# Cadmium extension for the DEVS WebViewer

## DEVS models visualization

### Include the web extension in the makefile

1. Add the following include variable to the include list at the top:



1. Add the include path to all your build commands, see example below:



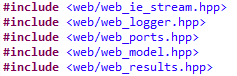
### Use the web extension namespace in your coupled model file

1. Remove the **cadmium** #include statements. Replace them by the **web** include statements
   1. Remove any **cadmium** namespace include statements:

Text

Description automatically generated

* 1. Add the following include statements for the **web** namespace:



\* **web\_ie\_stream** is only required if the model reads input data from a text file.

* 1. Remove any using statements that alias cadmium namespaces namespaces. For example:



1. For every **coupled**, **atomic**, **test** and **data** **structure** files, replace any struct, class or from the **cadmium** or **dynamic** namespaces by the same struct, class or function from the **web** namespace:
   1. Any **in\_port** or **out\_port** should be replaced by **web::in\_port** or **web::out\_port**
   2. Any **iestream\_input** should be replaced by **web::iestream\_input**
   3. Any **iestream\_input\_defs** should be replaced by **web:: iestream\_input\_defs**
   4. Any calls to the **make\_dynamic\_atomic\_model** or **make\_dynamic\_coupled\_model** functions should be replaced by a call to **web::make\_atomic\_model** or **web::make\_coupled\_model**. Both functions require an additional parameter in the second position, a string representing the type of the model. For example:

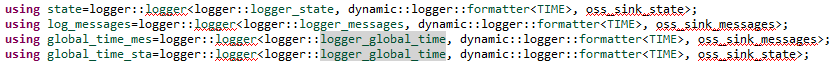


\* Multiple instances of an atomic model can share the same model type.

* 1. Any uses of the **atomic** or **coupled** classes should be replaced by **web::atomic** or **web::coupled** respectively.When building shared pointers of these models using **make\_shared**, an additional parameter in the second position should be provided in the same manner as the previous step.
  2. Any references to **model**, **Models**, **Ports**, **EICs**, **EOCs**, or **ICs** should be replaced by **web::model**, **web::Models**, **web::Ports**, **web::EICs**, **web::EOCs** or **web::ICs** respectively.
  3. Any calls to **make\_EIC**, **make\_EOC** or **make\_IC** should be replaced by **web::make\_EIC**, **web::make\_EOC** or **web::make\_IC** respectively.
  4. The Top model should be instantiated as a **Top** model rather than a **Coupled** model. This can be achieved by calling the **web::make\_top\_model** function. The arguments to this function are the same as the **web::make\_coupled\_model**.



1. Remove the cadmium the logger
   1. Remove the **oss\_sink\_messages** and **oss\_sink\_state** struct definitions and the corresponding **ofstream** definitions
   2. Remove the existing **logger** objects:



* 1. Remove the **multilogger** object



1. Modify the runner instantiation
   1. Replace the reference to the **runner** class by **web::runner**. The second parameter should be replaced by **web::logger\_top**
2. Call the web output function
   1. Add this line at after running the simulation.



### Add missing functions on atomic models and messages

1. Modify the state output function for each atomic model
   1. Remove any include statement that references the **cadmium** or **dynamic** namespaces. Add the following include statements:



* 1. On each atomic model, make sure the state output operator formats the messages as comma separated values. The function should follow this pattern:

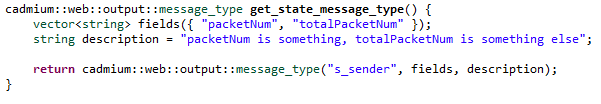
Text

Description automatically generated with medium confidence

* 1. Implement a **get\_state\_message\_type** function on your atomic model. This function must output a **web::message\_type object**.The constructor for a **message\_type** receives 3 arguments:

1. A unique identifier string that is unique among all message types.
2. A vector of strings that represent the names of the output fields.
3. A string containing a short description for the message type object.

The function should follow this pattern:



\* Note: It’s important that the length of the fields vector matches the number of values output by the operator above.

1. Modify the output function for each message data structure:
   1. Remove any include statement that references the **cadmium** or **dynamic** namespaces. Add the following include statement:

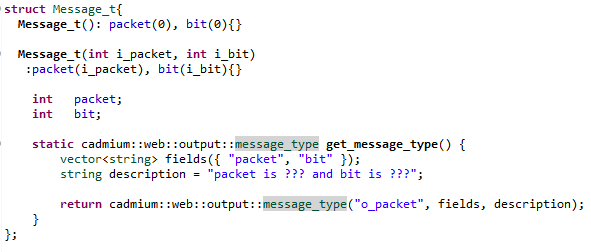


* 1. On each message data structure, make sure the output operator formats the messages as comma separated values. The function should follow this pattern:

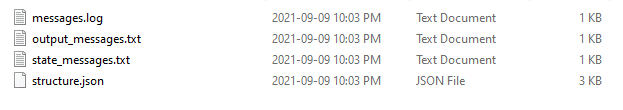
Text

Description automatically generated with medium confidence

* 1. Implement a static **get\_message\_type** function on the struct representing your message object. This function must output a **web::message\_type** object, constructed in the same manner as specified previously. The struct should follow this pattern:



1. At this point, the model should be ready to simulate. Once simulated, the following files should be in the simulation\_results folder:



### Prepare a diagram for your model

1. Go to <https://app.diagrams.net/>
2. In the menu, go to **Extras** > **Configuration…**
3. Copy paste the following in the Configuration text area

{"simpleLabels":true}

1. Click **Apply** and refresh the page.
2. Draw your diagram, including atomic models, coupled models, couplings and ports. Keep in mind that overlapping may cause interaction issues in the viewer. For labels, the underlying rectangle should be as tightly fit as possible on the text.

Here is a sample diagram for the alternate bit protocol model:

Diagram

Description automatically generated

1. Export the diagram to the SVG format. In the menu, go to **File > Export as > SVG …** then, click the **export** button. Name the file **diagram.svg**.

### Visualize the model

1. Go to <http://206.12.94.204:8080/arslab-web/1.6/app-simple/index.html>. You will see the following UI:

Graphical user interface, text, application

Description automatically generated

1. Go to the **/simulation\_results** folder. Drag and drop the **structure.json** and **messages.log** files on the dashed box.
2. Drag and drop your SVG diagram file (**diagram.svg**) on the dashed box.
3. Click the **Load simulation** button.
4. You should now see your diagram. The following is an example for the alternate bit protocol model:

Diagram

Description automatically generated

1. Click the **Link** button from the menu on the right. A popup window will appear. The popup shows a list of model components on the left and your diagram on the right:

A picture containing diagram

Description automatically generated

1. Associate each component on the left to its corresponding SVG element on the right. This is accomplished by first clicking on a box on the left then any number of diagram elements on the right. Elements on the right are highlighted in red once they are properly associated.

A picture containing graphical user interface

Description automatically generated

1. Once finished, close the popup and hit the **play** button from the playback bar at the bottom to animate your model.
2. Use the record button to save a video of your model animation. To do this, first click on the record button, then the play button, then the record button again to stop the recording and download the video.
3. Download your modified SVG file by clicking on the **download** button

## Cell-DEVS models visualization

### Include the web extension in the makefile

1. Add the following include variable to the include list at the top:



1. Add the include path to all your build commands, see example below:



### Use the web extension namespace in your coupled model file

1. Include the following include statement for the **web** extension:



1. Modify the state output operator of your model so that it outputs in a comma separated value format. The following is an example:

A picture containing text

Description automatically generated

1. Define a vector of strings representing the output fields for the state of your model. This vector must contain the same number of fields as your state output operator outputs.



1. Call the conversion function. This must be done after running the simulation. The function requires 4 parameters: the vector defined above, a path to the scenario file, the path to the results file, the path to the output folder.:



1. Once simulated, the following files should be in the output folder:

Text

Description automatically generated

### Visualize the model

1. Go to <http://206.12.94.204:8080/arslab-web/1.6/app-simple/index.html>. You will see the following UI:

Graphical user interface, text, application

Description automatically generated

1. Go to the **/simulation\_results** folder. Drag and drop the **structure.json,** **messages.log** and **style.json** files on the dashed box.
2. Click the **Load simulation** button.
3. At this point, you should see one purple or grey square per field that your model outputs. This is due to the default **style.json** file not being tailored to your model.
4. You can modify the styles by clicking on the palette button. This will open up the palette interface:

Graphical user interface

Description automatically generated with low confidence

1. From this interface you can modify the visualization for your simulation.
   1. The top part of the UI allows you to add or remove layers in your simulation and associate a style to each of them.
   2. The bottom part allows you to define a color palette for the style selected in the dropdown list.
2. Once your style is defined, you can download it by clicking on download button. This allows you to reuse it subsequently.