XML Input for INIT

Redesigning the Input File Format for INIT

Remy Oukaour
Dept. of Computer Science
Stony Brook University
Stony Brook, NY 11794

August 25, 2014

Contents

1. Introduction

The Brain Organization Simulation System (BOSS) developed at Stony Brook University models electrical activity in neuron networks. It runs in two stages: the initializer, INIT, generates a cerebellar model, and then the simulator, RUNSIM, processes activity in the model over time.

INIT takes as input a set of cell types and parameters controlling how cells of each type are distributed and synaptically connected in the overall model. It then generates a particular model with cells, synapses, and gap junctions placed according to its input.

Previously input to INIT was given by text files in a custom format. It has now been extended to take XML files as input, with a schema of elements and attributes that specify the same parameters as the original text files, but in a more standardized and extensible format.

2. Version History

The original version of BOSS in 1970 used a custom text-based input format. Parameter names were in all capital letters and had to be terse due to a length limit. Floating-point calculations were slow and inconsistent across platforms, so all values were specified as integers.

Khan “Sadh” N. Mostafa began developing an XML input format in November 2013. He settled on the RapidXml parser and began integrating it with INIT’s existing data structures.

Remy Oukaour, Jack Zito, and Heraldo Memelli finished designing and implementing the XML input format in August 2014. The current version of BOSS can read from either text or XML files, and a suitably written XML file can exactly replicate the output of a corresponding text file.

3. XML Syntax

An XML file consists of arbitrarily nested elements formatted as follows:

```
<name>content</name>
```

The element is delimited by an opening tag `<name>` and a closing tag `</name>`. Its content can be plain text, or it can contain one or more sub-elements. If an element has no content, it can be abbreviated as `<name/>`.

Elements also possess attributes formatted as follows:

```
<name attr1='value1' attr2="value2">content</name>
```

The attributes' values are delimited by single or double quotes. The values are plain text only, and cannot contain sub-elements.

Comments are formatted as follows:

```
<!--comment-->
```

Comments are delimited by "`<!--`" and "`-->`" sequences and can include any text, even syntactically invalid XML. They cannot be nested, since a comment ends at the first appearance of "`-->`".

The characters "<" and "&" are special and cannot appear in plain text.

An XML file must have a single root element that contains the content of the entire file (although comments can appear outside the root element). INIT input files have an `<input>` root element.

3.1. Value Syntaxes

Empty or missing attributes are usually taken to have a default value, although some attributes are required and INIT will halt if they are missing.

Most attributes are 32-bit signed integers. Integers are formatted as a sequence of digits, optionally preceded by a "+" or "-" sign, and with any number of "_" characters throughout. This allows long numbers to be more easily read; for example, the number 123,456,789 can be formatted "`123_456_789`". Integers also have optional suffixes for large amounts: "K" or "k" multiplies the value by one thousand, "M" or "m" by one million, and "B" or "b" by one billion. For example, the number −10,000,000 can be formatted "`-10M`".

Some attributes are pairs of integers representing a minimum and maximum. These pairs are formatted as two integers separated by a ":". For example, the pair [−10, 10] can be formatted "`-10:10`". If either integer in the pair is missing, it will take a default value.

The contents of some elements are lists of integers. These are sequences of integers separated by "," characters. List items can also be repeated by following them with a "*" and a second integer, which is the item count. For example the list {1, 1, 1, 2} can be formatted "`1*3, 2`". Non-positive counts, and incorrectly formatted or missing items, do not add any items to the list. For example, "`1*1, 2*2, 3*0, 3*-1, , 4,`" and "`1,2,2,4`" are both parsed as the list {1, 2, 2, 4}.

4. RapidXml

INIT uses the RapidXml library to parse its XML input files. RapidXml is a C++ library without any dependencies of its own. It is an *in-situ* parser, so it does not use significantly more memory than is needed to store the unparsed text content of an XML file. This avoidance of data copying also makes it very fast. RapidXml v1.13 can be downloaded from rapidxml.sourceforge.net.

5. Internal Data Structures

The internal data structures of INIT were slightly changed to allow for taking input from two different file formats.

5.1. `Params` Class

A single instance of the `Params` class stores general parameters not specific to any cell type. This includes parameters regarding simulation cycles, simulation space limits, Izhikevich model parameters, forced firing filters, and STDP adjustments. Originally the `Params` class had a `GetParams` method which took a `Config` instance as an argument and used it to parse its own parameters. The `GetParams` method now belongs to the `Config` class, which is passed the `Params` instance.

5.2. `CellTypeInfo` Class

A single instance of the `CellTypeInfo` class stores type-specific parameters. This includes parameters regarding soma centers, axonal fields, dendritic fields, synapse, gap junctions, and Izhikevich model parameters. Originally the `CellTypeInfo` class had a `GetCellTypeInfo` method which took a `Config` instance as an argument and used it to parse its own parameters. The `GetCellTypeInfo` method now belongs to the `Config` class, which is passed the `CellTypeInfo` instance.

5.3. `Config`, `TextConfig`, and `XmlConfig` Classes

The `TextConfig` and `XmlConfig` classes inherit from the abstract `Config` class. A single instance of one stores data read from a configuration file. Originally a `Config` instance was passed to `Params` and `CellTypeInfo` instances, which were responsible for using the `Config` instance to parse their own parameters. Now the `TextConfig` and `XmlConfig` classes have different implementations of the `GetParams` and `GetCellTypeInfo` methods, and these differences are encapsulated since both inherit from the `Config` class.

6. <input> Element

The <input> element is the root element of the input file. Its children are the <params> and <types> elements.

6.1. <params> Element

The <params> or <parameters> element contains the general parameters not specific to any cell type. Its attributes are:

- maxc or maxcycle: The last cycle for a simulation run. Required.
- mspcT or mspercycleT: The cycle step time, in thousandths of milliseconds per cycle. Required. Default 1,000 (i.e. 1 ms/cycle). Valid values are {100, 200, 250, 500, 1,000, 2,000, 3,000, 4,000, 5,000, 6,000, 7,000, 8,000, 9,000, 10,000}.
- minfire or minfireactive: The minimum number of somas that must fire during one cycle for it to be considered "active." Default 2. Not yet implemented.
- maxnofirec or maxinactivecycles: The maximum number of inactive cycles before halting the simulation. Default 10. Not yet implemented.
- synminpT or synapseminprobT: The minimum probability of forming a synapse, in thousandths. Default 1000 (i.e. 1.0 or 100%).
- gjs or gapjunctions: Whether or not there will be gap junctions. Default 0 (i.e. false).

Its children are the <simspace>, <izhi>, <filters>, and <stdpsT> elements.

6.1.1. <simspace> Element

The <simspace> element defines parameters specifying the size of the overall simulation space. Its attributes are:

- mmx or xminmax: The minimum and maximum *x* coordinates for the width of the simulation space, in micrometers. Default [0 μm, 10,000 μm].
- mmy or yminmax: The minimum and maximum *y* coordinates for the height of the simulation space, in micrometers. Default [0 μm, 10,000 μm].
- mmz or zminmax: The minimum and maximum *z* coordinates for the depth of the simulation space, in micrometers. Default [0 μm, 2,000 μm].

6.1.2. <izhi> Element

The <izhi> or <izhikevich> element defines parameters of the Izhikevich model for spiking neurons. Its attributes are:

- v or voltage: The initial voltage (potential) for each cell, in millivolts. Default −70 mV.
- u: The Izhikevich recovery parameter *u* for each cell. Default 0.
- vth or voltagethreshold: The threshold voltage for a firing spike. Default 30 mV.
- v2cT or v2coefficientT: The quadratic coefficient in the Izhikevich voltage equation, in thousandths. Default 40 (i.e. 0.04). Not yet implemented.
- v1c or v1coefficient: The linear coefficient in the Izhikevich voltage equation. Default 5. Not yet implemented.
- v0c or v0coefficient: The constant coefficient in the Izhikevich voltage equation, in millivolts. Default 140 mV. Not yet implemented.

6.1.3. `<filters>` Element and `<filter>` Elements

The `<filters>` or `<forcedfiringfilters>` element contains multiple `<filter>` elements for artificially force-firing certain cells. Each `<filter>` element's attributes are:

- id or index: The index of the filter (e.g. the first is 0, the second is 1, etc.). The first filter is a special progressive mesh (flyswatter) pattern.
- type: The letter or ID of the cell type affected by the filter.
- mmt or timeminmax: The start and end time for the filter, in milliseconds. The start time is clamped to at least 0, and the end time is clamped to at least −1.
- r or recur: The recurrence frequency of the filter, in milliseconds.
- mmx or xminmax: The minimum and maximum *x* coordinates of the filter's bounding box width, in micrometers.
- mmy or yminmax: The minimum and maximum *y* coordinates of the filter's bounding box height, in micrometers.
- mmz or zminmax: The minimum and maximum *z* coordinates of the filter's bounding box depth, in micrometers.
- fracT or fractionT: The fraction of cells forced to fire, in thousandths. The values "-1" or "image" will use a 2D firing pattern defined by a PGM image named "image_filter_*ID*.pgm" (or "image_filter.pgm" if an ID-specific one does not exist).

6.1.4. `<stdpsT>` Element

The `<stdpsT>` or `<stdpadjustmentsT>` element contains a list of STDP (spike-timing-dependent plasticity) adjustment factors in thousandths. INIT warns if the number of factors is even. Factors must be in the range [0, 1,000].

6.2. <types> Element and <type> Elements

The <types> element contains multiple <type> elements, each of which defines parameters for a single cell type. Its attributes are:

- n or count: The number of defined cell types. Must match the actual number of <type> elements. Required.
- chars or characters: A space-separated list of the types' letters, ordered by type ID. Must match the actual types' letters. Required.

Each <type> element's attributes are:

- id or index: The ID of the type. The first must be 0, the second must be 1, etc. Required.
- char or character: The capital letter representing the type. Required.

Each <type> element's children are the <somas>, <axons>, <dens>, <syns>, <gjs>, and <izhi> elements.

6.2.1. <somas> Element and <zlvs> Element

The <somas> or <somacenters> element defines parameters for the location and spacing of cells' soma centers. Its attributes are:

- dx or xspacing: The transverse spacing of the somas' *x* coordinates, in micrometers. Default 100 μm.
- dy or yspacing: The longitudinal spacing of the somas' *y* coordinates, in micrometers. Default 1,000 μm. Gets multiplied by a dilution factor.
- dz or zspacing: The vertical spacing of the somas' *z* coordinates, in micrometers. Default 100 μm.
- altox or altxoffset: The offset of the somas' *x* coordinates in alternate transverse rows, in micrometers. Default 0 μm.
- mmx or xminmax: The minimum and maximum *x* coordinates for the somas, in micrometers. Default [1 μm, MSIMXMX μm].
- mmy or yminmax: The minimum and maximum *y* coordinates for the somas, in micrometers. Default [1 μm, MSIMYMX μm].

- mmz or zminmax: The minimum and maximum *z* coordinates for the somas, in micrometers. Default [1 μm, *D* μm]. The minimum is clamped to at least MSIMZMN, and the maximum is clamped to at most MSIMZMX. The default maximum value *D* is calculated as follows:

```
D = MCZLVL + MCZ * CZLN;
if (D < 0 && MCZ < 0) { D++; }
else { D--; }
```

  D is thus set to 1 μm tighter than the space required for another row of cells. This maximizes the universe without creating edges too close to the extreme cell centers.

The <somas> element's child is the <zlvs> element. The <zlvs> or <zlevels> element contains parameters that vary over the different vertical *z* levels of somas. Its attributes are:

- n or count: The number of *z* levels for soma centers. Default 1. Range [1, 100].

The <zlvs> element's children are the <psT> and <asfsT> elements.

The <psT> or <probabilitiesT> element contains a list of probabilities that a soma center will be on each level, in thousandths. Default 1,000 (i.e. 1.0 or 100%).

The <aspfsT> or <axonspeedfactorsT> element contains a list of factors to get axonal speeds of somas on each level, in thousandths. Default 1,000 (i.e. 1×).

6.2.2. <axons> Element and <field> Elements

The <axons> or <axonal> element contains multiple <field> elements, each of which defines parameters for a single axonal field. Its attributes are:

- n or count: The number of defined axonal fields. Must match the actual number of <field> elements.

Each <field> element's attributes are:

- wx or xwidth: The field's width along the *x* axis, in micrometers. Default 0 μm.
- wy or yheight: The field's height along the *y* axis, in micrometers. Default 0 μm.
- wz or zdepth: The field's depth along the *z* axis, in micrometers. Default 0 μm.
- ox or xoffset: The field's transverse offset along the *x* axis from the soma center, in micrometers. Default 0 μm.
- oy or yoffset: The field's longitudinal offset along the *y* axis from the soma center, in micrometers. Default 0 μm.
- oz or zoffset: The field's vertical offset along the *z* axis from the soma center, in micrometers. Default 0 μm.

- asp or axonalspeed: The field's axonal speed, in millimeters per second (or micrometers per millisecond). Default 300 mm/s.
- dcfT or densitycorrectionfactorT: The field's density correction factor, in thousandths. Default 1,000 (i.e. 1×).

6.2.3. `<dens>` Element and `<field>` Elements

The `<dens>` or `<dendritic>` element contains multiple `<field>` elements, each of which defines parameters for a single dendritic field. Its attributes are:

- n or count: The number of defined dendritic fields. Must match the actual number of `<field>` elements.

Each `<field>` element's attributes are:

- wx or xwidth: The field's width along the *x* axis, in micrometers. Default 0 µm.
- wy or yheight: The field's height along the *y* axis, in micrometers. Default 0 µm.
- wz or zdepth: The field's depth along the *z* axis, in micrometers. Default 0 µm.
- ox or xoffset: The field's transverse offset along the *x* axis from the soma center, in micrometers. Default 0 µm.
- oy or yoffset: The field's longitudinal offset along the *y* axis from the soma center, in micrometers. Default 0 µm.
- oz or zoffset: The field's vertical offset along the *z* axis from the soma center, in micrometers. Default 0 µm.
- mmdsp or dendriticspeedminmax: The field's dendritic speeds at the tip of the neuritic region (minimum) and at the soma (maximum), in millimeters per second (or micrometers per millisecond). Both are clamped to at least 1, and swapped if necessary to keep the minimum less than the maximum. Default [300 mm/s, 300 mm/s].
- mmpfT or pulseattenuationfactorminmaxT: The field's minimum and maximum pulse attenuation factors, in thousandths. Both are clamped to within the range [1, 1,000] (i.e. [0.001×, 1×]), and swapped if necessary to keep the minimum less than the maximum. Default [1,000, 1,000] (i.e. [1×, 1×]).
- lnrxT or linearroundingfactorexponentT: The exponent of the dendritic linear rounding factor, in thousandths. Clamped to the within range [100, 10,000] (i.e. [0.1, 10]). Default 1,000 (i.e. 1).
- dcfT or densitycorrectionfactorT: The field's density correction factor, in thousandths. Default 1,000 (i.e. 1×).

6.2.4. <syns> Element and Child Elements

The <syns> or <synapses> element contains parameters for synapses between pairs of cell types. Its children are the <ots>, <daodsT>, and <owsT> elements.

The <ots> or <outputtypes> element defines parameters for types of individual output synapses. Values are a bitmask of the following flags:

- 1 (binary 00001) indicates a learning synapse.
- 2 (binary 00010) indicates a suppressible connection.
- 4 (binary 00100) indicates a binary firing synapse.
- 8 (binary 01000) indicates that the synapse triggers suppression according to its weight.
- 16 (binary 10000) indicates a synapse that does not incur a global delay.

The default type bitmask is 0.

The <daodsT> or <denaxonoverlapdensitiesT> element defines parameters for synapse densities where the input type's dendritic regions overlap the output type's axonal regions, in thousandths. The default density factor is 0.

The <owsT> or <outputinitialweightsT> element defines parameters for initial synaptic weights of output synapses, in thousandths of millivolts (or microvolts). The default initial weight is 0 mV.

The <ots>, <daodsT>, and <owsT> elements define their pairwise parameters as attributes named by output types' letters. The "rest" attribute specifies a value for any output types which are not otherwise specified. For example, to specify a P-to-D initial weight of −16 mV, with all other initial weights from P-type somas being 0 mV, write the following:

```
<type id="0" char="P">
    <syns>
        <owsT D="-16K" rest="0"/>
    </syns>
</type>
```

6.2.5. <gjs> Element

The <gjs> or <gapjunctions> element defines parameters for gap junctions. Its attributes are:

- aapT or axonaxonprobT: The probability of an axon-axonal gap junction, in thousandths. Default 0.

- ddpT or dendenprobT: The probability of a dendro-dendritic gap junction, in thousandths. Default 0.
- sspT or somasomaprobT: The probability of a soma-somatic gap junction, in thousandths. Default 0.
- adpT or axondenprobT: The probability of an axon-dendritic gap junction, in thousandths. Default 0.
- aspT or axonsomaprobT: The probability of an axon-somatic gap junction, in thousandths. Default 0.
- dspT or densomaprobT: The probability of a dendro-somatic gap junction, in thousandths. Default 0.
- condT ot conductanceT: The conductance of individual gap junctions, in thousandths of milliSiemens (i.e. microSiemens). Default 0 µS. Range [0 µS, 1,000,000 µS].

6.2.6. `<izhi>` Element

The `<izhi>` or `<izhikevich>` element defines type-specific parameters of the Izhikevich model that affect the type's firing patterns. Its attributes are:

- aT: The *a* parameter, which describes the time scale of the recovery variable *u*, in thousandths. Default 20 (i.e. 0.02).
- bT: The *b* parameter, which describes the sensitivity of the recovery variable *u* to subthreshold fluctuations of the membrane potential *v*, in thousandths. Default 200 (i.e. 0.2).
- cT: The *c* parameter, which describes the after-spike reset value of the membrane potential *v* caused by the fast high-threshold K^+ conductances, in thousandths of millivolts. Default −65,000 (i.e. −65 mV).
- dT: The *d* parameter, which describes after-spike reset of the recovery variable *u* caused by slow high-threshold Na^+ and K^+ conductances, in thousandths. Default 8,000 (i.e. 8).

A. Source Code Files

These files are part of the source code for INIT. Only the files which were changed to implement XML input are listed.

- Params.cpp and Params.h: Define the `Params` class for storing general model parameters.
- CellTypeInfo.cpp and CellTypeInfo.h: Define the `CellTypeInfo` class for storing type-specific parameters.
- Config.cpp and Config.h: Define the abstract `Config` class for reading and parsing a configuration file.

- TextConfig.cpp and TextConfig.h: Define the `TextConfig` class for reading and parsing a text configuration file.
- XmlConfig.cpp and XmlConfig.h: Define the `XmlConfig` class for reading and parsing an XML configuration file.
- Init.cpp: Implements methods of the `Init` class for generating a particular brain model from a general description. Implementing XML input required changing how the configuration file was read, and adding debug output for all the parsed parameters to compare text and XML results.
- Utils.cpp and Utils.h: Define utility classes and methods for performing common functions. Implementing XML input required a method for getting a file's extension.
- rapidxml/*.hpp: Implement the RapidXml library for quickly parsing XML.
- InputCat.txt: Defines a model of the cat cerebellum using 12 cell types. Some deprecated and unused parameters were removed.
- InputCat.xml: Defines a model of the cat cerebellum using 12 cell types. The XML format exactly replicates the model defined by InputCat.txt.