

*insilico*IDE 1.1 Getting Started (CLI)

In this document, we make Bonhoeffer van der Pol model (BVP) model in *insilico* terminal. Each section is same as Getting Started GUI vesion.

Initial setting

Create handlers and set prefix 'm' at MathML tag.

Set prefix 'm' at MathML tag (example: <math>, <apply> etc.)

```
>>> cstr2mml.set_ns_prefix("m")
```

At *insilico* terminal, users access *insilico*ML through handlers. This document uses the following handlers. "model" means model-handler. When users start *insilico*IDE, this model-hanlder is created automatically.

Create handlers

```
>>> hh = model.get_header_handler()
>>> mh = model.get_module_handler()
>>> ph = model.get_port_handler()
>>> eh = model.get_edge_handler()
>>> pqh = model.get_physicalquantity_handler()
>>> dh = model.get_definition_handler()
>>> dih = model.get_definition_initialvalue_handler()
>>> aih = model.get_article_info_handler()
>>> cih = model.get_creator_info_handler()
>>> uh = model.get_unit_handler()
>>> bl = model.get_builder()
>>> conv = model.get_converter()
```

File operations

Following commands deal with file operation. *insilico*IDE can export *insilico*ML as three file formats.

Open

Open *insilico*ML model from the file path: sample/HH-pulse-187-164.isml

```
>>> model.load_new_isml_model("sample/HH-pulse-187-164.isml")
```

Save

Dump and Save *insilico*ML model to the file path: sample/test.isml

```
>>> save_file("sample/test.isml", model.dump_document())
```

Export

Export CellML file to the path: sample/test.cellml

```
>>> save_file("sample/test.cellml", conv.isml2cellml(model.dump_document()))
```

Export as TeX file to the path: sample/test.tex

```
>>> conv.isml2tex("sample/test.tex")
```

Export cpp file to the path: sample/test.cpp

```
>>> conv.isml2cpp("sample/test.cpp")
```

Execute python script

insilico terminal is based on Python and so it is possible to execute Python script. Select python script files from the menu [Script]-[Open Script...] (Fig. 1), then the scripts are executed.

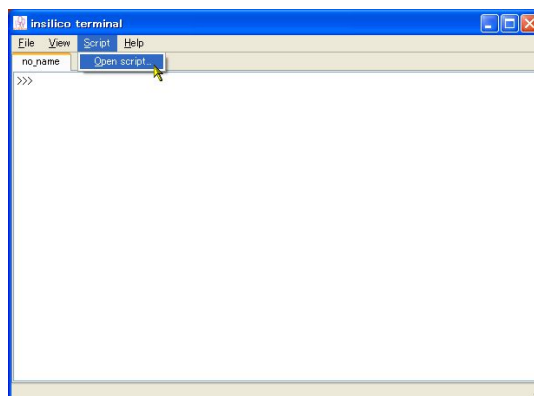


Fig. 1 : Open Python script file.

1. Header and Unit

1.1. Edit model name

Set model name

```
>>> hh.set_name("Bonhoeffer van der Pol model")
```

1.2. Edit discretization of time for numerical integration

Create a new unit "milli_second" and define milli_second

```
>>> uid_ms = uh.create("milli_second")
>>> uh.set_element(uid_ms, "second", "0", "m")
```

Set time step 0.001 (milli second) for numerical integration

```
>>> hh.set_time_discretization(uid_ms, "0.001")
```

1.3. Edit article information

Create new article

```
>>> aid = aih.create()
```

Get PubMed information of the article from TogoWS ※If your machine is not connected to internet, skip this command

```
>>> aih.setup_pubmedinfo_from_togows(aid, "19431309")
```

1.4. Edit creator information

Create new creator

```
>>> cih.create()
```

Set each creator information

```
>>> cih.set_person_name("1", "Yoshiyuki", "", "Asai")
>>> cih.add_affiliation("1", "Osaka University")
>>> cih.set_affiliation_address("1", "Osaka University", "1-1 Suita, Osaka, Japan")
```

2. BVP_excitation module

2.1. Make BVP_excitation module

Create a new module with name “BVP_excitation”

```
>>> m_bvp_x = mh.create("BVP_excitation")
```

2.2. Edit ports

Create input port Iext

```
>>> p_Iext = ph.create(m_bvp_x, "in", "Iext")
>>> ph.set_description(m_bvp_x, p_Iext, "Stimulation from external source")
```

Create input port y_i

```
>>> p_y_i = ph.create(m_bvp_x, "in", "y_i")
>>> ph.set_description(m_bvp_x, p_y_i, "Refractoriness of the ipsilateral side")
```

Create output port x

```
>>> p_x = ph.create(m_bvp_x, "out", "x")
>>> ph.set_description(m_bvp_x, p_x, "Output excitation level of this BVP")
```

2.3 Edit physical-quantity

2.3.1 x

Create physical-quantity “x” and set each property

```
>>> pq_x = pqh.create(m_bvp_x, "x")
>>> pqh.set_type(m_bvp_x, pq_x, "state")
```

Set initial-value definition of x

```
>>> mathml = cstr2mml.cstr2mml('x = 0.2')
>>> dih.set_initialvalue_definition(m_bvp_x, pq_x, '0', mathml, 'ae')
```

Set implementation definition of x

```
>>> mathml = cstr2mml.cstr2mml('diff(x, time) = c * ( ( x - x * x * x/3.0 - y_i ) + Iext ) ')
>>> dh.set_implementation_definition(m_bvp_x, pq_x, '0', mathml, 'ode', '')
```

Bind physical-quantity x to the output port x

```
>>> ph.set_reference(m_bvp_x, p_x, pq_x)
```

2.3.2 c

Create physical-quantity “c” and set each property

```
>>> pq_c = pqh.create(m_bvp_x, "c")
>>> pqh.set_type(m_bvp_x, pq_c, "static-parameter")
```

Set implementation definition of c

```
>>> mathml = cstr2mml.cstr2mml('c = 1.75')
>>> dh.set_implementation_definition(m_bvp_x, pq_c, '0', mathml, 'ae', '')
```

2.3.3 y_i and Iext

Create physical-quantity “ y_i ” and set each property

```
>>> pq_y_i = pqh.create(m_bvp_x, "y_i")
>>> pqh.set_type(m_bvp_x, pq_y_i, "variable-parameter")
```

Set implementation definition of y_i

```
>>> dh.set_implementation_definition(m_bvp_x, pq_y_i, '0', p_y_i, 'assign', 'port')
```

Create physical-quantity “Iext” and set each property

```
>>> pq_Iext = pqh.create(m_bvp_x, "Iext")
>>> pqh.set_type(m_bvp_x, pq_Iext, "variable-parameter")
```

Set implementation definition of Iext

```
>>> dh.set_implementation_definition(m_bvp_x, pq_Iext, '0', p_Iext, 'assign', 'port')
```

3. BVP_refractoriness module

In this section, make BVP_refractoriness module same as section2.

Make BVP_refractoriness module

```
>>> m_bvp_y = mh.create("BVP_refractoriness")
```

Input port x

```
>>> p_x_i = ph.create(m_bvp_y, "in", "x_i")
>>> ph.set_description(m_bvp_y, p_x_i, "Excitation level of the ipsilateral side")
```

Output port y

```
>>> p_y = ph.create(m_bvp_y, "out", "y")
>>> ph.set_description(m_bvp_y, p_y, "Output refractoriness of this BVP")
```

Physical-Quantity a

```
>>> pq_a = pqh.create(m_bvp_y, "a")
>>> pqh.set_type(m_bvp_y, pq_a, "static-parameter")
>>> mathml = cstr2mml.cstr2mml('a = 0.7')
>>> dh.set_implementation_definition(m_bvp_y, pq_a, '0', mathml, 'ae', '')
```

Physical-Quantity b

```
>>> pq_b = pqh.create(m_bvp_y, "b")
>>> pqh.set_type(m_bvp_y, pq_b, "static-parameter")
>>> mathml = cstr2mml.cstr2mml('b = 0.675')
>>> dh.set_implementation_definition(m_bvp_y, pq_b, '0', mathml, 'ae', '')
```

Physical-Quantity c

```
>>> pq_c = pqh.create(m_bvp_y, "c")
>>> pqh.set_type(m_bvp_y, pq_c, "static-parameter")
>>> mathml = cstr2mml.cstr2mml('c = 1.75')
>>> dh.set_implementation_definition(m_bvp_y, pq_c, '0', mathml, 'ae', '')
```

Physical-Quantity x_i

```
>>> pq_x_i = pqh.create(m_bvp_y, "x_i")
>>> pqh.set_type(m_bvp_y, pq_x_i, "variable-parameter")
>>> dh.set_implementation_definition(m_bvp_y, pq_x_i, '0', p_x_i, 'assign', 'port')
```

Physical-Quantity y

```
>>> pq_y = pqh.create(m_bvp_y, "y")
>>> pqh.set_type(m_bvp_y, pq_y, "state")
>>> mathm1 = cstr2mml.cstr2mml('diff(y, time) = ( x_i - b * y + a )/c')
>>> dh.set_implementation_definition(m_bvp_y, pq_y, '0', mathm1, 'ode', '')
>>> mathm1 = cstr2mml.cstr2mml('y = 0.2')
>>> dih.set_initialvalue_definition(m_bvp_y, pq_y, '0', mathm1, 'ae')
>>> ph.set_reference(m_bvp_y, p_y, pq_y)
```

4. current_pulse_generator_membrane module

4.1 Make module and edit ports

Create “current_pulse_generator_membrane” module and output port “current”

```
>>> m_stim = mh.create("current_pulse_generator_membrane")
>>> p_current = ph.create(m_stim, "out", "current")
```

4.2 Edit physical-quantity

4.2.1 initial_current, pulse_height, pulse_onset_time and pulse_width

initial_current

```
>>> pq_initial = pqh.create(m_stim, "initial_current")
>>> pqh.set_type(m_stim, pq_initial, "static-parameter")
>>> pqh.set_description(m_stim, pq_initial, "The initial value of the stimulus current.")
>>> mathml = cstr2mml.cstr2mml('initial_current = 0')
>>> dh.set_implementation_definition(m_stim, pq_initial, '0', mathml, 'ae', '')
```

pulse_height

```
>>> pq_height = pqh.create(m_stim, "pulse_height")
>>> pqh.set_type(m_stim, pq_height, "static-parameter")
>>> pqh.set_description(m_stim, pq_height, "The pulse height.")
>>> mathml = cstr2mml.cstr2mml('pulse_height = 0.7')
>>> dh.set_implementation_definition(m_stim, pq_height, '0', mathml, 'ae', '')
```

pulse_onset_time

```
>>> pq_onset = pqh.create(m_stim, "pulse_onset_time")
>>> pqh.set_type(m_stim, pq_onset, "static-parameter")
>>> pqh.set_description(m_stim, pq_onset, "The time instance when the pulse takes place.")
>>> mathml = cstr2mml.cstr2mml('pulse_onset_time = 20')
>>> dh.set_implementation_definition(m_stim, pq_onset, '0', mathml, 'ae', '')
```

pulse_width

```
>>> pq_width = pqh.create(m_stim, "pulse_width")
>>> pqh.set_type(m_stim, pq_width, "static-parameter")
>>> pqh.set_description(m_stim, pq_width, "The pulse width in milli-second.")
>>> mathml = cstr2mml.cstr2mml('pulse_width = 40')
>>> dh.set_implementation_definition(m_stim, pq_width, '0', mathml, 'ae', '')
```

4.2.2 Set unit

Get unit-id of milli_second and define as uid_ms

```
>>> uid_ms = uh.get_id_with_name('milli_second')
```

Set unit milli_second to *pulse_onset_time* and *pulse_width*

```
>>> pqh.set_unit(m_stim, pq_onset, uid_ms)
>>> pqh.set_unit(m_stim, pq_width, uid_ms)
```

4.2.3 stimulus_current

Create physical-quantity “stimulus_current” and set each property

```
>>> pq_stimulus = pqh.create(m_stim, "stimulus_current")
>>> pqh.set_type(m_stim, pq_stimulus, "variable-parameter")
```

Set implementation definition of stimulus_current

```
>>> csid = dh.set_implementation_definition(m_stim, pq_stimulus, '0', '', 'conditional', '')
>>> cid = dh.set_implementation_case(m_stim, pq_stimulus, csid, '0')
>>> mathml = cstr2mml.cstr2mml('pulse_onset_time <= time && time <= pulse_onset_time + pulse_width')
>>> dh.set_implementation_condition(m_stim, pq_stimulus, cid, mathml)
>>> mathml = cstr2mml.cstr2mml('stimulus_current = initial_current + pulse_height')
>>> dh.set_implementation_definition(m_stim, pq_stimulus, cid, mathml, 'ae', '')
>>> cid2 = dh.set_implementation_case(m_stim, pq_stimulus, csid, cid)
>>> mathml = cstr2mml.cstr2mml('stimulus_current = initial_current')
>>> dh.set_implementation_definition(m_stim, pq_stimulus, cid2, mathml, 'ae', '')
```

Bind physical-quantity to the output port y

```
>>> ph.set_reference(m_stim, p_current, pq_stimulus)
```

5. Link ports and Capsulation

5.1 Link ports

Link output port x (in BVP_excitation module) and input port y (in BVP_refractoriness module)

```
>>> bl.link_ports(m_bvp_x, p_x, m_bvp_y, p_x_i, "functional")
```

Link output port y (in BVP_refractoriness module) and input port x (in BVP_excitation module)

```
>>> bl.link_ports(m_bvp_y, p_y, m_bvp_x, p_y_i, "functional")
```

5.2 Capsulation

5.2.1. Capsulate current_pulse_generator_membrane module

Get module-id list of “current_pulse_generator_membrane”

```
>>> m_ids = mh.get_ids_with_name('current_pulse_generator_membrane')
>>> m_stim = m_ids[0]
```

Capsulate current_pulse_generator_membrane module

```
>>> mc_stim = bl.encapsulate(m_stim)
```

Create port “current” at capsule module of current_pulse_generator_membrane module

```
>>> pc_current = ph.create(mc_stim, 'out', 'current')
```

Link two current ports by forwarding edge

```
>>> bl.link_ports(m_stim, p_current, mc_stim, pc_current, 'forwarding')
```

5.2.2. Capsulate both BVP_excitation and BVP_refractoriness

Get module-ids

```
>>> m_bvp_x = mh.get_ids_with_name('BVP_excitation')[0]
>>> m_bvp_y = mh.get_ids_with_name('BVP_refractoriness')[0]
```

Encapsulate BVP_excitation and BVP_refractoriness

```
>>> mc_bvp = bl.encapsulate([m_bvp_x, m_bvp_y])
```

Create port “Iext” at BVP capsule module

```
>>> pc_Iext = ph.create(mc_bvp, 'in', 'Iext')
```

Link two Iext ports by forwarding edge

```
>>> bl.link_ports(mc_bvp, pc_Iext, m_bvp_x, p_Iext, 'forwarding')
```

Create port “x” at capsule module of BVP_refractoriness module

```
>>> pc_x = ph.create(mc_bvp, 'out', 'x')
```

Link two x ports by forwarding edge

```
>>> bl.link_ports(m_bvp_x, p_x, mc_bvp, pc_x, 'forwarding')
```

5.3. Link capsule modules by functional edge

Link capsule modules by functional edge

```
>>> e_stim = bl.link_ports(mc_stim, pc_current, mc_bvp, pc_Iext, 'functional')
```

7. Couple two BVP models

Load sample/ bvp_pulse.isml

```
>>> model.load_new_isml_model('sample/bvp_pulse.isml')
```

7.1 Create gap_junction module

Make gap_junction module

```
>>> m_gap = mh.create('gap_junction')
```

Physical-Quantity k

```
pq_k = pqh.create(m_gap, "k")
pqh.set_type(m_gap, pq_k, "static-parameter")
mathml = cstr2mml.cstr2mml("k = 0.6")
dh.set_implementation_definition(m_gap, pq_k, "0", mathml, "ae", "")
```

Input port V_1 and physical-quantity V_1

```
>>> p_V_1 = ph.create(m_gap, 'in', 'V_1')
>>> pq_V_1 = pqh.create(m_gap, 'V_1')
>>> pqh.set_type(m_gap, pq_V_1, 'variable-parameter')
>>> dh.set_implementation_definition(m_gap, pq_V_1, '0', p_V_1, 'assign', 'port')
```

Input port V_2 and physical-quantity V_2

```
>>> p_V_2 = ph.create(m_gap, 'in', 'V_2')
>>> pq_V_2 = pqh.create(m_gap, 'V_2')
>>> pqh.set_type(m_gap, pq_V_2, 'variable-parameter')
>>> dh.set_implementation_definition(m_gap, pq_V_2, '0', p_V_2, 'assign', 'port')
```

Output port I_1 and physical-quantity I_1

```
>>> p_I_1 = ph.create(m_gap, 'out', 'I_1')
>>> pq_I_1 = pqh.create(m_gap, 'I_1')
>>> pqh.set_type(m_gap, pq_I_1, 'variable-parameter')
>>> mathml = cstr2mml.cstr2mml('I_1 = k * ( V_2 - V_1 )')
>>> dh.set_implementation_definition(m_gap, pq_I_1, '0', mathml, 'ae', '')
>>> ph.set_reference(m_gap, p_I_1, pq_I_1)
```

Output port I_2 and physical-quantity I_2

```
>>> p_I_2 = ph.create(m_gap, 'out', 'I_2')
>>> pq_I_2 = pqh.create(m_gap, 'I_2')
```

```

>>> pqh.set_type(m_gap, pq_I_2, 'variable-parameter')
>>> mathml = cstr2mml.cstr2mml('I_2 = k * ( V_1 - V_2 )')
>>> dh.set_implementation_definition(m_gap, pq_I_2, '0', mathml, 'ae', '')
>>> ph.set_reference(m_gap, p_I_2, pq_I_2)

```

7.2 Create Isum module

Make Isum module

```

>>> m_sum = mh.create('Isum_current')

```

Input port Iarg_1 and physical-quantity I_1

```

>>> p_Iarg_1 = ph.create(m_sum, 'in', 'I_1')
>>> pq_Iarg_1 = pqh.create(m_sum, 'I_1')
>>> pqh.set_type(m_sum, pq_Iarg_1, 'variable-parameter')
>>> dh.set_implementation_definition(m_sum, pq_Iarg_1, '0', p_Iarg_1, 'assign', 'port')

```

Input port Iarg_2 and physical-quantity Iarg_2

```

>>> p_Iarg_2 = ph.create(m_sum, 'in', 'I_2')
>>> pq_Iarg_2 = pqh.create(m_sum, 'I_2')
>>> pqh.set_type(m_sum, pq_Iarg_2, 'variable-parameter')
>>> dh.set_implementation_definition(m_sum, pq_Iarg_2, '0', p_Iarg_2, 'assign', 'port')

```

Input port Isum and physical-quantity Isum

```

>>> p_sum = ph.create(m_sum, 'out', 'Isum')
>>> pq_sum = pqh.create(m_sum, 'Isum')
>>> pqh.set_type(m_sum, pq_sum, 'variable-parameter')
>>> mathml = cstr2mml.cstr2mml('Isum = I_1 + I_2')
>>> dh.set_implementation_definition(m_sum, pq_sum, '0', mathml, 'ae', '')
>>> ph.set_reference(m_sum, p_sum, pq_sum)

```

7.3 Copy BVP module and link modules by functional edge

Copy BVP module and get the newly-created root module-id. The root module-id is overlap between newly-created module-id list and all root module-id list.

Set each ids

```

>>> mc_bvp = '19586a4d-e98f-48f8-8856-df42535602dd'
>>> pc_Iext = ph.get_ids(mc_bvp, "in")[0]
>>> e_stim = 'e45e8fe0-4997-433c-853c-748c51577b0f'
>>> m_bvp_x = '56f05090-c974-4d26-8f2e-debae430acfc'

```

```
>>> p_x = '3'
>>> mc_stim = '8798a6fa-420c-4d01-a8b5-f6705edd3b6b'
>>> pc_current = '1'
```

Delete functional edge between current and Iext

```
>>> bl.remove_edge(e_stim)
```

Create a new output port and link forwarding edge at BVP model

```
>>> pc_x = ph.create(mc_bvp, 'out')
>>> bl.link_ports(m_bvp_x, p_x, mc_bvp, pc_x, "forwarding")
```

Copy BVP module subtree and get root module-id of newly-created module subtree

```
>>> mc_bvp2 = bl.copy_subtree(mc_bvp, 'root')[0]
```

Get port-id of x and physical-quantity Iext in new BVP module

```
>>> pc_x2 = ph.get_ids(mc_bvp2, 'out')[0]
>>> pc_Iext2 = ph.get_ids(mc_bvp2, 'in')[0]
```

7.4 Link modules

Isum (sum) to port1 (capsule_of_BVP_refractoriness)

```
>>> bl.link_ports(m_sum, p_sum, mc_bvp, pc_Iext, 'functional')
```

port2 (capsule_of_BVP_refractoriness) to V_1 (gap_junction)

```
>>> bl.link_ports(mc_bvp, pc_x, m_gap, p_V_1, 'functional')
```

port2 (capsule_of_BVP_refractoriness__1) to V_2 (gap_junction)

```
>>> bl.link_ports(mc_bvp2, pc_x2, m_gap, p_V_2, 'functional')
```

I_1 (gap_junction) to I_1 (sum)

```
>>> bl.link_ports(m_gap, p_I_1, m_sum, p_Iarg_1, 'functional')
```

I_2 (gap_junction) to port_1 (capsule_of_BVP_refractoriness__1)

```
>>> bl.link_ports(m_gap, p_I_2, mc_bvp2, pc_Iext2, 'functional')
```

port_1 (capsule_of_current_pulse_generator_membrane) to I_2 (sum)

```
>>> bl.link_ports(mc_stim, pc_current, m_sum, p_Iarg_2, 'functional')
```