FEDERICO SANTA MARÍA TECHNICAL UNIVERSITY

ELECTRONICS DEPARTMENT

VALPARAÍSO - CHILE

Neuronment v1.0

Reference Manual

Version 0.1

Author Pedro F. Toledo
pedrotoledocorrea@gmail.com
Revisers María José Escobar

June 2, 2015

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Glossary

N

Neuronal Network:

TBC
[3, 10]

Neuronment Procedure: NPROC

TBC
[3–6]

Neuronment Sequencer: NS

TBC
[3, 4, 7, 12]

Neuronment Sequencer Syntax: NSS

TBC
[3, 5]

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Purpose

This document is the Reference Manual for the software Neuronment in his version 1.0 and is intended to provide all the information required for a full usage of its features.

1.2 Style convention

This document uses the following styles:

- **bold**
  Words in bold are used for commands or situations in which they need to be used specifically as indicated including case type.
- *italic*
  Words in italic correspond to words that need to be substituted before using.
- `monospaced`
  Used for examples.
- `<name>`
  In the examples is used for parts that need to be substituted with the real value. Equivalent to italic in normal text.
- `[option]`
  In the examples is used to identify optional arguments.

Bold and italic may also be used to highlight words.

1.3 Problem reporting

If you find a problem, inconsistency or ambiguous explanation please contact the author at pedrotoledocorrea@gmail.com.

1.4 How to read this document

This document is divided in self explanatory chapters presented in 2 groups. Chapters 2, 3 and 4 refer to the common application environment and the following chapters have the instructions and details for the different possible neurological simulations and training procedures.

Chapter 2

User's Guide

2.1 Overview

The Neuronment project is a software intended for discreet simulation and training of complex neural networks for neuroscience studies. Its name comes from the words "Neurological" and "Environment" as the intention is to create a context where different neurological structures can be model, simulated and trained from a simple procedure description file, abstracting all the computational complexities required for the implementation.

Neuronment works by reading a Procedure Description file (standardized extension `*.nproc`) which should contain the list of all the neural network description parameters with a specification for the simulations and/or training intended to be calculated. This file, also called Neuronment Procedure (NPROC), should comply with the Neuronment Sequencer Syntax (NSS) in order to be correctly interpreted by the Neuronment Sequencer (NS).

The NSS has been developed with the intention to cover all the possible use scenarios of the intended purposes of the Neuronment project; nevertheless, it is susceptible to changes in future versions that may not be backwards compatible.

The Neuronment project has been build mainly on the experience acquire on the development of the thesis work of Pedro F. Toledo[1].

2.2 How to execute Neuronment

To execute this program it is required the executable file and a "Neuronment Procedure" file. To run it you must use the following line on the shell:

```
Neuronment -nproc <file.nproc> \
           [-verbose_messages|-no_verbose_messages] \
           [-no_output] \
           [-time]
```

- `Neuronment`:
  Name of the Neuronment executable to use.

- `-nproc <file.nproc>`:
  The flag **-nproc** is used to identify the NPROC file that should be read by the neuronment sequencer.

- `-verbose_messages|-no_verbose_messages`:
  This is an optional setting to force (or to not force) the apparition of explanatory descriptions for the coded information, warning and error messages returned by the neuronment sequencer. You can check the default behavior by checking the private define **DEFAULT_MESSAGES**. CHECK LINKS

- `-no_output`:
  This is an optional setting that eliminates any print to standard output; nevertheless, it doesn't affect the behavior of re-directions to files. It doesn't cause a notorious improvement on the total elapsed time required to run a NPROC .

- `-time`:
  This is an optional setting that initiates Neuronment with printing the elapsed time by default ON. The printing of the elapsed time corresponds to the standard output printing of the elapsed time between the end of a previous command and the end of the current one. This time is calculated in base to the `clock()` function of **time.h**, presented in seconds.
  The elapsed time printing can be enabled or disabled at any point of the execution (including when it was set by the `-time` option) by enabling or disabling the `ENV:show_elapsed_time` variable. CHECK LINKS

2.3 Neuronment sequencer

The neuronment sequencer is the module inside Neuronment responsible of interpreting the NPROC file specified at the program call. It bases its operation on loading settings and executing commands as indicated on the file.

2.3.1 Commands

All the operations that can be done by Neuronment are managed by NPROC file lines named "commands".

The different possible commands are divided in groups of related functionality and their results usually depend on the values of the Neuronment variables.

2.3.2 Variables

The neuronment sequencer considers a list of valid variables that are loaded at the starting of the program. These variables are defined in compiling time; nevertheless, there are options to create custom variables if required by the user. PENDING IMPLEMENTATION

The variables are internally called by the different commands as they require the information in order to use it to define their behavior.

A variable always have a “default” value also specified at compiling time. This value can be retrieved by the user but once the variable has been written the default value is lost and only the user value will be used for future configuration.

The commands that are variable dependent do not link dynamically to the variable values. If a command is executed under certain conditions, if the variable values required for its operation change these changes do not have any effect on the command behavior (for example: if a command defines a neural net configuration but after building the net and before simulating it one of the variables used by the command changes, this doesn’t have effect over the simulation and it will behave as if the variable value where the one at the time of the neural net configuration).

2.3.3 Message system

The basic Neuronment output system is a list of coded messages. Each one of these messages are identified information, warnings, problems or errors that have been considered at the developing stage.

An exhaustive list of messages codes as well as their description can be found at chapter 4. Also, it is possible to obtain extra information within the program by calling the **rescue man** command as indicated at **??**. PENDING IMPLEMENTATION

2.3.4 Assertions

When a problem arises on the program execution, there are 3 levels of warnings and/or early termination depending on the severity of the issue:

- Development Assertion:
  This happens when the program arrives to a known unexpected set of conditions. If the program can recover from this set of conditions it will continue, but it’s highly recommended to check for the root cause of this issue. This assertion will deploy the coded message **ER-001**.

- Implementation Assertion:
  This happens when there is a known incomplete implementation of a required feature to continue the program execution. Under this case the program will terminate and deploy the coded message **ER-008**.

- Run-time Assertion:
  This is the higher severity exception and it occurs when the program has arrived to an unknown and unexpected set of conditions. The program will terminate and deploy the codded message **ER-002**.

2.4 Neuronment sequencer syntax

The following section describes the different aspects of the neuronment sequencer syntax as it is required to be used on the development of a NPROC file.

In order to have a clear terminology, the following definitions will be used for future reference:

- Line:
  A line is a set of characters between:
  - The beginning of a file and a new line character or the end of file if there is no new line characters in the file.
  - Two new line characters
  - A new line character and the end of file.

  On this area there are 2 considerations that should be taken in to account:
  - The new line character(s) isn't part of the line
  - The new line character cannot be escaped as in BASH or CShell
- Comment:
  A line or part of a line that only has explanation or documentation proposes and shouldn't be considered by the sequencer.
- Command:
  A line or part of a line used to modify the Neuronment behavior, to calculate results or to retrieve values.
- Directive:
  Name received by a group of commands with common characteristics.
- Sub-Directive:
  Name received by an specific instruction of a directive in order to execute some task.
- Instruction:
  Name received by a Directive followed by a Sub-Directive.
- Arguments:
  Name received by the set of character strings at the right of an instruction on a command, strings that are or may be required for the command execution.
- Flags:
  Name received by an argument string that starts with the character "-". There are two types of flags:
  - Indicator Flags:
    Its only apparition has a well defined meaning.
  - Signaling Flags:
    A signaling flag is used to signal that the following string corresponds to an specific value indicated by the flag. This is normally used to avoid instructions receiving a list of values with non in-line declared meaning.

2.4.1 Basic rules

A NPROC file should be written in ascii and it will be divided in lines using the line breaks as line termination. The resulting lines will be interpreted according their content.

2.4.1.1 Empty line

An empty line prints an empty line to the standard output.

An empty line is a line without characters or only composed by spaces and/or tabs.

2.4.1.2 Comments

The character "#" divides a line between a command (everything to the left) and a comment (everything to the right).

If the command is empty or only composed of spaces and/or tabs, the whole line will be interpreted as a comment.

If the whole line is a comment, it will be printed out to the standard output, otherwise, the command will be executed and, if exists, no comment will be printed to the standard output.

2.4.1.3 Redirections

The character "¿" can be used to redirect a command result to a file instead of the standard output.

The text at the right of the "¿" character should be one valid file name without spaces. After the file name it is possible to add a comment as indicated previously.

2.4.1.4 Variables

All Neuronment interactions are managed by variables and/or command arguments which values should be set prior or at the command execution.

The variables are named as sets of strings separated by the ":"symbol as indicated in the following example:

```
VariableGroup1:Variable1
VariableGroup2:Variable1:ChildVariable1
```

The first string is called "Variable Group" and the second is called "Variable Name". If a third or any other string appear it will be called "Child Variable", "Grand Child Variable", etc.

All the strings of a variable must be composed exclusively by letters from a to z, A to Z, numbers and the underscore character.

At the moment of using a variable, the symbol "'" can be used to employ trailing zeros. It basically tells the neuronment sequencer to ignore all the digits equal to zero at the right of the symbol until a digit different of zero or and end of string is found. This allows the use of synonyms:

```
Sim:Var5
Sim:Var'5
Sim:Var'0000000005
```

This symbol also can be used also or for a child variable name:

```
Sim:Var7:Var6
Sim:Var'07:Var6
Sim:Var7:Var'000006
```

In case there are only zeros before a letter, the number is replaced by a zero:

```
Sim:Var420a
Sim:Var42'0a
Sim:Var42'0000000a
```

There can be any number of "'" in a variable.

2.4.1.5 Substitutions

If in order to use a command you would like to use a variable value as parameter instead of a hard coded string, you can substitute the name of the variable for its value by using the character "$" just before the variable group as in the following example:PENDING IMPLEMENTATION

```
$VariableGroup1:Variable1
$VariableGroup2:Variable1:ChildVariable1
```

2.4.1.6 Command results

If a command is intended to return a value after its execution, it will return the values through a variable as it will be indicated in the instruction specification.

If you want to store a result you should save it on another variable by creating a personal variable and then assigning the value to it by using substitution.PENDING IMPLEMENTATION

Chapter 3

Instructions

A command is composed by an instruction, their required arguments and, optionally, a redirection. The redirection has a well defined specific behavior (described on the previous section) and the the arguments will depend on the specific instruction for execution. Under this circumstances, it is possible to use the terms Instruction and Command as synonyms.

As indicated before, an instruction is composed by a Directive and a Sub-Directive. The directives is a way to group different commands with related functionality and the sub-directive is the specific name of the action to execute.

On the following sections there is an exhaustive list of sub-directives grouped by their corresponding directive.

3.1 Variable Management (varman)

The "Variable Management" or "varman" directives is the set of all the commands that can be use to set, retrieve or inquiry a variable. These instructions only have an effect (if any) at the environment level as the variables are not dinamically linked in Neuronment.

3.1.1 varman set

This command allows to set a list of values to a variable. To work, it sould have the following format:

```
varman set <variable name> <list of values>
```

The number of values of the list of values (number of strings separated by spaces and/or tabs) should be exactly the number supported by the variable. By other side, each value on the list will be interpreted as the expected type for the variable independently of their format.

3.1.2 varman print

This command allows to get the list of values currently stored on a variable. To work it should have the following format:

```
varman print <variable name>
```

If the variable doesn't exist or if it does not hold any value it will answer with a coded message.

3.2 Run Commands (runsim)

The "Run Commands" or "runsim" directives is the set of all the commands related to the configuration, simulation and training of the neural network. This instruction will use the values stored in the variables for their execution, but they will only retrieve the values at the execution moment and they will not link the values dynamically. This means that all the values required for these instructions needs to be set by the propper **varman** instructions prior their call.

3.2.1 runsim ss_initialize

This command reads all the global variables that descrive a single neural network and builds the data structure required for the storage of all the required values for a full simulation. This may require some variables to be set only in certain configuration conditions, therefore, it is not required an exhaustive list of variables set.

3.2.2 runsim ss_add_V1_diffusion

This command is used to superpose a V1 difussion activation to be used as V1 eternal exitation for a single simulator. In order to work it should have the following format:

```
runsim ss\_add\_V1\_diffusion
```

All the values required to add the diffussion activation are retrieved depending of the value of the variable **SIM:V1_external_excitation_method** which defines the mathematical method that will be used to compute the difussion.

3.2.3 runsim ss_simulate

This command will run the simulation for a single simulator. This is, it will calculate the activation and derivate activation for each neuron in the structure for each instant of time and will store the results in the data structure to be retrieved by reporting instructions.

3.3 Reporting Commands (report)

The "Reporting" directive or "report" is the set of instructions dedicated to retrieve the simulation data and print it to the standar output or, if specified, to a redirected file.

3.3.1 report ss_print_V1_activation

This command must be used as:

```
report ss\_print\_V1\_activation
```

It will display a column for each V1 neuron and in the rows the activation value for each time step calculated.

3.3.2 report ss_print_V1_external_excitation

This command must be used as:

```
report ss\_print\_external\_excitation <timestep>
```

It will display a row for each V1 neuron including the neuron name. orientation, prefered spatial frequency, preferred temporal frequency and the exitation for the specifi time step selected.

3.3.3 report ss_print_MT_activation

This command must be used as:

```
report ss\_print\_MT\_activation
```

It will display a column for each MT neuron and in the rows the activation value for each time step calculated.

3.4 I/O Interaction Commands (rescue)

The "I/O Interaction Commands" or "rescue" directive are the set of instructions related to the access to external files from the sequencer.

3.4.1 rescue nproc

This command is used to start the execution of another nproc file as part of the current nproc execution. This command should be used as:

```
rescue nproc [-silence] <nproc name>
```

Where the **nproc name** is the file to be read and **-silence** is a flag that allows to silence all the output of this file interpretation so it does not appears on the standard output; nevertheless, this does not affect the redirections.

3.4.2 rescue return

This command is used to stop the interpretation of an nproc file at any point. This command should be used as:

```
rescue return
```

Chapter 4

Messages

As indicated on the previous sections, there is a list of messages that the neuronment sequencer could use to indicate an information, warning or error.

The following section includes the list of all the possible interface messages with their description.

4.0.3 IN-001

- Interface Message:

  NProc directive not recognized

- Development Assertion: YES
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The directive (first word on the nproc command) trying to get interpreted is not on the list of possible directives. Please go to the Reference Manual chapter ”Directives” to get a full list of valid directives.

4.0.4 IN-002

- Interface Message:

  Trying to report an undeclared variable

- Development Assertion: YES
- Implementation Assertion: NO
- Runtime Assertion: NO

- Message Description:

  The Simulator as well as the Simulation Environment has a list of predefined variables so store and retrieve information. The variable been addressed is not part of the list. The user is not allowed to create new variables.

4.0.5 IN-004

- Interface Message:

  Unidentified sub directive, ignoring line

- Development Assertion: YES
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The sub-directive (second word on the nproc command) trying to get interpreted is not on the list of possible directives. Please go to the Reference Manual chapter "Directives" to get a full list of valid directives.

4.0.6 IN-005

- Interface Message:

  Unidentified setting, ignoring line

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  A value has tried to be written on an un-identified or un-available configuration variable. This nproc line will be ommited.

4.0.7 IN-006

- Interface Message:

  Flag not found or without a value

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO

- Message Description:

  The command on execution requires the definition of a flag (a word that starts with ”-” in the arguments) that is not present or without a value (if a flag requires a value, the next word after it should be the string representing the value for the flag. This string must NOT start with a ”-”. If the value is a negative number put it between quotes).

4.0.8 IN-007

- Interface Message:

  A boolean argument has not been properly written. Will be interpreted as false

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  A string been read as boolean does not match any of the possible true values (true, True, T, t, 1) neither false (false, False, F, f, 0). It will be interpreted and stored as false.

4.0.9 IN-008

- Interface Message:

  The maximum amount of possible nested ȓescue nprocc̈alls has been reached

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  Each time a ”rescue nproc” instruction is called, the execution of this new nproc file increases in 1 the nesting level of execution. This message appears when the nesting level has reached the maximum allowed.

4.0.10 IN-010

- Interface Message:

  Incorrect amount of arguments for the command, ignoring line

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The number of arguments of the command does not match the minimum required. This line will be ignored.

4.1 Command Line Issues

4.1.1 UI-001

- Interface Message:

  Duplicated or contradictory flags on command call

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The Neuronment command-line call has a flag declared more than once or two flags that are different are trying to set a contradictory behaivior.

4.1.2 UI-002

- Interface Message:

  Flag expected

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The Neuronment command-line call holds a value in a place where should be a flag (a tring starting with ”-”).

4.1.3 UI-003

- Interface Message:

  Flag not recognized, flag omitted

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The Neuronment command-line call has detected a flag that is not on the list of possible flags. This flag will be ignored.

4.1.4 UI-004

- Interface Message:

  Label without content

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The Neuronment command-line call has a flag that requires a value, but the value has not been found. If you are trying to use a negative number put it between quotes.

4.1.5 UI-006

- Interface Message:

  Empty NProc name

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The call to Neuronment requires a mandatory nproc file to be processed.

4.2 File IO Issues

4.2.1 ER-001

- Interface Message:

  Development Assetion

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The program arrived to an unexpected set of conditions.

4.2.2 ER-002

- Interface Message:

  Runtime Assertion

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  Something went wrong running the program. Terminating.

4.2.3 ER-003

- Interface Message:

  File could not be opened

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The nproc required for execution file could not be oppened.

4.2.4 ER-004

- Interface Message:

  File could not be properly closed

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The nproc file previously executed did notreturned a propper closed status from the OS when trying to close the file.

4.2.5 ER-005

- Interface Message:

  Trying to get a new line from a non ready file

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  This happens when for some reason the interpreter is trying to get a new line from a file that has not been properly oppened.

4.2.6 ER-006

- Interface Message:

  Trying to get a new line from a file already at the end

- Development Assertion: YES
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  This happens when for some reason the interpreter is trying to get a new line from a file already at the EOF.

4.2.7 ER-007

- Interface Message:

  Fail on getting a new line from nproc file

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  This happens when for some reason the file stream under interpretation is not able to retreive a new line from the nproc file, even if it is not at the EOF.

4.2.8 ER-008

- Interface Message:

  Implementation Assertion

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  There is a problem or an incompete implementation of a required feature.

4.2.9 ER-009

- Interface Message:

  Trying to close an unopened file

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The interpreter is trying to close a file that has not been oppened.

4.2.10 ER-010

- Interface Message:

  Required file is empty

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The interpreter requires a file for execution. In this case the file name is empty.

4.3 Implementation Issues

4.3.1 DV-001

- Interface Message:

  Missing implementation

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The build in use has identified a procedure wich has not been implemented.

4.3.2 DV-002

- Interface Message:

  Trying to declare a previously declared setting

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  At the moment of creating a new variable for use by Neuronment, it has been detected that a previous variable with the same name has already been declared.

4.3.3 DV-003

- Interface Message:

  Trying to load a setting of an unsupported type

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  Neuronment has many internal data types but only a few of them are intended for data handling. In this case, a data type not intended for data handling is trying to be used for data handling.

4.3.4 DV-005

- Interface Message:

  Trying to use an unsupported setting

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  Neuronment is trying to get an information parameter from a variable that does not exists or have not been declared jet.

4.3.5 DV-006

- Interface Message:

  Trying to read an incorrect data type for the setting

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  In Neuronment the variables have a specific type of value declared at their declaration. In this case, the variable value is trying to be used as a diferent type of the one declared.

4.3.6 DV-007

- Interface Message:

  The setting count is different than the declared setting count on the HashEntry

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  When retrieving a variable list of values, the number of values retrieved is different from the number of values that where declared at the variable configuration.

4.3.7 DV-008

- Interface Message:

  The Log manager is being copied and this should NEVER happen

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The Neuronment sequencer as instance is trying to be copied to another instance. This is not suppose to happen as a sequencer should never be copied.

4.3.8 DV-012

- Interface Message:

  Trying a quick access of incorrect type

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  When trying to quick retrieve a value from a single element value list, the function in use is triying to retrieve a value from a a variable of different type that the one that is supose to retrieve.

4.3.9 DV-013

- Interface Message:

  Hash Table Full

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The hash table has all the keys filled with values. The hash tables in Neuronment does not store lists under colissions.

4.3.10 DV-015

- Interface Message:

  Setting declared but has not been initialized

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The variable has been declared but it does not have any value stored. This is unexpected as at the variable declaration a default value should have been declared and stored.

4.3.11 DV-017

- Interface Message:

  Discrepancy on neuron type on assignment

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  A instance of a neuron of certain type is trying to be copied on an instance of a neuron of another incompatible type.

4.3.12 DV-018

- Interface Message:

  Trying to quick retrieve un-existent value

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  When trying to quick retrieve a value from a single element value list, the function in use is triying to retrieve a value from a a variable that does not stores any value. This is unexpected as it should have stored a default value at the variable declaration.

4.3.13 SD-030

- Interface Message:

  Internal simulation pointer corruption

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The simulator is trying to start the claculation of a new step but the counter of steps for the calculations for the activation and the derivate of activation are different. This means that there is an uncomplete calculation from the previous step.

4.3.14 SD-031

- Interface Message:

  Trying to compare 2 identical neurons

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO

- Message Description:

  When comparing 2 different instances of a neuron of a same type, there is no difference in the instance attributes that allows to identify each one unequivocaly. This is unexpected as at minimum 2 different instances of a neuron should have al least a different name.

4.3.15 SD-032

- Interface Message:

  Empty pointer to function entry

- Development Assertion: NO
- Implementation Assertion: YES
- Runtime Assertion: NO
- Message Description:

  The interpreter has identified a directive that has not a valid function to execute.

4.4 Reporting Issues

4.4.1 RP-002

- Interface Message:

  Trying to close a non opened stream

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The interpreter is trying to close a redirection file that has not been oppened.

4.4.2 RP-003

- Interface Message:

  Stream did notoppened correctly

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES

- Message Description:

  When trying to open a file for redirection the OS failed to give a successful operation result.

4.5 Simulator Interface Issues

4.5.1 SD-001

- Interface Message:

  Simple Simulator Not Initialized

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  Trying to start a simulation on a simulator that has not been build.

4.5.2 SD-015

- Interface Message:

  No V1_Neuron created

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  There are not V1 neurons declared and no neuron has been build for the simulation. The simulator requires at least one V1 neuron to work.

4.5.3 SD-016

- Interface Message:

  No MT_Neuron created

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES

- Message Description:

  There are not MT neurons declared and no neuron has been build for the simulation. The simulator requires at least one MT neuron to work.

4.5.4 SD-021

- Interface Message:

  Invalid timing for eternal excitation

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The simulator is trying to access a negative time step. All time steps have a equal or greater than zero value.

4.5.5 SD-025

- Interface Message:

  Unordered external excitation phase insertion attempt

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  When declaring the external exitations, the fases should be declared in consecutive order. In this case there seems to be a gap at the moment of declaring the different phases.

4.5.6 SD-027

- Interface Message:

  First diffusion phase should be always zero

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES

- Message Description:

  When declaring the external exitations, the fases should be declared in consecutive order. The first phase should be declared as phase 0.

4.5.7 SD-028

- Interface Message:

  The number of steps for simulate needs to be at least 1

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  The miminum number of steps to calculate a simulation is 1. In this case, the value configured for this number of steps is zero or negative.

4.5.8 SD-033

- Interface Message:

  Trying to access an invalid activation TimeStep

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  Each time step has a index number from zero up to the number of steps simulated minus one. In this case, there is an attempt to access a timestep outside this range.

4.5.9 SD-036

- Interface Message:

  Setting not recognized

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES

- Message Description:

  This happens when a user or default configured setting has a value that cannot be used for simulation.

4.5.10 SD-037

- Interface Message:

  Setting required

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: YES
- Message Description:

  When trying to retrieve a variable value, the variable does not exists.

4.6 Warnings

4.6.1 WN-006

- Interface Message:

  There are undocumented calculations in use

- Development Assertion: NO
- Implementation Assertion: NO
- Runtime Assertion: NO
- Message Description:

  The simulator is calculating using mathematical functions that does not have documented references.

Bibliography

[1] Pedro F. Toledo. *Implementation of a multithreaded numeric genetic algorithm with decreasing mutation impact for training of a visual cortex simulator*. UTFSM, Valparaíso, Chile, 2014.

Appendix A

First Appendix