

Prepare for the Modelica Crash Course!

Filip Jorissen, Damien Picard, Jelger Jansen, Iago Cupeiro Figueroa

KU Leuven – September 13, 2022

This guide will help you to install Dymola and already get the hang of Modelica, such that you can profit maximally from the crash course.

Installation

1. First of all, install Dymola (commercial software). Download the trial version on [this page](#). You will be able to use a license from our license pool during the crash course.
2. After installation you **have** to install a C compiler, otherwise you cannot simulate any model. We can suggest to use [Visual Studio 2019](#), but other compilers are possible as well. For detailed instructions see <http://www.Dymola.com/compiler>.
3. Do not forget to select your C compiler in **Simulation>Set Up>Compiler**. In order to access this menu item, open Dymola and change to the Simulation tab (tab bar at the top).
4. Test your installation by running a demo (e.g. open **File>Demos>Robot**, then click **Simulation>Simulate** and wait till you see a graph). More detailed instructions can be found [here](#).

Optional preparation material

To get the most out of the crash course, we suggest to come prepared by installing Dymola (which occasionally takes some time) and making a preparatory homework exercise, which can be found below.

We also suggest to read the following chapters of the open-access book [Modelica by Example](#) by M. Tiller carefully.

1. [Basic equations](#): General introduction to the Modelica language, illustration of the model structure, basic concepts such as derivative, initialization, parameter, variable and type.
2. [Polynomial Evaluation](#): Example of [function definition](#), protected variable and time.
3. [One-Dimensional Heat Transfer](#): Introduction to arrays and loop in Modelica.
4. [Connectors](#): Component-oriented modeling, classes of variables.

Additionally, we strongly recommend to read Chapter 1 “What is Dymola?” (p. 9-20) of the Dymola User Manual. This chapter gives more information on the features of Dymola and Modelica and explains the Dymola environment (graphical user interface, model editor, parameters change, simulation, ...). The Dymola User Manual can be found in your installation folder of Dymola: `C:\Program Files (x86)\Dymola xx\Documentation\Dymola User Manual 1A`, where `xx` stands for the version you installed.

For further background, the following sections from “Modelica by Example” by M. Tiller are interesting. However, it is not required to read before the course:

1. [Discrete Behavior](#): State event handling and hysteresis.
2. [Vectors and Arrays](#): Array declarations and array construction
3. [Components](#): Syntax, arguments and examples (e.g. heat transfer and electrical components).
4. [Architecture](#): Configuration management.

Finally, the [Modelica Language Specification](#) is a complete description of the Modelica language.

Homework exercise

Consider the electrical circuit shown in Fig. 1. The circuit has a sinusoidal source, two resistors, a diode, a capacitor and the ground. The purpose of this exercise is to learn some basic Modelica concepts by creating the model of this electrical circuit from scratch. You will learn how to create connectors, partial models and models and how to re-use your code.

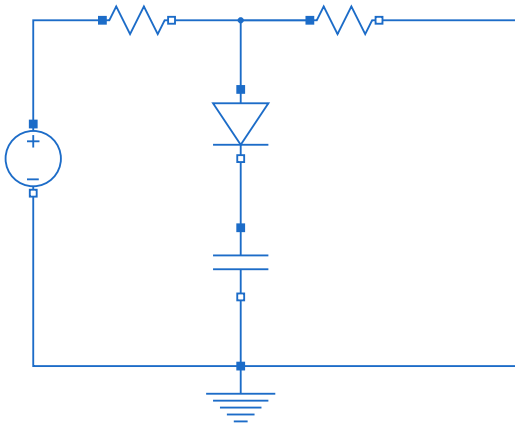


Figure 1: Electrical circuit

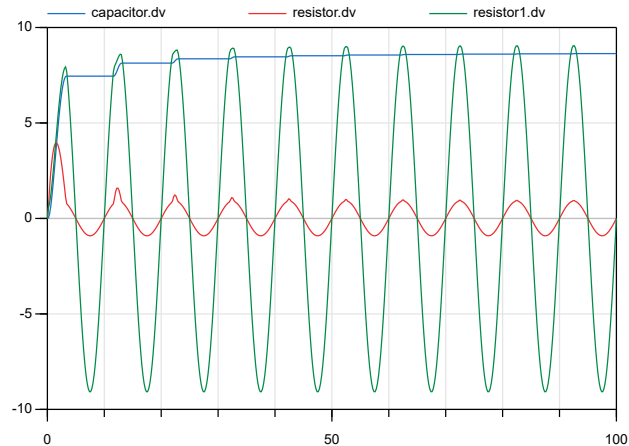


Figure 2: Result of simulation

Some tips:

- Make sure you end statements in Modelica code with a semicolon (;). With some exceptions (e.g., **(partial) model**, **equation** and **algorithm** lines), this is always necessary.
- You can check your model using the **check** button in the toolbar. This will usually point you to possible mistakes you made, so make sure to read error and warning messages carefully. Although Modelica has a bad reputation for debugging, its syntax checker is not bad.
- Dymola has a nifty autocompletion function, which helps you reduce how much of the code you have to type yourself. Just start typing something and press **Ctrl+space** to open a context menu with some suggestions.

Below you can find a more detailed procedure in order to get a simulation model of the electrical circuit. Summarised, you will start by making a new library (package). In that package you will define an electrical connector in which the current and voltage are declared as variable. Then, you will create a positive and negative pin by extending the general electrical connector. Subsequently, the positive and negative connector can be used to define a partial model that describes any electrical component with two connectors. This partial model can be extended in a next step to create customised electrical components (resistor, capacitor, ...). Once you have made these models, you can build and simulate the electrical circuit shown in Fig. 1.

1. After opening Dymola, create a new *package* called *MyLib* (through **File>New>Package**). Don't forget to provide an appropriate description so you know what your library is for when you don't touch it for a longer time. It is useful to deselect "Save contents of package in one file" if you are going to use version control (git).

All component models you will define in the following step need to be placed in this folder.

2. Each component uses electrical connectors. Create a **partial connector** (right-click *MyLib*, then **New>Connector**). Make sure to select the option "Partial" in the dialog that pops up. Call the **partial connector** *Pin*. This connector will need two variables: *i* (current) and *v* (voltage). You can add variables while you are in the *text editor* of Dymola (**Text>Modelica Text**). If you want to know how to create variables in the text editor, you can review [this page](#).

Which one is a *flow* variable and which one is a *potential* variable, and how should this be reflected in your code? Check [this link](#) for a hint.

3. Create the **connectors** *Pin_a* and *Pin_b* for the positive and negative pins. Use **extends** to inherit the variables *i* and *v* from the **partial connector** you have just defined. An easy way to create extending models is by right-clicking on the model you want to extend from, and click **New>Extend from**.

Make two different graphical illustrations for the pins. To do so, select the *icon context* from the toolbar (**Graphics>Icon**). For example, you can draw a square in the available canvas and choose a different color for each of the pins. A good convention is to use the same shape for pins of the same flow type.

4. Create a **partial model** *OnePin* which contains the variables *i* (current) and *v* (voltage) and one connector *Pin_a*. To add the connector, drag *Pin_a* from your library folder to the modelling window while you are in the *diagram context* (**Graphics>Diagram**). The pin should be located on the top border of the white canvas. This model will be the basis for all models which have only one pin (i.e. the ground node).
5. Create variables *i* and *v* in *OnePin* and write **equations** to link these variables to those you have defined in the *Pin* you use in the component. In order to access variables, you can use dot notation as follows: **componentName.variable**.

Your code should look something like this:

```
partial model OnePin "Model with a single pin"
  flow Modelica.Units.SI.Current i "Current flowing through this component"
  ;
  Modelica.Units.SI.Voltage v "Voltage of this component";

  Pin_a pin annotation...;
equation
  pin.i = i;
  pin.v = v;
  annotation...;
end OnePin;
```

Notice that the model *OnePin* contains an instance of *Pin_a* named **pin**. In the **equation** section, you can see that the current through this **pin** is accessed with the dot-notation and made equal to the current of the *OnePin*. The same is done for the voltages.

However, it is not possible to access a variable on something that is not an instance of a Modelica model. For instance, *OnePin.v* would make no sense.

6. Now create a **partial model** *TwoPin* with the same variables and two *Pin* connectors, one positive pin on the left and one negative on the right. Add two variables: voltage difference between the two pins *dv* and current *i*.

While still in **partial model** *TwoPin*, add equations that link the variables in this model with the variables of the pins. When connecting currents, use the convention that when the current flows into the component from the pin, it is positive and vice versa.

7. Create the models for the source, the resistor, the diode and the capacity by extending *OnePin* or *TwoPin*. Make use of the following equations:

(a) Resistor: $Ri = v$ ¹

(b) Capacity: $C \frac{dv}{dt} = i$

(c) Source: $v = V \sin(2\pi ft + \omega) + \beta$

(d) Diode: if $\frac{v}{V_t} \geq \alpha$: $i = I_{ds} \left(\exp \left(\alpha \left(1 + \frac{v}{V_t} - \alpha \right) \right) - 1 \right) + \frac{v}{R}$, else $i = I_{ds}(e^\alpha - 1) + \frac{v}{R}$

(e) Ground: $v = 0$

Define the constants *R*, *C*, *V*, ... as **parameter**, such that you can change their value using the parameter dialog in the *Diagram* context.

To review how to define the time derivative of a variable, revise [this link](#).

¹*v* in this equation is equivalent to variable *dv* in the Modelica model.

8. Create a model called *Circuit* in which you include all these components as shown in Fig. 1.

You are very welcome to experiment with model parameters, but to verify your solution against Fig. 2, use following parameters:

- Voltage source: $f = 0.1 \text{ Hz}$, $V = 10 \text{ V}$
- Left resistance: $R = 1 \Omega$
- Right resistance: $R = 10 \Omega$
- Diode: $I_{ds} = 1.e-6$, $V_t = 0.04$, $\alpha = 1$, $R = 1.e8$.
- Capacitor: $C = 1 \text{ F}$, $V_0 = 0 \text{ V}$ (start voltage)

9. Simulate the circuit for 100 seconds.

Optional exercise

This assignment aims at testing your comprehension of the basic Modelica concepts learned during the above mentioned reading by setting a model of a building and simulating its thermal behaviour. **This assignment will not be treated during the Crash Course.** However, you may find it a useful exercise to get more acquainted with Modelica and Dymola.

Let us consider a simplified building as represented in Fig. 3. The building consists of a single room called *zone*, walls, a floor (foundation), a roof and a window. The building thermally interacts with the environment: ambient air ($TAmb$), the ground, and the sun ($QSol$). The building foundation is approximated by a thick concrete layer called *slab* separating the zone from the ground.

On the right-hand side of the figure, a thermal model of the building is proposed using the resistance-capacitance approach. The heat transfer through the building envelope is approximated by a 1D heat flow (conduction) through equivalent thermal resistances. The thermal storage in the building envelope, due to its thermal inertia, is approximated by equivalent thermal capacities. $CZone$ represents the thermal capacity of the zone (consisting of the internal walls, the furnitures, and a part of the external wall). The $CSlab$ represents the thermal capacity of the slab. Finally, the ground is discretized in $n = 5$ layers, each of them having an identical heat capacity $CGro[i]$. Through the window, the sun heats up the room with a thermal power $QSol$.

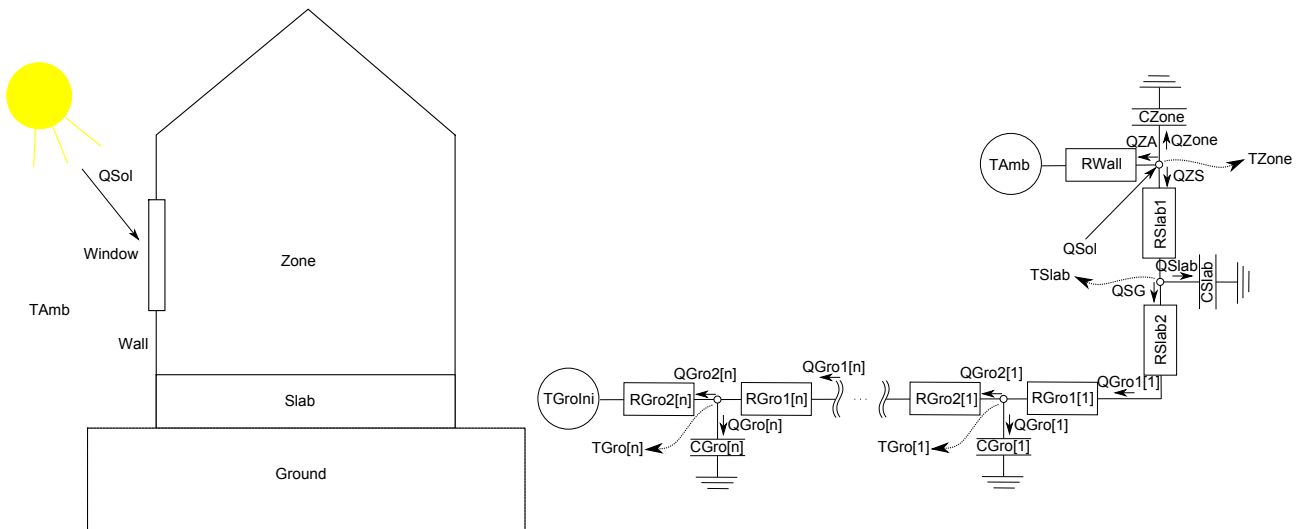


Figure 3: Building model.

The values of the parameters are given in Table 1. Using the electrical analogy, the governing equations of the system are the following:

Thermal resistance: $T_1 - T_2 = R Q_{1 \rightarrow 2}$ with R the thermal resistance between node 1 and 2 and $Q_{1 \rightarrow 2}$ the heat flow, positive defined from 1 to 2.

Thermal capacity: $C \frac{dT}{dt} = Q$ with C the thermal capacity and Q the heat flow, positive defined flowing to the capacity.

Conservation of energy (Kirchhoff): $\sum Q_i = 0$ or the sum of the heat flows through one node is zero.

Thermal resistances [K/W]	<u>RWall</u>	<u>RSlab1</u>	<u>RSlab2</u>	<u>RGro1[i]</u>	<u>RGro2[i]</u>
	0.00806	0.016	0.016	0.033	0.033
Thermal capacities [J/K]	<u>CZone</u>	<u>CSlab</u>	<u>CGro[i]</u>		
	$2.4096 * 10^8$	$3.36 * 10^8$	$2.52 * 10^8$		
Temperatures [K]	<u>TGroIni</u>	<u>TGro[i](start)</u>	<u>TSlab(start)</u>	<u>TZone(start)</u>	
	283.15	283.15	293.15	293.15	

Table 1: Building parameters.

Questions

1. Apply what you have learned during the reading and the homework exercise by setting up a model for the building using the above mentioned equations. What is the zone temperature after a year under these conditions?

Approximate the ambient temperature by a sine using following code:

```
TAmb = 10*cos(2 * Modelica.Constants.pi * time * 3 * 10^(-8)) + 276.15;

QSol = floor( cos(2 * Modelica.Constants.pi * time / 86400) + 1) * 5000 *
        cos(2 * Modelica.Constants.pi * time / 86400);
```

2. *Optional:* Try to obtain the same results using the components of the Modelica Standard Library (MSL) instead of writing the equations yourself. This library is automatically loaded in Dymola and can be found on the left-hand side of the Dymola window. For the thermal components, look in the MSL at Modelica.Thermal.HeatTransfer.Components.

Good luck!

Further reading

- M. Tiller, *Modelica by Example*: <https://mbe.modelica.university/>
- M. Wetter and T. Noudui, *Introduction to Modelica*: <http://simulationresearch.lbl.gov/modelica/downloads/workshops/2015-06-22-lbnl/slides/modelica-intro.pdf>