ANNOTATIONS

Lecture 3.3

Modelon

OVERVIEW

- Annotations
- Variables
- Connectors
- Documentation
- Icons
- Functions

ANNOTATIONS

ANNOTATIONS

- Annotations contains additional meta data

Variable annotation
- Parameter dialog layout

```
model HeatCapacitor "Lumped thermal element storing heat"
    parameter .Modelica.SIunits.HeatCapacity C "Heat capacity of element (= cp*m)"  annotation (•••);
    .Modelica.SIunits.Temperature T(start = 293.15,displayUnit = "degC") "Temperature of element" annotation (•••);
    .Modelica.SIunits.TemperatureSlope der_T(start = 0) "Time derivative of temperature (= der(T))" annotation (•••);
    .Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port annotation(•••);
equation
    T=port.T;
    der_T=der(T);
    C * der(T)=port.Q_flow;
    annotation(•••);
end HeatCapacitor;
```

Component annotation
- Size and placement on canvas

Class annotation
- Documentation
- Icon

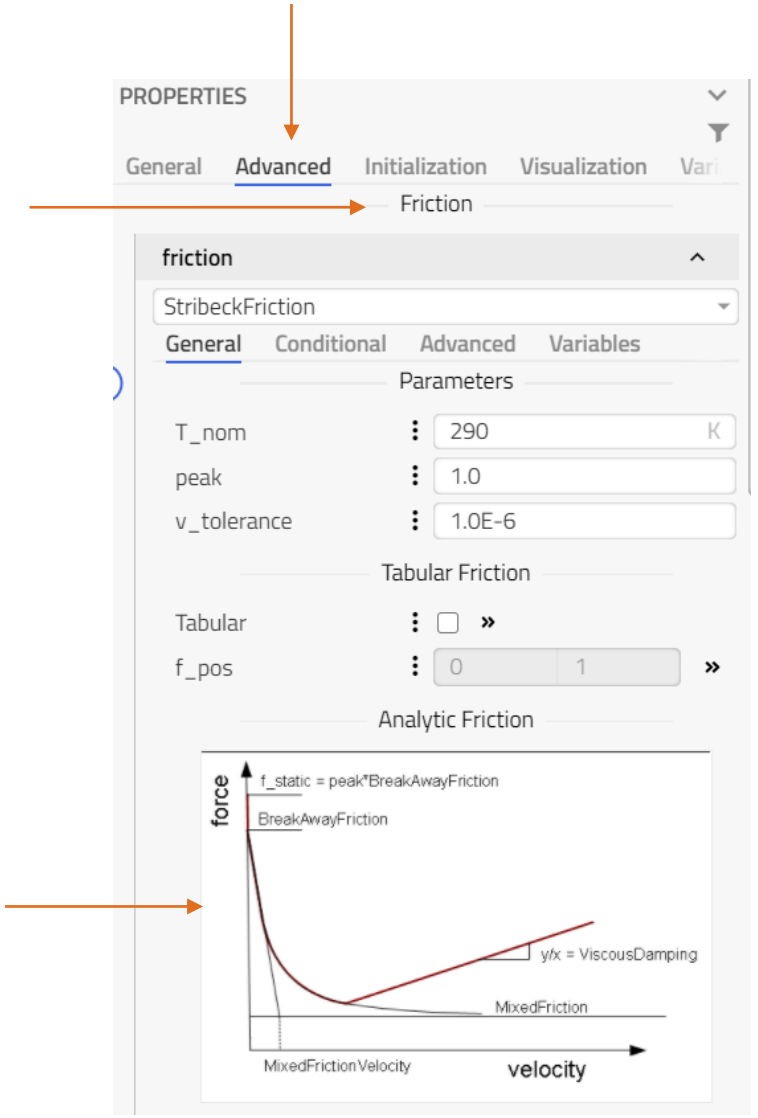
SOME VARIABLE ANNOTATIONS

SOME VARIABLE ANNOTATIONS

- Evaluate
  - Syntax: `parameter Real myParam annotation(Evaluate=true);`
  - Set to true, the parameter will be substituted by its value. Changing its value require model recompilation.
  - Set to false, the parameter will remain parametrizable after compilation – as far as possible (i.e. depending on dependencies). Changing its value should not require recompilation but re-initialization of the model.
- HideResult
  - Syntax: `Real myVar annotation(HideResult=true);`
  - Applicable to any instance (class, model, record, type etc.)
  - Set to true, the values will not be accessible for plotting (indeed, not stored at all).

ORGANIZE PARAMETER DIALOG

- annotation(Dialog(
  enable = true, → can be Boolean condition based
  tab = "Advanced",
  group = "Friction",
  showStartAttribute = false,
  colorSelector = false,
  groupImage="modelica://MyPath/image.png",
  connectorSizing = false));

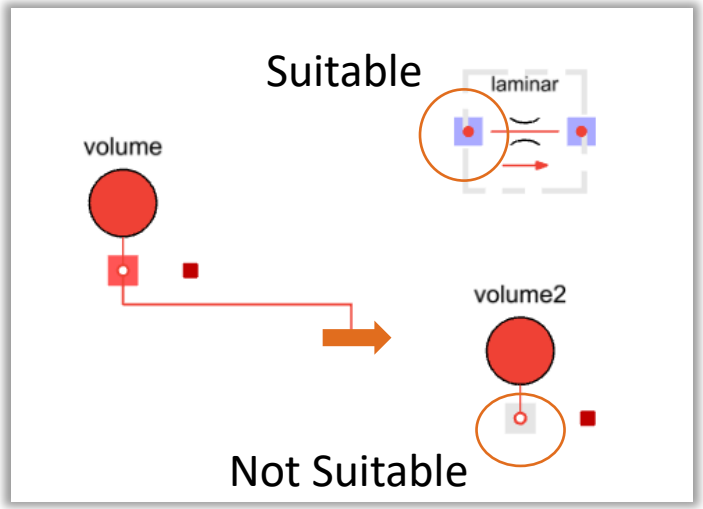


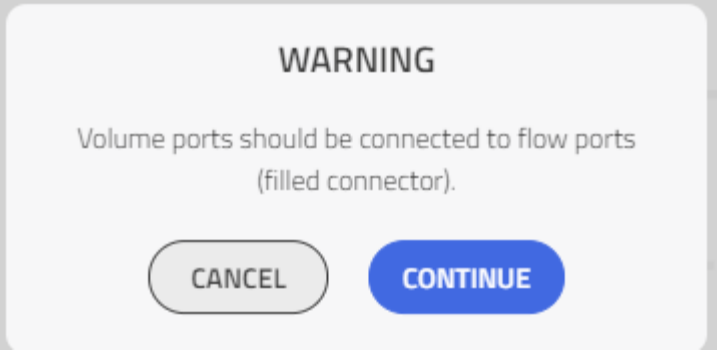
CONNECTOR ANNOTATIONS

ANNOTATION ON CONNECTORS

- Enable to specify which connectors should not connect together even if they have the same variables inside

```
connector VolumePort "Volume port"
  extends Hydraulics.Interfaces.Port;
  annotation (
    __Modelon(ConnectionRestrictions(invalidConnectionWarning="Volume ports should be connected to flow ports (filled connector).",
          canConnectTo={Hydraulics.Interfaces.FlowPort,Hydraulics.Interfaces.NeutralPort})),
```



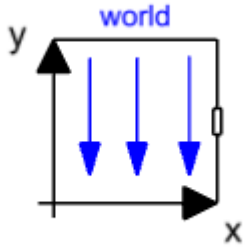


SOME MODEL ANNOTATIONS

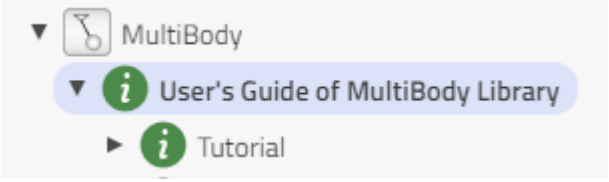
SOME MODEL ANNOTATIONS

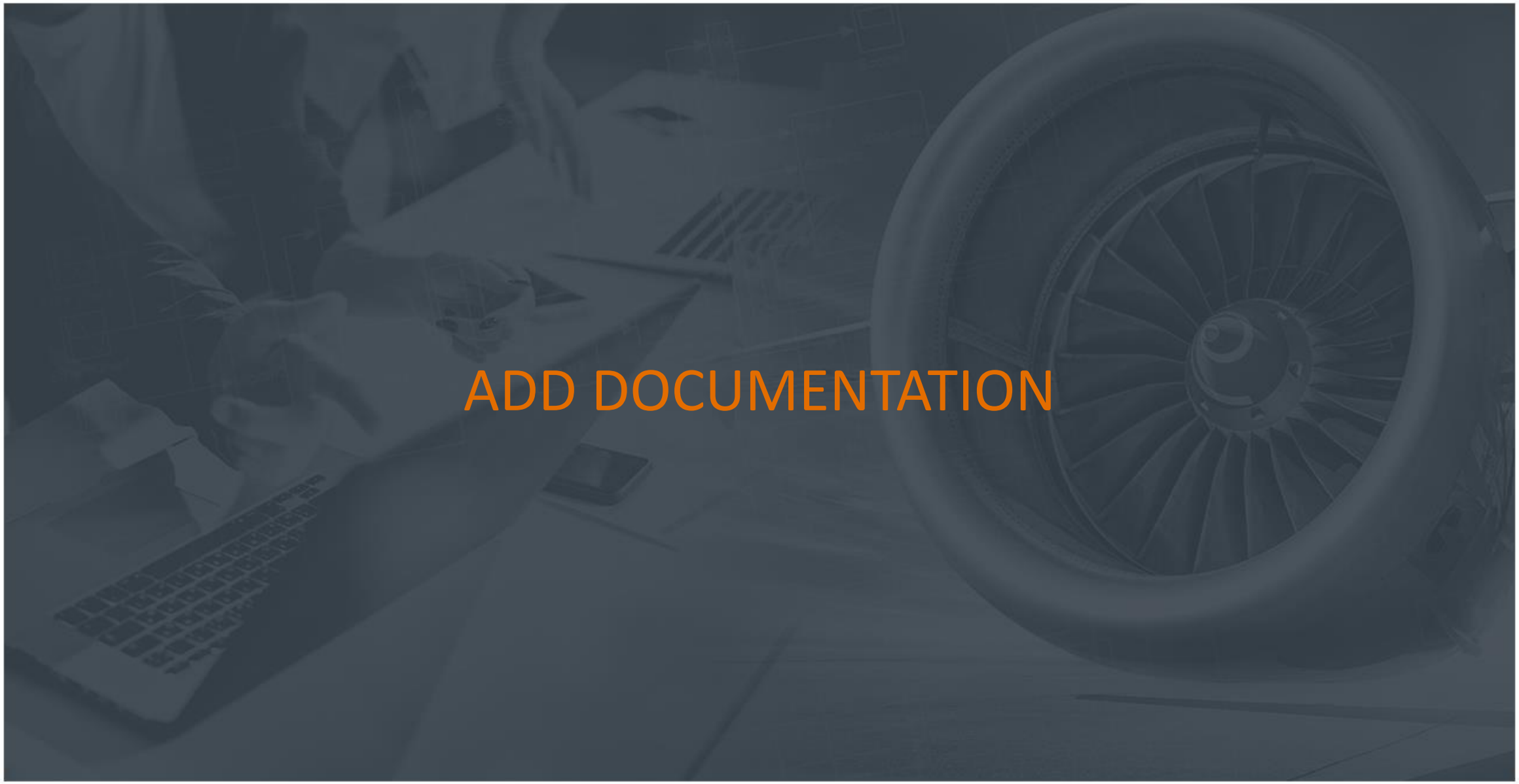
- defaultComponentName
- defaultComponentPrefixes
- missingInnerMessage

```
annotation (
  defaultComponentName="world",
  defaultComponentPrefixes="inner",
  missingInnerMessage="No \"world\" component is defined. A default world
component with the default gravity field will be used
(g=9.81 in negative y-axis). If this is not desired,
drag Modelica.Mechanics.MultiBody.World into the top level of your model.",
```



- DocumentationClass
  - Synthax: `annotation(DocumentationClass=true);`
  - States that this class and encapsulated ones are only containing documentation
  - Set `preferredView="info"`
- preferredView
  - Can be set to info, diagram or text
  - Define which view opens in priority (text not yet supported)

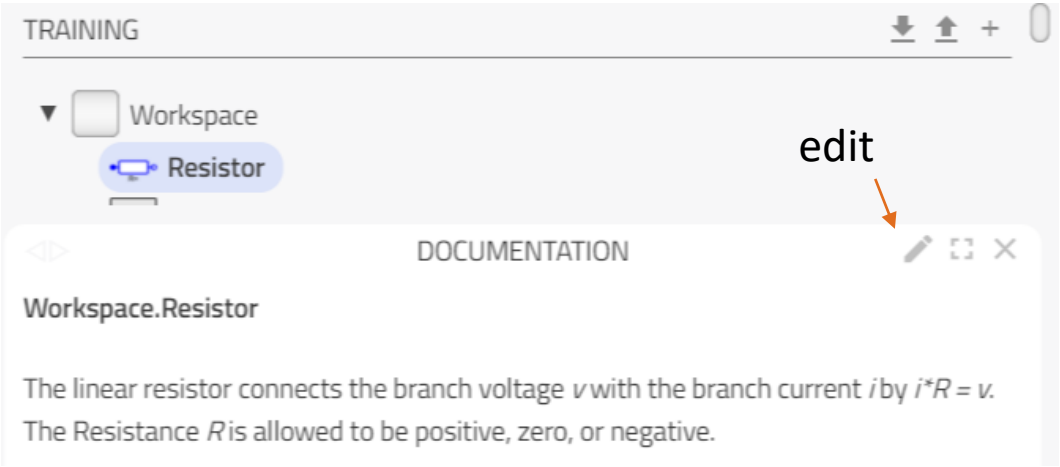


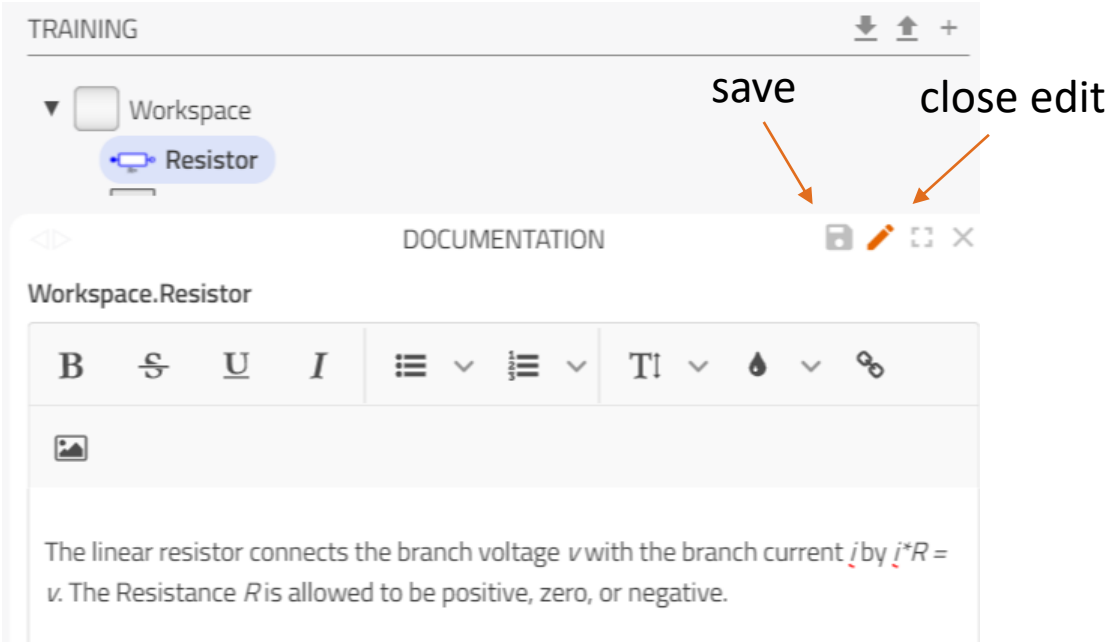


ADD DOCUMENTATION

ADD DOCUMENTATION

- Show Documentation > Edit pen
- Possible to write in Microsoft Word and copy paste (preserve style and get correction)
- Activate edit, Write, Save, De-activate edition

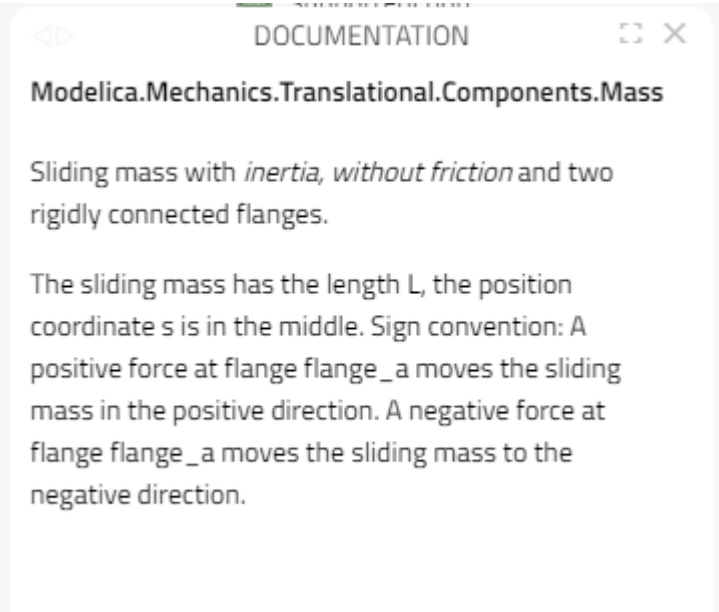




DOCUMENTATION

- Stored as HTML code inside the component

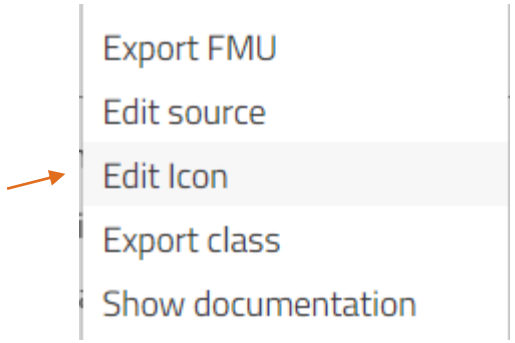

```
      annotation (
        Documentation(info="<html>
<p>
Sliding mass with <em>inertia, without friction</em> and two rigidly connected flanges.
</p>
<p>
The sliding mass has the length L, the position coordinate s is in the middle.
Sign convention: A positive force at flange flange_a moves the sliding mass in the positive direction.
A negative force at flange flange_a moves the sliding mass to the negative direction.
</p>

</html>"),
```

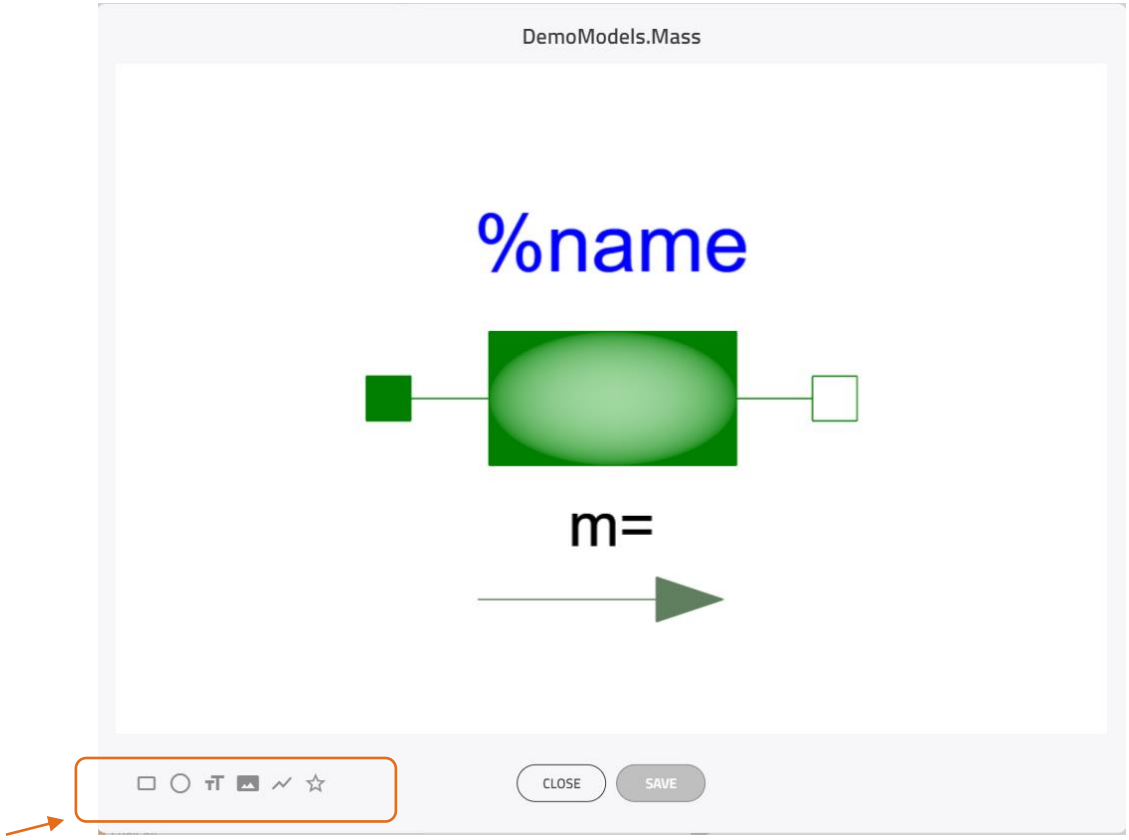


DRAW ICON

DRAW ICON

- “Edit Icon” from context menu
- Draw shapes, add text, add image from resource folder
- Edit colors
- Etc.





ICON PRIMITIVES

- Stored in model annotation Icon()
- Example:

```
Rectangle(
        extent={{-55,-30},{56,30}},
        fillPattern=FillPattern.Sphere,
        fillColor={160,215,160},
        lineColor={0,127,0})
```

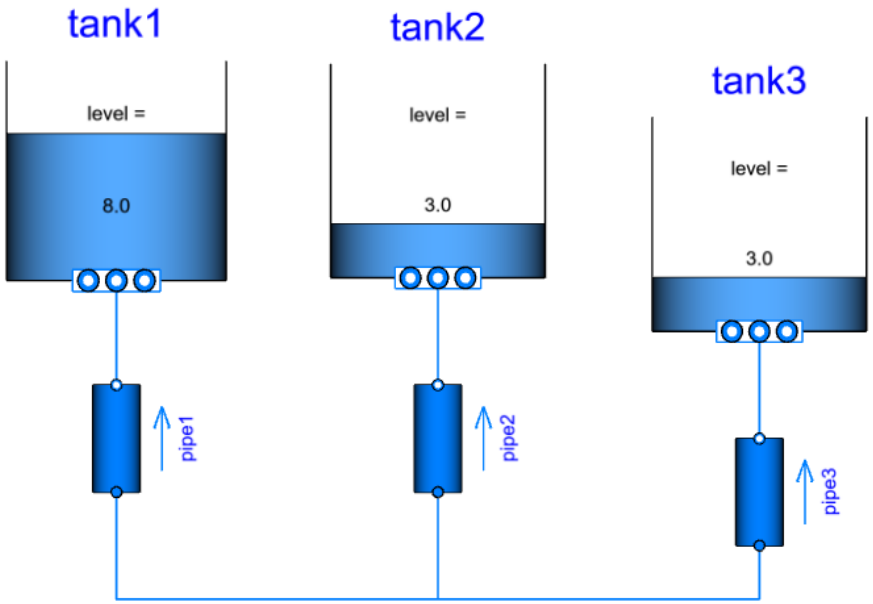
```
Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{
        100,100}}), graphics={
    Line(points={{-100,0},{100,0}}, color={0,127,0}),
    Rectangle(
      extent={{-55,-30},{56,30}},
      fillPattern=FillPattern.Sphere,
      fillColor={160,215,160},
      lineColor={0,127,0}),
    Polygon(
      points={{50,-90},{20,-80},{20,-100},{50,-90}},
      lineColor={95,127,95},
      fillColor={95,127,95},
      fillPattern=FillPattern.Solid),
    Line(points={{-60,-90},{20,-90}}, color={95,127,95}),
    Text(
      extent={{-150,85},{150,45}},
      textString="%name",
      lineColor={0,0,255},
      fillColor={110,210,110},
      fillPattern=FillPattern.Solid),
    Text(
      extent={{-150,-45},{150,-75}},
      textString="m=%m",
      fillColor={110,221,110},
      fillPattern=FillPattern.Solid,
      fontSize=0)}),
```

ANIMATE ICONS

- Its possible to animate icons using simulation result data
- DynamicSelect(DefaultValue , CalculatedValue)

- Example Tank:

```
Rectangle(
  extent=DynamicSelect({{-100,-100},{100,10}}, {{-100,-100},{100,(-100
      + 200*level/height)}}),
```



FUNCTIONS AND ANNOTATIONS

FUNCTIONS

- Functions contain algorithm sections
- Algorithms are not as simple to manipulate symbolically
- There are specific annotations to help the tool such as:
  - smoothOrder()
  - derivative()
  - inverse()
  - inline()

FUNCTIONS-DEFINITION

- Order inputs and outputs separate for readability
- Any input can have a default value defined in the function
- Internal variables are defined in the `protected` section

```
function polyCube
  input Real x;
  input Real c0=2;
  input Real c1=3;
  input Real c2=1;
  input Real c3=1;
  output Real y;
protected

algorithm
  y:= c0 + c1*x + c2*x^2 + c3*x^3;
end polyCube;
```

FUNCTIONS-DEFINITION

```
function polyCube
  input Real x;
  input Real c0=2;
  input Real c1=3;
  input Real c2=4;
  input Real c3=5;
  output Real y;
algorithm
  y:= c0 + c1*x + c2*x^2 + c3*x^3;
end polyCube;
```

- The following calls are equivalent

```
y=polyCube(1,2,3,4,5)
y=polyCube(x=1,c0=2,c1=3,c2=4,c3=5)
y=polyCube(1,2,3,c2=4,c3=5)
y=polyCube(1,c2=4)
```

- Positional arguments goes first

FUNCTIONS-DEFINITION

```
function f
  input Real x1;
  input Real x2;
  input Real x3;
  output Real y1;
  output Real y2;
  output Real y3;
algorithm

end f;
```

- Handling multiple returns:

```
(y1,y2,y3)=f(x1,x2,x3)
```

- Catching parts of the function return: (note the ordering)

```
y1=f(x1,x2,x3)
(y1,y2)=f(x1,x2,x3)
```

FUNCTION - DERIVATIVE

```
function polyCube
  input Real x;
  input Real c0=2;
  input Real c1=3;
  input Real c2=4;
  input Real c3=5;
  output Real y;
algorithm
  y:= c0 + c1*x + c2*x^2 + c3*x^3;
annotation(smoothOrder=N);
end polyCube;
```

- Modelica tools cannot differentiate a generic function (except in simple cases)
- Using the annotation `smoothOrder=N` tells the complier that the function is N times differentiable

FUNCTION - DERIVATIVE

- Specifying the derivative function explicitly
- User is responsible to make sure the derivative function is correct!
- For each Real input add a derivative input in derivative function:

```
function polyCube
  input Real x;
  input Real c0;
  input Real c1;
  input Real c2;
  input Real c3;
  output Real y;
algorithm
  y:= c0 + c1*x + c2*x^2 + c3*x^3;
annotation(
derivative(order=1)=der_polyCube);
end polyCube;
```

```
function der_polyCube
  input Real x;
  input Real c0;
  input Real c1;
  input Real c2;
  input Real c3;
  input Real der_x;
  input Real der_c0;
  input Real der_c1;
  input Real der_c2;
  input Real der_c3;
  output Real der_y;
algorithm

end der_polyCube;
```

`order` specifies the derivative order:

`order=1` f’(x)

`order=2` f’’(x)

FUNCTION -DERIVATIVE

- What if some of the inputs are parameters
  - Use `noDerivative (zeroDerivative` treated in the same way)
  - Possible to define multiple derivative functions

```
function polyCube
  input Real x;
  input Real c0;
  input Real c1;
  input Real c2;
  input Real c3;
  output Real y;
algorithm
  y:= c0 + c1*x + c2*x^2 + c3*x^3;
annotation(
derivative(order=1,
noDerivative=c0,
noDerivative=c1,
noDerivative=c2,
noDerivative=c3)=der_polyCube2,
derivative=der_polyCube);
end polyCube;
```

```
function der_polyCube2
  input Real x;
  input Real c0;
  input Real c1;
  input Real c2;
  input Real c3;
  input Real der_x;
  output Real der_y;
algorithm

end der_polyCube2;
```

DEFINING INVERSES

- User can define inverse functions
- Can define multiple inverses (one for each input to original function)

- Example from media library functions:

```
function h_pTX
input Real p "pressure";
input Real T "temperature";
input Real X[:] "mass fractions";
output Real h "specific enthalpy";
algorithm

annotation(inverse(T=T_phX(p,h,X)));
end h_pTX;
```

```
function T_phX
input Real p "pressure";
input Real h "specific enthalpy";
input Real X[:] "mass fractions";
output Real T "temperature";
algorithm

end T_phX;
```

CODE GENERATION ANNOTATION

- It is possible to include inline function calls directly into the model wherever it’s called.
- This enables Impact to do further symbolic manipulations
- Improves simulation speed in many cases:
  - inline = true

WORKSHOP

- In this workshop you will:
  - Create an icon to a solar collector
  - Document the solar collector