Flowsheet Visualizer Tutorial

> Author: Dan Gunter \ Created: 2022-07-05

Outline

- Introduction
- Example flowsheet
- Running the Flowsheet Visualizer
- Running from a script
- Further reading

Introduction

The IDAES Flowsheet Visualizer (FV) is a Python tool that provides a web-based visualization of any existing IDAES model or flowsheet. The visualization shows a diagram of the flowsheet as well as a stream table. You can interact with the diagram and export it as an image for inclusion in presentations or publications.

This tutorial will show the basic steps of running the FV on an example flowsheet, interacting with the resulting GUI, saving your work, and exporting the diagram as an image. It will also show how the Visualizer can be updated to reflect changes in the model components and/or variable values. The tutorial will also show how to run the Visualizer from a Python script.

Example flowsheet

This initial section creates an example flowsheet.

Setup

Module imports and any additional housekeeping needed to initialize the code.

In [ ]:
```
import visualizer_tutorial as vistut
vistut.quiet()   # turn off default logging and most warnings
from idaes.core.util.model_statistics import degrees_of_freedom
from IPython.display import Markdown
```

Create the flowsheet

In [ ]:
```
# use the pre-defined function to create the flowsheet
model = vistut.create_model()

# description of the flowsheet we created
display(Markdown(vistut.function_markdown(vistut.create_model)))
```

```
vistut.quiet()

# initialize the flowsheet as a square problem (dof=0)
vistut.initialize_model(model)

# verify that there are zero degrees of freedom
print(f"DOF = {degrees_of_freedom(model)}")
```

Running the Flowsheet Visualizer

In most cases, you will run the FV by calling the `visualize()` method attached to your flowsheet. This function takes a number of optional arguments, which we will look at briefly later, and one required argument: the **title** to give the visualization. Unless you give more information, this title also is used as the filename in which to save its current state.

In the following, we start the FV with the title "Hydrodealkylation". This will pop up a new browser tab (and save the status in a file called *Hydrodealkylation.json*).

After the visualizer starts, we recommend making its tab into its own browser window and viewing it side-by-side with this notebook window.

In [ ]:
```
model.fs.visualize("Hydrodealkylation")
```

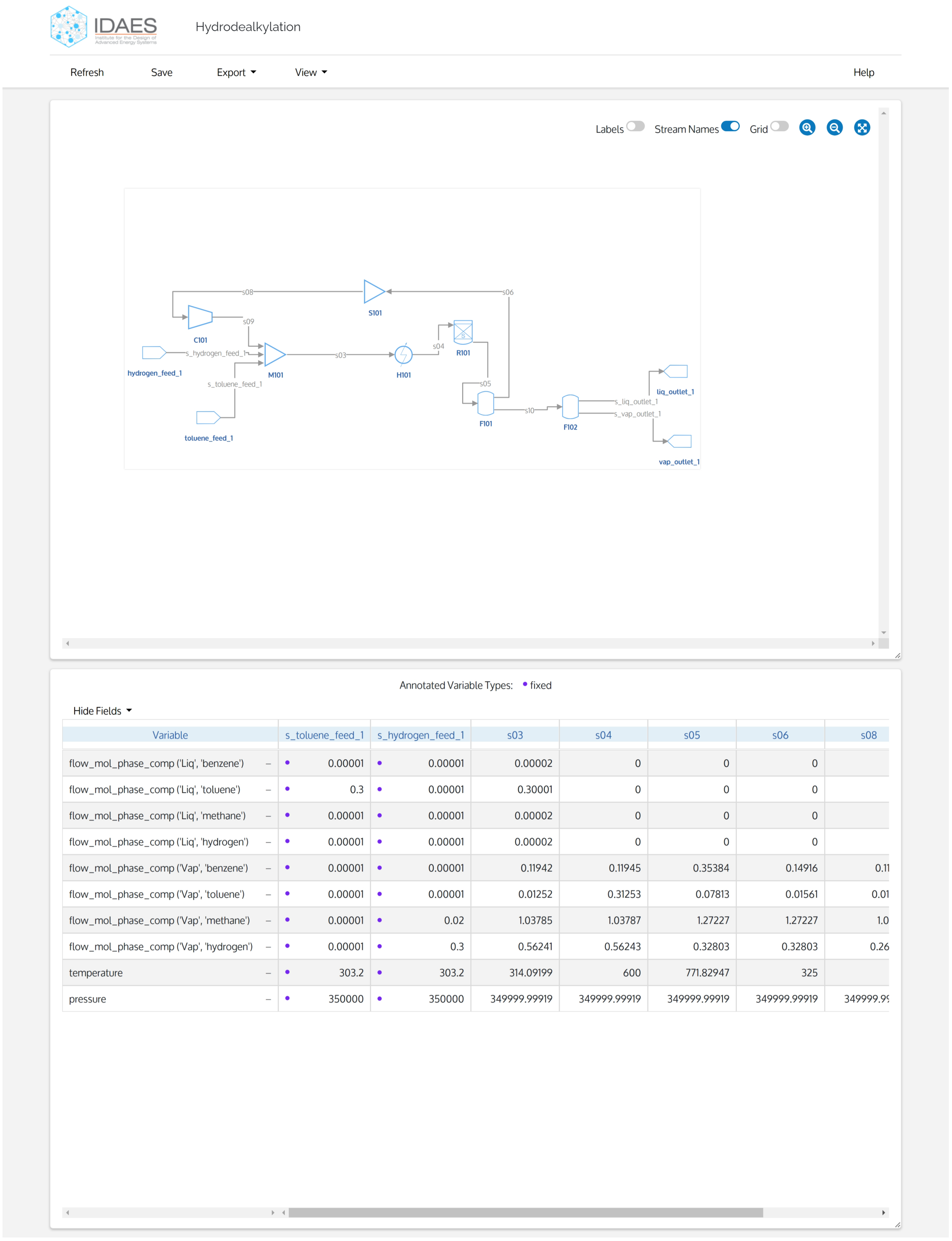
Optional arguments

The optional (keyword) arguments are documented in the base function, which can be found in `idaes.core.ui.fsvis.visualize`:

- name: Name of flowsheet to display as the title of the visualization
- load_from_saved: If True load from saved file if any. Otherwise create a new file or overwrite it (depending on 'overwrite' flag).
- save: Where to save the current flowsheet layout and values. If this argument is not specified, "`name`.json" will be used (if this file already exists, a "-`<version>`" number will be added between the name and the extension). If the value given is the boolean 'False', then nothing will be saved. The boolean 'True' value is treated the same as unspecified.
- save_dir: If this argument is given, and `save` is not given or a relative path, then it will be used as the directory to save the default or given file. The current working directory is the default. If `save` is given and an absolute path, this argument is ignored.
- save_time_interval: The time interval that the UI application checks if any changes has occurred in the graph for it to save the model. Default is 5 seconds
- overwrite: If True, and the file given by `save` exists, overwrite instead of creating a new numbered file.
- browser: If true, open a browser
- port: Start listening on this port. If not given, find an open port.
- log_level: An IDAES logging level, which is a superset of the built-in `logging` module levels. See the `idaes.logger` module for details
- quiet: If True, suppress printing any messages to standard output (console)
- loop_forever: If True, don't return but instead loop until a Control-C is received. Useful when invoking this function at the end of a script.

Interacting with the visualizer

The first things you need to learn about the FV are how to manipulate the overall layout and control the view. The UI should initially look something like the screenshot below:



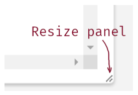
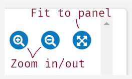
Annotated Variable Types: • fixed

Variable	s_toluene_feed_1	s_hydrogen_feed_1	s03	s04	s05	s06	s08
flow_mol_phase_comp ('Liq', 'benzene') –	• 0.00001	• 0.00001	0.00002	0	0	0	
flow_mol_phase_comp ('Liq', 'toluene') –	• 0.3	• 0.00001	0.30001	0	0	0	
flow_mol_phase_comp ('Liq', 'methane') –	• 0.00001	• 0.00001	0.00002	0	0	0	
flow_mol_phase_comp ('Liq', 'hydrogen') –	• 0.00001	• 0.00001	0.00002	0	0	0	
flow_mol_phase_comp ('Vap', 'benzene') –	• 0.00001	• 0.00001	0.11942	0.11945	0.35384	0.14916	0.11
flow_mol_phase_comp ('Vap', 'toluene') –	• 0.00001	• 0.00001	0.01252	0.31253	0.07813	0.01561	0.01
flow_mol_phase_comp ('Vap', 'methane') –	• 0.00001	• 0.02	1.03785	1.03787	1.27227	1.27227	1.0
flow_mol_phase_comp ('Vap', 'hydrogen') –	• 0.00001	• 0.3	0.56241	0.56243	0.32803	0.32803	0.26
temperature –	• 303.2	• 303.2	314.09199	600	771.82947	325	
pressure –	• 350000	• 350000	349999.99919	349999.99919	349999.99919	349999.99919	349999.99

As you can see, the FV has two main panels. We will call the top panel the **diagram** and the bottom panel the **stream table**.

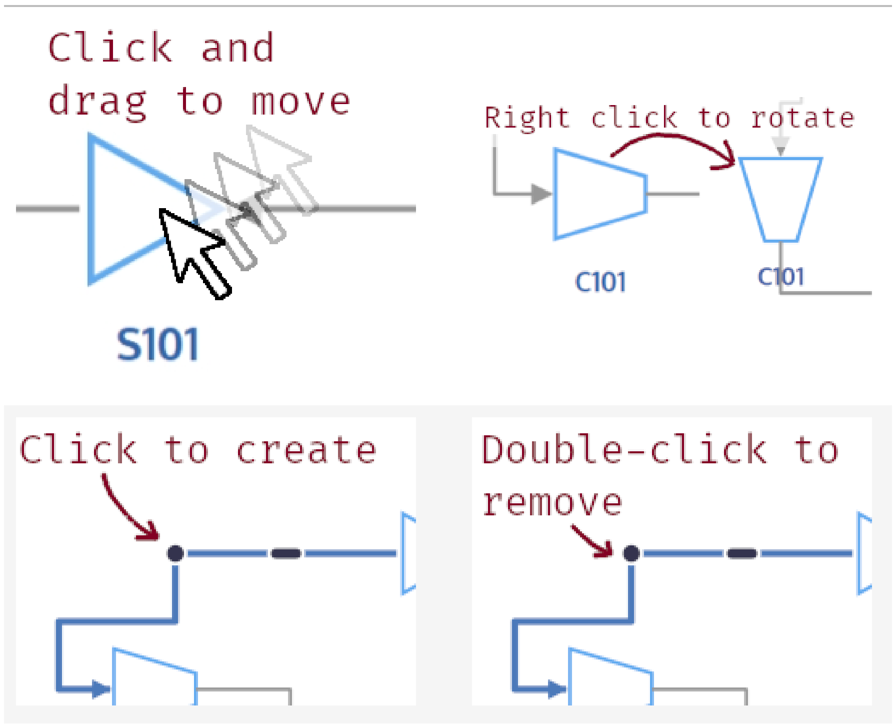
View controls

Before looking at the two panels in detail, it helps to know some basic controls for making them easier to view.

Control	Description	Illustration
Panel height	Change the height of the panels by grabbing the small handle in the lower right corner with your mouse.	
Diagram size	Zoom in/out on the diagram with the magnifying glass "+" and "-" buttons in the upper-right corner of the top panel. The button labeled with two crossing arrows fits the diagram into the current panel height and width.	

Rearranging the diagram

The diagram shown in the top panel is interactive. You can move the units shown there into different positions. Whatever arrangement you end up with will be saved for next time. The arcs (i.e., lines representing streams) connecting the units will automatically re-route themselves as you move them. Below is a summary of the different actions you can take when rearranging the diagram.



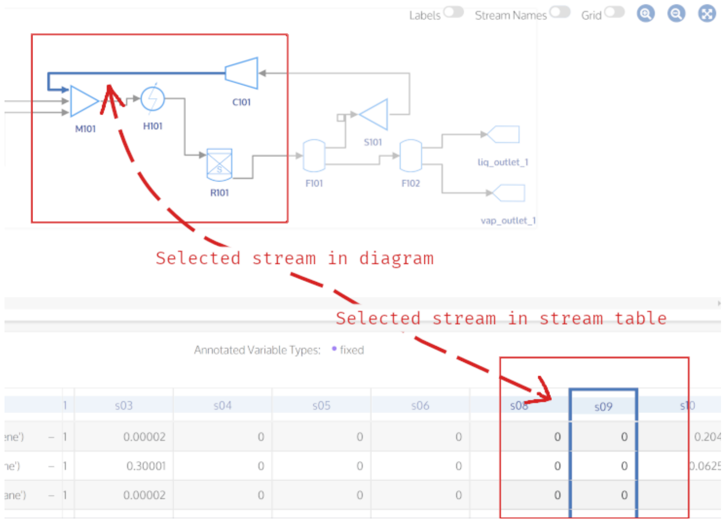
Stream table

The stream table panel shows the values of variables on all the streams between units, and also from units to outlets.

Stream table "brushing"

Brushing refers to the ability to have actions in one visual area influence the display in another. It is commonly used in statistics to show how points in one scatterplot correspond to their points in another, for the same samples. Here, we use it to link the position of a stream in the diagram with its variable values in the stream table.

Controls

Action	Result	Illustration
moving the mouse over an **arc** in the diagram	highlights the corresponding **column** in the stream table	
moving the mouse over a **column** in the stream table	highlights the corresponding **arc** in the diagram	

Labels
Stream Names
Grid
M101
H101
C101
R101
F101
S101
F102
liq_outlet_1
vap_outlet_1
Selected stream in diagram
Selected stream in stream table
Annotated Variable Types:
fixed

		1	s03	s04	s05	s06	s08	s09	s10
ene')	–	1	0.00002	0	0	0	0	0	0.204
ne')	–	1	0.30001	0	0	0	0	0	0.0625
ane')	–	1	0.00002	0	0	0	0	0	

Example

Stream table brushing is useful for answering questions like:

> How much benzene are we losing in the F101 vapor outlet stream?

To answer this question, we will use some interactive elements of the stream table.

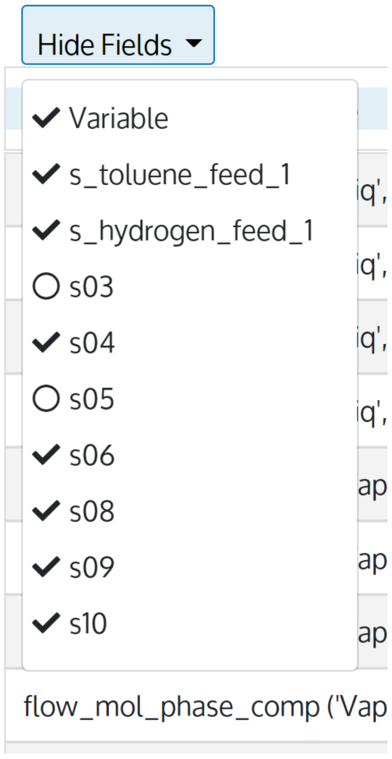
1. Find the inlet of F101 on the diagram. Mouse over this to see the values for that stream highlighted in the stream table below. This is stream `s05`. Look across at the row for Benzene vapor (`flow_mol_phase_comp('Vap', 'benzene')`) and see that the value is 0.35384
2. Find the vapor outlet of F101 by looking for the arc connecting to the splitter and compressor feedback loop. This is stream `s06`. Then look at the same row for the Benzene vapor mol fraction and see that the value is 0.14916
3. Thus the amount of benzene lost is (in mole fractions) about 0.2

Showing and hiding streams

For complex diagrams, there are a lot of streams and the stream table does not fit in the window. To avoid having to scroll back and forth, there is the ability to "hide" selected streams in the stream table.

- Click on the "Hide Fields" menu and select which fields to hide

- The mark will toggle between a check (shown) and open circle (hidden)



For example, we can hide all the streams except the feeds and the flash inlets and outlets.

Saving and loading

The current layout and status can be saved to a file, and this file can then be loaded when the model is viewed again. The main benefit is that interactive layout of the diagram is saved for re-use.

File name

This file is named, by default, for the title of the visualizer (e.g., "Hydrodealkylation") with a ".json" extension to indicate the data format and saved in the same directory as the Jupyter notebook.

You can select a different filename and location when you start the visualization, e.g.

```
model.fs.visualize("The Title", save="thefilename.json",
save_dir="/path/to/save/the/file")
```

Reloading

To reload the saved layout, simply choose the same title (since the filename, by default, matches the title) or explicitly use the `save` and `save_dir` keywords for the `visualize()` function to select a previously saved file. This means you only need to manually lay out the diagram once. Of course, if you add new pieces to the flowsheet you will need to position them correctly (as discussed below).

Exporting

Exporting the diagram as an image

You can export an image of the flowsheet diagram in the Scalable Vector Graphics (SVG) format, which can render without fuzziness at arbitrary sizes. Almost all presentation and drawing programs, including MS

Word and Powerpoint, can use SVG images.

From the top menu select *Export -> Flowsheet*. You will get a preview of the flowsheet that you can then download to a file.

Exporting the stream table as CSV

You can export the stream table as comma-separated values. From the top menu select *Export -> Stream Table*.

Updating when the flowsheet changes

The FV has a connection to the Python program that has the flowsheet (model) in memory. Therefore, when the underlying flowsheet changes, the visualization can be quickly updated to show the new state. This feature is particularly useful for interactive flowsheet creation and debugging in Jupyter Notebooks.

To illustrate the feature, below is some IDAES modeling code that adds another Flash unit to the model, connecting the liquid outlet of the first flash unit to its inlet. There is a little more code that updates some of the output values of the model and sets initial values for this new unit, and then re-initializes the model.

After this code executes, the model will have a unit called "F102" connected to "F101".

In [ ]:
```
# Add a second flash unit
from idaes.generic_models.unit_models import Flash
from pyomo.network import Arc
from pyomo.environ import Expression, TransformationFactory
m = model # alias
m.fs.F102 = Flash(default={"property_package": m.fs.thermo_params,
                           "has_heat_transfer": True,
                           "has_pressure_change": True})
# connect to 1st flash unit
m.fs.s10 = Arc(source=m.fs.F101.liq_outlet, destination=m.fs.F102.inlet)
# update expressions for purity and cost
m.fs.purity = Expression(
        expr=m.fs.F102.vap_outlet.flow_mol_phase_comp[0, "Vap", "benzene"] /
             (m.fs.F102.vap_outlet.flow_mol_phase_comp[0, "Vap", "benzene"]
              + m.fs.F102.vap_outlet.flow_mol_phase_comp[0, "Vap", "toluene"]))
m.fs.heating_cost = Expression(expr=2.2e-7 * m.fs.H101.heat_duty[0] +
                                1.9e-7 * m.fs.F102.heat_duty[0])
# fix unit output and pressure drop
m.fs.F102.vap_outlet.temperature.fix(375)
m.fs.F102.deltaP.fix(-200000)

# expand arcs
TransformationFactory("network.expand_arcs").apply_to(m)
# re-initialize
_ = vistut.initialize_model(m)
```

Since the FV is connected to the current state of the model in memory, simply hitting "Refresh" in the FV window will show the new flash unit in the diagram, and the new stream (liquid) in the stream table. We can then interactively rearrange the unit to be in the position we want in the diagram.

In [ ]:
```
#v model.fs.visualize("Hydrodealkylation-new")
```

Showing new values

The previous step showed how a new unit in the flowsheet will be automatically added to the diagram. Similarly, if the values in the flowsheet change these will be reflected in the stream table. Below, we solve the initialized flowsheet. To make comparison a little easier, we will open a second UI window with the new values (the old values will not be updated unless we decide to hit the "Refresh" button).

In [ ]:
```
# Create the solver object
from pyomo.environ import SolverFactory
solver = SolverFactory('ipopt')
solver.options = {'tol': 1e-6, 'max_iter': 5000}

# Solve the model
results = solver.solve(model, tee=False)

# Open a second window
model.fs.visualize("HDA_solved")
```

When we look at the stream table, we can see the values in the stream between the first and second flash unit changing.

Running from a script

Finally, although all examples have been shown in a Jupyter Notebook, there is nothing preventing the use of the FV from within a plain Python script (or module).

For example, the code to run this same tutorial as a Python script is also in the file visualizer_tutorial.py in the same folder as this notebook.

If you have installed the IDAES examples (using `idaes get-examples`), then you can do the following to import and run the module:

```
from idaes_examples.Tutorials.Basics import visualizer_tutorial
visualizer_tutorial.main()
```

Further reading

Reference documentation for the FV is available in the IDAES main documentation, online at https://idaes-pse.readthedocs.io/

In [ ]:
```
```