Advanced Training: Recent Features, Output Analysis and Current Best Practices Part 1

Bryan Tomsula
CPFD, LLC

2019 Barracuda Virtual Reactor Users Conference
June 27-28, 2019
Chicago, IL

Outline of Topics

Graphical Selection of ICs and BCs

Base Materials

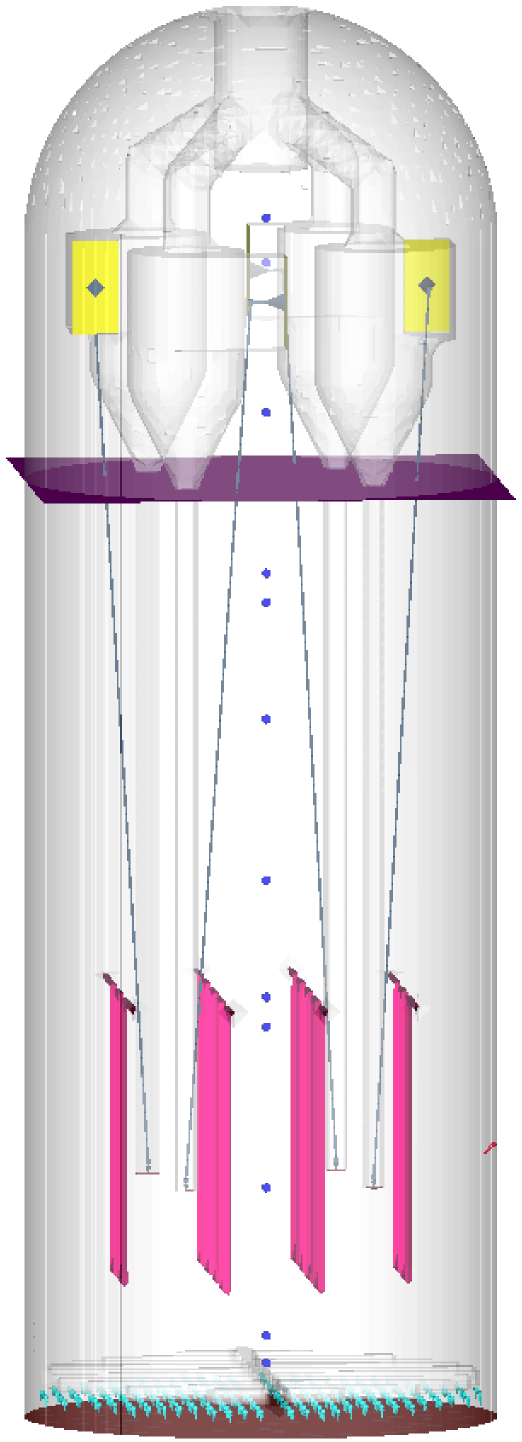
- Updated Base Materials Editor
- “Interpolated from SFF File”
- Material Property Limiters

Plotting Pressure Profiles

Initial Conditions from IC files

New GMV Output Options

Chemistry Reaction Rate Coefficient User-Defined Expression Parser



Example based on Wednesday Gasifier from Barracuda training class

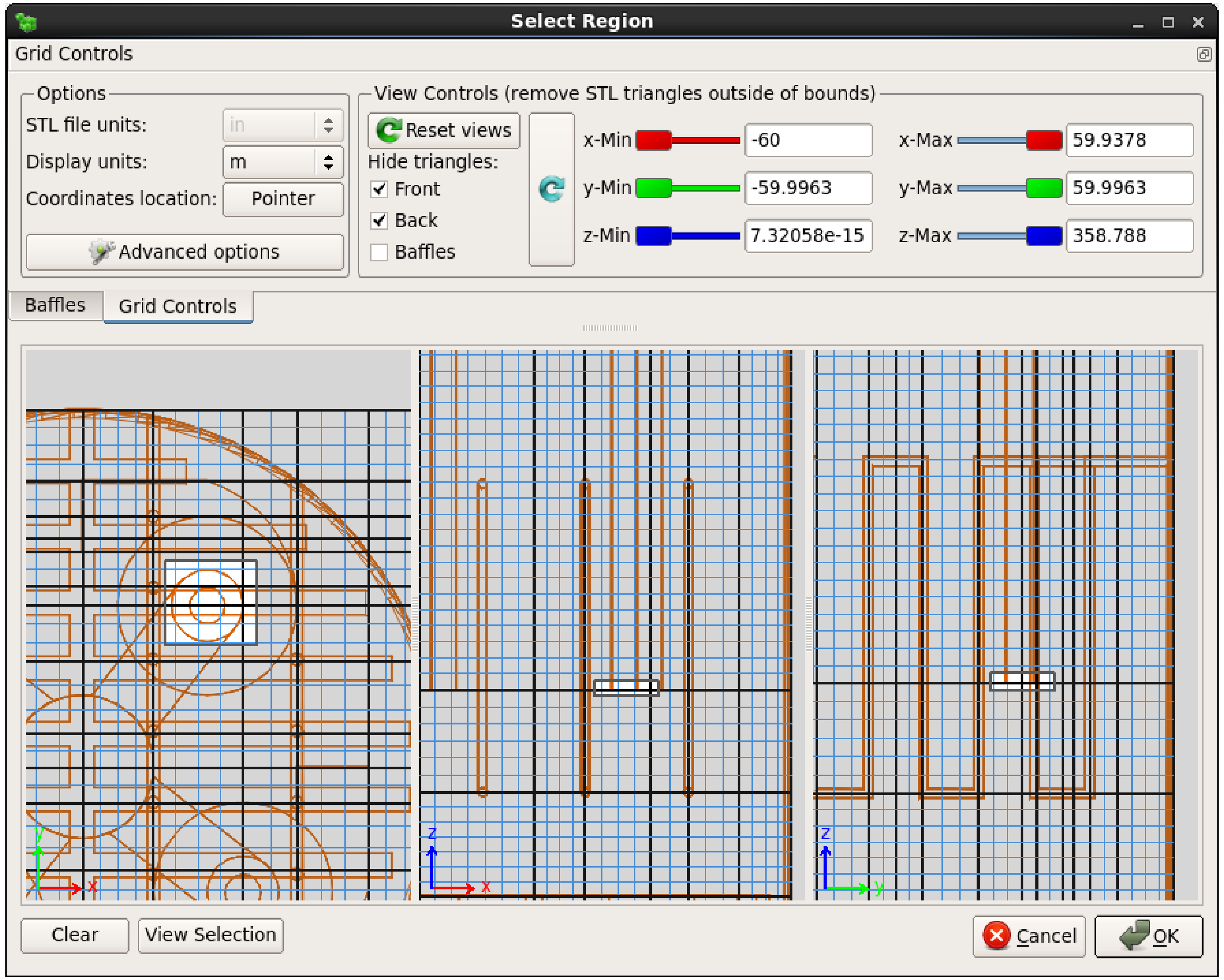
Graphical Selection of ICs and BCs

Added in release 17.3.0

No longer using IJK cell indices to locate ICs and BCs

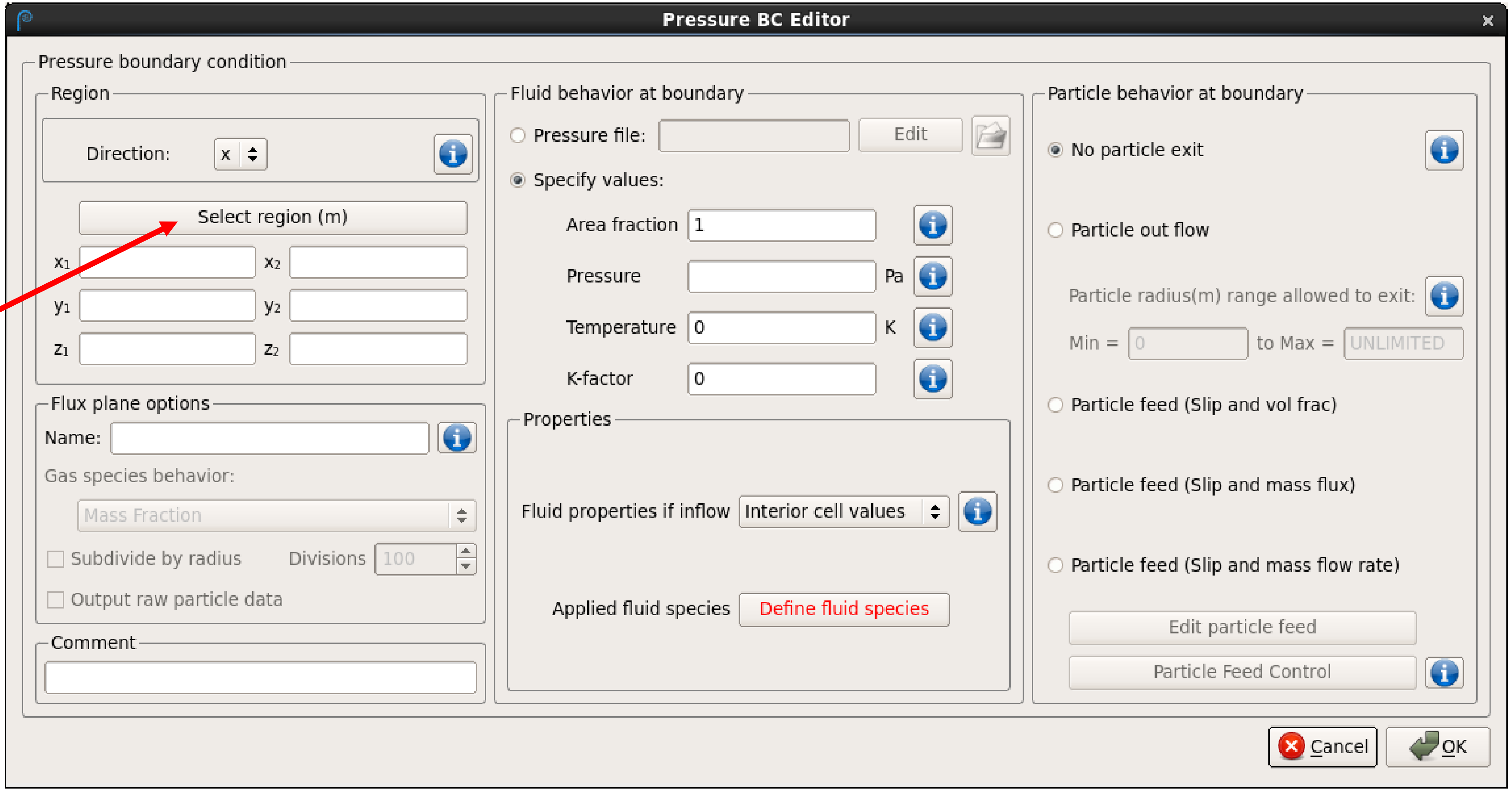
Now using XYZ physical coordinates

- Will not change when grid resolution is increased or decreased
- Graphical point-and-click selection of IC and BC locations
- Quicker and easier to define



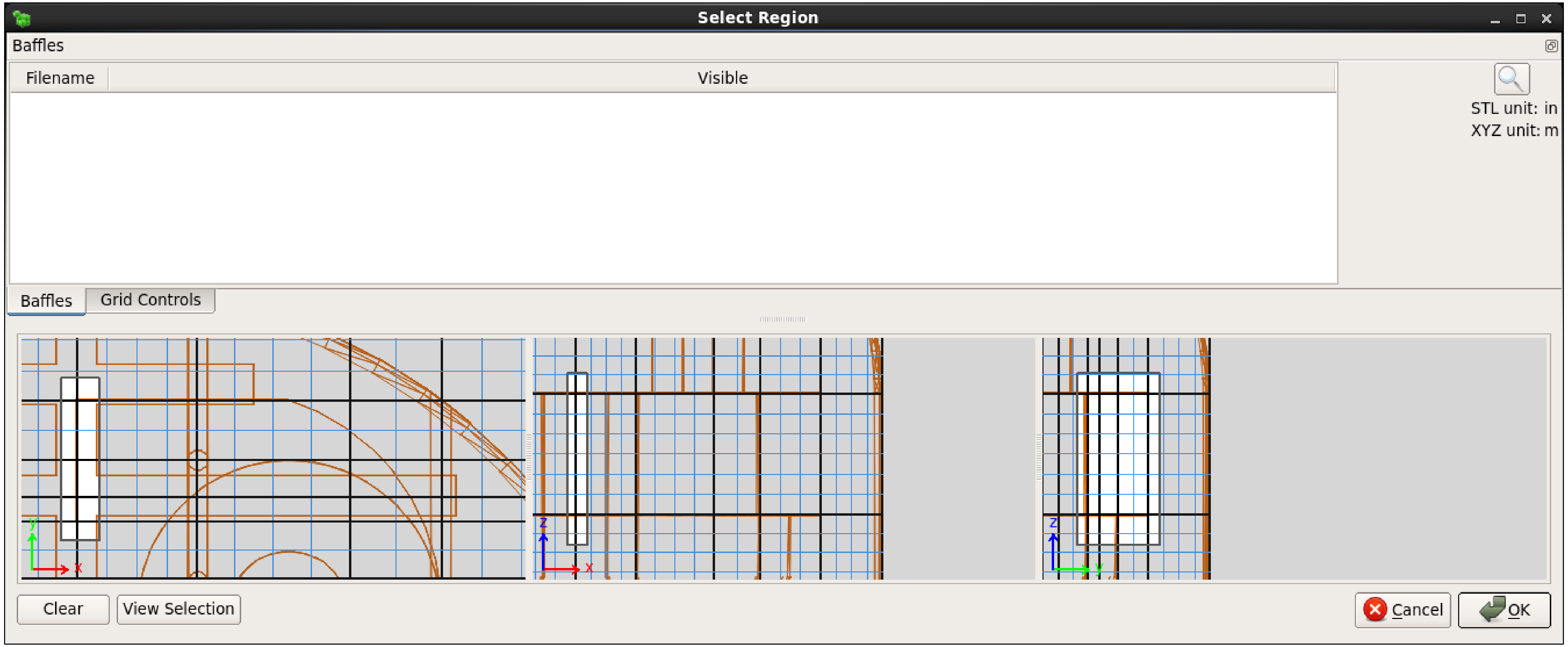
Selecting Pressure BC for Cyclone Inlets

Click on *Select region*

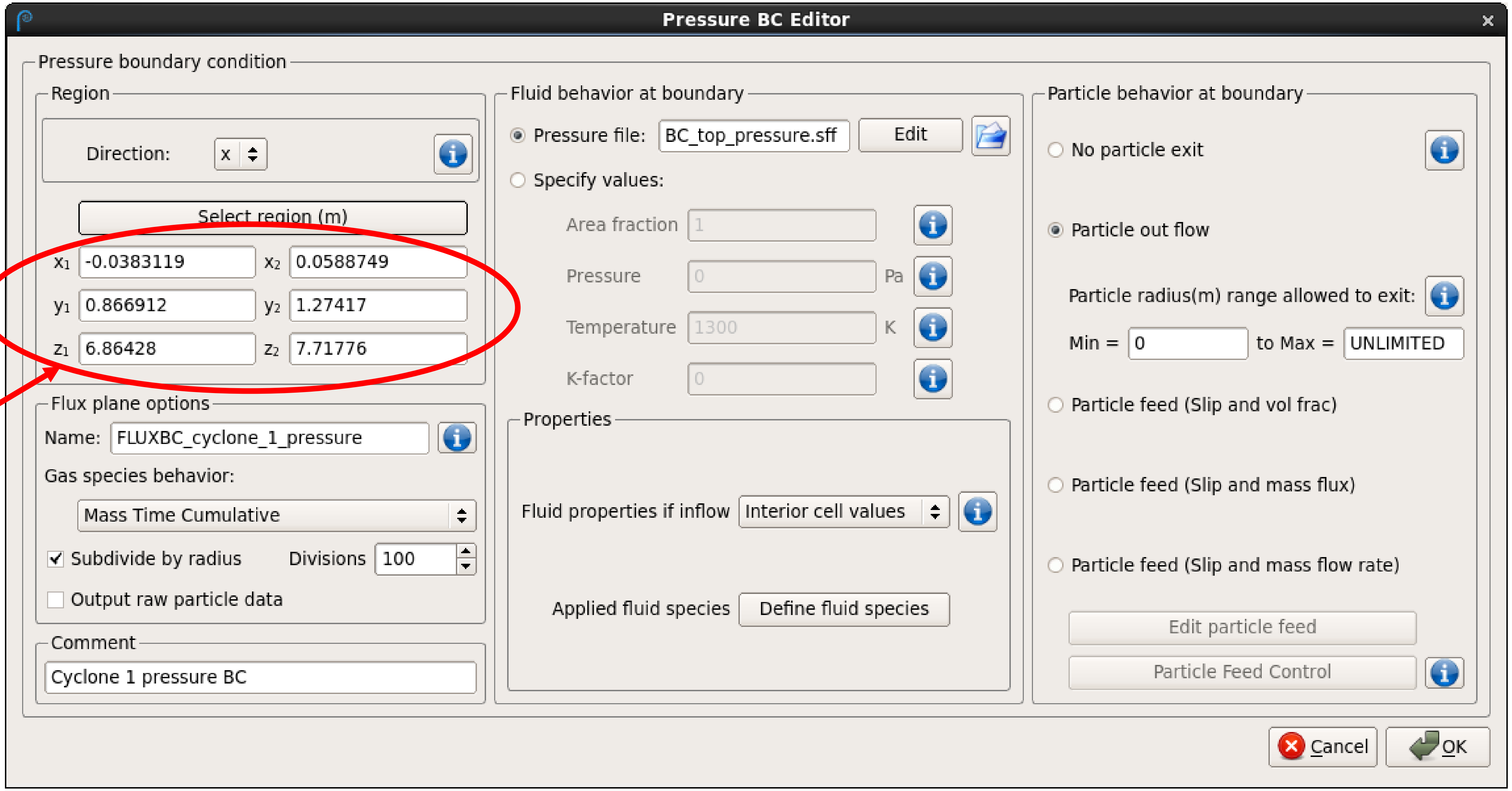


Selecting Pressure BC for Cyclone Inlets

Click and drag to graphically select the cyclone inlet horn face



Selecting Pressure BC for Cyclone Inlets



x, *y*, *z* automatically populated from graphical selection

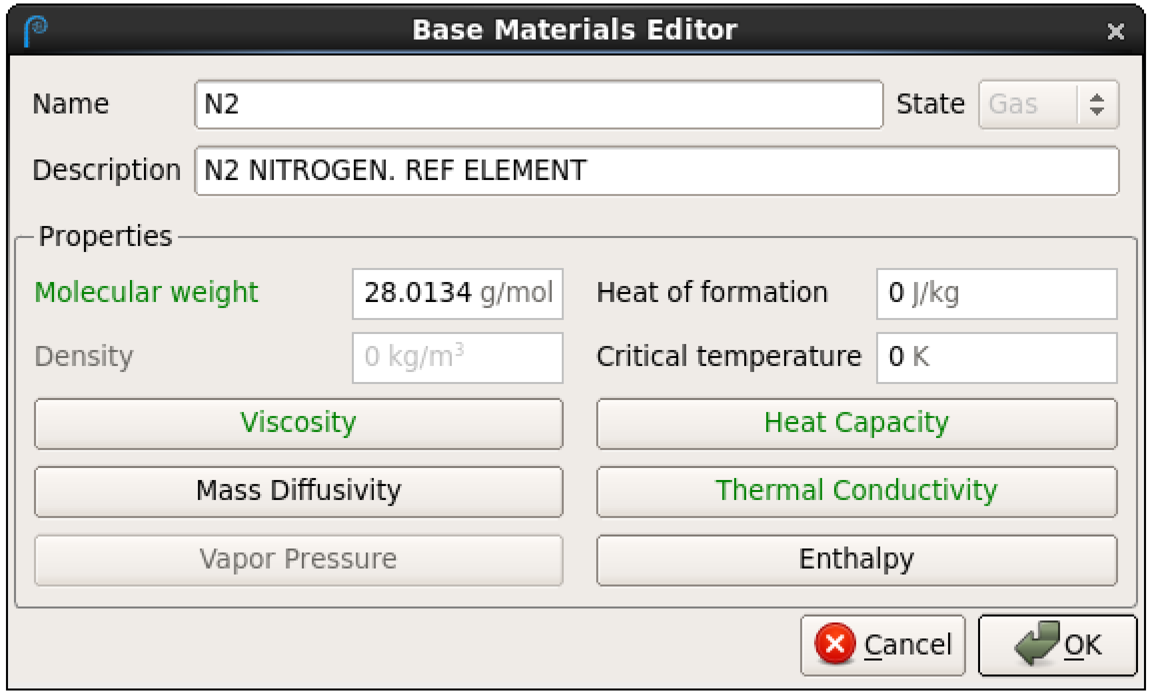
Updated Base Materials Editor

Added in release 17.3.0

More compact and logical arrangement of properties in the GUI

Color-coded labels

- Green = properties are valid across entire temperature range
- Red = properties are invalid at some temperatures



Material Properties “Interpolated from SFF File”

Added in release 17.3.0

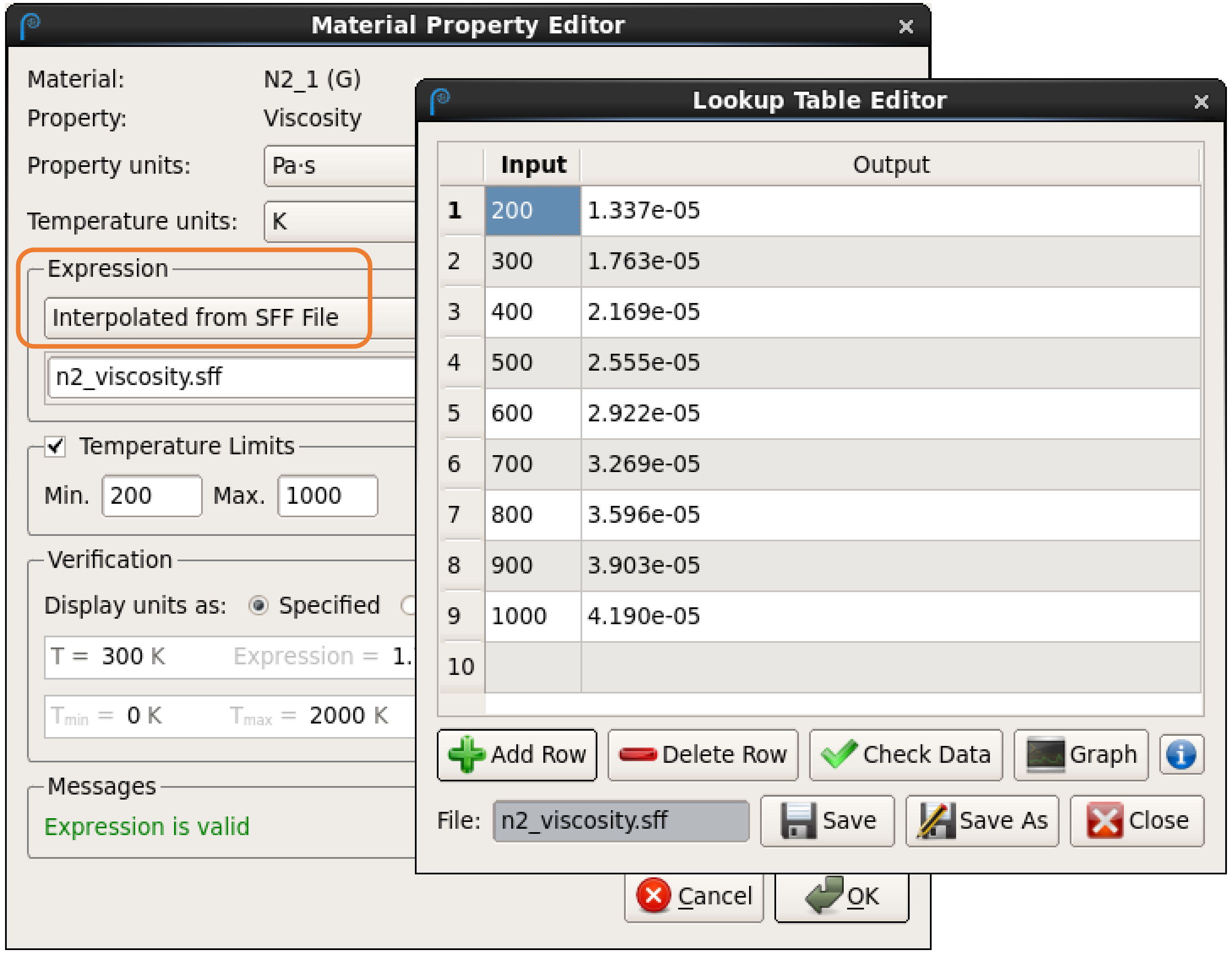
New functional form available for all material properties

- “Interpolated from SFF file”

“Input” is temperature in the user-selected units

“Output” is the current property in the user-selected units

Linear interpolation between rows



Material Property Temperature and Value Limits

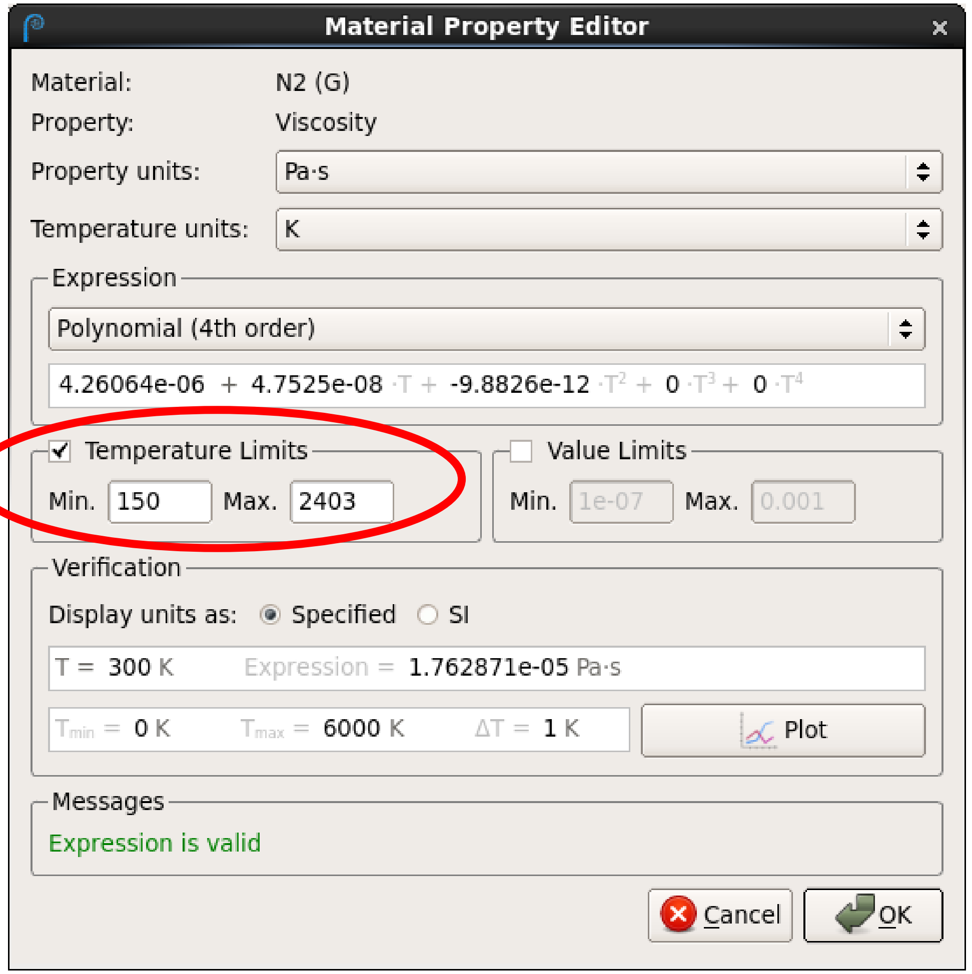
Example: the 4th-order polynomial expression for the viscosity of N2 goes negative at about 4900 K

- Negative viscosity (or other material properties) can lead to invalid results and/or solver crashes

Temperature and/or Value Limits allow for bounds to be placed for each property

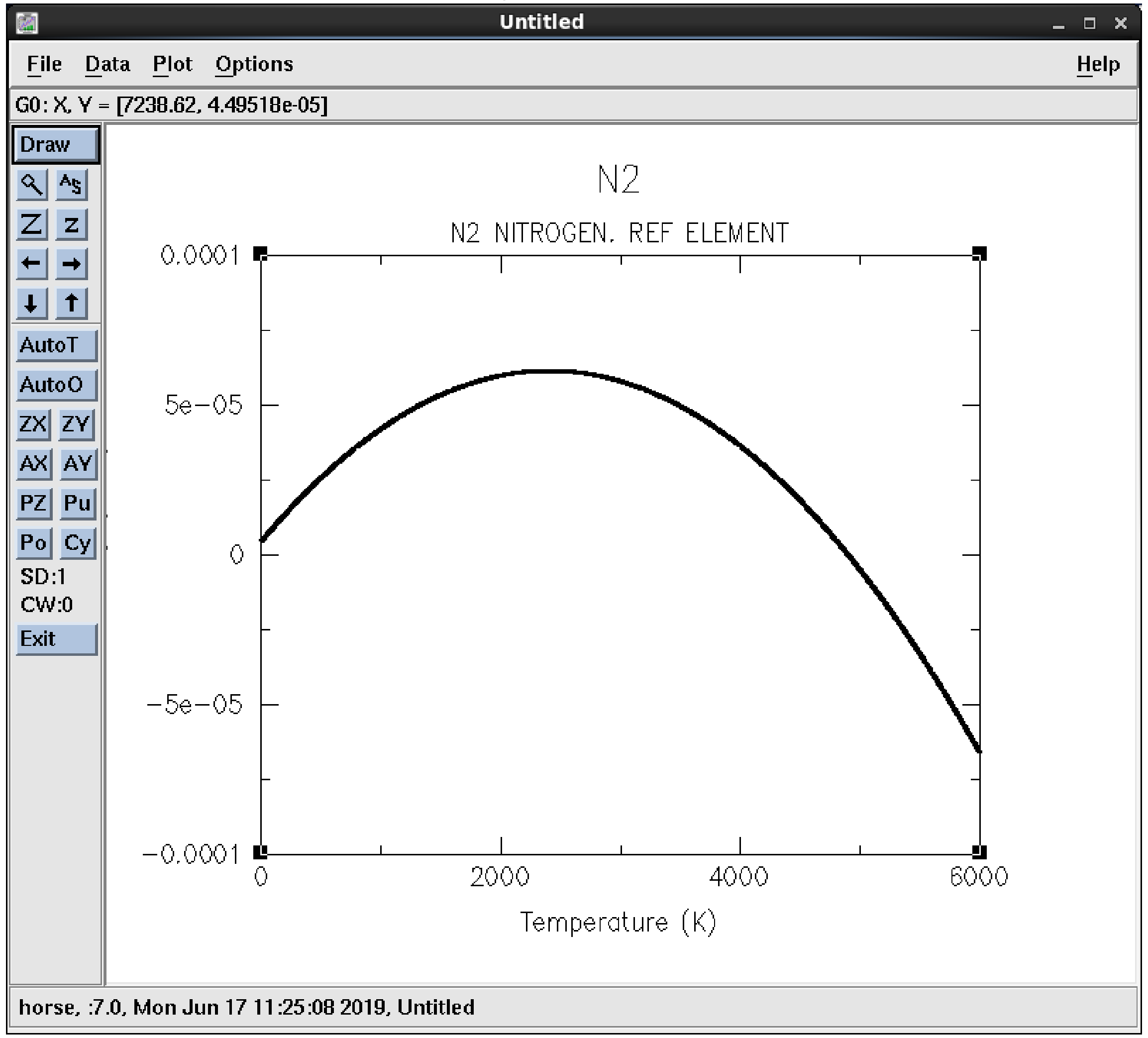
- If T < Min. value, property is evaluated at T=Min.
- If T > Max. value, property is evaluated at T=Max.
- Min. and Max. Value Limits can also be defined

Built-in database materials have limits pre-applied when available from the reference sources

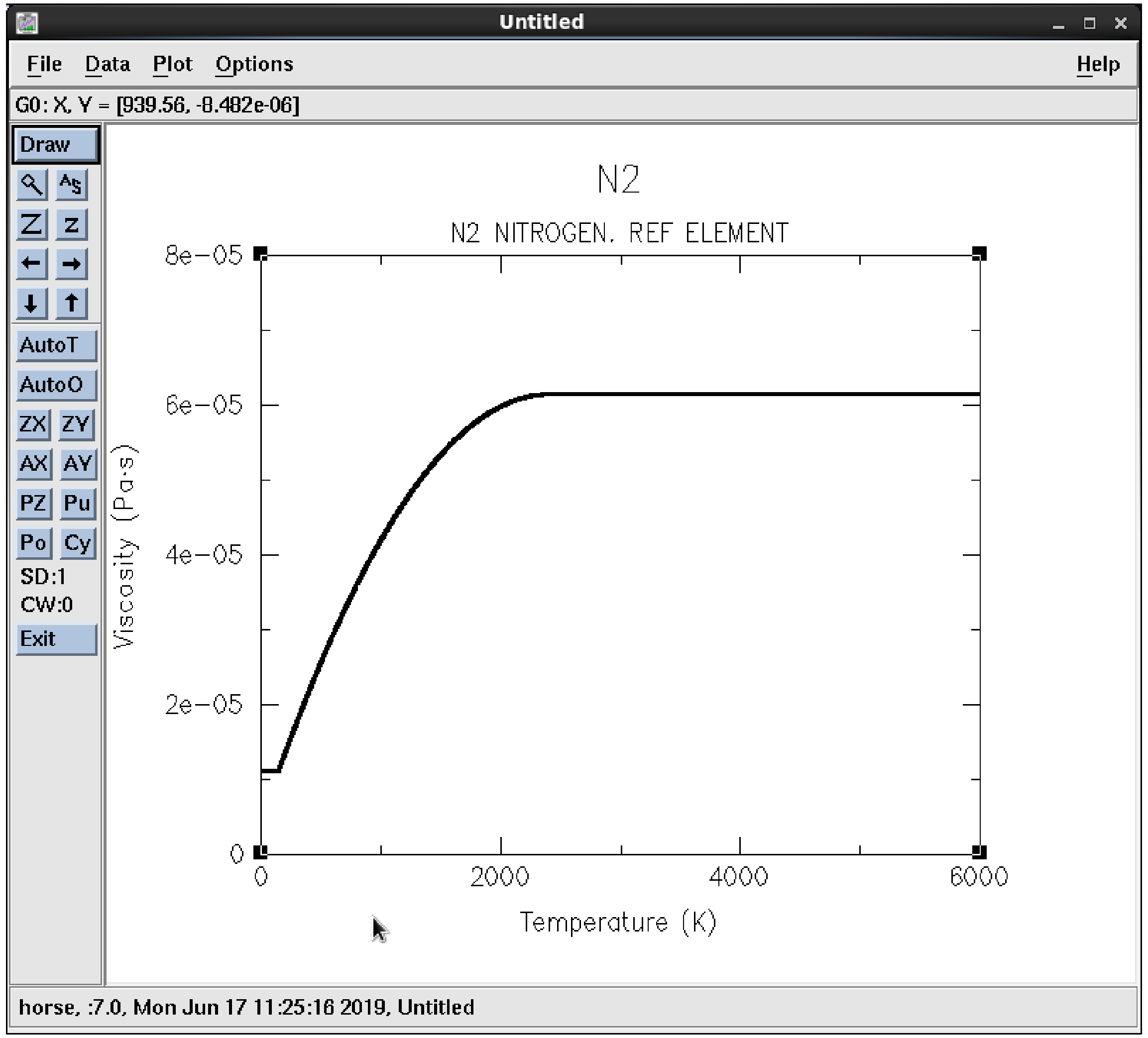


Material Property Limiter Example: N2 Viscosity

No Temperature Limits



With Temperature Limits



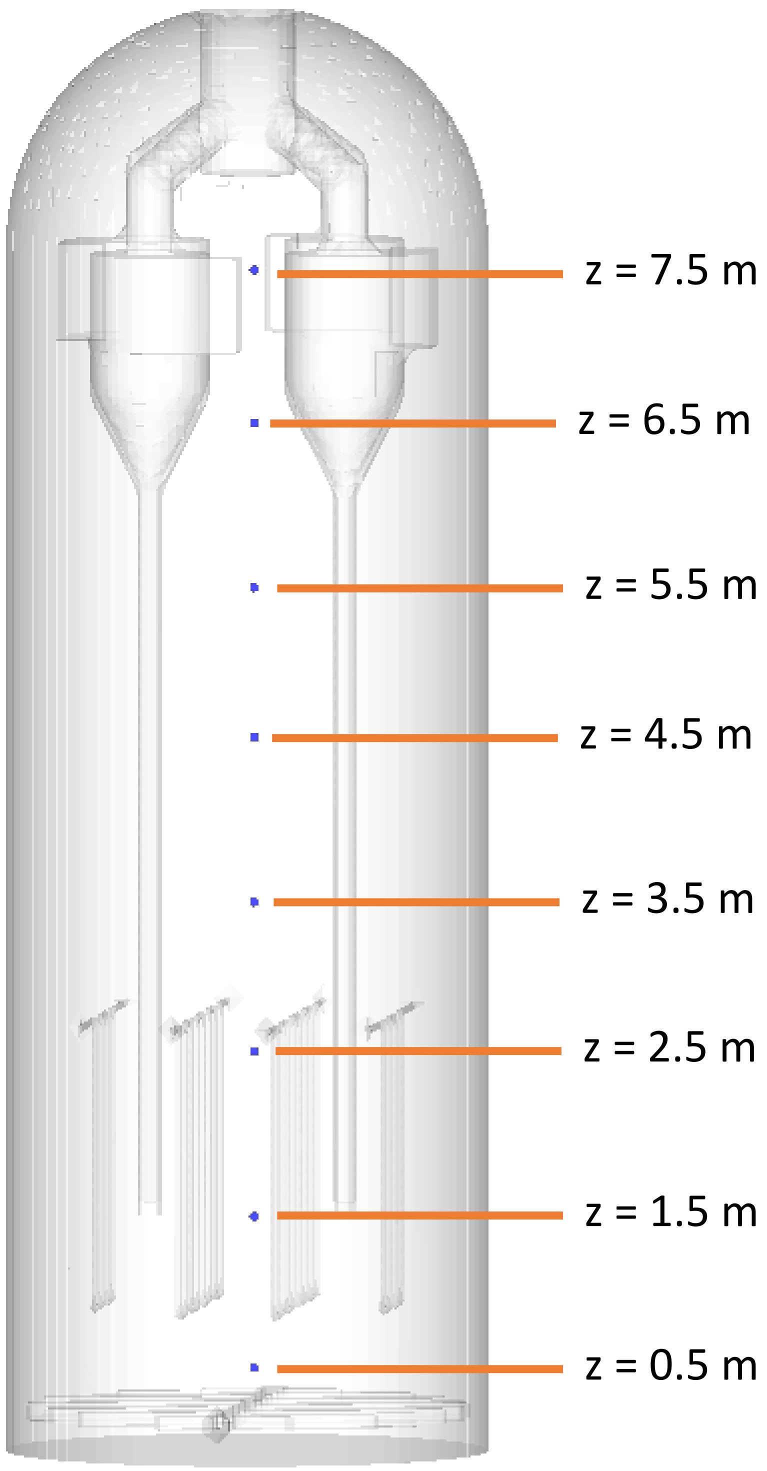
Plotting Pressure Profiles

Transient data points defined along the vertical axis of the vessel

- Record pressure every time-step
- Analogous to physical pressure taps

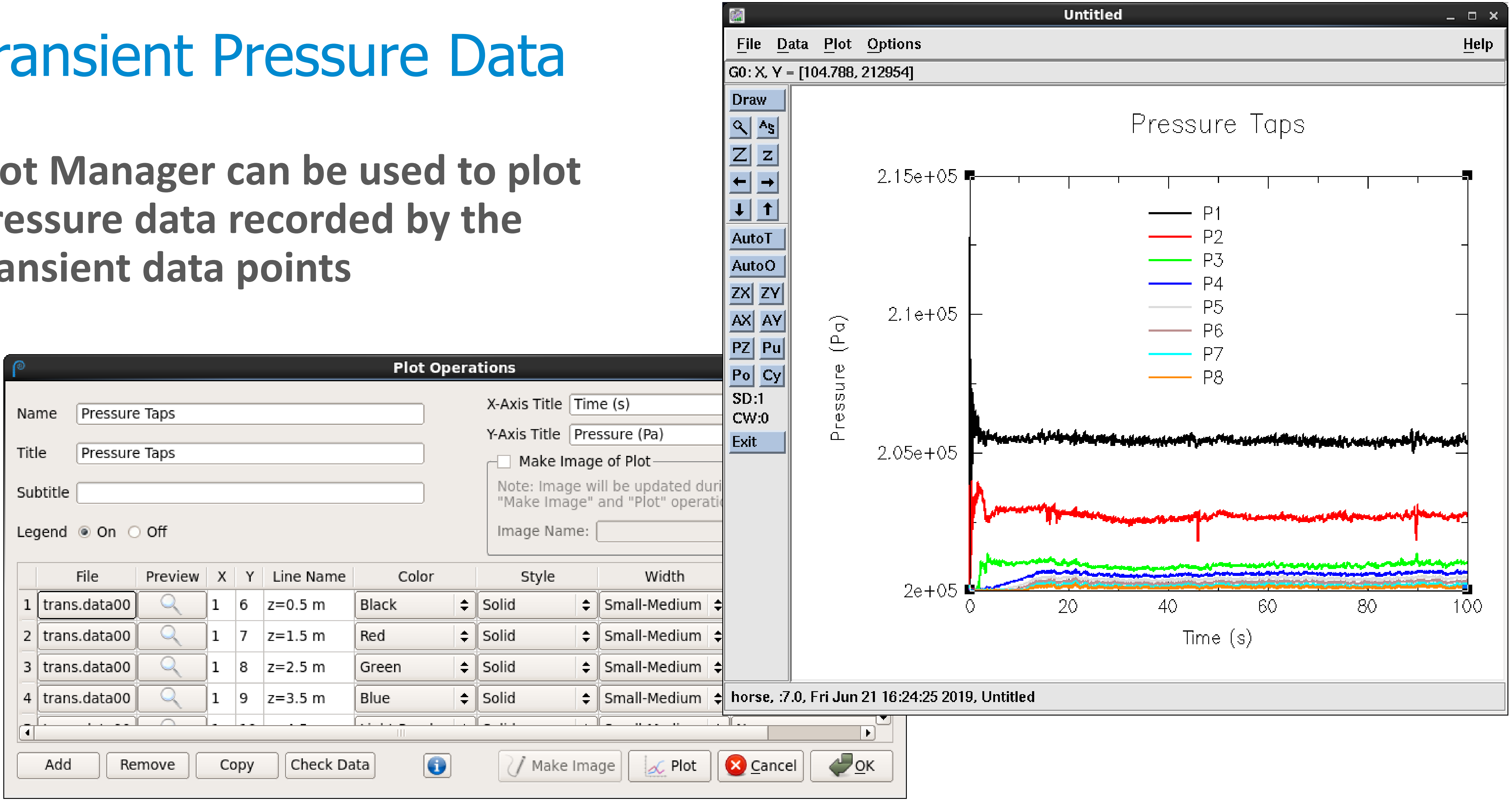
For convenience, space points evenly at 1-meter intervals

- This makes apparent bed density calculations very easy, since Δz is simply 1 m



Transient Pressure Data

Plot Manager can be used to plot pressure data recorded by the transient data points



Plotting the Pressure Profile at a Specific Time

Python can be used to perform data analysis on the transient pressure data

- Select data for a certain time-period: 90 to 100 s
- Perform a time-average for each location
- Plot the pressure profile vs height

Start a Jupyter Notebook in the simulation directory:

```
jupyter-notebook pressure_profile_example.ipynb
```

Start with a typical "import" cell in the Jupyter notebook:

In [1]:
```
%matplotlib inline
from __future__ import print_function
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np
```

Read from trans.data00 and Calculate Time-Average

In [2]:

```
# Read data from transient data file
#       Variable name          Units  ijk                    xyz(m)                          Comment
#       --------------------   ------ ------------   ----------------------------------------   ----------
#@   1  "Time"                 "s"
#@   ...
#@   6  "Pressure"             "Pa"   " 17  19   8"  " 3.75307e-02  3.58722e-02  5.18390e-01"  "P1"
#@   7  "Pressure"             "Pa"   " 17  19  18"  " 3.75307e-02  3.58722e-02  1.47646e+00"  "P2"
#@   8  "Pressure"             "Pa"   " 17  19  29"  " 3.75307e-02  3.58722e-02  2.52164e+00"  "P3"
#@   9  "Pressure"             "Pa"   " 17  19  39"  " 3.75307e-02  3.58722e-02  3.46846e+00"  "P4"
#@  10  "Pressure"             "Pa"   " 17  19  50"  " 3.75307e-02  3.58722e-02  4.50997e+00"  "P5"
#@  11  "Pressure"             "Pa"   " 17  19  60"  " 3.75307e-02  3.58722e-02  5.45679e+00"  "P6"
#@  12  "Pressure"             "Pa"   " 17  19  71"  " 3.75307e-02  3.58722e-02  6.49830e+00"  "P7"
#@  13  "Pressure"             "Pa"   " 17  19  81"  " 3.75307e-02  3.58722e-02  7.46836e+00"  "P8"

# List of data columns we want to read:
dataColumnList = [6, 7, 8, 9, 10, 11, 12, 13]

# Array of height values corresponding to each transient data point:
zArray = np.array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5])

# Min and max time values we want to use for calculating average pressure:
tMin = 90  # (s)
tMax = 100 # (s)

# Array to hold calculated average pressure values:
avePressureArray = np.zeros_like(zArray)

for i, dataColumn in enumerate(dataColumnList):
    t, P = np.genfromtxt('trans.data00', usecols=(0,dataColumn-1), unpack=True)
    tFilter = (t > tMin) & (t <= tMax)
    avePressureArray[i] = np.mean(P[tFilter])
```

How to filter for a specific time period:

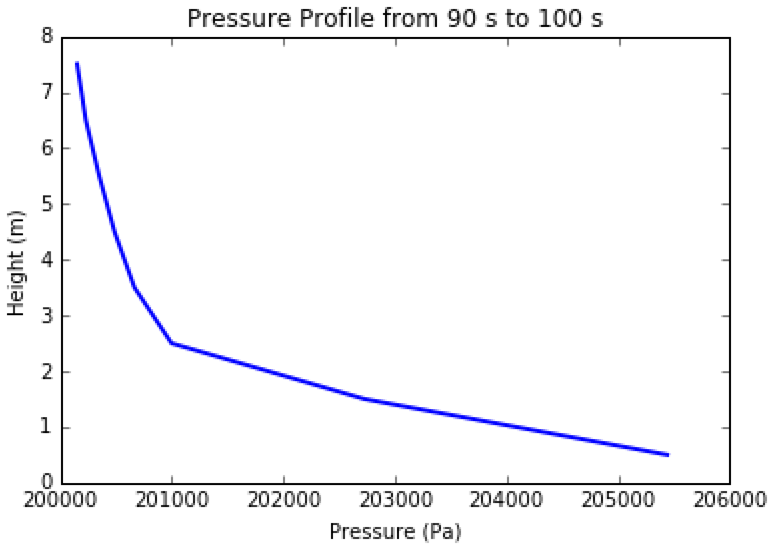
```
tFilter = (t > tMin) & (t <= tMax)
```

You can then use this as a filter on any other data array with matching dimensions:

```
myAverage = np.mean(P[tFilter])
```

Plot the Average Pressure Profile, 90 s to 100 s

In [3]:
```
# Plot the pressure profile calculated above
fig, ax = plt.subplots()
ax.plot(avePressureArray, zArray, linewidth=2)
ax.set_title('Pressure Profile from 90 s to 100 s')
ax.set_xlabel('Pressure (Pa)')
ax.set_ylabel('Height (m)')
p = 'pressure_profile_90_s_to_100_s'
fig.savefig(p + '.png', format='png')
fig.savefig(p + '.pdf', format='pdf')
```



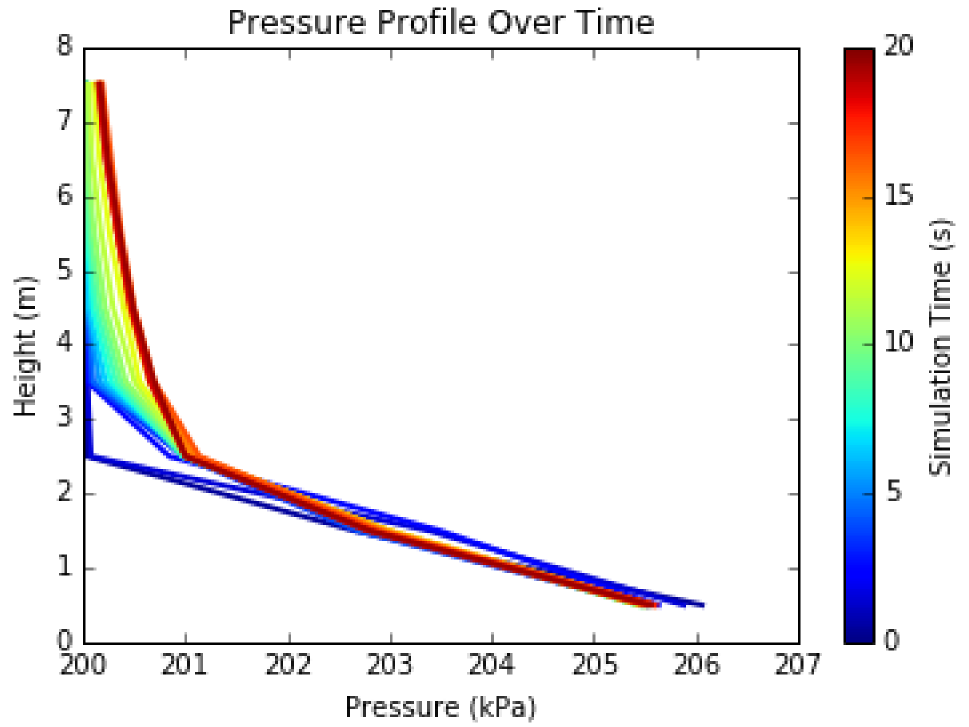
Plotting Pressure Profile vs Time

Useful to assess whether steady state has been reached

Lines are colored using a "colormap"

- Matplotlib includes many useful colormaps, see the web page [Colormap reference]

A color bar is placed on the right-hand side of the plot to indicate the time value of each line



Define Time Periods and Read trans.data00

In [1]:

```
%matplotlib inline
from __future__ import print_function
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
```

Import the `cm` module to use colormaps

…

In [4]:

```
# Define minimum and maximum time, and number of lines you want to plot
tMin = 0
tMax = 20
N = 20

# For our plot, let's use "center" values for each time period
tArray = np.linspace(tMin,tMax,N+1)
tMinArray = tArray[:-1]
tMaxArray = tArray[1:]
tCenters = (tMinArray + tMaxArray) / 2
#print(tCenters)

# Read transient data into a 2-dimensional array *once* (don't want to read multiple times in for loops)
# Use unpack=True so that individual columns can be accessed easily
transData = np.genfromtxt('trans.data00', unpack=True)
```

`transData` is a 2-dimensional array

Create Plot for Pressure Profile Over Time

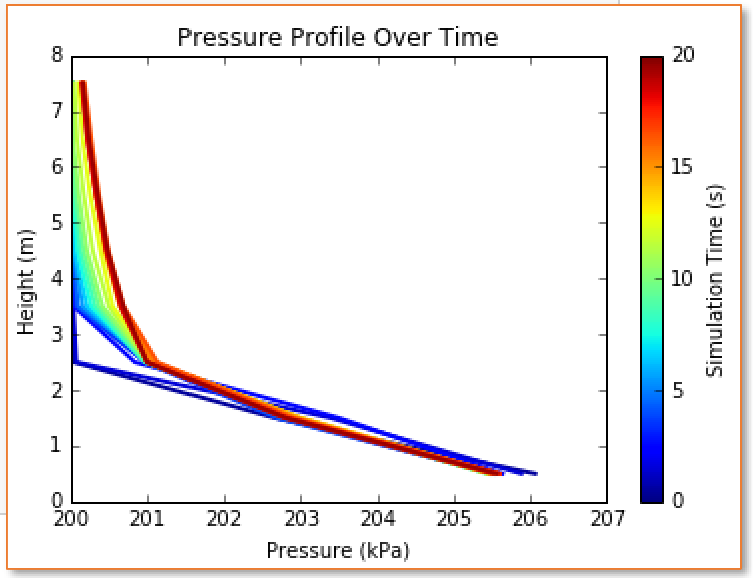
In [5]:

```
fig, ax = plt.subplots()
my_cmap = cm.jet

t = transData[0]

for tMin, tMax in zip(tMinArray, tMaxArray):
    tFilter = (t >= tMin) & (t < tMax)

    avePressureArray = np.zeros_like(zArray)

    for i, dataColumn in enumerate(dataColumnList):
        P = transData[dataColumn-1]
        avePressureArray[i] = np.mean(P[tFilter])

    myTime = (tMin + tMax) / 2
    myColor = (myTime - min(tArray)) / (max(tArray) - min(tArray))
    ax.plot(avePressureArray / 1000, zArray, linewidth=2, color=my_cmap(myColor))

# Create a colorbar to put on the right side of the plot
my_norm = plt.Normalize(vmin=min(tArray), vmax=max(tArray))
sm = cm.ScalarMappable(norm=my_norm,cmap=my_cmap)
sm.set_array([])
cbar = fig.colorbar(sm, ticks=tArray[::5])
cbar.set_label('Simulation Time (s)')

# Set the usual axis labels
ax.set_title('Pressure Profile Over Time')
ax.set_xlabel('Pressure (kPa)')
ax.set_ylabel('Height (m)')
p = 'pressure_profile_0_s_to_20_s'
fig.savefig(p + '.png', format='png')
fig.savefig(p + '.pdf', format='pdf')
```

Set the desired colormap here

Outer loop is for time values

Inner loop is for data columns



Fluid and Particle ICs: “Initialize from IC file”

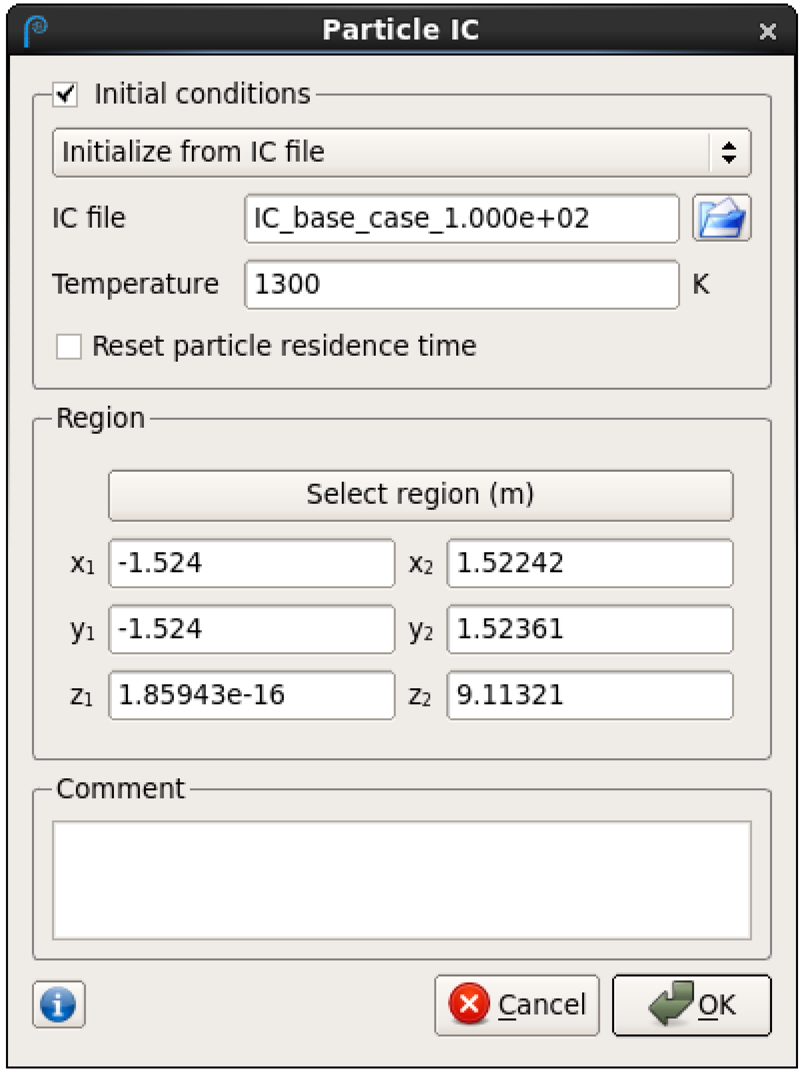
Added in release 17.3.0

Use results from a previous Barracuda simulation as fluid and particle ICs for a <u>new simulation</u> (not a restart)

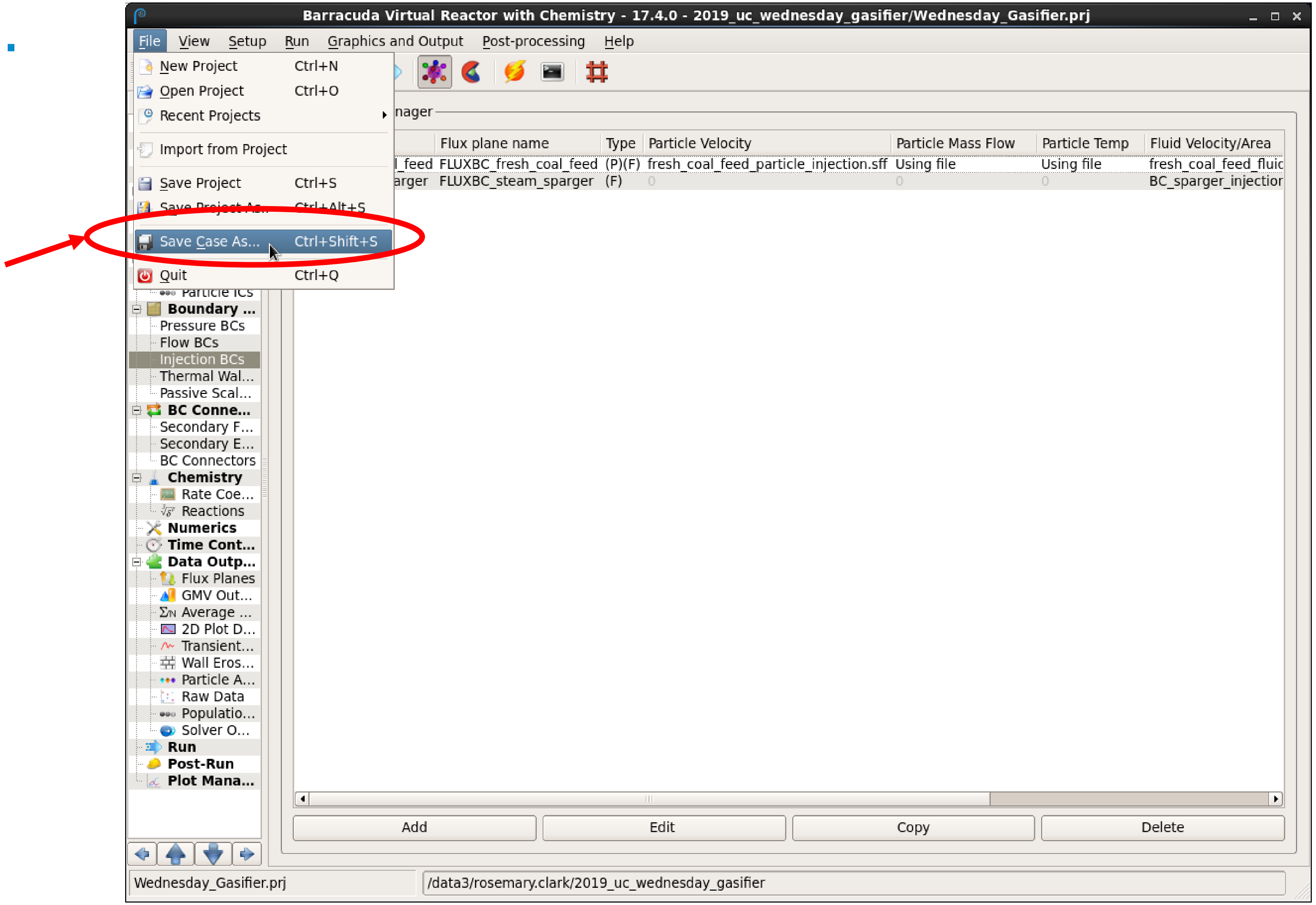
- Run a “base case” to steady state
- Start variations based on the base case steady state results
- All fluid and particle properties are used for the ICs

Current example using Wednesday Gasifier:

- Add additional Injection BC locations
- Add a new internal flux plane
- Select additional GMV output variables

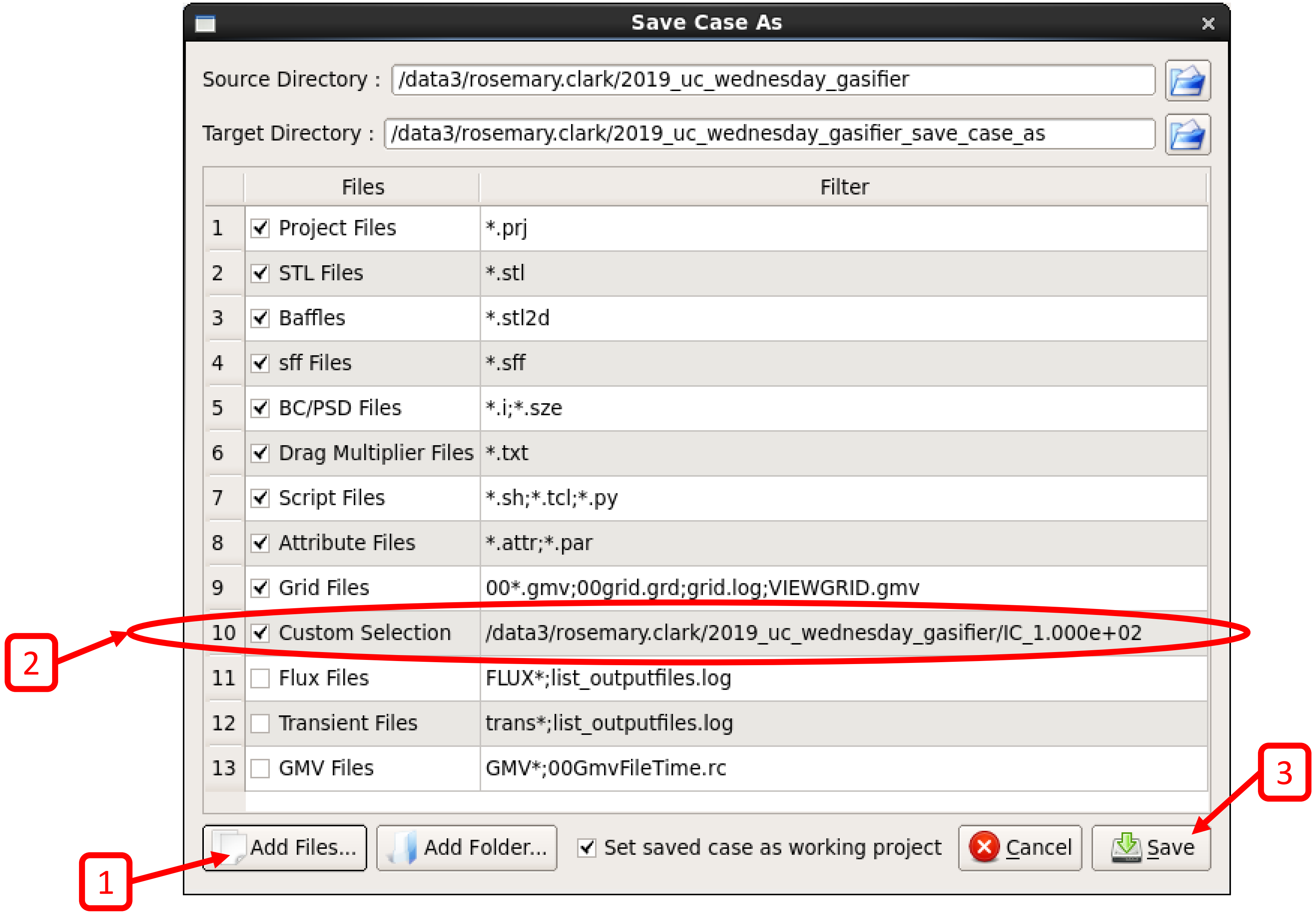


Save Case As…

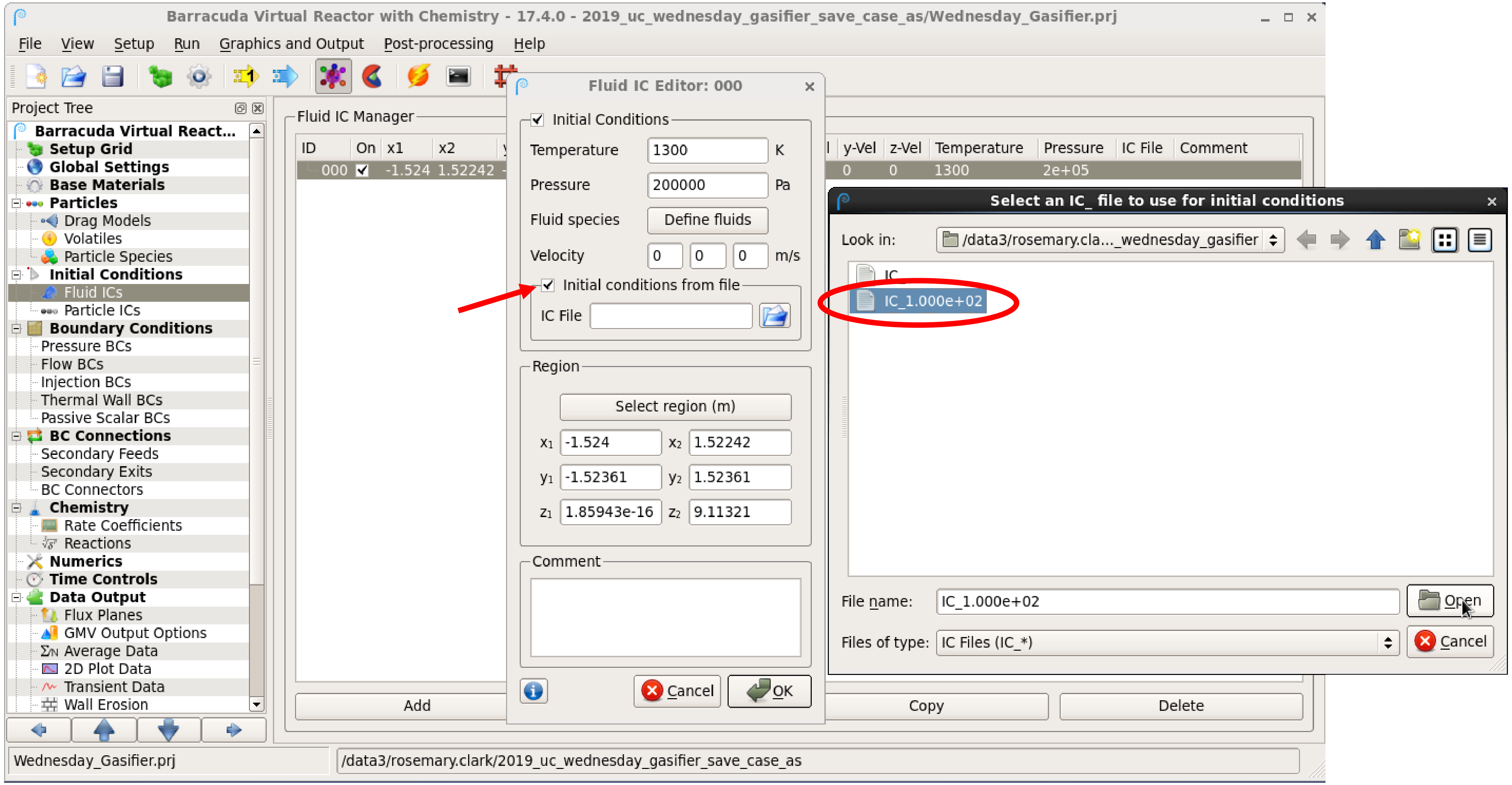


Barracuda Virtual Reactor with Chemistry - 17.4.0 - 2019_uc_wednesday_gasifier/Wednesday_Gasifier.prj
File
View
Setup
Run
Graphics and Output
Post-processing
Help
New Project Ctrl+N
Open Project Ctrl+O
Recent Projects
Import from Project
Save Project Ctrl+S
Save Case As... Ctrl+Shift+S
Quit Ctrl+Q
Flux plane name
Type
Particle Velocity
Particle Mass Flow
Particle Temp
Fluid Velocity/Area
FLUXBC_fresh_coal_feed
(P)(F)
fresh_coal_feed_particle_injection.sff
Using file
Using file
FLUXBC_steam_sparger
(F)
Boundary ...
Pressure BCs
Flow BCs
Injection BCs
Thermal Wal...
Passive Scal...
BC Conne...
Secondary F...
Secondary E...
BC Connectors
Chemistry
Rate Coe...
Reactions
Numerics
Time Cont...
Data Outp...
Flux Planes
GMV Out...
Average ...
2D Plot D...
Transient...
Wall Eros...
Particle A...
Raw Data
Populatio...
Solver O...
Run
Post-Run
Plot Mana...
Add
Edit
Copy
Delete
Wednesday_Gasifier.prj
/data3/rosemary.clark/2019_uc_wednesday_gasifier

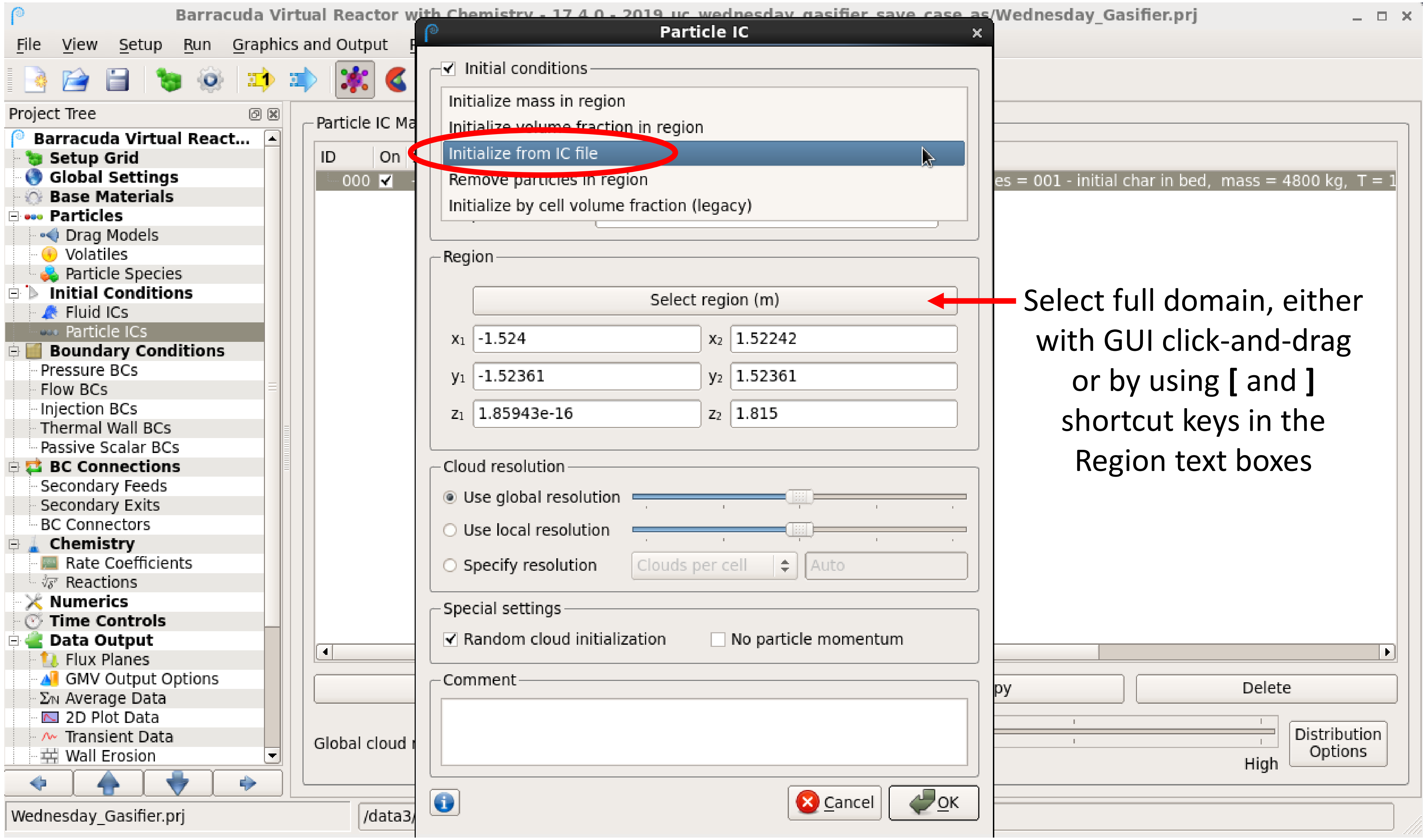
Remember to Select IC file



Edit Fluid IC

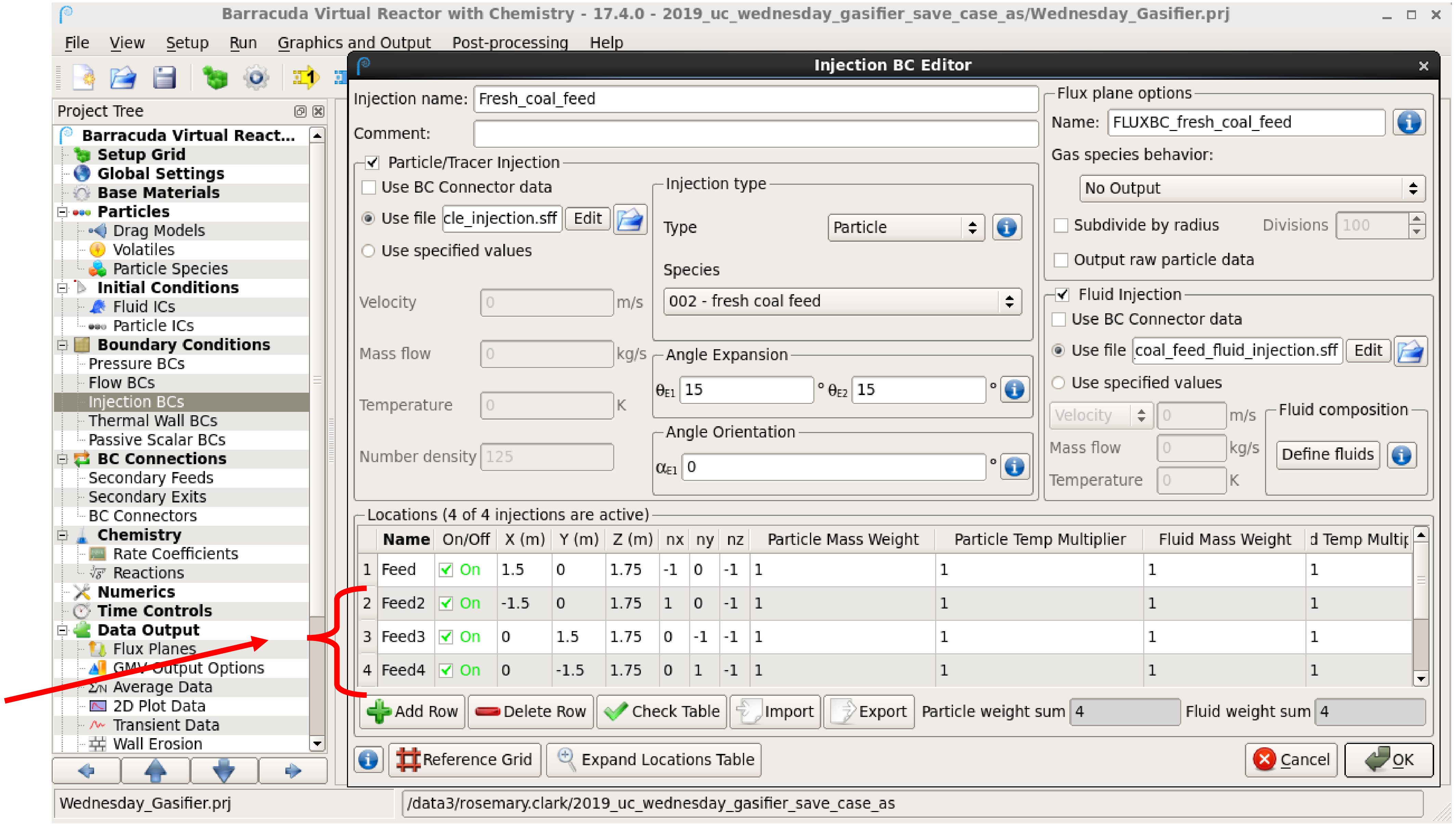


Edit Particle IC

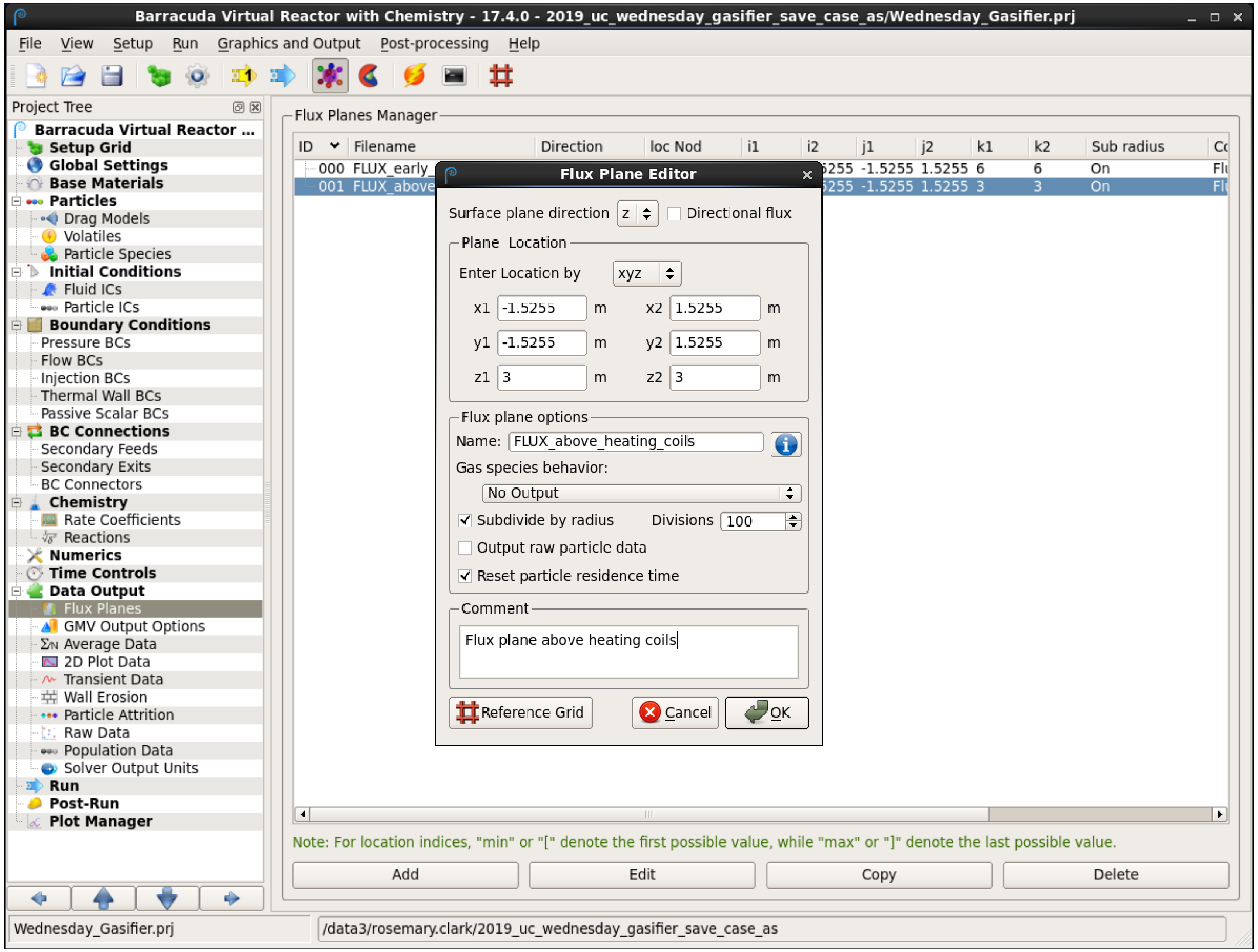


Select full domain, either with GUI click-and-drag or by using [and] shortcut keys in the Region text boxes

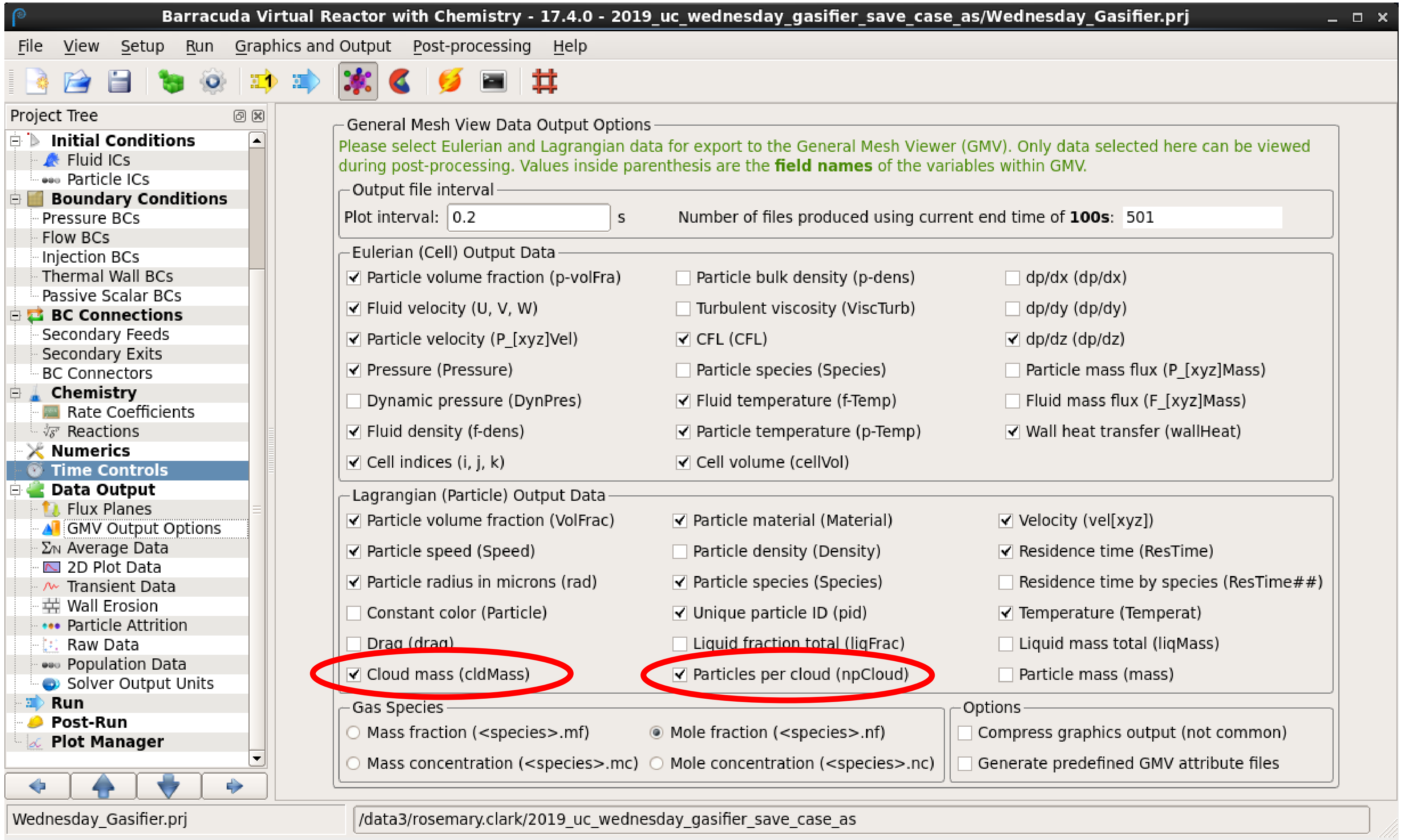
Add Injection BCs



Add Flux Plane



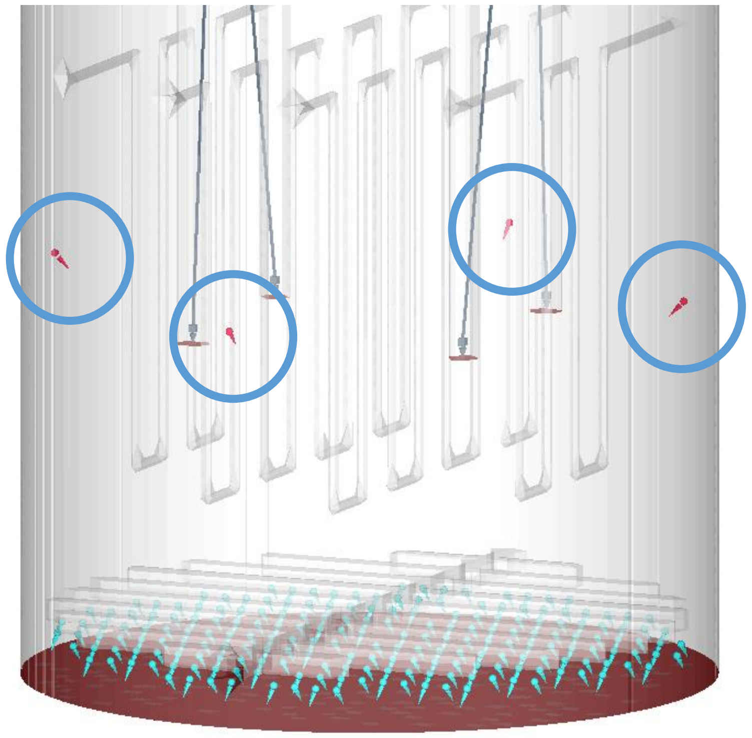
GMV Output Options



Run for a Single Time-Step to Verify Changes

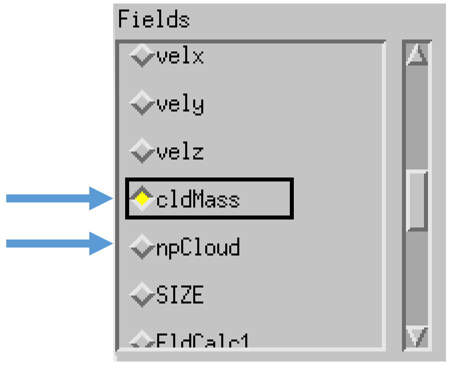
New coal feed Injection BC locations have been added

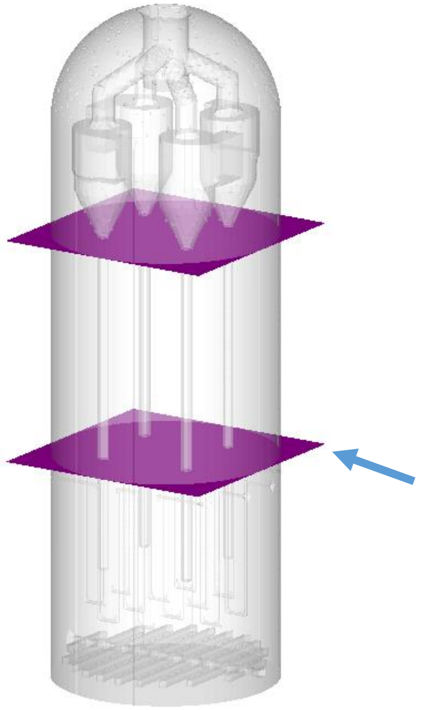
- Original simulation had 1 Injection BC for coal feed
- New simulation has 4 Injection BCs for coal feed



New internal flux plane has been added

New GMV variables have been added



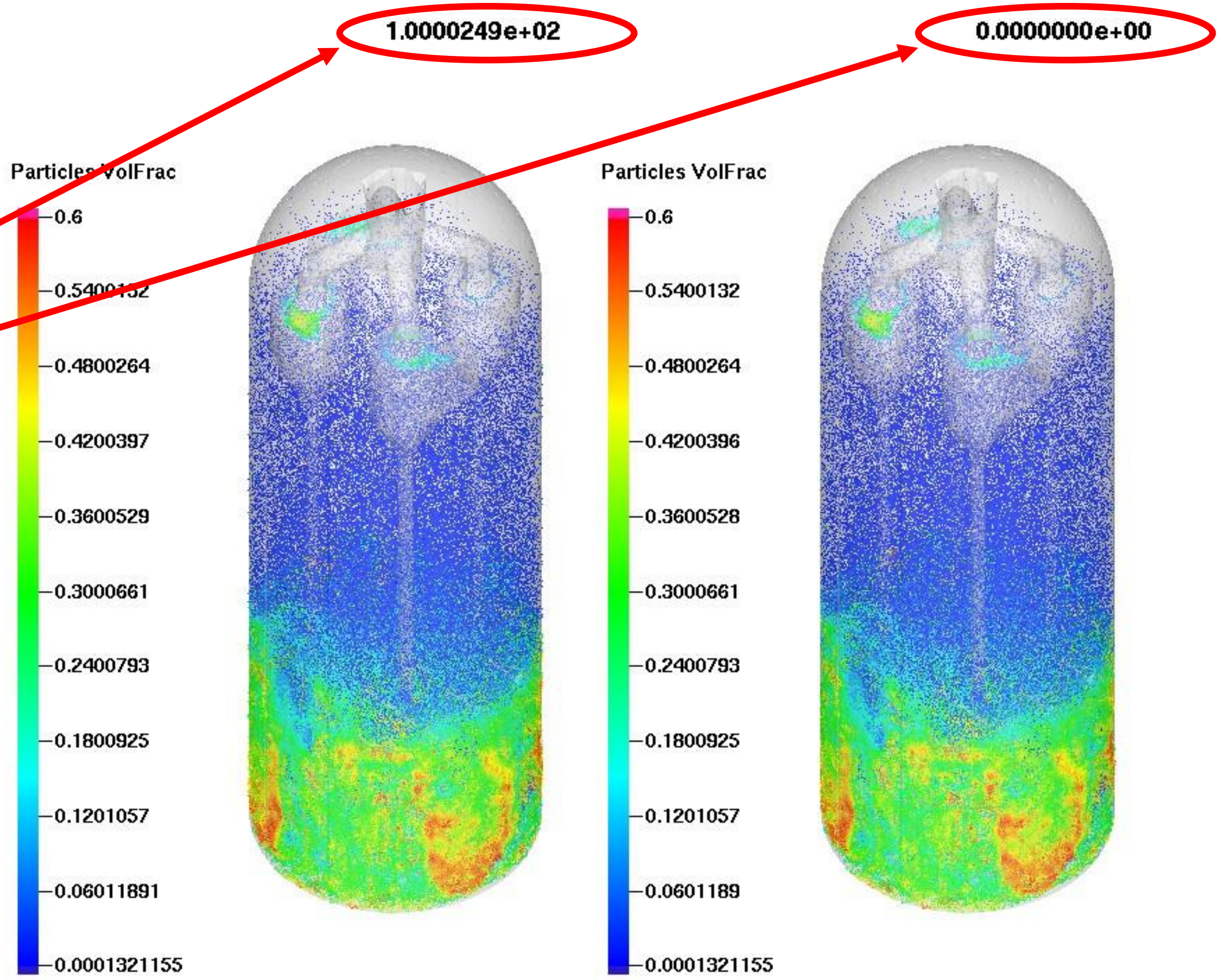


It's Not a Restart!

This is a new simulation, not a restart.

- Original simulation at t = 100 s
- Save Case As… simulation with changes at t = 0 s

The IC file from the original simulation was only used to define the initial fluid and particle fields in the new simulation



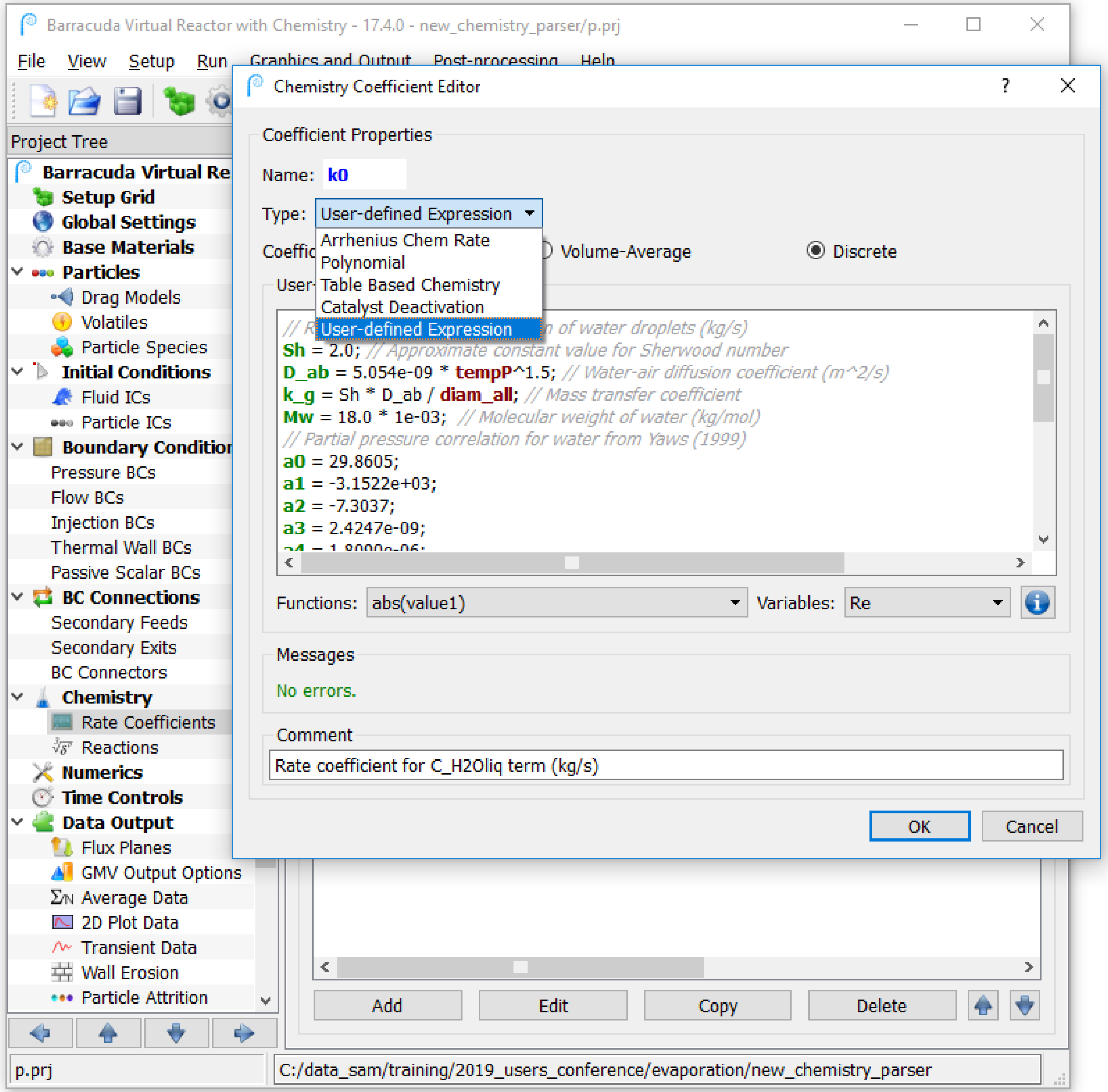
User-Defined Expressions for Reaction Rate Coefficients

New feature in 17.4.0

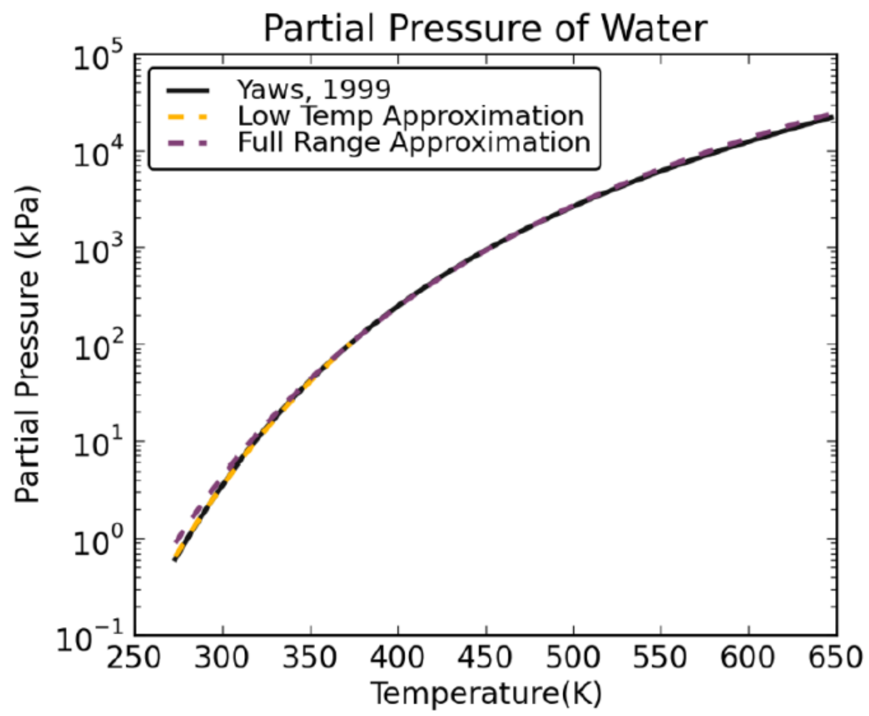
- Same parser interface as updated User-Defined Drag Model GUI

Interface is improved to allow user to simply and clearly define chemical reaction rate coefficients

- Subexpressions can be used
- Much more flexible and powerful than older constants-based input method



Example: Evaporation of Water Droplets



See CPFD Support Site post Evaporation Model for Droplets and Solid Particle Drying (Updated)

In previous versions of Barracuda:

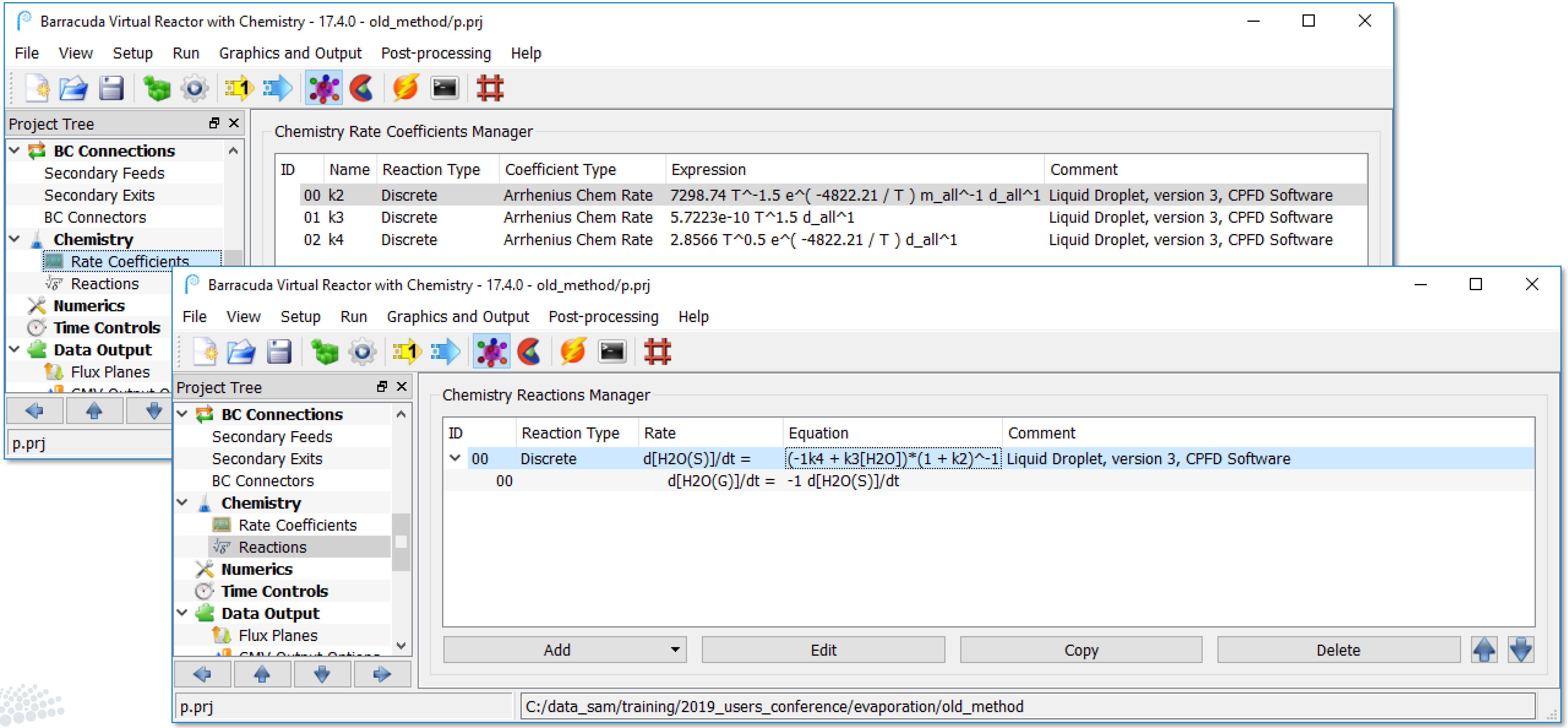
- The expression for partial pressure of water (Yaws, 1999) could not be entered directly. A curve-fit approximation was needed.

$$\log_{10} P = 29.8605 - \frac{3.1522\times10^3}{T} - 7.3037\log_{10} T + 2.4247\times10^{-9} T + 1.8090\times10^{-6} T^2$$

- Implementing the time-step stability term required multiple inter-related rate coefficient definitions.

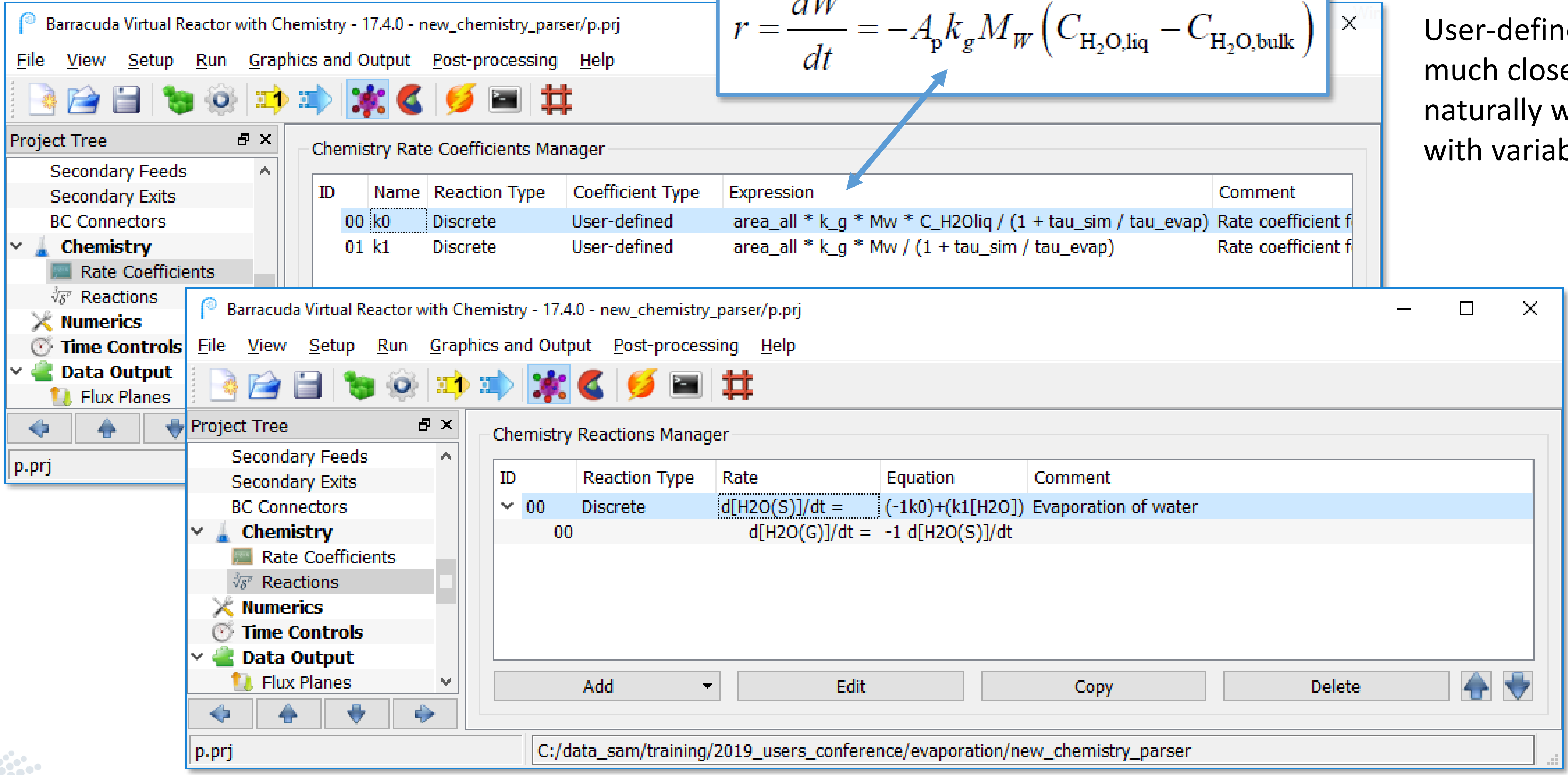
$$\frac{dW}{dt} = \frac{\kappa C_{H_2O,bulk} - \kappa R^{-1} T_p^{-1} P_0 \exp\left(-\beta / T_p\right)}{1 + \tau_{sim} / \tau_{evap}}$$

Reaction Definitions Using pre-17.4.0 Input (Arrhenius)



Reaction Definitions with User-Defined Expressions

$$r = \frac{dW}{dt} = -A_p k_g M_W \left(C_{H_2O,liq} - C_{H_2O,bulk} \right)$$



User-defined expression is much closer to the way we naturally write the reaction rate with variables.

Syntax of User-Defined Expressions

C-like input

- End statements with **;**
- Final **return** statement

Syntax highlighting

Comments are supported

- Use **//**

Real time error-checking