as visible in the Connection Group Table. This part of the Network Viewer tab is very useful for showing different aspects of the connections between neurons and it is also used to select the neurons for monitoring or noise injection in the Simulation tab. If you want to select a subset of the connections for viewing, the following options are available:

- **All connections.** Shows positive and negative connections.
- **Positive connections.** Only connections with positive weights are shown.
- **Negative connections.** Only connections with negative weights are shown.
- **from/to**. Connections from and to the selected neuron in the selected neuron group are shown

- **from**. Connections from the selected neuron in the selected neuron group are shown
- **to** . Connections to the selected neuron in the selected neuron group are shown
- **between**. Connections between the first selected neuron and the second selected neuron are shown. Use this mode to select an individual synapse for monitoring during a simulation.

The connection details check box displays information about the selected connections (see Figure 6.3. In this table, “Saved Weight” is the weight that is loaded up at the beginning of a simulation as the synapses' starting weight. As the simulation progresses, this weight may change and the user can view the current value of the weights by pressing “View Weights” in the Simulation tab. The synapse's current weight is then visible in the “Temp Weight” column of this table. If the user chooses to permanently save the weights during a simulation, their values are written to the Saved Weight field and will become the starting weights when the simulation is next initialised.



ConnGrpID	PreConnNeurID	PostConnNeurID	Saved Weight	Temp Weight	Delay
7	1832	1210	96	0	215
7	1811	1210	86	0	216
7	1621	1210	79	0	228
7	1559	1210	122	0	128
7	1443	1210	104	0	231
6	1210	257	85	0	114

Figure 6.3. Connection Details Table

6.2 Network Viewer

The Network Viewer shows all of the visible neurons and connections in three dimensions. This display starts out with the Z axis vertical, the X axis horizontal and to the right and the Y axis going

into the display away from the viewer. You can navigate around this window using the following controls:

- **Arrow-left**. Moves camera left.
- **Arrow-right.** Moves camera right.
- **Arrow-up**. Moves camera up.
- **Arrow-down**. Moves camera down.

- **Ctrl** + **Arrow-left**. Rotates camera left.
- **Ctrl** + **Arrow-right**. Rotates camera right.
- **Ctrl** + **Arrow-up**. Rotates camera up.
- **Ctrl** + **Arrow-down**. Rotates camera down.

- **Ctrl** + =. Zooms in.
- **Ctrl** + **-**. Zooms out.
- **Ctrl + Y**. Zooms out to show all layers.

When viewing connections from/to, from and to an individual neuron, the neuron will be highlighted in red and the selected neuron can be changed using the following controls:

- **ALT + Arrow-right**. Selects the next neuron within the group moving along X positive.
- **ALT + Arrow-left**. Selects the next neuron within the group moving along X negative.
- **ALT + Arrow-up**. Selects the next neuron within the group moving along Y positive.
- **ALT + Arrow-down**. Selects the next neuron within the group moving along Y negative.

When viewing connections *between* two individual neurons, the from neuron will be highlighted in red and the selected neuron can be changed using the controls that have just been outlined. The to neuron will be highlighted in green and the selected to neuron can be changed using the following controls.

- **SHIFT** + **ALT** + **Arrow-right**. Selects the next neuron within to the group moving along X positive.
- **SHIFT** + **ALT + Arrow-left**. Selects the next neuron within the to group moving along X negative.
- **SHIFT** + **ALT + Arrow-up**. Selects the next neuron within the to group moving along Y positive.

- **SHIFT** + **ALT + Arrow-down**. Selects the next neuron within the to group moving along Y negative.

WARNING: Occasionally the Network Viewer loses keyboard focus, which may cause the keyboard to control other aspects of SpikeStream. This is rarely serious, but I have accidentally quit the application on occasions by inadvertently navigating through the file menu. Click on the Network Viewer to restore keyboard focus.

6.3 View Menu

The view menu on the main menu bar allows you to selectively refresh information in SpikeStream:

- **View->Reload devices Ctrl+D**. Reloads the list of devices in the Simulation tab.
- **View->Reload patterns Ctrl+P**. Reloads the list of patterns in the Simulation tab.
- **View->Reload everything Shift+F5**. Reloads everything, including neuron and connection groups, parameters, patterns and devices.

7. Running a Simulation

7.1 Simulation Tab

The Simulation tab (see Figure 7.1) is used to control all aspects of a simulation.

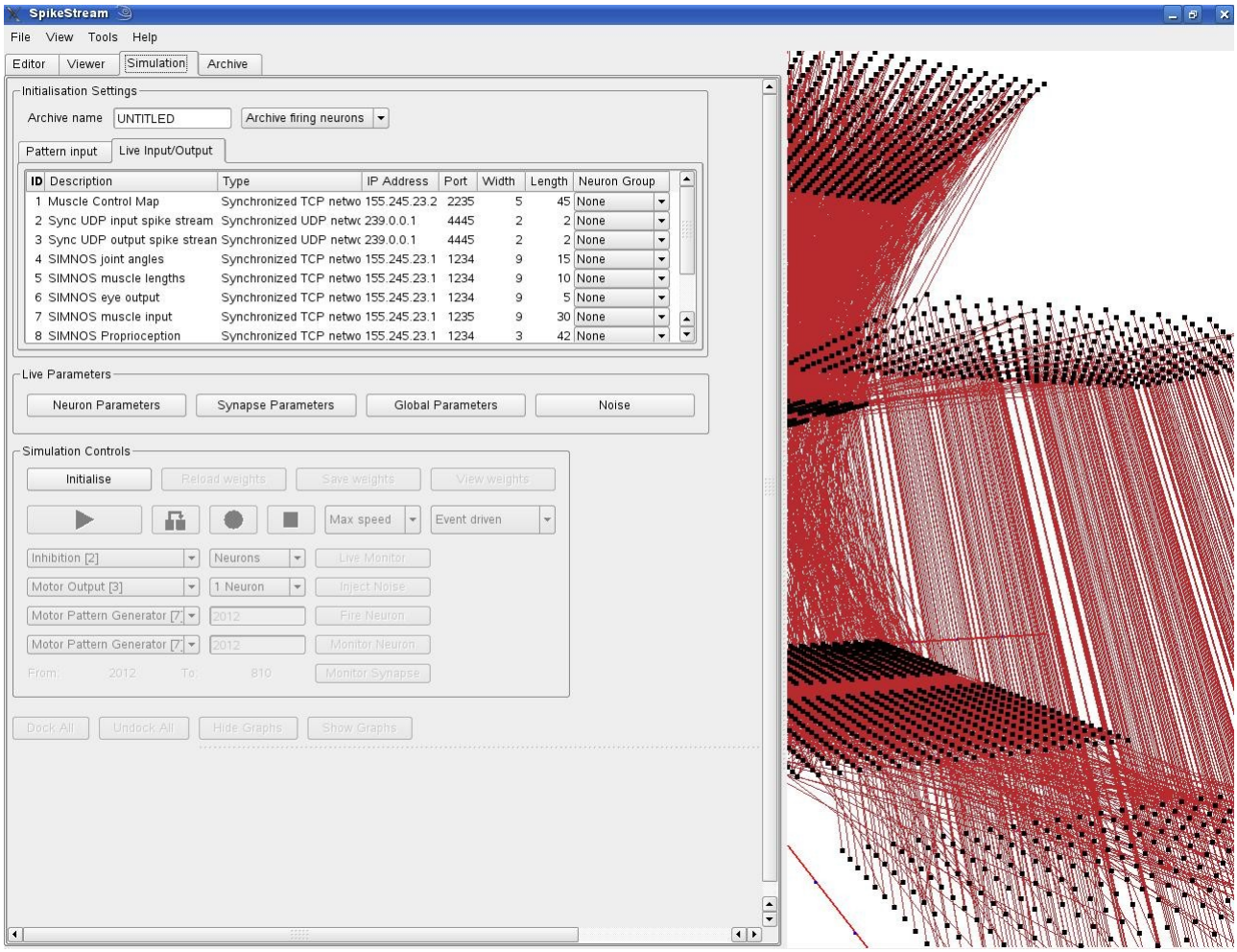


Figure 7.1. Simulation tab

7.2 Archive Name and Type

At the top of the Simulation tab is a box where you can enter a name for the archive. This archive will only be stored if you record data from the simulation. There is also an combo box that enables you to select between recording the firing neuron patterns from a layer or the spikes emitted from a layer. The firing neurons option is recommended because it has been more thoroughly tested. The archive name can be changed at a later point using the Load Archive Dialog.

7.3 Patterns and Devices

The next part of the Simulation tab is another set of tabs that let you connect patterns and devices up to layers in the simulation. Each of the combo boxes in these tables only displays the layers that are the correct size for the pattern or device. Selecting the layer in the combo box will connect the pattern or device up to the layer when the simulation is initialised. If you add a new device to the Devices table you can refresh the devices table by clicking on "View->Reload devices" or pressing CTRL+D. At the bottom of the pattern table is a text box where you can set the number of time steps between each pattern. For example, if you set this to ten, a pattern will be applied every ten time steps. This is particularly important when you are using patterns that are spread over time. See Chapter 9 for more information on devices and Chapter 10 for more information about patterns.

7.4 Parameters

Parameters for the simulation are set using the four buttons in the "Parameters" section of the Simulation tab.

7.4.1 Neuron Parameters

Clicking on the "Neuron Parameters" button brings up the dialog shown in Figure 7.2, where you can set the parameters for the simulation. This dialog edits the neuron parameters table in the database that matches the neurons' type and these parameters can be changed at any point during a simulation run.

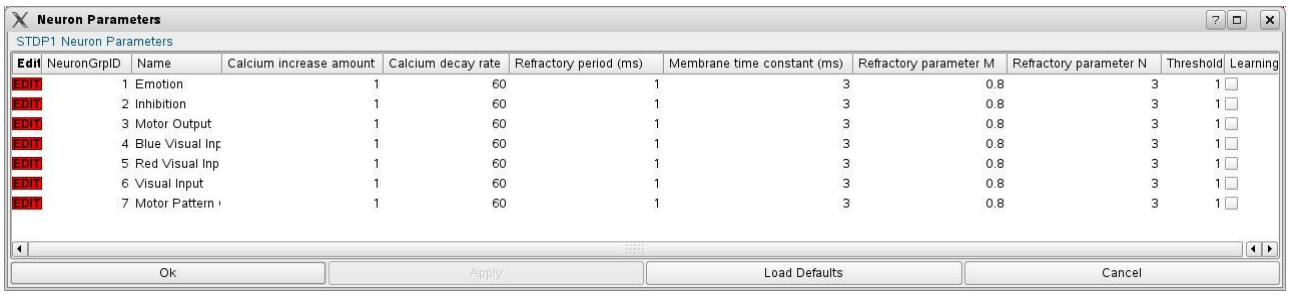


Figure 7.2. Neuron Parameters Dialog

To change the parameters, click on the edit button for a particular layer and an Edit Neuron Parameters Dialog will be launched that enables you to adjust the parameters (see Figure 7.3).

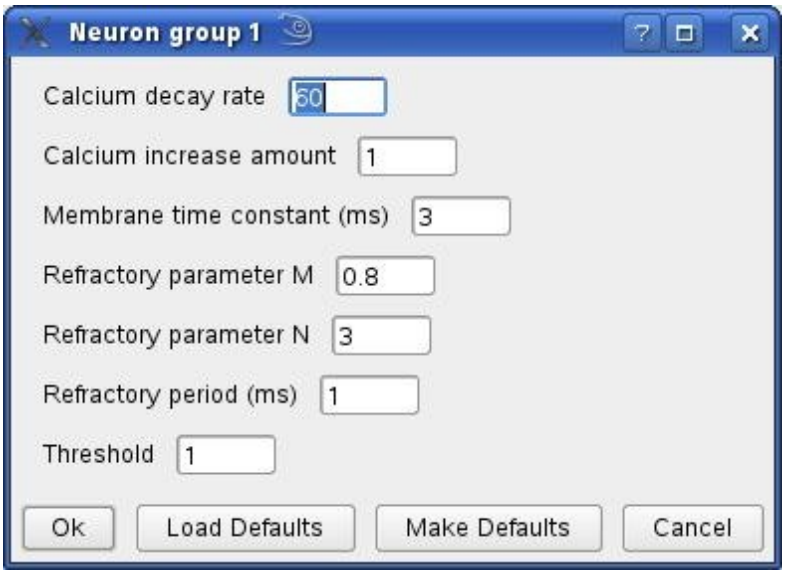


Figure 7.3. Edit Neuron Parameters Dialog

Pressing "Ok" in this second dialog updates the Neuron Parameters Dialog, but *will not update the simulation until you press "Ok" or "Apply" within the Neuron Parameters Dialog*. Boolean parameters are set using the check boxes within the Neuron Parameters Dialog.

IMPORTANT NOTE: The "Load Defaults" button is not implemented in the Neuron Parameters Dialog and the "Make Defaults" button has not been implemented in the Edit Neuron Parameters Dialog.

7.4.2 Synapse Parameters

The editing of synapse parameters proceeds in an identical way to the editing of neuron parameters.

7.4.3 Global Parameters

This dialog (see Figure 7.4) controls parameters that are global to the simulation. Checking "Run simulation in real time" will update the simulation clock in real time instead of using the time step duration value. "Time step duration" enables you to set the amount of time that is simulated by each time step. Smaller values will lead to a more accurate simulation, but may also increase the amount of time taken to compute the simulation.

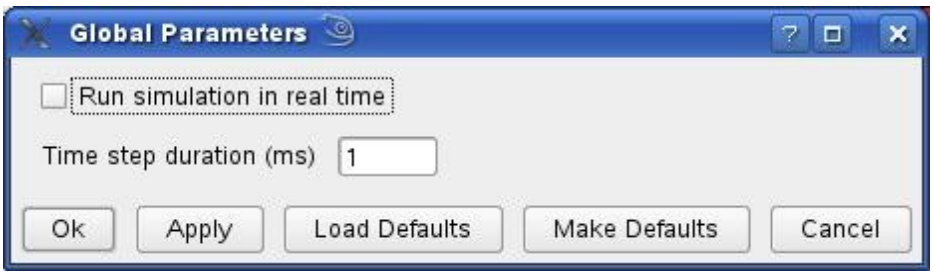


Figure 7.4. Global Parameters Dialog

7.4.4 Noise

This dialog (see Figure 7.5) enables you to add random noise to the neuron groups.

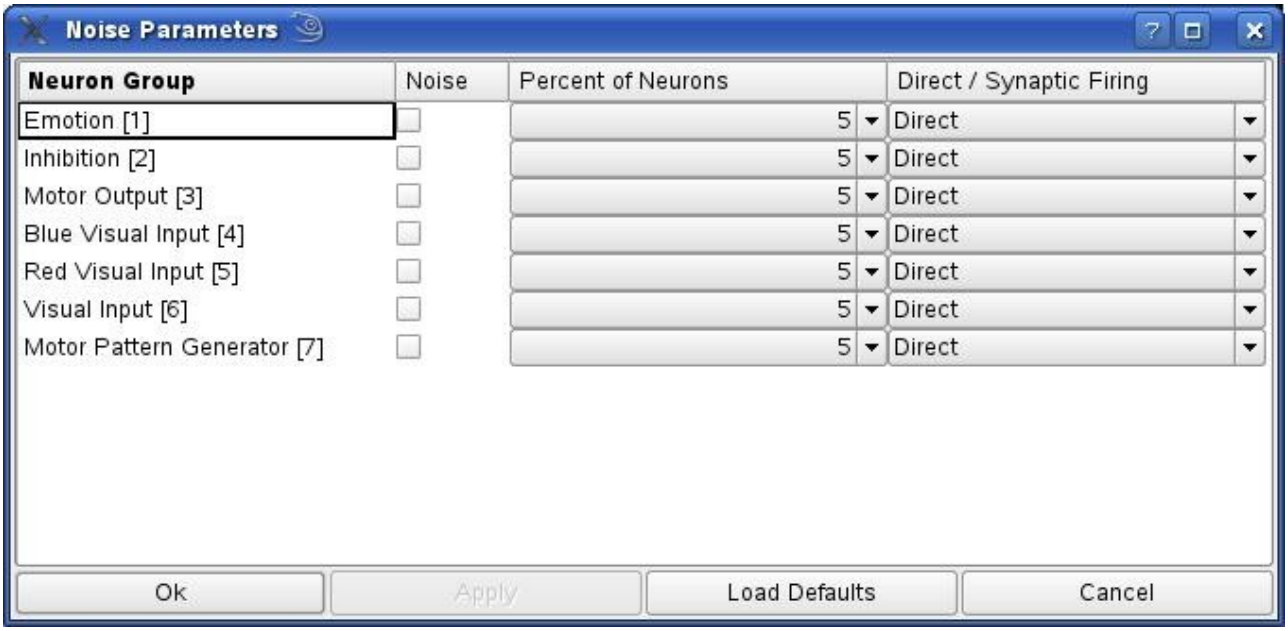


Figure 7.5. Noise Dialog

The second column enables or disables noise for the neuron group. The third column selects the percentage of neurons that will be randomly selected from each neuron group at each time step. There is also a "random" option that selects a random percentage of neurons at each time step. The last column selects between direct and synaptic noise. In direct noise mode, the selected neurons are directly fired by the simulation. In synaptic noise mode, the specified synaptic current is injected into the neuron at each time step, which may or may not lead to firing.

7.5 Simulation Controls

The next set of controls are for running and monitoring of simulation and for the manual injection of noise. These controls are only enabled when the simulation is initialised (see Figure 7.6).

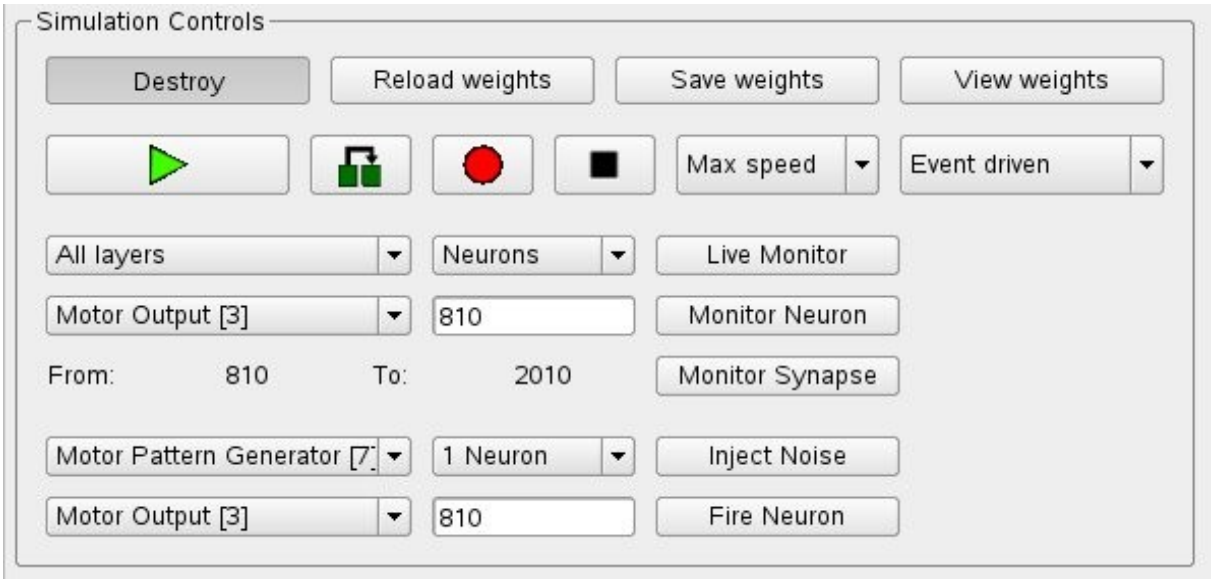


Figure 7.6. Simulation controls

7.5.1 Initialise / Destroy

When initialise is pressed, pvm is used to launch the simulation across all the hosts that have been added to the virtual machine. These are created as separate tasks running in parallel, with one task per neuron group. An extra task is created for the archiving of the simulation. Pressing "Destroy" causes all of these tasks to exit.

7.5.2 Weight Buttons

During a simulation run these buttons offer the following functions:

- **Reload weights**. Requests each task to reload its weights from the database.
- **Save weights**. Requests each task to save its current weights to the database.
- **View weights.** Requests each task to save its current weights to the "Temp Weight" field in the database. This enables the user to view the weights without permanently changing them.

7.5.3 Transport Buttons

The simulation is run using a standard set of transport buttons:

- **Play**. Plays and stops the simulation.
- **Step**. Advances to the next time step. Strange behaviour with pvm message passing can lead each step to take a second or two.
- **Record**. Records the simulation using the specified archive name.
- **Stop**. Stops the simulation.

A combo box after the stop button can be used to slow the simulation down, which is extremely useful for monitoring what is going on in the simulation. The last combo on this row is used to control the update mode of the simulation:

- **Event driven**. The fastest update mode. Neuron and synapse classes are only updated when they receive a spike.
- **Update all neurons**. All neuron classes are updated at each time step. Synapses are only updated when they receive a spike. Useful for neural models that display spontaneous activity.
- **Update all synapses**. All synapse classes are updated at each time step. Neurons are only updated when they receive a spike.
- **Update everything**. All neuron and synapse classes are updated at each time step. In this mode, SpikeStream operates like a synchronous simulator.

7.5.4 Monitoring

The next set of controls are used to monitor what is going on in the simulation.

- **Live Monitor.** Launches a window displaying the firing state of the selected neuron group or all the neuron groups. This window can display the spikes emitted by the neuron group or the firing of the neurons in the neuron group.
- **Monitor Neuron**. Each neuron class can define its own set of variables for live monitoring. Select a neuron using the Network Viewer or type in a neuron id and click this button to draw a live graph of the monitored variables for the neuron (see Figure 7.7). *NOTE: If this is launched part way through a simulation, it may take a little while to adjust itself.*

- **Monitor Synapse**.. Each synapse class can define its own set of variables for live monitoring. To select a synapse you need to set the Network Viewer tab to 'between' mode. You should have a green neuron and a red neuron highlighted. Select a synapse using the Network Viewer and click “Monitor Synapse” to draw a live graph of the monitoring variables for the synapse. *NOTE: If this is launched part way through a simulation, it may take a little while to adjust itself.*

Closing these windows stops the monitoring data being sent from the tasks simulating the neuron group.

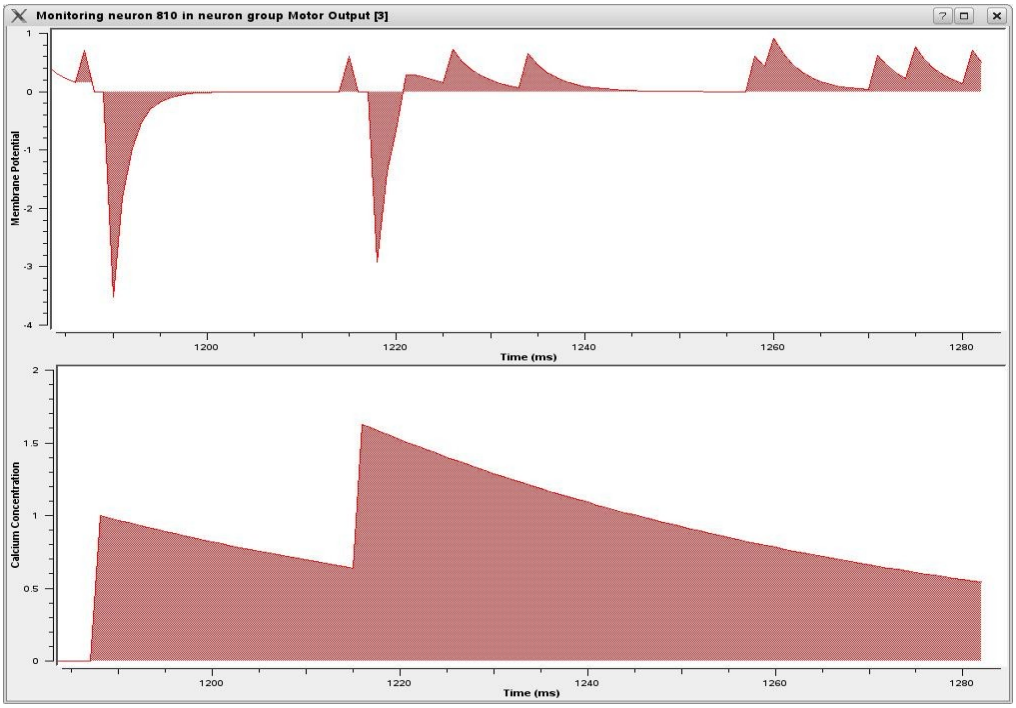


Figure 7.7. Graphs of monitored neuron variables

NOTE: The values in this graph are sampled every time step so with a high time step value of 10ms, for example, you may not see any change on the membrane potential in response to incoming spikes because the neuron will have reset itself to zero at each time step.

7.5.5 Noise Injection

Controls that can be used to manually inject noise into a neuron group within a single simulation step:

- **Inject Noise.** Fires the specified percentage of neurons once within a simulation step.
- **Fire Neuron**. Fires the specified neuron once within a simulation step. The neuron's id can be typed into the field or selected using the Network Viewer.

7.5.6 Docking Controls

A number of buttons are available to selectively hide and show monitoring information.

- **Dock All**. Places all live monitor windows in the docking area. These windows will continue to display the neuron patterns whilst they are in the docking area and they can be dragged around and rearranged.
- **Undock All**. Restores all live monitor windows to their original location.
- **Hide Graphs**. Makes all graphs invisible and switches their plotting off.
- **Show Graphs**. Makes all the current graphs visible and switches their plotting on.

7.6 Network Probes

Clicking on "Tools->Probe manager" launches a dialog to manage the probes. Network Probes are designed to run alongside the simulation and carry out actions on the neural network for testing purposes. For example, a network probe might be created to stimulate parts of the network with noise in order to identify its causal dependencies. *NOTE: This feature is still under development and should be ignored.*

8. Archives

8.1 Archive Tab

The recording of archives is carried out in the Simulation tab. Archives are played back in the Archive tab shown in Figure 8.1.

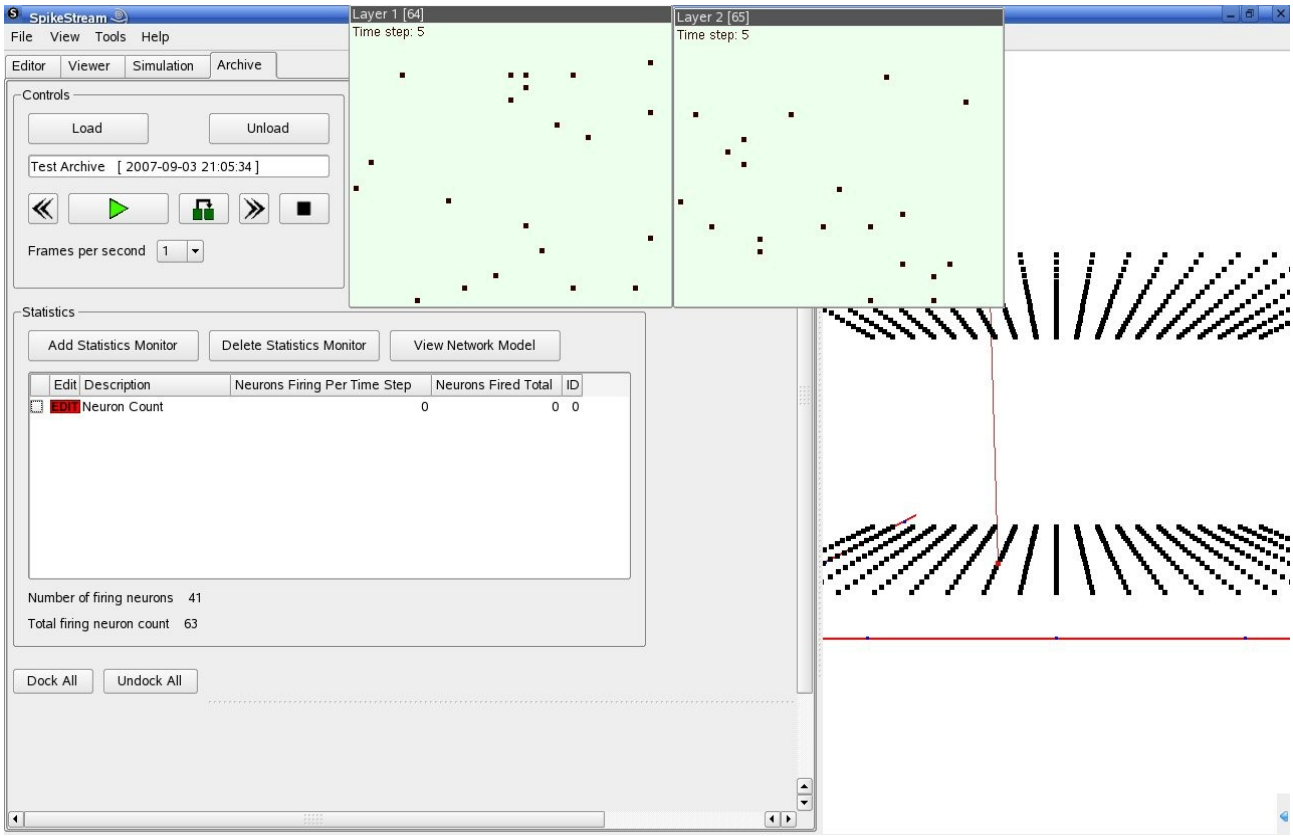


Figure 8.1. Archive tab

8.1.1 Loading and Playing Back an Archive

To load an archive press the "Load" button, which will open up the Load Archive Dialog, shown in Figure 8.2, with controls to rename and delete archives. When you have selected your archive and pressed "Ok", the archive will be loaded and can be replayed, stepped through, rewound etc. using the controls available in the Archive tab.

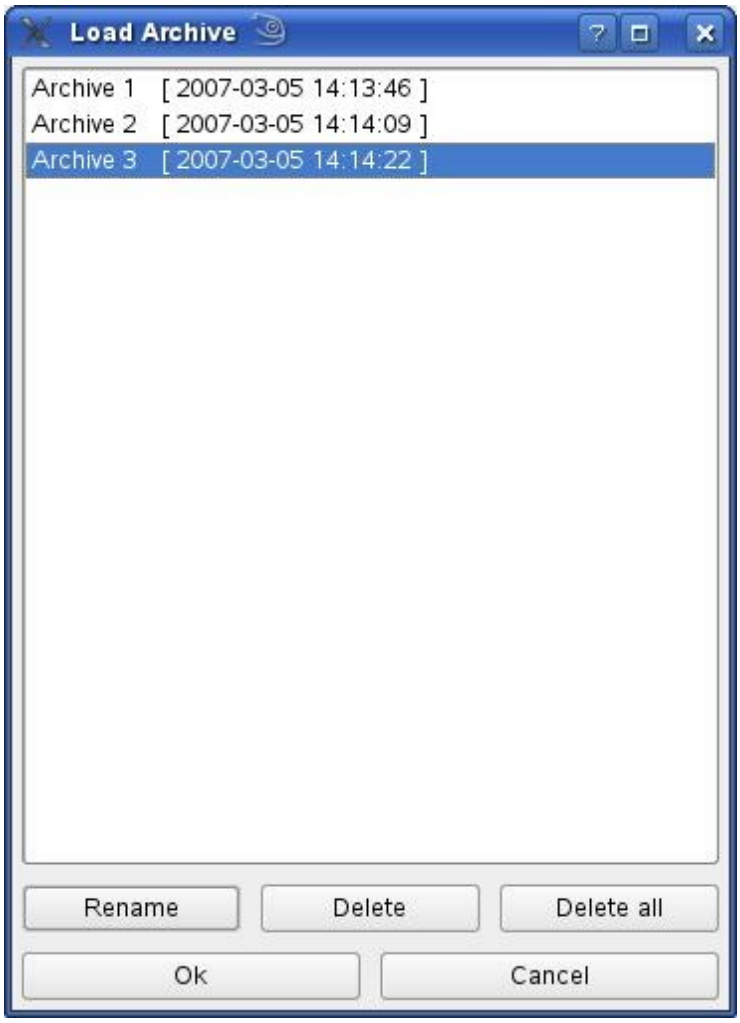


Figure 8.2. Load Archive Dialog

8.1.2 Archive Statistics

Statistics about the archive can be gathered by adding a statistics monitor to count the number of times a neuron or range of neurons fires or the number of times neurons fire in a particular neuron group. Clicking on the "Add Statistics Monitor" button launches the dialog shown in Figure 8.3. In this dialog you can choose to monitor the number of times neurons fire in a particular layer or count the number of times one or a number of neuron IDs fire, which is done by adding the neuron IDs as a comma separated list. OR, AND and range operators are supported, for example: 12121 & 12121, 1323-56565, 123213|098098.

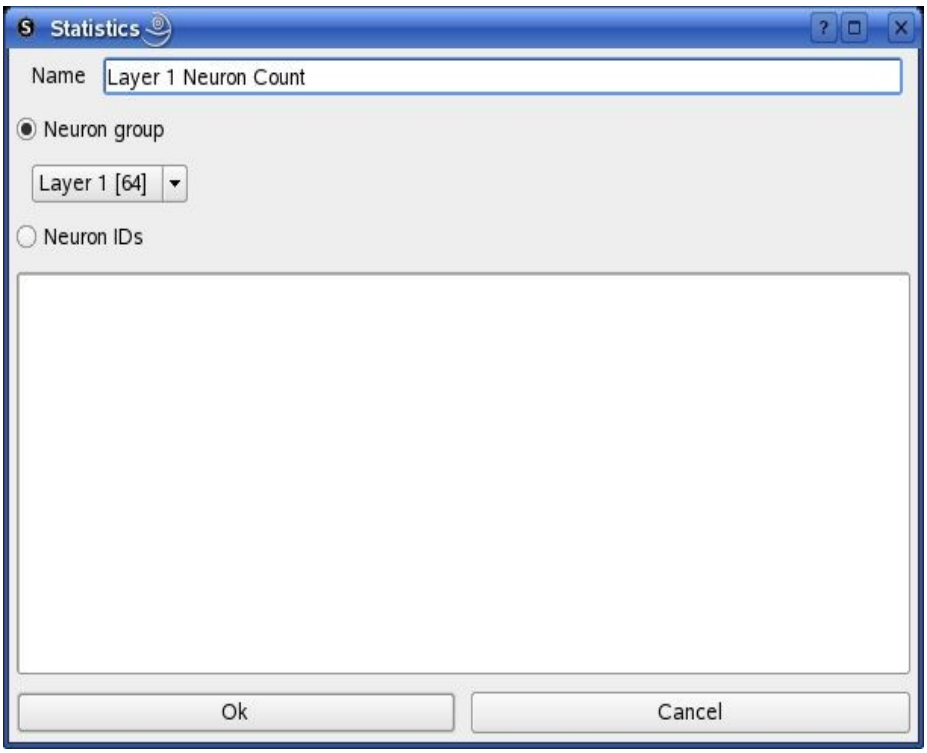


Figure 8.3. Archive Statistics Dialog

There is also a button that allows you to view the XML network model associated with the archive (see next section), which may be different from the network model that is currently loaded into SpikeStream.

8.2 Archive Structure

Each archive contains a summary of the neuron groups stored in XML format in the NetworkModels table. An example of a network model is given below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<neural_network>
      <neuron_groupid="19">
            <name>Learner</name>
            <start_neuron_id>161429</start_neuron_id>
            <width>1</width>
            <length>1</length>
            <location>10,1,10</location>
            <spacing>1</spacing>
            <neuron_type>6</neuron_type>
      </neuron_group>
      <neuron_group id="17">
```

```
            <name>Generator</name>
            <start_neuron_id>161427</start_neuron_id>
            <width>1</width>
            <length>1</length>
            <location>1,1,1</location>
            <spacing>1</spacing>
            <neuron_type>6</neuron_type>
      </neuron_group>
</neural_network>
```

Each network model is associated with one or more rows of firing patterns in the NetworkData table, which are also stored in XML format. An example of NetworkData for one time step is given below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<network_pattern>
      <neuron_group id="17">161427</neuron_group>
      <neuron_group id="18">161428</neuron_group>
</network_pattern>
```

9. Devices

9.1 Introduction

SpikeStream can send and receive spikes across a network to and from an external device, such as a real or virtual robot, camera, etc. This feature is still under development and only the TCP synchronized method has been fully tested between SpikeStream and the SIMNOS virtual robot.

9.2 Sending and Receiving Spike Messages

A number of different methods exist for sending and receiving spike messages across a network. Not all of them have been implemented and the synchronized TCP methods have been most thoroughly tested. The next few sections outline the general procedure for sending and receiving messages. More detail about this can be found in the SpikeStream Simulation code.

9.2.1 Synchronized TCP Network Input

This method uses TCP to send and receive spike packets across the network. This is designed to work with devices that run in their own simulation time, such as the SIMNOS virtual robot (see section 9.4), and it enables the two devices to remain perfectly synchronized. The procedure for receiving this type of message is as follows:

- Wait to receive packet containing the data.
- Unpack the first four bytes, which contain the number of spikes in the message.
- Unpack the spikes, each of which is four bytes long.
- The first byte is the X position of the spike within the layer.
- The second byte is the Y position of the spike within the layer.
- The third and fourth byte contain the time delay of the spike. *WARNING: This is untested for non-zero values and should be set to zero for the moment.*
- When all spikes have been unpacked send a confirmation message containing a single byte to confirm that the data has been received. This has the value SPIKESTREAM_DATA_ACK_MSG (defined in $SPIKESTREAM_ROOT/include/DeviceMessages.h), which is currently set to 1, but may change.
- Fire neurons in the layer that received spikes from the device.

Since the layer connected to the device will not complete its simulation step until it has updated itself, this method synchronises SpikeStream with the external device, which should also wait until it receives the acknowledgment message.

9.2.2 Synchronized TCP Network Vision Input

This method is similar to the previous one, except that no delay is included within the packet and the X and Y positions are defined using two bytes. The procedure for receiving this type of message is as follows:

- Wait to receive packet containing the data.
- Unpack the first four bytes, which contain the number of spikes in the message.
- Unpack the spikes, each of which is four bytes long.
- The first two bytes are the X position of the spike within the layer.
- The next two bytes are the Y position of the spike within the layer.
- When all spikes have been unpacked send a message containing a single byte to confirm that the data has been received. This has the value SPIKESTREAM_DATA_ACK_MSG (defined in $SPIKESTREAM_ROOT /include/DeviceMessages.h), which is currently set to 1, but may change.
- Fire neurons in the layer that receive spikes from the device.

Since the layer connected to the device will not complete its simulation step until it has updated itself, this method synchronises SpikeStream with the external device, which should also wait until it has received the acknowledgment message.

9.2.3 Synchronized TCP Network Output

This method sends spikes in a synchronized manner from SpikeStream to an external device. The procedure is as follows:

- Add the number of spikes as a four byte value to the packet.
- Add the spikes to the packet. The first byte is the X position, the second byte is the Y position and the next two bytes are the delay, currently not used.
- Send the packet.
- Wait to receive a packet containing an acknowledgment that the data has been received. This has the value DEVICE_DATA_ACK_MSG (defined in $SPIKESTREAM_ROOT/ include/ DeviceMessages.h), which is currently set to 3, but may change.

9.2.4 Synchronized UDP Network Input

This method creates a loose synchronization between the external device and SpikeStream by timing the interval between spike packets and slowing the simulator down to match. This method only works if the device can slow itself down as well. This method has been implemented on SpikeStream, but it has not been fully tested and some tweaking of the SpikeStream Simulation code may be necessary to get it working. The basic approach is as follows:

- The receive method runs as a separate thread which receives the spike messages and unpacks them into a separate buffer.
- The first two bytes of each packet contain the synchronization information. The first 7 bits are the time step count on the external device. This can overflow without problems since it is there to indicate the rate of increase of the time steps in the external device. The remaining bit is a flag to indicate whether the external device was delaying itself on the previous time step.
- The rest of the packet is filled with spikes, with the first byte being the X position, the second byte the Y position and the next two bytes a delay value, which is not currently used.
- When the packet has been unpacked, the receive method calculates the update time per time step for the external device.
- When SpikeStream Simulation completes a simulation step, it sleeps if its own update time per time step is less than that of the external device and if the external device is not delaying itself.
- The SynchronizationDelay table in the Devices database is used to coordinate delay information between independent SpikeStream tasks.

UDP is a potentially lossy method of transmission and the synchronization is also approximate. This makes this approach a useful halfway step between the loss free TCP synchronization and the potentially highly lossy sending and receiving of information to and from a live hardware device, such as a robot, which is interacting with the real world.

9.2.5 Synchronized UDP Network Output

This method is virtually identical to synchronized UDP network input. SpikeStream needs both input and output connections to a device to make this synchronization method work.

9.2.6 Asynchronous UDP Network Input/ Output

This method has been designed for using SpikeStream with a live device, but has not yet been implemented. The procedure is something like the following.

- Input spikes are received by a separate thread that unpacks them into a buffer, which is used to fire the neurons at each time step.
- Output spikes are transmitted at the end of each time step.

When it is implemented, the code will be similar to that used for the synchronized UDP input and output, only without the delay.

9.3 Adding Devices

The Devices table in the Devices database contains a list of available devices that SpikeStream can connect to and details about any new devices should be added here. The communication protocol between SpikeStream and the device is determined by the Type field in this table. Definitions of the different device types can be found in $SPIKESTREAM_ROOT/include/DeviceTypes.h. When a device is selected in the Simulation tab, SpikeStream will attempt to connect to it using the information provided. The “Firing Mode” option in the Devices table in the Simulation tab is used to select whether the spikes from the device fire the neuron directly or inject the specified post synaptic potential into the neuron.

9.4 SpikeStream and SIMNOS

9.4.1 Overview

The main external device that has been used and tested with SpikeStream is the SIMNOS virtual robot created by Richard Newcombe, shown in Figure 9.1.

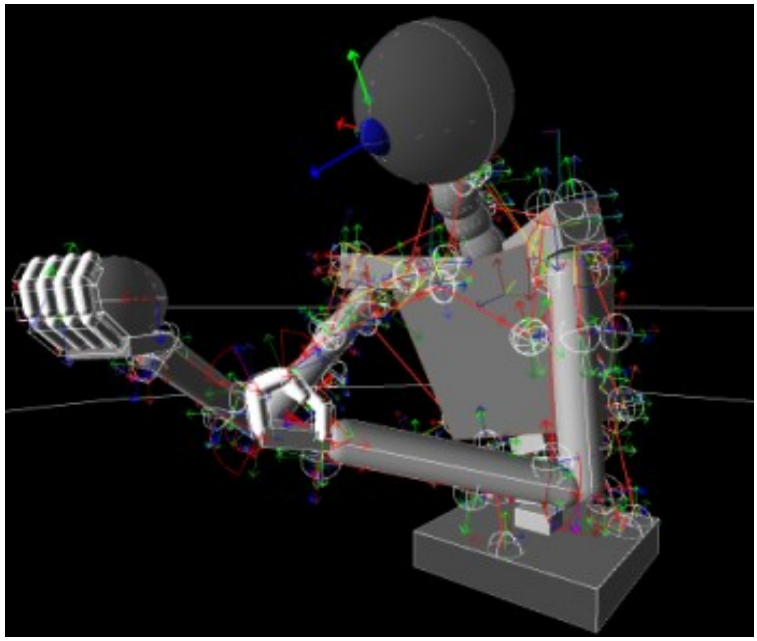


Figure 9.1. SIMNOS virtual robot

SIMNOS is a humanoid anthropomimetic robot whose body is inspired by the human musculoskeletal system. Information about muscle length, joint angles and visual information (available with a wide variety of preprocessing methods) is encoded by SIMNOS into spikes using a selection of methods developed by Newcombe (Gamez, Newcombe, Holland & Knight, 2006) and passed across the network to SpikeStream. SIMNOS can also receive muscle length data from SpikeStream in the form of spiking neural events, which are used to control the virtual robot. Together SIMNOS and SpikeStream provide an extremely powerful way of exploring sensory and motor processing and integration. More information about SIMNOS can be found at www.cronosproject.net, SIMNOS will be released soon and anyone interested in using it should contact Richard Newcombe (r.a.newcombe@gmail.com) if they would like a free copy of the current version.

9.4.2 SIMNOS Device Database

The Devices database works a little differently when you are using SIMNOS and SpikeStream together. In this case, the Devices table in the Devices database is created automatically by the SIMNOS spike servers, which enter their information into the Devices and SIMNOSSpikeReceptors tables when they start. To use SIMNOS and SpikeStream you will need to enter the details of the SIMNOS Device database into your spikestream.config file on the main workstation. You will know that you are connecting correctly if you see the four entries in the Devices table shown in Figure 9.2 (the exact entries depend on the configuration of SIMNOS):

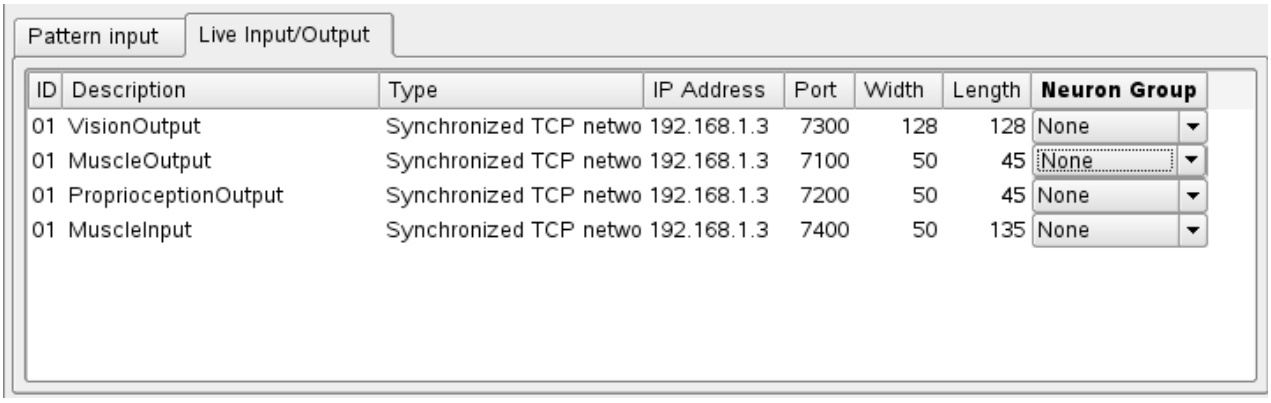


Figure 9.2. SIMNOS device entries

When using SIMNOS, you need to manually create the SynchronizationDelay and SIMNOSReceptors tables in the SIMNOS Devices database by pasting in the appropriate SQL from Devices.sql.

9.4.3 SIMNOS Receptors and Components

Information is exchanged between SIMNOS and SpikeStream in the form of relatively large layers, which connect to layers of equivalent size within the simulator. However, in many cases one wants to connect neuron groups up to part of this incoming information, such as the data coming from a single arm. It is to solve this kind of problem that the SIMNOS Receptors and Components framework was created. The SIMNOSSpikeReceptors table contains a list of the receptors that are available in SIMNOS, which are associated with a particular device. The SIMNOS Components table consists of lists of receptors, which together constitute a SIMNOS component. These lists of receptor IDs could correspond to the head, neck, arm, part of the visual field or any other abstraction that you want to make of the data from a particular device. Entries in the SIMNOSComponents database have to be created manually by the user and they can then be used to connect a neuron group up to a part of an input or output layer, as explained in the next section.

9.4.4 Using SIMNOSComponents

1. Create a layer that matches the input width and length of the SIMNOS device. For this example we will create a layer to connect to the Muscle Output of SIMNOS, which is currently 50 neurons wide and 45 neurons long (NOTE: The width varies depends on the spike conversion settings in SIMNOS).
2. Create an entry in the SIMNOSComponents database listing the receptors that you want to connect to in this layer. You need to look in SIMNOSSpikeReceptors table for the receptor IDs, which are associated with a description of the receptor. For example, to connect to the

first third and fourth receptor in the SIMNOS muscle output, we need to add an entry as follows: `INSERT INTO SIMNOSComponents (Name, ReceptorIDs, Width, Length) VALUES ("Example component description", "2001,2003,2004", 50, 3);`

3. Click on the “Add Neurons” button to launch the Neuron Group Properties Dialog, enter a name for the layer and select “SIMNOS Component from the “Neuron group type” combo box. The Neuron Group Properties Dialog should look like Figure 9.3.
4. Since there is only one component and one input layer, you don't have any choices in the other combo boxes and you just have to set a location for the new layer.
5. Press “Ok” and you will be presented with a dialog to set the properties for the connection between the device input layer and the component layer that you have just created (see Figure 9.4).
6. When you have set the connection properties, click “Ok” and you should see a new layer with connections to the first third and fourth row of the device muscle output layer (see Figure 9.5).

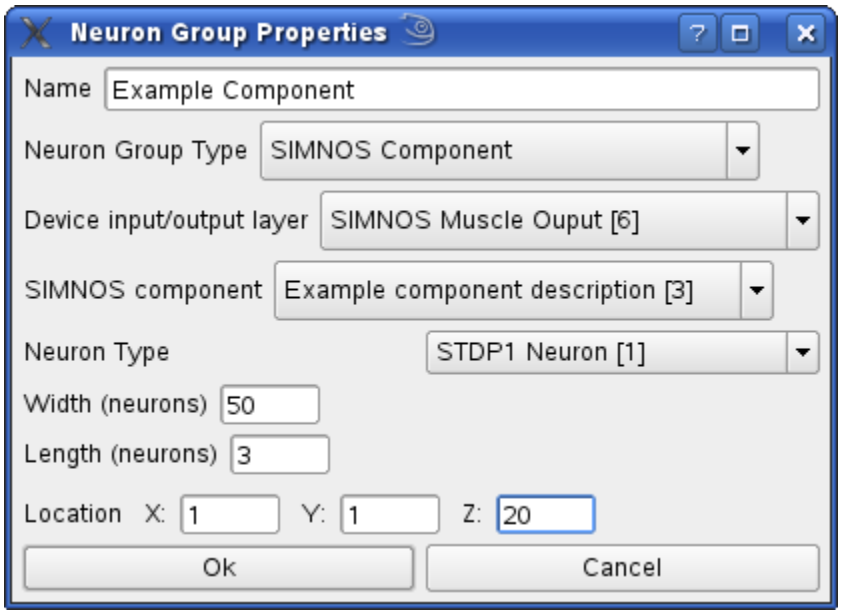


Figure 9.3. Creating a SIMNOS component

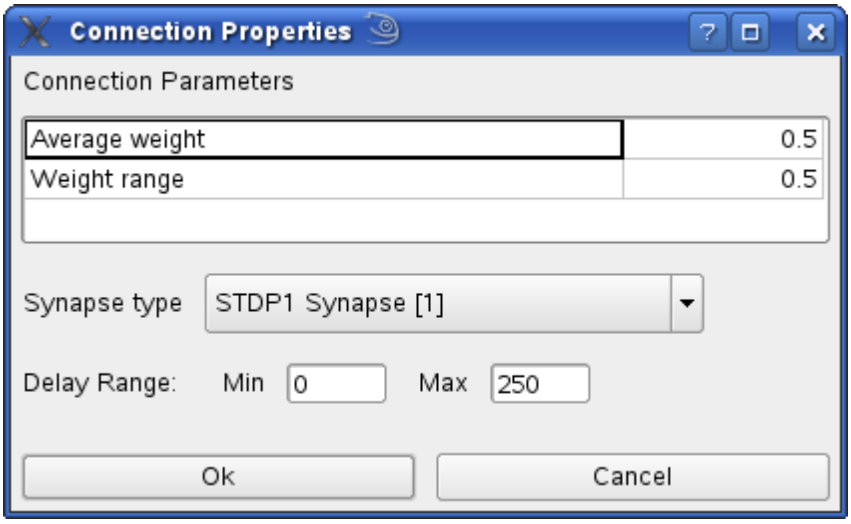


Figure 9.4. Setting connection properties for a SIMNOS component

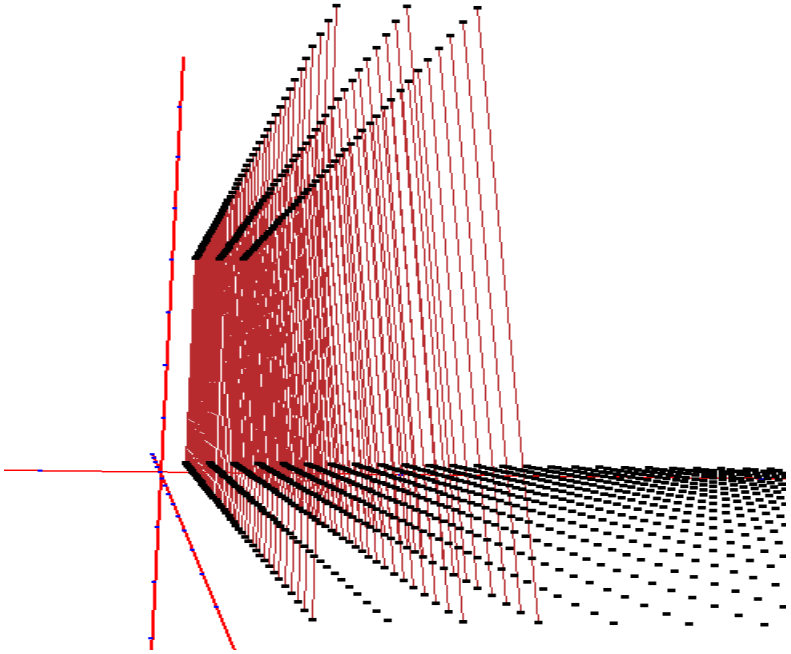


Figure 9.5. SIMNOS component layer connected to device receptors

10. Patterns

10.1 Introduction

Patterns can be applied to layers in the network for training or testing purposes. Two different types of pattern are available:

- **Static**. A snapshot of a firing pattern in the layer at a single point in time. This pattern will be held for every time step that the pattern is held.
- **Temporal**. The pattern codes a firing pattern that is spread out over several time steps. Each neuron will only be fired once at its specified time.

10.2 Adding Patterns

10.2.1 Pattern Manager

The Pattern Manager (see Figure 10.1) is used to load patterns from a file into the SpikeStream database. Click on Tools->Pattern manager to launch the Pattern Manager, which will display a list of patterns currently stored in the database. Patterns can be deleted by checking their associated box and clicking the "Delete Pattern(s)" button. To load a pattern into the database from a file, click on "Add Pattern(s)", navigate to the file(s) that you want to add and then click "Ok". If the pattern file(s) loads up successfully you will see the new pattern(s) listed in the Pattern Manager. Instructions for creating pattern files are given in the next section.

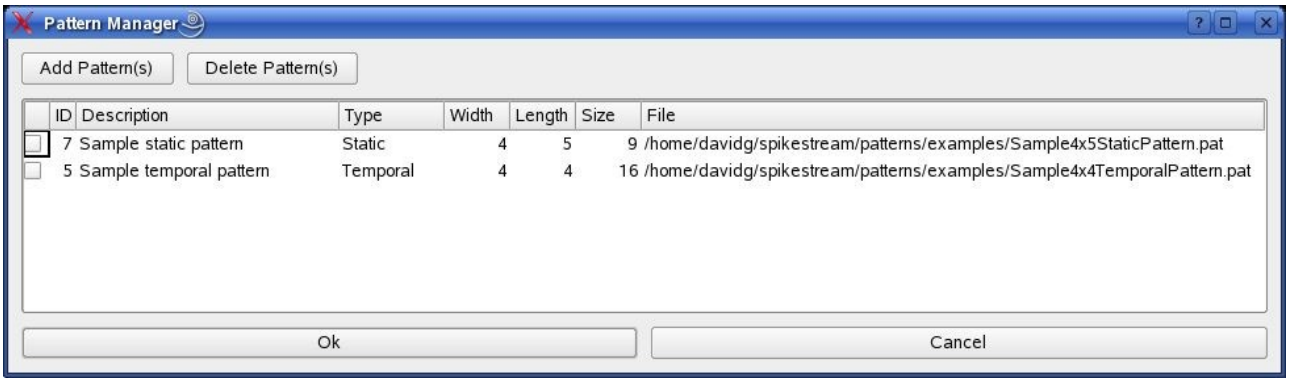


Figure 10.1. Pattern Manager

10.2.2 Pattern Files

The easiest way to create patterns is to manually or programatically generate pattern files and load them into the database using the Pattern Manager. The format is as follows.

- **First lines.** Can contain any information you wish, such as comments, authorship, etc., but must not contain hashes. All lines will be skipped by the parser until the information about the pattern is reached.
- **# Type.** The type of the pattern. This line should either be "# Type: static" or "# Type: temporal".
- **# Width.** The width of the pattern, for example "# Width: 4".
- **# Height.** The height of the pattern, for example "# Height: 4".
- **# Description**. A short description of the pattern that will be added to the pattern database, for example "# Description: Sample static pattern".
- **# Pattern data**. After the information about the pattern, the file can contain one or more pieces of pattern data. After each "#Pattern data:" heading there should be a width x height matrix of numbers, separated by spaces, containing the pattern at that point in time. For static patterns, these numbers must be either 1 or 0. For temporal patterns, they must be between 0 and 250 (currently the maximum number of time steps). The numbers in temporal patterns code the time that the neuron will be fired after the pattern has been loaded. For example, if you create a pattern containing a number of fives and set the "Number of time steps per pattern" in the Simulation tab to ten, then five time steps after the pattern was loaded, the neurons corresponding to the fives in the pattern would be fired and after another five time steps, the next pattern would be loaded. All of this will become much clearer when you try out the static and temporal sample pattern files given in SPIKESTREAM_ROOT/patterns/examples

NOTE: If your pattern does not behave as expected, make sure that you have the static / temporal field set correctly for your pattern.

10.2.3 Direct Pattern Generation

Whilst the automatic generation of pattern files is probably the easiest way to generate patterns, it is also possible to directly add patterns directly to the Patterns database without using the Pattern Manager. In this case, you need to generate a pattern description and one or more rows of pattern data. When you have added a couple of test patterns to the database using the Pattern Manager, a look at the structure of the data will show you how to directly generate your own patterns.

11. Saving and Loading Databases

11.1 Introduction

SpikeStream Application directly edits the database and so there is no need to explicitly save anything when you close it apart from any weights that have been changed during a simulation run. To enable users to save and load different neural networks, SpikeStream can save its databases to a file and reload them at a later point.

11.2 Saving Databases

Click on "File->Save database" and you will be prompted to choose the file to save the databases into. When the file is selected you will be presented with the Database Dialog shown in Figure 10.1. This enables you to select which of the databases you want to save – for example, you may only want to save the NeuralNetwork database into the file and leave out the Neural Archive, Patterns and Devices databases. When you have checked the databases that you want to save, press "Ok" and they will be saved into the specified file. Saving and loading of databases is carried out by the SaveSpikeStreamDatabase and LoadSpikeStreamDatabase scripts, which use mysqldump.

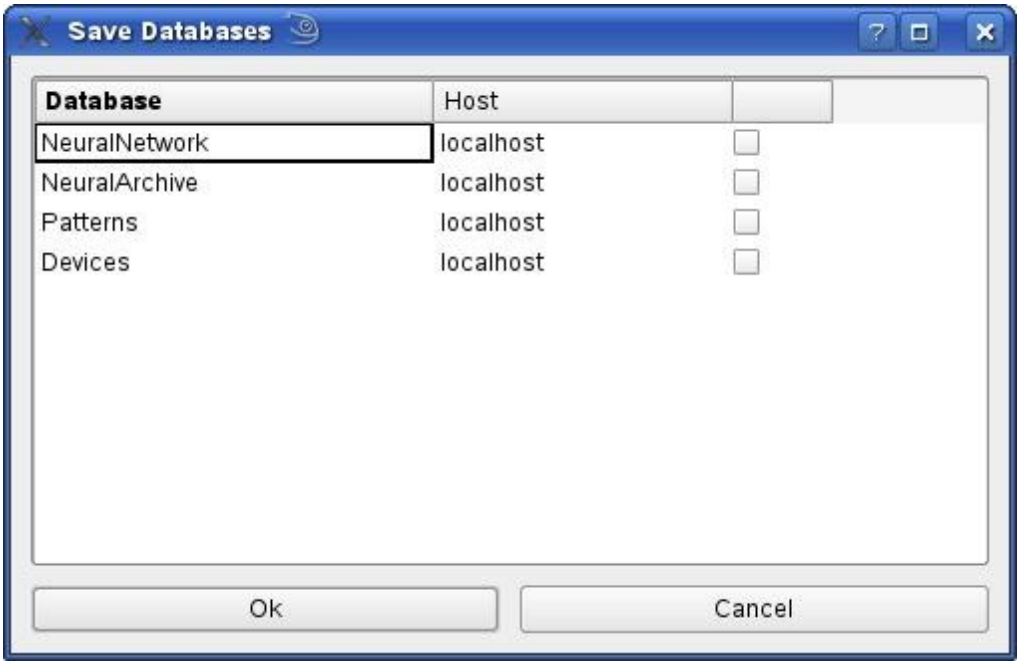


Figure 11.1. Database Dialog

This operation stores everything in the database including the neuron, synapse, global and noise parameters.

11.3 Loading Databases

Databases can only be loaded when the simulation is *not* initialized and an archive is not currently being played back. The loading of databases follows the reverse procedure to saving of databases. Click on "File->Load databases". This will first warn you that the loading operation will overwrite any of the databases that you choose to load. If you want to keep the current database you should cancel the loading operation and save the current database in a separate file. When you are ready to load the database, click "Yes" on this warning and use the file dialog to select the database that you want to load. SpikeStream will then inspect this file to determine which databases are stored inside it and present you with a Database Dialog containing a list of the databases that are available in the file. Select the databases that you want to load and click ok.

IMPORTANT NOTE: In the present implementation, the adding and removing of neuron and synapse types must be done without SpikeStream running. Loading up a database with different neuron and synapse classes from the ones currently loaded will lead to errors. The database should be ok, but you will need to restart SpikeStream to resolve the problem.

11.4 Clear Databases

The databases can only be cleared when the simulation is *not* initialized and an archive is not currently being played back. Clicking on "File->Clear databases" resets all data in the databases except the neuron, synapse and probe types. This operation is not reversible, so make sure that you do not have any important information or saved simulation runs that you want to keep before pressing "Yes" when the confirm dialog is displayed. If you want to reset everything back to its default state including the neuron, synapse and probe types, use the load database feature (section 10.3) to load the file $SPIKESTREAM_ROOT/database/DefaultDatabase.sql.tar.gz. The CreateSpikeStreamDatabases script can also be used to reset all the databases.

11.5 Import Connection Matrix

This feature is at an early stage of development and it is used to create a neuron group and set of connections based on a connection matrix in which the x and the y axes are the neuron ids and the values are the weights. After you have clicked File->Import connection matrix and selected the file containing the connection matrix it will create the new layer at (0, 0, 0) using the default neuron and synapse types. Before running this function you will need to create enough space at (0, 0, 0) for the new layer.

12. Neuron and Synapse Classes

12.1 Introduction

The dynamic class loading features of SpikeStream make it relatively easy to change the neuron and synapse models without modifying the whole application. However, a certain amount of work is required to get a new neuron or synapse class recognized by SpikeStream so that it can run in a distributed manner.

IMPORTANT NOTE: Adding and removing synapse classes should be done without SpikeStream running or you will get errors from the Neuron and Synapse parameters dialogs, which only load up the Neuron and Synapse type information once during initialization of SpikeStream. This can also occur when you have loaded a database with different neuron and synapse types or with a different TypeID for the existing types. Restarting SpikeStream usually resolves the problem.

12.2 Creating Neuron and Synapse Classes

12.2.1 Extend the Neuron or Synapse Class

The first stage is to write the code for the neuron or synapse classes, which have to inherit from the Neuron or Synapse classes in $SPIKESTREAM_ROOT/spikestreamsimulation/src. More information about these classes can be found in the online source documentation, available on the project website http://spikestream.sourceforge.net/pages/documentation.html

The easiest place to start when writing your own neurons or synapses is to look at STDP1Neuron and STDP1Synapse and to tweak these to match your own neuron or synapse model or learning rule. These examples also illustrate some of the areas that need to be handled carefully by a neuron or synapse class. The methods that you need to extend are covered in the next two sections.

12.2.2 Synapse.h

- **virtual const string* getDescription() = 0;** Returns a descriptive name for the synapse, which can be useful for debugging class loading. The class that invokes this method is responsible for cleaning up the string.
- **virtual short getShortWeight() = 0;** Returns the weight as a short between MIN_SHORT_WEIGHT and MAX_SHORT_WEIGHT (defined in Synapse.h). This is a virtual method because some implementations may need the state of the weight to be calculated retrospectively.

- **virtual double getWeight() = 0;** Returns the weight as a double between MIN_DOUBLE_WEIGHT and MAX_DOUBLE_WEIGHT. This is a virtual method because some implementations may need the state of the weight to be calculated retrospectively.
- **virtual bool parametersChanged() = 0;** Called when the parameters of the synapse have changed. The parameters of the synapses are held as references to parameter maps and when these are reloaded this method is called.
- **virtual void processSpike() = 0;** Called when a spike is routed to this synapse. In event-based simulation the synapse should be updated by this method.
- **virtual void calculateFinalState() = 0;** Called to update synapse class when all synapses are being updated at each time step. This method is never called during event based simulation. In this mode, the synapse class is only updated whenever it processes a spike.
- **virtual string getMonitoringInfo();** This method returns a string containing an XML description of the variables that are available for monitoring within this class. Overload this method and getMonitoringData() if you want to send monitoring information back to the main application. This will enable you to view a graph of the weight, for example, as described in section 7.5.4.
- **virtual MonitorData* getMonitoringData();** Returns a monitor data struct (defined in GlobalVariables.h) containing the data that is being monitored. This returned data must match that defined in the string returned by getMonitoringInfo();

12.2.3 Neuron.h

- **virtual void calculateFinalState() = 0;** Calculates the final state of the neuron after all spikes have been received. In synchronous simulation mode all neurons have this method called on them at the end of each simulation step.
- **virtual void changePostSynapticPotential(double amount, unsigned int preSynapticNeuronID) = 0;** This method is called when a synapse changes the membrane potential of the neuron. The neuron should update itself when this method is called by calling calculateFinalState().
- **virtual const string* getDescription() = 0;** Returns a description of this neuron class for debugging only. Destruction of the new string is the responsibility of the invoking method.
- **virtual bool setParameters(map<string, double> paramMap) = 0;** Sets the parameters of the neuron. These should be defined in their own database, whose name is listed in the NeuronTypes database. This method is called on only one instance of the neuron class with

the parameters being set and held statically. The parametersChanged() method is called after the static setting of the parameters to inform each neuron class that the parameters have changed.

- **virtual void parametersChanged() = 0;** Called after the parameters have been statically changed to inform each neuron class that the parameters have been changed. This enables them to update their learning state, for example, after learning has been switched off.
- **virtual string getMonitoringInfo();** This method returns an string containing an XML description of the variables that are available for monitoring within this class. Overload this method and getMonitoringData() if you want to send monitoring information back to the main application. This will enable you to view a graph of the membrane potential, for example, as shown in section 7.5.4.
- **virtual MonitorData* getMonitoringData();** Returns a monitor data struct (defined in GlobalVariables.h) containing the data that is being monitored. This returned data must match that defined in the string returned by getMonitoringInfo();

12.3 Build and Install Library

When you have created your neuron and synapse classes, compile them as .so libraries and copy them to $SPIKESTREAM_ROOT/lib. They need to have the standard library name format, such as libstdp1neuron.so for a "stdp1neuron" library. More information about this procedure can be found at: http://www.linux.org/docs/ldp/howto/Program-Library-HOWTO/shared-libraries.html. When your neuron class calls methods that are unique to the synapse class – i.e. methods that are not present in Synapse.h – you need to link against the synapse library to build the neuron class. This can be done by passing information about the dynamic synapse library to gcc when you build the neuron class. However, to *run* a simulation using the neuron class, the dynamic library that you have linked against needs to be accessible by the operating system in one of the known locations.[1] This can be done in one of three ways, which have to be carried out on every machine that you run the simulation on.

Method 1: Change the LD_LIBRARY_PATH Environment Variable

One way to ensure that the operating system can find the dynamic libraries is to add the location of your neuron and synapse libraries to the system path. This can be done by adding the following line

1 This step could probably be avoided by linking the neuron or synapse class against a static version of the other neuron or synapse class. However, I have not tried this yet and it is probably more memory efficient to use a dynamic library.

to your .bashrc file:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${SPIKESTREAM_ROOT}/lib
```

This can work fine if you are running SpikeStream on a single workstation, but it is likely to cause problems running across multiple machines and is not recommended anyway.

Method 2: Add Links to Library in /usr/local/lib

This method creates a link from /usr/local/lib to the location of your libraries. For example, to install STDP1Synapse, change to /usr/local/lib, log in as root and create the links using the following command:

```
ln -s /home/davidg/spikestream/lib/libstdp1synapse.so
libstdp1synapse.so.1
```

This may have to be done using the full address of the library if SPIKESTREAM_ROOT has only been defined for the user shell. The advantage of this approach is that it makes it easy to update the libraries when developing the neuron and synapse classes and it is more portable across systems. This approach is implemented by the InstallSpikeStream script, which is used to install the neuron and synapse classes included in the SpikeStream distribution (see section 2.4.5).

IMPORTANT NOTE: You should only install links to these libraries as root if you are the sole user of SpikeStream on the system. Otherwise you may end up dynamically loading another user's libraries!

Method 3: Copy Library to /user/local/lib

If your dynamic libraries are rarely going to change, it makes more sense to install them permanently by copying them to /usr/local/lib, rather than linking from /usr/local/lib to somewhere else on the system. This approach only makes sense if the other parts of SpikeStream were installed in /usr/local/bin as well. Since SpikeStream is still in the process of development, this option is not recommended at this stage.

IMPORTANT NOTE: You should only install these libraries as root if you are the sole user of SpikeStream on the system. Otherwise you may end up dynamically loading another user's libraries!

12.4 Update Database

The final stage is to add appropriate entries and tables to the Neural Network database so that networks can be created and simulated using the new neuron classes. This involves updating the neuron and synapse types and adding tables for the neuron and synapse parameters. In these examples, the neuron and synapse classes will be called Example Neuron and Example Synapse.

12.4.1 Add Neuron and Synapse Types

The NeuronTypes and SynapseTypes tables in the NeuralNetwork database hold information about all of the available neuron and synapse types. To use your new neuron and synapse classes in SpikeStream, they must have an entry in these tables. Before adding a new neuron type, select a TypeID. This is a unique identifier for your neuron type which must not conflict with any of the existing types. In this example, I have selected a TypeID of 2 since the only neuron class currently in the database is an STDP1Neuron with a TypeID of 1. To add a new neuron type, use the following SQL:

```
USE NeuralNetwork;

INSERT INTO NeuronTypes(TypeID, Description, ParameterTableName,
ClassLibrary) VALUES (1, "Example Neuron",
"ExampleNeuronParameters", "libexampleneuron.so");
```

The SQL for adding a new synapse type is similar:

```
USE NeuralNetwork;

INSERT INTO SynapseTypes(TypeID, Description, ParameterTableName,
ClassLibrary) VALUES (1, "Example Synapse",
"ExampleSynapseParameters", "libexamplesynapse.so");
```

12.4.2 Add Parameter Tables

Each neuron and synapse class has an associated parameter table in which the parameters for the neuron or synapse model can be set individually for each neuron or connection group, which have entries in the appropriate table. In order for this to work, the parameter table has be set up in a specific fashion. The SQL for the STDP1Neuron and STDP1Synapse parameter tables is given below:

```
USE NeuralNetwork;


CREATE TABLE STDP1NeuronParameters (
NeuronGrpID SMALLINT UNSIGNED NOT NULL,
CalciumIncreaseAmnt_val DOUBLE DEFAULT 1.0,
CalciumIncreaseAmnt_desc CHAR(100) DEFAULT "Calcium increase amount",
CalciumDecayRate_val DOUBLE DEFAULT 60.0,
CalciumDecayRate_desc CHAR(100) DEFAULT "Calcium decay rate",
RefractoryPeriod_val DOUBLE DEFAULT 1.0,
RefractoryPeriod_desc CHAR(100) DEFAULT "Refractory period (ms)",
MembraneTimeConstant_val DOUBLE DEFAULT 3.0,
MembraneTimeConstant_desc CHAR(100) DEFAULT "Membrane time constant (ms)",
RefractoryParamM_val DOUBLE DEFAULT 0.8,
RefractoryParamM_desc CHAR(100) DEFAULT "Refractory parameter M",
RefractoryParamN_val DOUBLE DEFAULT 3.0,
RefractoryParamN_desc CHAR(100) DEFAULT "Refractory parameter N",
Threshold_val DOUBLE DEFAULT 1.0,
Threshold_desc CHAR(100) DEFAULT "Threshold",
Learning_val BOOLEAN DEFAULT 0,
Learning_desc CHAR(100) DEFAULT "Learning",
PRIMARY KEY (NeuronGrpID));

CREATE TABLE STDP1SynapseParameters (
ConnGrpID SMALLINT UNSIGNED NOT NULL,
Learning_val BOOLEAN DEFAULT 0,
Learning_desc CHAR(100) DEFAULT "Learning",
Disable_val BOOLEAN DEFAULT 0,
Disable_desc CHAR(100) DEFAULT "Disable",
CalciumThreshUpLow_val DOUBLE DEFAULT 30.0,
CalciumThreshUpLow_desc CHAR(100) DEFAULT "Calcium threshold up low",
CalciumThreshUpHigh_val DOUBLE DEFAULT 120.0,
CalciumThreshUpHigh_desc CHAR(100) DEFAULT "Calcium threshold up high",
CalciumThreshDownLow_val DOUBLE DEFAULT 30.0,
CalciumThreshDownLow_desc CHAR(100) DEFAULT "Calcium threshold down low",
CalciumThreshDownHigh_val DOUBLE DEFAULT 40.0,
CalciumThreshDownHigh_desc CHAR(100) DEFAULT "Calcium threshold down high",
WeightChangeThreshold_val DOUBLE DEFAULT 0.8,
WeightChangeThreshold_desc CHAR(100) DEFAULT "Weight change threshold",
WeightIncreaseAmnt_val DOUBLE DEFAULT 0.1,
WeightIncreaseAmnt_desc CHAR(100) DEFAULT "Weight increase amount",
WeightDecreaseAmnt_val DOUBLE DEFAULT 0.1,
WeightDecreaseAmnt_desc CHAR(100) DEFAULT "Weight decrease amount",
PRIMARY KEY (ConnGrpID));
```

As you can see from the examples, each parameter table has a neuron or connection group id as its primary key. The parameters themselves can either be boolean, which appears as a check box in the parameter dialog, or doubles. Each value is defined using ExampleName_val, which stores the value of the parameter and has the specified default, and ExampleName_desc, whose default is the description of the value. As long as these conventions are adhered to in your parameter tables, you should be able to set the parameters using the Neuron Parameters Dialog and Synapse Parameters Dialog and the simulation should be able to access them without problems.

References

Gamez, David (2007). SpikeStream: A Fast and Flexible Simulator of Spiking Neural Networks. *Proceedings of ICANN 2007,* forthcoming.

Gamez, David, Newcombe, Richard, Holland, Owen and Knight, Rob (2006). Two Simulation Tools for Biologically Inspired Virtual Robotics. *Proceedings of the IEEE 5th Chapter Conference on Advances in Cybernetic Systems*, Sheffield, pp. 85-90.