LCG - electrophysiology on the command-line
User manual

Daniele Linaro
João Couto
Michele Giugliano

24th December 2013

Contents

Chapter 1

Introduction

This manual describes the functionalities of LCG, a suite of programs – called *commands* – that can be used to perform electrophysiological experiments.
LCG was developed by Daniele Linaro and João Couto while at the Theoretical Neurobiology and Neuroengineering Laboratory of the University of Antwerp. Michele Giugliano developed the concept and meta-description of stimulus files and implemented the first version of the related source code.

The main features of LCG are the following:

- dynamic clamp capabilities, with built-in active electrode compensation Brette et al. (2008);
- straightforward design and implementation of closed loop and hybrid experiments;
- command-line operability;
- ease of automation by scripting;
- compact description of stimulation waveforms;
- simple implementation of non real time protocols (e.g., current and voltage clamp);
- support for multiple real time engines;
- simple installation and operation procedures.

The entry point of all LCG protocols is the program called `lcg`. Its purpose is merely to parse its arguments and call the appropriate protocol with the arguments with which it was invoked. To clarify this concept, let's look at the following example:

```
1 lcg steps -a -200,100,50 -d 2
```

This command instructs `lcg` to look for a program called `lcg-steps` in the directories where executable files are located,[1] and invoke it with the subsequent arguments unchanged. The previous call is therefore equivalent to

[1] These are usually stored in the `$PATH` environment variable.

```
lcg-steps -a -200,100,50 -d 2
```

This approach is particularly suited to adding extensions to LCG: to do so, it is sufficient to follow the naming scheme just described and place additional scripts or programs where `lcg` can find them, i.e., anywhere in the path.

LCG comes with a `help` command, that can be used to display general information about LCG and a list of the most commonly used protocols, by simply invoking

```
lcg help
```

Alternatively, `lcg-help` accepts one single argument, which must be the name of a recognized protocol and invokes it with the `-h` option. For example, the following four commands produce the same result:

```
lcg help steps
lcg-help steps
lcg steps -h
lcg-steps -h
```

For this to work, all protocols must accept the `-h` option and interpret it as a request for help. All protocols that come with LCG adhere to this convention, and additional protocols should do the same.

The manual is organized as follows: Chap. 2 covers in detail the installation of a real time operating system and of LCG and its dependencies. Chapter 3 describes the most commonly used electrophysiological protocols that are embedded within LCG, while Chap. 4 explains how stimulus files can be used to apply arbitrary stimulation protocols, in traditional voltage and current clamp experiments. Chapter 5 covers the internal organization of the data files saved by LCG and contains a few examples that show how to load LCG data files using both Matlab and Python. Chapters 6 through 8 explain how configuration files can be used to describe custom experimental protocols, and which basic building blocks – called *entities* or *streams* – are available in LCG. Finally, Chap. 9 covers some additional features of LCG.

In the simplest scenario, in which LCG is already installed on a machine and the experimentalist only wants to perform conventional voltage and/or current clamp experiments, without the need for real time control over the experiment, Chapters 3 and 4 with some parts of Chap. 5 are sufficient to get going.

Chapter 2

Installation

The primary application of LCG is to perform electrophysiology experiments. Albeit not necessary, as it will be described in the following, it is however recommended to install LCG on a Linux machine with a data acquisition card and a working installation of the Comedi library. A real-time kernel is required only to perform closed-loop or dynamic clamp experiments.

It is also possible to install LCG on any UNIX compatible operating system (Linux or Mac OS X, for instance) without Comedi and real-time capabilities. This approach can be useful to familiarize with the program and to develop new experiments. A detailed description of how to do this is given in Sec. 2.4.

In the following section, we describe how to set up a Linux machine with a real-time kernel and Comedi and how to install LCG. We assume that you have a working Linux installation: if that is not the case, refer to the documentation of one of the multiple distributions available. In our laboratory we use Debian and therefore the following instructions will be based on this distribution.

2.1 Configuration of the Linux box

LCG can be installed on a variety of Linux distributions. Here, we will refer to the stable version of Debian at the time of this writing (*Squeeze 6.0*).

2.1.1 Patching and installing the realtime kernel

If you don't want to use the real-time capabilities of LCG you can skip this paragraph.

This is potentialy the most difficult part of the installation. In order to achieve nanosecond precision, LCG requires a kernel with real-time capabilities. Both RTAI or **PREEMPT_RT** can be used. The latter is advisable since RTAI does not include support for the latest kernels, which might work better with the most recent hardware.

You may need to install tools to build the kernel. In Debian you can type, as root:

```
apt-get update
apt-get install build-essential binutils-dev libelf-dev libncurses5
libncurses5-dev git-core make gcc subversion libc6 libc6-dev
    automake
libtool bison flex autoconf flex libgsl0-dev
```

1. **Check which kernels and patches are available.** It is advisable to install the real-time patch on a "vanilla" kernel. To do so, navigate to the kernel.org "rt" project page and check which patches are available for the kernel that you wish to install.

2. **Download the kernel and the realtime patch** The directory **/usr/src** is a common place to install the Linux kernel. You will need to have root access to perform these operations.

```
    cd /usr/src
    wget https://www.kernel.org/pub/linux/kernel/v3.x/linux
        -3.8.4.tar.bz2
    wget http://www.kernel.org/pub/linux/kernel/projects/rt
        /3.8/patch-3.8.4-rt2.patch.bz2
```

3. **Decompress and patch the kernel.** Patching the kernel will add real-time support to the kernel you have downloaded.

```
    tar xvf linux-3.8.4.tar.bz2
    ln -s linux-3.8.4 linux
    cd linux
    bzcat ../patch-3.8.4-rt2.patch.bz2 | patch -p1
```

4. **Configure the kernel.** The easiest way to do this is to use the configuration file from a kernel that was previously installed in your system or from a similar installation/system.

```
    cp /boot/config-`uname -r` .config
    make oldconfig
    make menuconfig
```

   This will evoke a user interface to configure the kernel options. You need to set the `Preemption model` to `Fully Preemptible Kernel (RT)` in the `Processor type and features`. Disable `CPU Frequency scaling` under `Power Management and ACPI options` and `Check for stack overflow` and the options under `Tracers` (the later to make the kernel smaller) in `Kernel hacking`. Additionally you should also disable the `Comedi drivers` under the `Staging drivers` of the `Device drivers` menu. Save and exit.

5. **Compile and install the kernel.** This step might take several minutes to hours, depending on the size of the kernel and on the speed of your machine.

```
make && make modules && make modules_install && make install
```

   When the installation is complete you will need to update the boot loader.

```
cd /boot
update-initramfs -c -k 3.8.4-rt2
update-grub
```

Check that grub has been updated. You should see an entry with the name of the newly installed kernel in `/etc/grub/menu.lst`. You can now reboot into your new kernel.

2.1.2 Installation of the dependencies

LCG makes use of several libraries:

1. **BOOST C++ library.** In order for LCG to work you will need to install the BOOST C++ library. Only the last command needs to be run as root.

```
    wget http://sourceforge.net/projects/boost/files/boost
        /1.53.0/boost_1_53_0.tar.bz2
    tar --bzip2 -xvf boost_1_53_0.tar.bz2
    cd boost_1_53_0.tar.bz2
    ./bootstrap.sh --prefix=/usr/local
    ./b2 install
```

2. **Comedi.** By default, LCG uses Comedi to interface to supported data acquisition cards. The installation of Comedi consists of three parts: a kernel-space library (`comedi` itself), a user-space interface (`comedilib`) and some additional programs for the calibration of the board (`comedi_calibrate`). They can be downloaded from the following Git repositories:

```
git clone git://comedi.org/git/comedi/comedi.git
git clone git://comedi.org/git/comedi/comedilib.git
git clone git://comedi.org/git/comedi/comedi_calibrate.git
```

You now need to install each of these modules, starting with the kernel module:

```
cd comedi
./autogen.sh
./configure --with-linuxdir=/usr/src/linux
make
```

Then as root:

```
make install
depmod -a
```

Now for comedilib, the user space interface to the kernel module (run `make install` as root):

```
cd comedilib
./autogen.sh
./configure --prefix=/usr/local
make
make install
```

And similarly for comedi_calibrate, the tools to calibrate the DAQ cards (install the Boost headers before this step):

```
cd comedi_calibrate
./autogen.sh
./configure --prefix=/usr/local
make
make install
```

Now that the drivers are installed you need to create the rules to allow users to access the devices. To do that, create, as root, a file called `/etc/udev/rules.d/99-comedi.rules` and add the following line to it: `KERNEL=="comedi0", MODE="0666"`. In case you have multiple acquisition cards, add a line for each of them.

3. **HDF5 library.** The HDF Group designed a set of libraries for data storage and management with high performance, efficiency and high volume/complexity in mind. **lcg** uses this library to save binary data files. The library can be installed from source in the following way (the last command should be run as root):

```
tar xjf hdf5-1.8.9.tar.bz2
cd hdf5-1.8.9
./configure --prefix=/usr/local --enable-shared
make
make install
```

Alternatively, it is possible to use a package manager, such as Debian's `apt-get`:

```
apt-get install hdf5-tools hdf5-serial-dev
```

2.2 Installation of lcg

To install LCG, start by getting it from the GithHub repository. We recommend to install LCG in a local folder, so that it is easier to have multiple installations for different users.

```
mkdir -p $HOME/local/src
cd $HOME/local/src
git clone https://github.com/danielelinaro/dynclamp.git lcg
cd lcg
autoreconf -i
./configure --prefix=$HOME/local
make
make install
```

If you are installing LCG on a real-time machine, you need to create a realtime group to allow non-root users to run LCG commands. To do so, as root add the following lines to `/etc/security/limits.conf`.

```
@realtime       -       rtprio  99
@realtime       -       memlock unlimited
```

Then, create a group `realtime` and add the users that you want to be able to use LCG.

```
groupadd realtime
usermod -a -G realtime USER
```

You need to log out and log back in for the changes to take effect.

2.2.1 Installation of the Python bindings

LCG comes with a set of Python scripts that can be used both to perform standard electrophysiology experiments and as starting point for developing new protocols.

The full list of available protocols is described in Chapter 3: here, we only describe how to install the Python bindings and their dependencies.

Start by installing `numpy`, `scipy`, `matplotlib` and `pytables` if they are not already present on your system. In Debian this can be accomplished with the following command.

```
apt-get install python-numpy python-scipy python-matlablib python-tables
```

Alternatively, to have the latest version of the modules, you can install them from source.

To install LCG Python module, from the base directory (i.e., where you cloned the Git repository) type:

```
cd python
python setup.py build
python setup.py install --prefix=$HOME/local
```

This will install the module in the directory `$HOME/local/lib/python2.6/site-packages`, which needs to be added to your `PYTHONPATH` environment variable, by adding the following line to your `.bashrc` file:

```
export PYTHONPATH=$PYTHONPATH:$HOME/local/lib/python2.6/site-packages
```

2.3 Final configuration

In order for LCG to function properly, a few configuration steps must be completed. First of all, add the directory where LCG binaries where installed to your path. To do this, add the following line to your `.bashrc` file:

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

After this, copy the files `lcg-env.sh` and `lcg-completion.bash` from LCG base directory to your home directory:

```
cp lcg-env.sh ~/.lcg-env.sh
cp lcg-completion.bash ~/.lcg-completion.bash
```

Source them any time you log in by adding the following lines to your `.bashrc` file:

```
source ~/.lcg-env.sh
source ~/.lcg-completion.bash
```

The script in `lcg-completion.bash` provides autocomplete capabilities to LCG but is not required for correct functioning. The environment variables exported in `lcg-env.sh`, on the other hand, provide necessary defaults to LCG and should be tailored to your system. In particular, the file exports the following variables:

- **COMEDI_DEVICE** The path to the device file from which data is read.
- **AI_CONVERSION_FACTOR_CC** The conversion factor to be used for the analog input, in current clamp mode.
- **AO_CONVERSION_FACTOR_CC** The conversion factor to be used for the analog output, in current clamp mode.
- **AI_CONVERSION_FACTOR_VC** The conversion factor to be used for the analog input, in voltage clamp mode.
- **AO_CONVERSION_FACTOR_VC** The conversion factor to be used for the analog output, in current clamp mode.
- **RANGE** The range of the output to the analog card.
- **AI_SUBDEVICE** The analog input subdevice on the acquisition card.
- **AI_CHANNEL** The default channel used for analog input.
- **AO_SUBDEVICE** The analog output subdevice on the acquisition card.
- **AO_CHANNEL** The default channel used for analog output.
- **AI_UNITS_CC** The units for the analog input, in current clamp mode.
- **AO_UNITS_CC** The units for the analog output, in current clamp mode.
- **AI_UNITS_VC** The units for the analog input, in voltage clamp mode.
- **AO_UNITS_VC** The units for the analog output, in voltage clamp mode.
- **SAMPLING_RATE** The default sampling rate of the acquisition.
- **GROUND_REFERENCE** The ground reference of the acquisition card. At present, Ground-Referenced Single Ended (GRSE) and Non-Referenced Single Ended (NRSE) are supported.

Most of the previous values depend on how your amplifier is configured and on how it is wired to the acquisition card. It is also important to note that the conversion factors and the units provided in `.lcg-env.sh` are meaningful when only one input and output channels are present. In all other cases, input/output conversion factors will have to be specified either in the configuration file or when invoking a script.

2.4 Installation without a realtime kernel

If you want to learn how to work with LCG or test configuration files, it is possible to install LCG on your personal computer. LCG can be compiled on any UNIX-like operating system, such as Linux and Mac OS X. To do so, you

just need to install `BOOST` and the `HDF5` library (see Sec. 2.1.2). The procedure to install LCG is the same as described in Sec. 2.2: the `configure` script will detect the absence of Comedi and of the real-time kernel and not compile parts of LCG. If you are installing LCG on Mac OS X, note that the equivalent of the `.bashrc` file is called `.profile`.

Chapter 3

Electrophysiology protocols

In this chapter, we describe the electrophysiology protocols that come with LCG.

All protocols allow the user to specify the sampling frequency (usually with a `-F` switch, but be careful of exceptions) and the input and output channels (usually with the `-I` and `-O` options, respectively, but be *extremely* careful of exceptions).

Of the following protocols, only those for the computation of f-I curves with a Proportional-Integral-Derivative (PID) controller and for clamping the frequency of a neuron require a real-time kernel. For all other protocols, a Comedi installation is sufficient.

Note that, whenever in the following voltage clamp or current clamp are mentioned, LCG does not communicate with the amplifier to switch modes: it is up to the user to do so.

An up-to-date list of the more commonly used protocols can be obtained by entering

```
1 lcg help
```

3.1 Action potential protocol

This protocol consists in injecting a brief (the default value is 1 ms) depolarizing pulse of current to elicit a single action potential to extract its salient electrophysiological properties. The duration of the pulse should be short enough in order not to interfere with the actual shape of the action potential. The user can specify the amplitude and duration of the pulse, as well as the number of repetitions and the interval between them.
For example, the following command

```
lcg ap -a 2000 -d 1 -n 10 -i 5
```

injects 10 pulses of amplitude 2000 pA and duration 1 ms with intervals of 5 s. The voltage is recorded for 0.5 s before and after the application of the pulse.

3.2 V-I curve protocol

This protocol consists in injecting a series of hyperpolarizing and/or depolarizing steps of current to compute a V-I curve. The default is to inject steps of current from −300 to 50 pA in steps of 50 pA, which is suited for cortical pyramidal neurons or other large types of cells. Alternatively, the user can specify the values of current by using the `-a` option.
For example, the following command

```
lcg vi -a -100,40,20 -d 1
```

instructs the protocol to inject steps of current from −100 to 40 pA in steps of 20 pA with a duration of 1 s.

3.3 Time constant protocol

This protocol is identical to the action potential protocol described previously, with the sole exception that the injected pulses of current are hyperpolarizing and can therefore be used to compute the time constant of a cell. The default values of amplitude and duration of the pulses are −300 pA and 10 ms, but they can be changed using the `-a` and `-d` options, respectively.
For example, the command

```
lcg tau -n 20
```

instructs the protocol to inject 20 pulses, instead of the default 30.

3.4 Current ramp protocol

This protocol injects an increasing ramp of current into a cell, to find its threshold for spiking. The user can specify the initial and final values of the ramp, as well as its duration.
For example, the command

```
lcg ramp -a 50 -A 300 -d 15
```

instructs the protocol to inject a 15 s ramp, starting from 50 pA and ending at 300 pA.

3.5 White noise protocol

This protocol injects white noise into a cell. It is primarily used to compute the electrode kernel for the Active Electrode Compensation technique described in Brette et al. (2008).
For example, the command

```
lcg kernel -s 100
```

instructs the protocol to inject a noisy trace lasting 10 s, with zero mean and standard deviation of 100 pA. The default value of standard deviation is 250 pA, which is suited for cortical pyramidal cells. The script also calls `lcg-extract-kernels` to extract the electrode kernel and saves the result in a file called `kernel.dat`. An additional option is provided (`-H`) to inject an additional *holding* current that is not taken into account in the actual computation of the electrode kernel. This option is useful for spontaneously active cells, like Purkinje cells, or for *in vivo* experiments.

3.6 Filtered noise protocol

This protocol injects a Ornstein-Uhlenbeck process into a cell. It can be used either to analyze the reliability and precision of cells, as was done in Mainen and Sejnowski (1995), or to extract the parameters of an exponential integrate and fire model, as described in Badel et al. (2008).
For example, the command

```
lcg ou -n 20 -i 5 -m 100 -s 100 -t 20 -d 3
```

instructs the protocol to inject 20 repetitions of a noisy current with mean and standard deviation equal to 100 pA and time constant of 20 ms. The duration of each injection is 3 s and the interval between repetitions is 5 s.

3.7 Input resistance

This protocol injects a train of hyperpolarizing pulses suited to compute the input resistance of a cell *in vivo*, as described in Crochet and Petersen (2006).
For example, the command

```
lcg rin -a -100 -d 300 -f 1 -D 90
```

instructs the protocol to inject 300 ms-long pulses of amplitude −100 pA at a frequency of 1 Hz. The total duration of the recording is specified by the `-D` option, in this case 90 s. Alternatively, the user can specify, with the `-N` option, the number of pulses to inject.

3.8 DC steps protocol

This protocol injects DC steps of current into a cell, and is a generalization of the previously described protocol for the computation of the V-I curve. It has additional parameters that allow to specify the duration of the recording after the application of the stimulus and an optional holding value.
For example, the command

```
lcg steps -a 100,400,50 -d 2 -n 4 -i 5
```

instructs the protocol to inject steps of current from 100 to 400 pA in steps of 50 pA with a duration of 2 s and interval of 5 s. Each amplitude is repeated 4 times. Additionally, this protocol has a `--vclamp` option, which instructs the program to consider the steps as voltage ones and use, as a consequence, the appropriate default conversion factors (contained in the `.lcg-env.sh` file, see Sec. 2.3). In this case, the command

```
lcg steps -a -100,0,20 --vclamp
```

instructs the program to inject *voltage* steps at values ranging from −100 to 0 mV, in steps of 20 mV. For all the other options, default values are used.
Note that in both cases, the amplitudes of the steps are shuffled, in order to minimize the effects due to adaptation to increasing (or decreasing) stimulation amplitudes. However, an option is provided (`--no-shuffle`) to inject the steps in a linear order.

3.9 f-I curve protocol

Input-output curves, also known as f-I curves, can be easily computed with LCG in two ways: the most straightforward and traditional one is to use `lcg-steps` to inject steps of current and subsequently count the number of spikes emitted for each amplitude.

An alternative way is to use a PID controller and have the neuron follow an increasing ramp of frequency. The protocol that implements this algorithm is `lcg-fi`, which requires a real-time kernel to operate correctly and can be called as follows:

```
lcg fi -a 150 -m 5 -M 30 -T 30 -n 2 -w 60
```

which instructs the protocol to initially inject a current of 150 pA and drive the cell from a minimum of 5 to a maximum of 30 Hz. The protocol is repeated twice, with each trial lasting 30 s and an interval of 60 s between trials. Note that here the inter-trial interval option is specified by the `-w` option, instead of the more common `-i`, which is used, together with `-p` and `-d`, to specify the integral, proportional and derivative gains of the controller, respectively. Note also that, for optimal results, the initial value of current (150 pA in the example) should be chosen such that it elicits spiking in the cell at a frequency as close as possible to the minimum required value (here equal to 5 Hz).

3.10 Frequency clamp protocol

In a sense, this protocol is the dual of an injection of suprathreshold current: in that case, a constant current is injected (clamped) and the firing frequency of the cell can be measured. Here, the protocol finds the current necessary to make the cell spike at a given frequency. To do so, it uses a PID controller in much the same way as the protocol for the computation of a f-I curve, with the difference that here the value of frequency is a constant instead of a ramp.
For example, the command

```
lcg fclamp -f 10 -a 100
```

instructs the protocol to clamp the firing frequency of the cell at a value of 10 Hz, starting with an injected current of 100 pA, which must be above threshold, in order for the frequency estimator to work properly.

3.11 Spontaneous activity protocol

This protocol can be used to record spontaneous activity from an arbitrary number of input channels and is suited both for *in vitro* and *in vivo* experiments. The user can specify the duration of the recording, the device, subdevice, channels and gains to be used in the recording. The only limitation is that all channels need be on the same device and subdevice.
For example, the command

```
lcg spontaneous -d 60 -I 0,1,2,3
```

instructs the protocol to record for 60 s from channels 0, 1, 2 and 3. If not specified, the default device file and input subdevice are taken from the `.lcg-env.sh` file, see Sec. 2.3.

3.12 Train of pulses protocol

This protocol injects a train of brief depolarizing protocols into a cell to elicit action potentials, while simultaneously recording from two cells, the stimulated one and another that might be connected to it. The purpose of this protocol is therefore to test the connectivity between pairs of cells and to measure their short-term synaptic properties.
For example, the command

```
lcg pulses -N 8 -O 0 -I 0,1 -f 30 -d 1 -A 3000
```

instructs the protocol to stimulate a presynaptic cell with a train of 8 pulses of duration 1 ms and amplitude 3000 pA at a frequency of 30 Hz. The output channel (i.e., the channel used to inject current into the presynaptic cell) is number 0, while the input channels are numbers 0 and 1, with 0 being the channel connected to the presynaptic cell and 1 that of the postsynaptic cell. The default is to record the presynaptic cell in current clamp and the postsynaptic

one in voltage clamp, but the option `--cclamp` allows to instruct the protocol to record both channels in current clamp mode.

3.13 Utility programs

LCG contains some additional programs that are not protocols, but that can be used in various occasions during an electrophysiology experiment. They are described in brief in the following, and we refer the user to their help (i.e., `lcg help utility_name`) for an in depth description of their options.

- `lcg zero` outputs zero on all channels of a DAQ board. It is particularly useful at the beginning of an experiment, when the output of the DAQ board is undefined.
- `lcg output` outputs a given value on some channels of the DAQ board. It can be useful when it is required to hold a cell with a certain current or to a certain voltage, for example in voltage clamp experiments.
- `lcg annotate` allows the user to add comments into an existing H5 data file. It accepts two options, `-f` for the filename and `-m` for the comment message. If no options are specified, the program prompts the user for the file name and for a message.

Chapter 4

Stimulus generator

Of course, if LCG were restricted only to the protocols described in Chapter 3, it would constitute a cheap replacement of commercial programs such as Pulse, which offer a vast array of additional functionalities. What makes LCG extremely powerful and *versatile* is the possibility to define arbitrary *stimulus* files to be used in voltage and current clamp experiments, described in this chapter, and the possibility to use *configuration* files to describe arbitrary experiments, (e.g., dynamic clamp or closed loop experiments) defined as the interactions of elementary objects. Configuration files and such objects, called *entities* in the context pf LCG, are described in detail in Chapters 6 and 8, respectively. This chapter describes how to use LCG to inject arbitrary waveforms, described in stimulus files.

In principle, all traditional electrophysiology protocols (i.e., those that do not require real-time control over the recorded quantities), can be implemented in LCG by using just two commands, `lcg-stimgen` and `lcg-stimulus`: the former produces stim-files[1] according to the specification of the user, and the latter applies the synthesized waveforms to the appropriate output channels while at the same time recording from an arbitrary number of input channels.

As a matter of fact, all the protocols described in Chapter 3, with the exception of f-I curve and frequency clamp protocols described in Sections 3.9 and 3.10, respectively, can be implemented using only `lcg-stimgen` and `lcg-stimulus`. The reason for their existence is simply the possibility to define appropriate default values, which make the application of the protocols even simpler.

[1] The syntax of stim-files is explained in great detail in the companion manual, which must be read to properly understand the following instructions.

Chapter 5

Data files

In this chapter we describe the structure of the H5 files used by LCG to store the recorded data.

Chapter 6

Configuration files

This chapter describes how to write custom configuration files to be used by LCG.

Chapter 7

Streams

This chapter describes all the streams contained in LCG

Chapter 8

Entities

This chapter describes all the entities contained in LCG Short definition of entity and the general properties of entities.

8.1 H5Recorder

The *H5Recorder* records the entities that are connected to it to HDF5 files.

Parameter	Type	Type	Description
`filename`	String	yyyymmddHHMMSS.h5	Name of the file to record
`compress`	Boolean	True	If true, file is saved with GZIP compression

Table 8.1: H5Recorder configuration file parameters

Chapter 9

Additional features

This chapter will describe some additional features present in LCG that most likely are not of daily use. However, they can prove rather useful in some particular scenarios.

9.1 Computation of the SHA checksum

9.2 Adding comments to data files

Bibliography

R. Brette, Z. Piwkowska, C. Monier, M. Rudolph-Lilith, J. Fournier, M. Levy, Y. Frégnac, T. Bal, and A. Destexhe, Neuron **59**, 379 (2008).

Z. F. Mainen and T. J. Sejnowski, Science **268**, 1503 (1995).

L. Badel, S. Lefort, R. Brette, C. C. H. Petersen, W. Gerstner, and M. J. E. Richardson, Journal of neurophysiology **99**, 656 (2008).

S. Crochet and C. C. H. Petersen, Nature neuroscience **9**, 608 (2006).