All the variables defined in EDLUT dictionaries are written in lower case using the snake_case format. Here can be seeing some examples: name, integ_meth, conf_filename, tab_filename, a, b, c, d, e_exc, e_inh, c_m, tau_exc, tau_inh, e_leak, g_leak, v_th, tau_ref, v_spk_peak, ep_cap, thr_slo_fac, tau_w.

1. NEURON MODELS

This section defines all the event-driven and time-driven neuron models included in EDLUT with their parameters.

1.1. NeuronModel

This class abstracts the behavior of a neuron in a spiking neural network. It includes internal model functions which define the behavior of the model (initialization, update of the state, synapses effect...). This is only a virtual function (an interface) which defines the functions of the inherited classes.

Parameters:

- newMap["name"] = boost::any(this->name); //string: neuron model name **(AUTOMATICALLY SET BY DERIVED CLASES)**.

1.1.1. EventDrivenInputDevice

This file declares a class which abstracts an event-driven input device in CPU. These devices can inject input spikes and current in the neural network.

Without parameters.

1.1.1.1. InputCurrentNeuronModel

This class defines the behavior of an input neuron layer that can propagate input currents to the neural network.

Without parameters.

1.1.1.2. InputSpikeNeuronModel

This class defines the behavior of an input neuron layer that can propagate input spikes to the neural network.

Without parameters.

1.1.2. EventDrivenNeuronModel

This class abstracts the behavior of an event-driven neuron model in a spiking neural network. It includes internal model functions which define the behavior of the model (initialization, update of

the state, synapses effect, next firing prediction...). This is only a virtual function (an interface) which defines the functions of the inherited classes.
Without parameters.

1.1.2.1. CompressTableBasedModel

This class implements the behavior of event-driven spiking neuron models using precalculated look-up tables to "predict" the neuron behavior. This "Compress" version can merge several look-up tables with the same indexes in just one, minimizing the look-up time (ideal for complex neuron models such as HH with several state variables and look-up tables).
Parameters:

- newMap["conf_filename"] = boost::any(this->conf_filename); //String: name of configuration file.cfg where is defined the look-up tables structure
- newMap["tab_filename"] = boost::any(this->tab_filename); //String: name of file.dat where are defined the look-up tables.

1.1.2.1.1. CompressSynchronousTableBasedModel

This class implements the behavior of event-driven spiking neuron models using precalculated look-up tables to "predict" the neuron behavior. This "CompressSynchronous" version not only can merge several look-up tables with the same indexes in just one, such as his parent class, but also can synchronize the output activity, minimizing the time required to process the output spikes.
Parameters:

- newMap["conf_filename"] = boost::any(this->conf_filename); //String: name of configuration file.cfg where is defined the look-up tables structure
- newMap["tab_filename"] = boost::any(this->tab_filename); //String: name of file.dat where are defined the look-up tables.

1.1.2.2. TableBasedModel

This class implements the behavior of event-driven spiking neuron models using precalculated look-up tables to "predict" the neuron behavior.
Parameters:

- newMap["conf_filename"] = boost::any(this->conf_filename); //String: name of configuration file.cfg where is defined the look-up tables structure
- newMap["tab_filename"] = boost::any(this->tab_filename); //String: name of file.dat where are defined the look-up tables.

1.1.2.2.1. SynchronousTableBasedModel

This class implements the behavior of event-driven spiking neuron models using precalculated look-up tables to "predict" the neuron behavior. This "Synchronous" version can synchronize the output activity, minimizing the time required to process the output spikes.
Parameters:

- newMap["conf_filename"] = boost::any(this->conf_filename); //String: name of configuration file.cfg where is defined the look-up tables structure
- newMap["tab_filename"] = boost::any(this->tab_filename); //String: name of file.dat where are defined the look-up tables.

1.1.3. TimeDrivenModel

This class abstracts the behavior of time-driven neuron models in spiking neural networks. It includes internal model functions which define the behavior of the model (initialization, update of

the state, synapses effect...). This is only a virtual function (an interface) which defines the functions of the inherited classes.
Without parameters.

1.1.3.1. TimeDrivenNeuronModel

This class abstracts the behavior of time-driven neuron models in spiking neural networks implemented in CPU. It includes internal model functions which define the behavior of the model (initialization, update of the state, synapses effect...). This is only a virtual function (an interface) which defines the functions of the inherited classes.
Parameters:

- newMap["int_meth"] = imethod; //INTEGRATION METHODS DEFINED IN SECTION 2.

1.1.3.1.1. AdExTimeDrivenModel

This class implement an AdEx Time-Driven neuron model with a membrane potential (V), a membrane recovery variable (w), three conductances (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["a"] = boost::any(this->a); //conductance (nS)
- newMap["b"] = boost::any(this->b); //spike trigger adaptation (pA)
- newMap["thr_slo_fac"] = boost::any(this->thr_slo_fac); //threshold slope factor (mV)
- newMap["v_thr"] = boost::any(this->v_thr); //effective threshold potential (mV)
- newMap["tau_w"] = boost::any(this->tau_w); //adaptation time constant (ms)
- newMap["e_exc"] = boost::any(this->e_exc); //excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); //inhibitory reversal potential (mV)
- newMap["e_reset"] = boost::any(this->e_reset); //reset potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); //effective leak potential (mV)
- newMap["g_leak"] = boost::any(this->g_leak); //leak conductance (nS)
- newMap["c_m"] = boost::any(this->c_m); //membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); //AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); //GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); //NMDA (excitatory) receptor time constant (ms)

1.1.3.1.2. EgidioGranuleCell_TimeDriven

This class implements a detailed Leaky Integrate-And-Fire Time-Driven neuron model for a cerebellar granule cell. This neuron model includes 15 state variables modelling the neural dynamics, three conductances (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Without parameters.

1.1.3.1.3. HHTimeDrivenModel

This class implements a Hodgkin and Huxley neuron model with four differential equations (membrane potential, m, h and n), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); //Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential(mV)
- newMap["g_leak"] = boost::any(this->g_leak); // Leak conductance (nS)
- newMap["c_m"] = boost::any(this->c_m); // Membrane capacitance (pF)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory NMDA) receptor time constant (ms)
- newMap["g_na"] = boost::any(this->g_na); // Maximum value of sodium conductance (nS)
- newMap["g_kd"] = boost::any(this->g_kd); // Maximum value of potassium conductance (nS)
- newMap["e_na"] = boost::any(this->e_na); // Sodium potential (mV)
- newMap["e_k"] = boost::any(this->e_k); // Potassium potential (mV)

1.1.3.1.4. IzhikevichTimeDrivenModel

This class implements an Izhikevich neuron model with two differential equations (membrane potential and membrane recovery), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["a"] = boost::any(this->a); // Time scale of recovery variable u (dimensionless)
- newMap["b"] = boost::any(this->b); // Sensitivity of the recovery variable u to the subthreshold fluctuations of the membrane potential v (dimensionless)
- newMap["c"] = boost::any(this->c); // After-spike reset value of the membrane potential v (dimensionless)
- newMap["d"] = boost::any(this->d); // After-spike reset of the recovery variable u (dimensionless)
- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["c_m"] = boost::any(this->c_m); // Membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)

1.1.3.1.5. LIFTimeDrivenModel

This class implements a Leaky Integrate-And-Fire (LIF) neuron model with one differential equation (membrane potential), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)

- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["c_m"] = boost::any(float(this->c_m)); // Membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)
- newMap["g_leak"] = boost::any(float(this->g_leak)); // Leak conductance (nS)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)

1.1.3.1.6. LIFTimeDrivenModel_IS

This class implements a Leaky Integrate-And-Fire (LIF) neuron model with one differential equation (membrane potential), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse. In this case, the model uses the units defined in the International System (IS): seconds (s), volts (V), faradiums (F) and siemens (S) instead of miliseconds (ms), milivolts (mV), picofaradiums (pF) and nanosiemens (nS).
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (V)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (V)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (V)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (V)
- newMap["c_m"] = boost::any(float(this->c_m)); // Membrane capacitance (F)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (s)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (s)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (s)
- newMap["g_leak"] = boost::any(float(this->g_leak)); // Leak conductance (S)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (s)

1.1.3.1.7. TimeDrivenInferiorOliveCell

This class implements an Inferior Olive cell as a Leaky Integrate-And-Fire (LIF) neuron model with one differential equation (membrane potential), three time dependent equations (excitatory, inhibitory and NMDA conductances), one external input current synapse, and one electrical coupling synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)
- newMap["c_m"] = boost::any(this->c_m); // Membrane capacitance (uF/cm^2)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)

- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)
- newMap["g_leak"] = boost::any(this->g_leak); // Leak conductance (mS/cm^2)
- newMap["area"] = boost::any(this->area); // Cell area (cm^2)

1.1.3.1.8. TimeDrivenPurkinjeCell

This class implements a Purkinje cell model with three differential equations (membrane potential, calcium and Muscariny channels), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)

1.1.3.1.9. TimeDrivenPurkinjeCell_IP

This class implements a Purkinje cell model with four differential equations (membrane potential, calcium and Muscariny channels and membrane capacitance), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)
- newMap["ep_c_m"] = boost::any(this->ep_c_m); // Epsilon capacitance (uF*ms/cm^2)

1.1.3.2. TimeDrivenNeuronModel_GPU

This class abstracts the behavior of time-driven neuron models in spiking neural networks implemented in GPU (with an additional C_INTERFACE class in CPU to manage the GPU class). It includes internal model functions which define the behavior of the model (initialization, update of the state, synapses effect...). This is only a virtual function (an interface) which defines the functions of the inherited classes.
Parameters:

- newMap["int_meth"] = imethod; //INTEGRATION METHODS DEFINED IN SECTION 2.

1.1.3.2.1. AdExTimeDrivenModel_GPU

This class implement an AdEx Time-Driven neuron model with a membrane potential (V), a membrane recovery variable (w), three conductances (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["a"] = boost::any(this->a); //conductance (nS)
- newMap["b"] = boost::any(this->b); //spike trigger adaptation (pA)
- newMap["thr_slo_fac"] = boost::any(this->thr_slo_fac); //threshold slope factor (mV)
- newMap["v_thr"] = boost::any(this->v_thr); //effective threshold potential (mV)
- newMap["tau_w"] = boost::any(this->tau_w); //adaptation time constant (ms)
- newMap["e_exc"] = boost::any(this->e_exc); //excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); //inhibitory reversal potential (mV)
- newMap["e_reset"] = boost::any(this->e_reset); //reset potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); //effective leak potential (mV)
- newMap["g_leak"] = boost::any(this->g_leak); //leak conductance (nS)
- newMap["c_m"] = boost::any(this->c_m); //membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); //AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); //GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); //NMDA (excitatory) receptor time constant (ms)

1.1.3.2.2. EgidioGranuleCell_TimeDriven_GPU

This class implements a detailed Leaky Integrate-And-Fire Time-Driven neuron model for a cerebellar granule cell. This neuron model includes 15 state variables modelling the neural dynamics, three conductances (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Without parameters.

1.1.3.2.3. HHTimeDrivenModel_GPU

This class implements a Hodgkin and Huxley neuron model with four differential equations (membrane potential, m, h and n), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); //Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential(mV)
- newMap["g_leak"] = boost::any(this->g_leak); // Leak conductance (nS)
- newMap["c_m"] = boost::any(this->c_m); // Membrane capacitance (pF)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)

- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory NMDA) receptor time constant (ms)
- newMap["g_na"] = boost::any(this->g_na); // Maximum value of sodium conductance (nS)
- newMap["g_kd"] = boost::any(this->g_kd); // Maximum value of potassium conductance (nS)
- newMap["e_na"] = boost::any(this->e_na); // Sodium potential (mV)
- newMap["e_k"] = boost::any(this->e_k); // Potassium potential (mV)

1.1.3.2.4. IzhikevichTimeDrivenModel_GPU

This class implements an Izhikevich neuron model with two differential equations (membrane potential and membrane recovery), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["a"] = boost::any(this->a); // Time scale of recovery variable u (dimensionless)
- newMap["b"] = boost::any(this->b); // Sensitivity of the recovery variable u to the subthreshold fluctuations of the membrane potential v (dimensionless)
- newMap["c"] = boost::any(this->c); // After-spike reset value of the membrane potential v (dimensionless)
- newMap["d"] = boost::any(this->d); // After-spike reset of the recovery variable u (dimensionless)
- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["c_m"] = boost::any(this->c_m); // Membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)

1.1.3.2.5. LIFTimeDrivenModel_GPU

This class implements a Leaky Integrate-And-Fire (LIF) neuron model with one differential equation (membrane potential), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["c_m"] = boost::any(float(this->c_m)); // Membrane capacitance (pF)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)
- newMap["g_leak"] = boost::any(float(this->g_leak)); // Leak conductance (nS)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)

1.1.3.2.6. LIFTimeDrivenModel_IS_GPU

This class implements a Leaky Integrate-And-Fire (LIF) neuron model with one differential equation (membrane potential), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse. In this case, the model uses the units defined in the International System (IS): seconds (s), volts (V), faradiums (F) and siemens (S) instead of miliseconds (ms), milivolts (mV), picofaradiums (pF) and nanosiemens (nS).
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (V)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (V)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (V)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (V)
- newMap["c_m"] = boost::any(float(this->c_m)); // Membrane capacitance (F)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (s)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (s)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (s)
- newMap["g_leak"] = boost::any(float(this->g_leak)); // Leak conductance (S)
  newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (s)

1.1.3.2.7. TimeDrivenPurkinjeCell_GPU

This class implements a Purkinje cell model with three differential equations (membrane potential, calcium and Muscariny channels), three time dependent equations (excitatory, inhibitory and NMDA conductances) and one external input current synapse.
Parameters:

- newMap["e_exc"] = boost::any(this->e_exc); // Excitatory reversal potential (mV)
- newMap["e_inh"] = boost::any(this->e_inh); // Inhibitory reversal potential (mV)
- newMap["v_thr"] = boost::any(this->v_thr); // Effective threshold potential (mV)
- newMap["e_leak"] = boost::any(this->e_leak); // Effective leak potential (mV)
- newMap["tau_exc"] = boost::any(this->tau_exc); // AMPA (excitatory) receptor time constant (ms)
- newMap["tau_inh"] = boost::any(this->tau_inh); // GABA (inhibitory) receptor time constant (ms)
- newMap["tau_nmda"] = boost::any(this->tau_nmda); // NMDA (excitatory) receptor time constant (ms)
- newMap["tau_ref"] = boost::any(this->tau_ref); // Refractory period (ms)

2. INTEGRATION METHODS

This section defines the integration methods used in all the time-driven neuron models, both in CPU and GPU. These methods are classified in two sub-groups: fixed-step and bifixed-step integration methods.

2.1. IntegrationMethod and IntegrationMethod_GPU

These classes respectively abstract the behavior of all the integration methods in CPU and GPU for all the time-driven neural models defined in CPU and GPU. They include internal model functions

which define the behavior of the integration methods (initialization, calculate next value, ...). These are only virtual functions (interfaces) which define the functions of the inherited classes.

Parameters:

- newMap["step"] = boost::any(this->elapsedTimeInSeconds); // Fixed-step size defined in seconds (larger step size in bifixed methods)
- newMap["name"] = boost::any(this->name); // Integration method name **(AUTOMATICALLY SET BY DERIVED CLASES)**.

2.1.1. IntegrationMethodFast and IntegrationMethodFast_GPU

These intermediate classes are respectively defined in CPU and GPU to include a template pointer to the neuron models that must be integrated. This configuration using templates allows the CPU and GPU compilers to perform several optimizations in the integration process for each specific combination of neuron model and integration methods (loops unrolling, inline functions even with inherit virtual methods, etc.). All the possible neuron model and integration method combinations are set in the IntegrationMethodFactory and IntegrationMethodFactory_GPU classes.

Without parameters.

2.1.1.1. *FixedStep and FixedStep_GPU*