# DEVS WebViewer tutorial

## Introduction

The DEVS web viewer is a Web application developed in HTML and Javascript. It provides an animated and interactive visualization outputs for DEVS and Cell-DEVS simulators (CD++, CD++ 2.0 and Cadmium). The application can read native Cell-DEVS results for the CD++ and CD++ 2.0 simulators. For other simulators or formalisms, files must be converted to a common specification detailed in this document. In this way, it decouples the visualization from the simulator.

Visualizations rely on SVG diagrams for DEVS models, and a Canvas based representation for Cell-DEVS models. In this document, we will first describe the user interface for the DEVS WebViewer.

the process to convert simulation log files to the common specification. Then we will provide an overview of the common specification format. Finally, we will explain the user interface and the different features of the viewer.

## DEVS WebViewer user interface and features:

The figure shows the user interface presented to the user when the DEVS Web Viewer is accessed. The user can click on the central drop zone to upload the files to convert or view, or they can drag and drop them directly into the drop zone. Users can upload the files to be converted as presented in section 1, or the common specification files as presented in section 2. For the latter, the files will be loaded directly in the application. For the former case, the files will be first converted then loaded in the application. The system automatically determines the simulator and formalism used.

Timeline

Description automatically generated

Figure : DEVS viewer initial display

The following figure shows the files ready to be loaded in the viewer. The file list can be emptied by clicking the “Clear” button or individual files can be removed by clicking the box with the corresponding file below the main input box. To load and visualize the simulation, users click the "Load simulation" button. At this point, the input files will be converted if required, then parsed and loaded. If the format of the provided files is adequate, the viewer will show the diagram if the analyzed model uses the regular DEVS formalism, or a grid if it follows the Cell-DEVS formalism.

Timeline

Description automatically generated

Figure : simulation files ready to be loaded in the viewer

There is a toolbar to the right of the file input box. The first button in the toolbar (cloud icon) allows users to load simulation results from the RISE platform. The RISE platform holds a collection of simulation results to use as demos. Below is a screenshot with some models currently available.

Graphical user interface, text, application

Description automatically generated

Figure : list with Models stored in RISE.

The second button, with a screwdriver and wrench icon, allows users to configure their visualization. This is only accessible once the simulation has been successfully loaded. Playback speed, layout, grid colours and diagram size can be configured. The figure below shows, on the left-hand side (a), the base configuration interface and, on the right-hand side (b), the grid configuration interface for Cell-DEVS models. The grid configuration allows users to specify which layers and which ports to show in the visualization as well as the colours used to draw cells.

Graphical user interface, application

Description automatically generated Graphical user interface

Description automatically generated with medium confidence

Figure : (a) Simulation settings. (b) Grid and style options.

The third button, with a download arrow icon, allows users to download the files in the common specification presented in section 2. This is only accessible once the simulation has been successfully loaded.

Finally, there is also a playback bar that allows users to navigate through the simulation time steps. The playback bar is located below the main simulation visualization. Users can move forward or backwards a single frame, animate the simulation backwards or forwards or jump to the end or the beginning of the simulation. Users can also use the slider to move through the time steps of the simulation.

To the right of the bar, a record button allows users to record their simulation as a *.webm* video. To do so the user must click the record button then use the playback options to animate the simulation (all navigation options can be used) and finally, click the record button again to stop and download the video.



Figure : The simulation playback bar.

## Reading simulation outputs

### Native CD++ and CD++ 2.0 results for Cell-DEVS

The viewer can process Cell-DEVS results directly from the CD++ or CD++ 2.0 simulators. The simulation files and results can be provided as output by the simulators. This section describes the files that must be provided to follow this path. Only the initial values (.val) file is optional.

#### Simulation results (.log)

This file contains the messages that are output as the simulation is executed. The name of the file does not matter but it must have a .log extension.

For CD++ the outputs are stored in a log file. This file always has the *.log* extension for CD++ but sometimes the *.log01* extension for CD++2.0 (Cell-DEVS),. In the case of Cadmium, there can be multiple log files, so the user has to choose the log file that contains the simulation message, specified with the “logger\_messages” option in the main file where the top model is declared.

For CD++, the .log file is formatted as follows:

Mensaje I / 00:00:00:000 / Root(00) para top(01)

Mensaje I / 00:00:00:000 / top(01) para lug(02)

Mensaje I / 00:00:00:000 / lug(02) para lug(0,0,0)(03)

...

Figure : A partial .log file for a CD++ Cell-DEVS simulation.

For CD++ 2.0, the .log file is formatted as follows:

0 / L / I / 00:00:00:000:0 / top(2168) / computer\_lab(01)

0 / L / D / 00:00:00:000:0 / computer\_lab(0,0)(02) / 00:00:00:000:0 / computer\_lab(01)

0 / L / D / 00:00:00:000:0 / computer\_lab(0,1)(03) / 00:00:00:000:0 / computer\_lab(01)

...

Figure : A partial .log file for a CD++ 2.0 Cell-DEVS simulation.

#### Model make file (.ma)

This file contains the model structure. It identifies atomic and coupled models, their components, links, ports, and other information related to the model. The name of the file does not matter but it must have a .ma extension.

The *.ma* file should contain the following elements. Some are optional:

* Models (required): each model is identified by a name within square brackets. In the example below, 2 models are identified (top and sender).
* Components (optional): each key-value pair is a submodel. A model with components is a coupled model. In the example below, the top model is coupled.
* Ports (optional): Ports are identified by the “neighborports” key. The value for this key-value pair is a list of named ports for the model. These will become available as layers in the visualization. The example below has no ports defined, only the “out” port is available.
* Dimensions (required): Dimensions are usually specified through the “dim” key but a combination of “width and “height” is also supported. These are the dimensions of the cell-space. In the example below, the dimensions are 100, 100, 2
* Initial Value (optional): The initial value used for the cell-space. These can be specified through the “initialvalue” or “initialrowvalue” keys. The *.val* file often supersedes these elements. In the example below, the initial global value is 0.

For CD++ and CD++ 2.0, the .ma file is formatted very similarly, as follows:

#include (macros.inc)

[top]

components : LUG

[LUG]

type : cell

dim : (100, 100, 2)

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : LUG(-1,-1,0) LUG(-1,0,0) LUG(-1,1,0)

neighbors : LUG(0,-1,0) LUG(0,0,0) LUG(0,0,1) LUG(0,1,0)

neighbors : LUG(1,-1,0) LUG(1,0,0) LUG(1,1,0)

initialvalue : 0

initialcellsvalue : map.val

localtransition : LUG-rule

[LUG-rule]

rule : {(0,0,0) + 2} 100 { #macro(NonUrban) and #macro(Logistic) }

rule : {(0,0,0) - 1} 100 { #macro(Growing) }

rule : {(0,0,0) + 1} 100 { #macro(Urban) and (#macro(UrbanNeighborCount) < 9) }

rule : {(0,0,0)} 100 { t }

Figure : A partial .ma file for the Logistic Urban Growth, a CD++ Cell-DEVS model.

#### Initial values file (.val)

For CD++, Cell-DEVS models will often rely on an initial values file. When that is the case, the *.val* file must be provided. CD++2.0 (Cell-DEVS), does not require the initial values file since they appear in the log file at the first-time step.

This file contains the initial values for the cell-space used when simulating the model. It is an optional file where each row identifies the cell coordinates and values used as an initial value for the cell. The name of the file does not matter but it must have a .val extension.

For CD++ and CD++ 2.0, the .val file is formatted very similarly, as follows:

(10, 8) = 500 -700 -1

(13, 8) = 500 -700 -5

(17, 8) = 500 -700 -7

(20, 8) = 500 -700 -9

(24, 8) = 500 -700 -10

(27, 8) = 500 -700 -11

(10, 13) = 500 -700 -1

(13, 13) = 500 -700 0

(17, 13) = 500 -700 5

(20, 13) = 500 -700 2

(24, 13) = 500 -700 1

(27, 13) = 500 -700 0

...

Figure : A partial .val file for a CD++ or CD++ 2.0 Cell-DEVS simulation.

#### Color palette file (.pal)

This file contains the color palette to use to draw the cell-space. Each line describes the color that a cell will be drawn according to its state value. There are two formats accepted. Note that in both cases, the minimum value for a range is inclusive and the maximum value is exclusive.

The first accepted format is:

[99.5;100.5] 255 255 255

[100.5;101.5] 0 204 204

[101.5;102.5] 0 153 153

[9.5;10.5] 128 128 128

[10.5;11.5] 96 96 96

[11.5;12.5] 64 64 64

[19.5;20.5] 0 0 255

[20.5;21.5] 0 0 196

[21.5;22.5] 0 0 132

[29.5;30.5] 255 0 0

[30.5;31.5] 196 0 0

[31.5;32.5] 132 0 0

[39.5;40.5] 0 128 0

...

Figure : A partial .pal file in the first accepted format.

The second accepted format is:

VALIDSAVEDFILE

255,255,0

255,0,255

133,127,56

...

0.0,3.0

3.0,5.0

5.0,10.0

...

Figure : A partial .pal file in the second accepted format.

#### Visualization file (visualization.json)

The optional visualization file contains a complete set of configurations for rendering a simulation output. It contains playback, layout, and style parameters. It can be provided instead of the palette (.pal) file described previously.

The easiest way to generate this file is to first load a simulation by providing the native CD++ or CD++ 2.0 files then, configuration the visualization through the WebViewer. Once this is done, you can download the visualization file by clicking the 4th button from the top in toolbar to the right of the cell-space.

Visualization files vary according to the formalism represented. For Cell-DEVS models an example can be found in Appendix 1.

## Step by step example

### Cell-DEVS visualization – Logistic urban growth model

A picture containing graphical user interface

Description automatically generated

Figure : visualization for the CO2 concentration model

1. Drag the simulation log (LUG.log), make (LUG.ma), initial values (map.val) and palette (map.pal) files and drop them onto the dashed blue square.

\* Note that the .log, .ma and .pal files are mandatory

1. Load the simulation by pressing the “Load simulation” button.
2. The cell-space should be displayed. Review the visualization settings to change the number of columns and size of the grids, and the style for each port and grid.
3. (Optional) Download the visualization files with the 4th button from the top in toolbar to the right of the cell-space.

## Appendix 1:

{

"playback": {

"speed": 10,

"loop": false,

"cache": 10

},

"grid": {

"columns": 2,

"width": 350,

"height": 350,

"spacing": 10,

"showGrid": false,

"aspect": true,

"layers": [{

"z": 0,

"ports": ["out"],

"style": 0,

"position": 0

}, {

"z": 1,

"ports": ["out"],

"style": 0,

"position": 1

}

],

"styles": [{

"buckets": [{

"start": 99.5,

"end": 100.5,

"color": [255, 255, 255]

}, {

"start": 100.5,

"end": 101.5,

"color": [0, 204, 204]

},

...

{

"start": 0.8,

"end": 0.9,

"color": [122, 17, 26]

}, {

"start": 0.9,

"end": 9,

"color": [128, 0, 0]

}

]

}

]

}

}

## Appendix 2: Overview of the common specification:

The DEVS WebViewer requires that simulation results be provided in a specifically designed format. This format is obtained by converting simulation results using the process described in the previous section. The section describes this format, a summary data model is provided below:

### 

Figure 12: An overview of the common specification data model

The data model presented above contains a sub-structure to hold structural elements, message emitters and links, and another sub-structure to hold messages organized by time frames. In the implementation we have adopted, a *json* file is used to represent the former and a csv like format is used for the latter. Both are explained in this section.

### structure.json

The *structure.json* file contains all information related to the structural elements of a DEVS or Cell-DEVS model (atomic and coupled models, ports, and links). The *messages.log* file contains all the messages output by the simulation. The figure below provides an example for the alternate bit protocol model (ABP):

|  |  |
| --- | --- |
| {  "name": "Alternate Bit Protocol",  "simulator": "CDpp",  "type": "DEVS",  "nodes": [{  "name": "sender",  "type": "atomic",  "svg": ["#m-01"]  }, {  "name": "receiver",  "type": "atomic",  "svg": ["#m-02"]  },  ...  ],  "ports": [{  "model": "network",  "name": "in1",  "type": "input",  "svg": ["#p-05"]  }, {  "model": "network",  "name": "out1",  "type": "output",  "svg": ["#p-06"]  },  ...  ],  "links": [{  "modelA": "network",  "portA": "out1",  "modelB": "receiver",  "portB": "in",  "svg": ["#l-06"]  }, {  "modelA": "network",  "portA": "out2",  "modelB": "sender",  "portB": "ackin",  "svg": ["#l-04"]  },  ...  ]  } | ...  00:00:20:000;7,11;9,1  00:00:22:987;12,11  00:00:32:987;5,1  00:00:50:000;7,11;9,1  00:00:51:957;12,11  00:01:01:957;5,1  00:01:04:992;14,1;10,1  00:01:14:992;7,20;9,2  00:01:17:174;12,20  00:01:27:174;5,0  00:01:44:992;7,20;9,2  00:01:48:841;12,20  00:01:58:841;5,0  00:02:02:496;14,0;10,0  00:02:12:496;7,31;9,3  00:02:15:942;12,31  00:02:25:942;5,1  ... |

Figure 13 example of a JSON structure file for the ABP   
model (left) and the corresponding messages file (right).

The *structure.json* file contains the following elements:

* name: The name of the simulation model
* simulator: The name of the simulator used
* type: The type of simulation model (DEVS or CellDEVS).
* nodes: an array of nodes representing the atomic and coupled models that compose the simulation model. Each model (atomic or coupled) contains the following elements:
  + name: the name of the model
  + type: the type of the model (atomic or coupled)
  + svg: the svg elements in the diagram that correspond to the model
  + size: (Cell-DEVS models only) an array of integer representing the dimensions of the cell-space for the model
* ports: an array containing ports that compose the simulation model. Each port contains the following elements:
  + model: the name of the model associated to the port
  + name: the name of the port
  + type: the type of the port (input or output)
  + svg: the svg elements in the diagram that correspond to the model
* links: an array containing links that relate different elements of the simulation model. Each link contains the following elements:
  + modelA: the name of the origin model for the link
  + portA: the name of the origin port for the link
  + modelB: the name for the destination model for the link
  + portB: the name of the destination port for the link
  + svg: the svg elements in the diagram that correspond to the model

### messages.log:

All messages output by a simulation are contained in the messages.log file. Each line of the file contains all messages emitted for a given time step. Messages are constructed as specified below:

Logo

Description automatically generated

Messages differ slightly for Cell-DEVS models. For each state or message data, the first three values will correspond to the X, Y and Z coordinates of the cell. The next value will correspond to the port index in the structure and the remaining values will correspond to the message emitted.

### diagram.svg

A scalable vector graphics (SVG) file is required to visualize DEVS models. The SVG file must show a diagram with the structure of the model that is going to be animated. It can be drawn with any SVG editing tool, such as Inkscape or diagrams.net. The visualizer gives a high degree of freedom in the design of the diagram, so the user can add any extra images or text that clarifies the content of the diagram. Here are recommended elements that should be in the diagram:

* Shapes for the atomic and coupled models included (can be rectangles, circles, or more complex shapes).
* A label with the name of each atomic or coupled model (can be inside or outside the model).
* Links composed by lines and arrow markers to show the direction of the link.
* Labels with the names of the input and output port of the models (the user could, eventually, obviate some of the ports that are not necessary for visualization, and still reproduce the visualization).

Each element in the diagram should have a unique ID that corresponds to an element in the *structure.json* file. The DEVS WebViewer requires this to highlight the active components in every time step.