**DRAFT – 17/01/19**

**Not finished, but it may help**

Keywords: repetitive actions, script, *Modelica*, *Dymola*.

► What is a script for *Dymola*?

It is just text, a piece of code (in this case Modelica code with some *Dymola* functions) that can be executed in a command prompt (in *Dymola*, the window called “Commands”). Scripts can be also stored in text files with extension *.mos*. The written code in the file will be executed every time it is opened by the program (*Dymola*). This files can be created in any text editor and also with the script editor included in *Dymola*.

► Is this guide for me?

This guide is indicated for users who need to do repetitive actions in *Dymola*. All “basic operations” (see next point) can be considered repetitive actions, but also something like running a simulation ten times changing a single value and plot just one parameter of each result. Can you do it by hand? Yes, but you can also do it with few lines of code. Sometimes, the repetitive actions can take a lot of time for a human being (Export 100 files, simulate 1000 models, perform simulations of the same model changing 6 parameters with 5 variations each -56 simulations hehe-, etc.).

►For? If? What?!

The explanations in this guide are for people who already know how a “*for*” loop works (and also their friends “*if*”, “*while*”, etc.). The objective of the guide is to show you how to do some actions by giving some examples of the code. So, if you are not familiar with this terms… Take a programming course! Maybe *Python*, this snake may appear in the future…

Of course, this guide is for people who already worked with *Dymola* (basic level). Do you need to take a course?

► What can I learn from this guide?

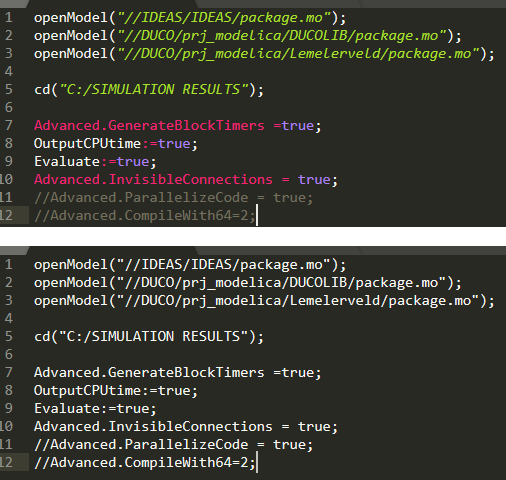
* Basic operations:
  + Open models and libraries.
  + Set standard and advanced parameters in Dymola.
  + Change the *Working* *Directory*.
* Force parameters to be changed AFTER translation of the model (*propagate*, *start* value)
* Perform many simulations:
  + The same model – change one or more parameters.
  + Different models – change zero or more parameters.
* Plot data after a simulation.
* Export selected data after a simulation in CSV files.
* Export data in .mat files. pending
* Export graphs. pending
* All previous points in one script. Yes, lazy human, you want it. Not all, pending graphs
* *Startup*.*mos* pending

► Before start. Recommendations.

For scripting you just need a text editor. You can use *Notepad* or even *Microsoft* *Word* (seriously?). If you plan to make some scripts I recommend you to install any of the following text editors:

* [*Sublime Text*](https://www.sublimetext.com/3): Fast, powerful and beautiful.
* [*Visual Studio Code*](https://code.visualstudio.com): Super powerful, beautiful but a bit slower than *Sublime* *Text* on the first run. It is not for everyone, but if you plan to code regularly and also do it in different programming languages you should try it.
* [*Atom*](https://atom.io): Similar to *Sublime* *Text*, but 100% Open Source.
* [*Notepad++*](https://notepad-plus-plus.org/download): It is not very beautiful, but it is fast, free and powerful.

All these editors have the option to highlight the code and they also include somehow intelligent code completion –not all in the same way- (In *Visual* *Studio* it is called *IntelliSense*). For *Modelica* it is necessary to download an external package for each text editor.

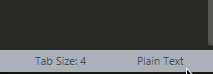


\*Highlighted text vs plain text (Sublime Text)\*

For highlighting in *Sublime* *Text*:

* 1. Open *Sublime* *Text*.
  2. Open *Sublime's* command palette (Ctrl+Shift+P).
  3. Enter: *Package Control: Install Package.*
  4. Enter: *Modelica.*

To select this specific highlight just click on the bottom-right corner of the window (on the text “Plain text” and select *Modelica* in the menu.

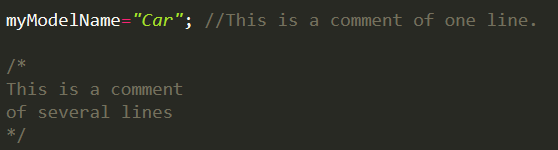


Use the Internet to find out how to install these highlights in the other applications.

► Scripting - Basics

* Comments: There are two ways for writing comments in the code. Everything after double slash (//) would be considered as comment in the same line. Comments of several lines start with slash and asterisk (/\*) and finish with asterisk and slash (\*/). Every character in between would be considered as comment.

Examples (comments are the characters in grey):

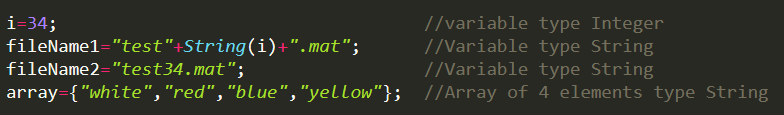


\*Note that the all statements have to finish with a semicolon (;)

except the comments (because they are not functional parts)\*

* Declaration of variables: It is totally different than the code in the model! In the model, you have to declare the variables with detail like the type of data, the size, etc. Here, you just need to give a unique name to every variable you declare and also a value. The type of the data is defined depending on the value.

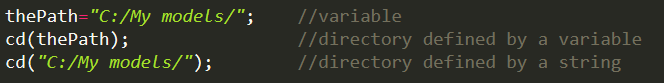
Examples:



\*The operator *+* can be used for summations (numbers) or for concatenation of strings\*

* Function *cd(route):* This function changes the *Working Directory* (WD). That directory is the default destination of simulation results, exported models, etc. By default, the working directory changes every time a model is open. It is important to specify a working directory to avoid problems, especially the ones related to relative paths. *route* can be a string or a variable or a combination of both. *route* can be **absolute (:/)** or **relative (//)** (more convenient for using models in other machines).

Example:



\*Note that routes in Modelica are written always with slash (**/**) NOT with backslash (**\**).

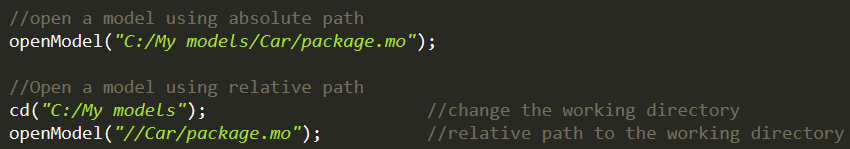
It is possible to use backslash twice (\\ instead of \), but it can create confussions.\*

NEW: In Dymola 2019 FD01 it is possible to use slash (/) and simple backslash (\) with the same results in scripts

Suggestion: When a directory is stored in a variable it might be useful to add the slash (/) at the end of the string. It is not necessary, but it can be convenient if we want to attach other information to the end of the string (e.g. the name of a file).

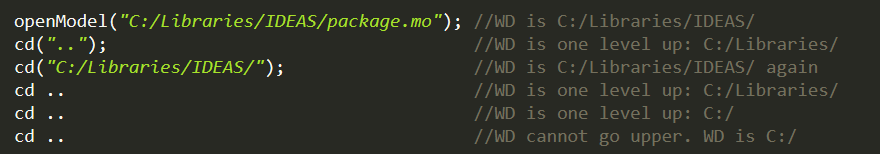
* Function *openModel(route):* It is valid for **models** and **libraries**. We can open several models and/or libraries just specifying the main file of them (**package.mo** in the case of models composed by different files).

Examples:



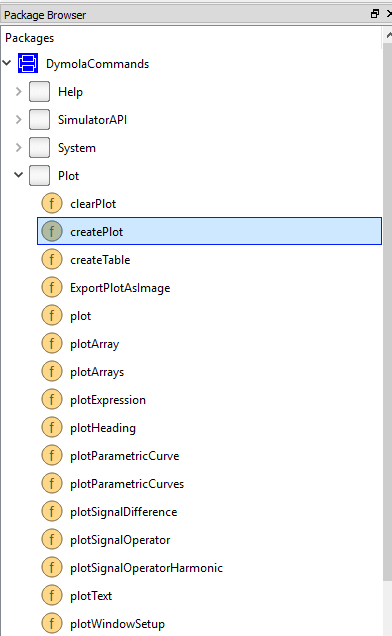
To use relative paths it is highly recommended to specify the working directory.

* Go to the upper directory argument *..*: As in many other systems, the argument “two points” (..) for the [cd command](https://en.wikipedia.org/wiki/Cd_(command)) changes the working directory one level up. This special argument can be set in two different ways. Both are shown in the example. Note that the simplest form does not need a semicolon (;) at the end of the line. It is an exception of the semicolon-end-of-lines rule.



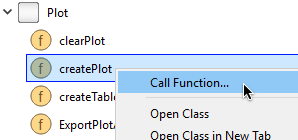
\*cd without arguments will show the working directory\*

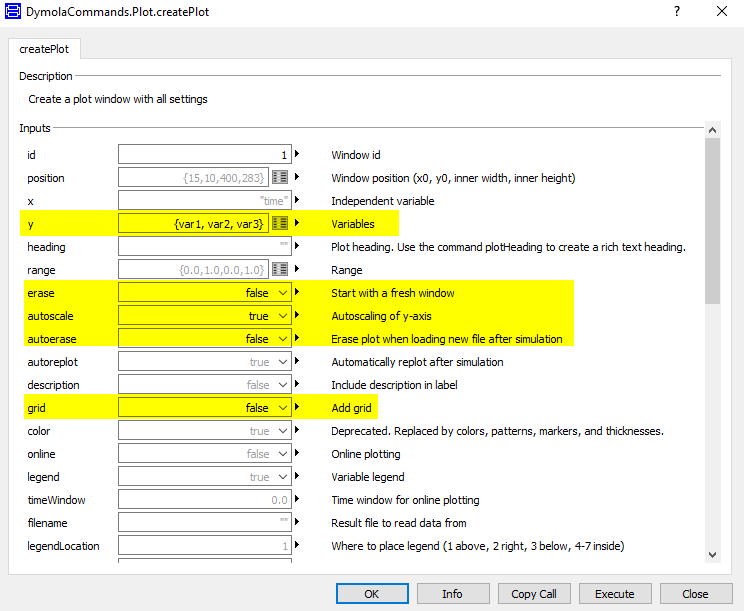
- Other functions: *Dymola* includes many useful function that cannot be explained in this guide. All relevant functions are into *DymolaCommands* Package.

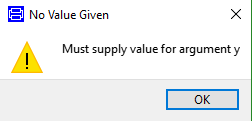


These functions can be called directly in the code just typing the name of the Modelica route and the name of the function and specifying the parameters between brackets. One easy way to use the function properly is to find it in the package browser, then right click on it and select “Call Function…”. A window will appear with all possible parameters for that function. We can fill the parameters we need and click on the button “Copy Call”. Some functions require certain parameters to be filled before enabling the copy of the call, so it is necessary to fill all these parameters and then use the “Copy Call” button. After that, the code of the function is copied in the clipboard and we can paste it in our script. After that we can continue making modifications directly in the code.

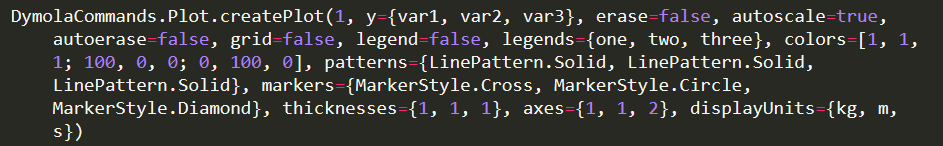
Example with function **createPlot:**





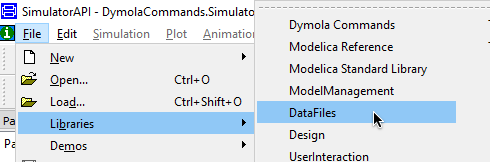


Message asking for a value for the argument *y* that appears after clicking on “Copy Call” button. It is required to set some parameters that are essential for executing a function.



Code obtained after clicking “Copy Call” button. Only filled arguments are present.

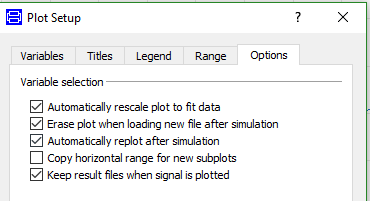
* The function createPlot has many arguments, but it is an exception. Normally, functions have few parameters. This complex function can be generated in another way (explained later).
* Any function can be called, including functions crated by the user.
* Not all arguments are required to use a function.
* Some functions are not loaded by default. To use them and it is necessary to load the packages where they are included. Some packages come with Dymola (e.g. DataFiles package, wich contains the function convertMATtoCSV). To load those packages just click on File-Libraries-[select the desired package to be loaded].



\*The functions included in these build-in libraries can be accessed without loading them graphically, but it is necessary to load them in this way to access the “Call Function…” option\*

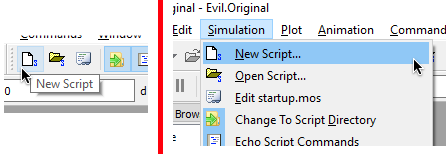
► Scripting - Pots

One of the most tedious work after running simulations is to plot variables in a nice way. Many times we will need to run dozens of simulations of the same model. There are options to update the plotted variables every time we simulate the same model, but this settings are not stored in anywhere by default. We can lose those settings just changing the simulated model, restarting Dymola, changing some parameters in the model or even changing some parameters in the graphs.



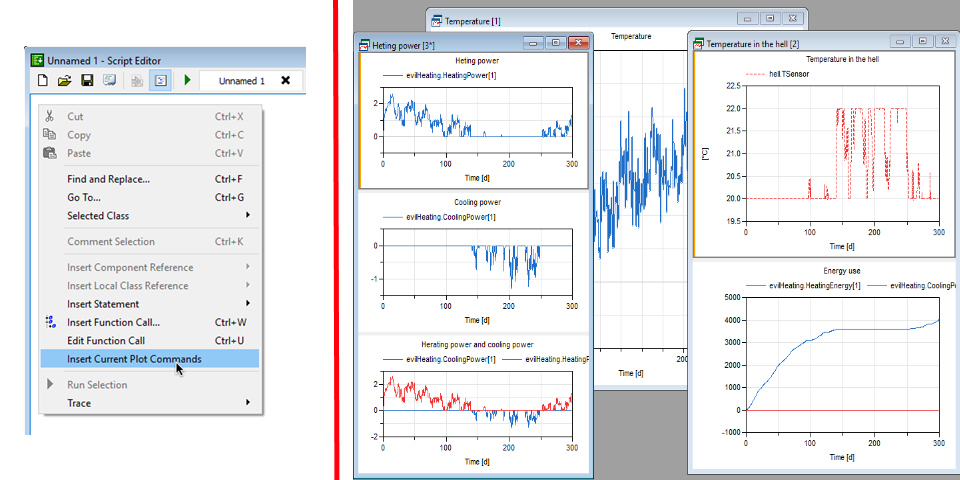
\*Specific options for updating the plots after re-simulations\*

The fastest way to keep those options is just to simulate our target model once and set all the properties in all plots we want to see: variables to plot, scales, colours, line types, position of the plot, titles, subplots, etc. When everything is set as we want, open the Dymola’s script editor.



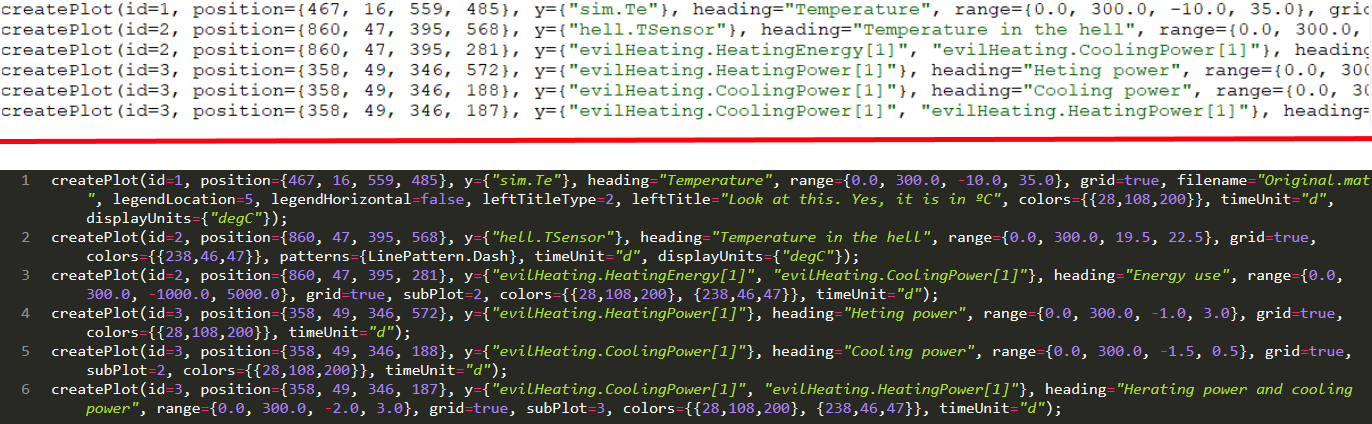
\*Two ways of opening the Script Editor\*

The editor should be totally empty. Right click with on a white space and select “Insert Current Plot commands”



\*Script Editor and desired plots\*

Then, all parameters for representing those plots will appear. We can save that code for future simulations.

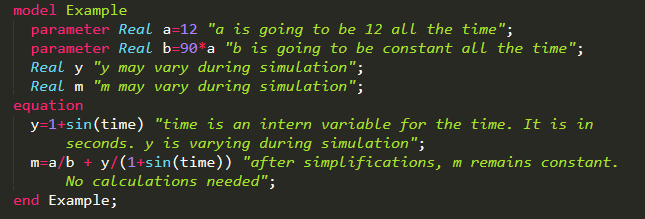


\*Code in Script Editor vs code in Sublime Text\*

The generated code can be modified for other models -changing the variables, of course- and adapted for other purposes. All parameters can be variables. With this guide there is an example of a model with some small scripts. One of them is for plotting with parameters. NOT YET

► Simulate a model multiple times by changing one or more parameters

The Modelica code is only valid to build models, but the real calculations during simulations are made by some solvers that use C code. C doesn’t have anything to do with Modelica and to simulate a model, first it is needed to translate the Modelica code into C code. That translation will transform variables, constants, formulas and also will try to simplify part of the model. After the translation, some parameters that are not changing during the simulation will remain as constants and other parameters will be included into the C code as parameters. The reason is to optimize the code and to reduce the number of equations of the translated model.



\*In this example there are 2 parameters that are constant all the time (a and b), one calculated parameter (y) that varies and another parameter (m) that can be a constant.\*

\*The example is only to show parameters that may not be really parameters. The translator works in a different way\*

When we simulate a model Dymola checks if the a translation is needed. A translation is always needed if:

* The Variable Browser is empty
* The last simulation was of another model
* The last simulation was of the target model but a change was made in the code after the previous translation –e.g. a parameter has been changed-.

The last point is, actually, the same as the second one. The process and the result of the translation can be different even if we change a single parameter in our model.



\**Translate*, *Simulation* and *Stop* buttons. Which one is the most important? The one for stop.\*

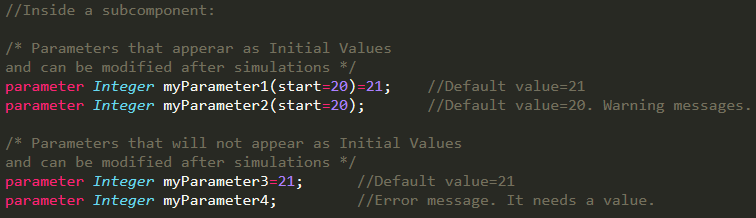
There are functions for translate and for simulate under *DymolaCommands.SimulatorAPI* (Package Browser). There are several that can be used for scripts. Check them!

***“I have a parameter in my model but I cannot change it.”***

My knowledge of the translator is limited and I don’t know all the ways to force the change the value of parameters after the translation process. The following explanation is valid for many models, but it may not work with all.

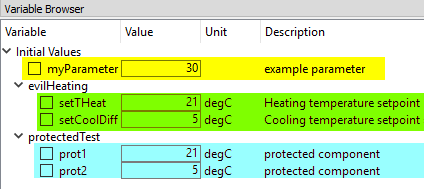
As far as I know, there are two possibilities to force a parameter to be changeable after the translation of a model:

* **Parameters declared in the top level of the model:** Normally, the parameters that were declared in the top level of the model can be changed after the translation. The top level is the package that contains all the subsystems in the model. It is the “thing” you select few seconds before you press the red bouncing ball button.
* **Parameters with start values:** The declaration of the parameter can be in any subcomponent of the model with the attribute *start=a\_value.* If the parameter is inside a component that is *protected*, it cannot be changed after first simulation. It will disappear from the results but, paradoxically it could be changed -The new *Sweep Parameters* function can help. It is explained later-.

****

\*Parameters with and without start values\*

The declared parameters with a start value and the ones declared in the top level appear, after translation, like this:



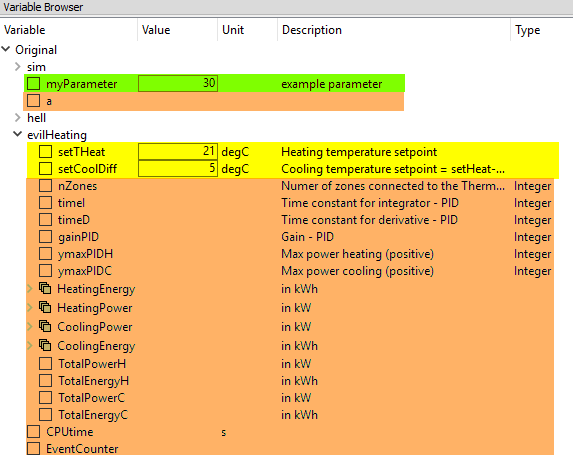
\*Yellow: top level parameter,

green=parameters with start values inside a non-protected component,

blue=parameters with start values inside a protected component\*

After the translation all parameters that can be modified are shown under the label *Initial Values*. They have a value in a text box that can be changed –with appropriate values- before the simulation.

Once the parameters are set, the simulation can be executed. The results include all parameters in the model, but only the ones that appear in Initial Values and also not included inside protected components can be modified:



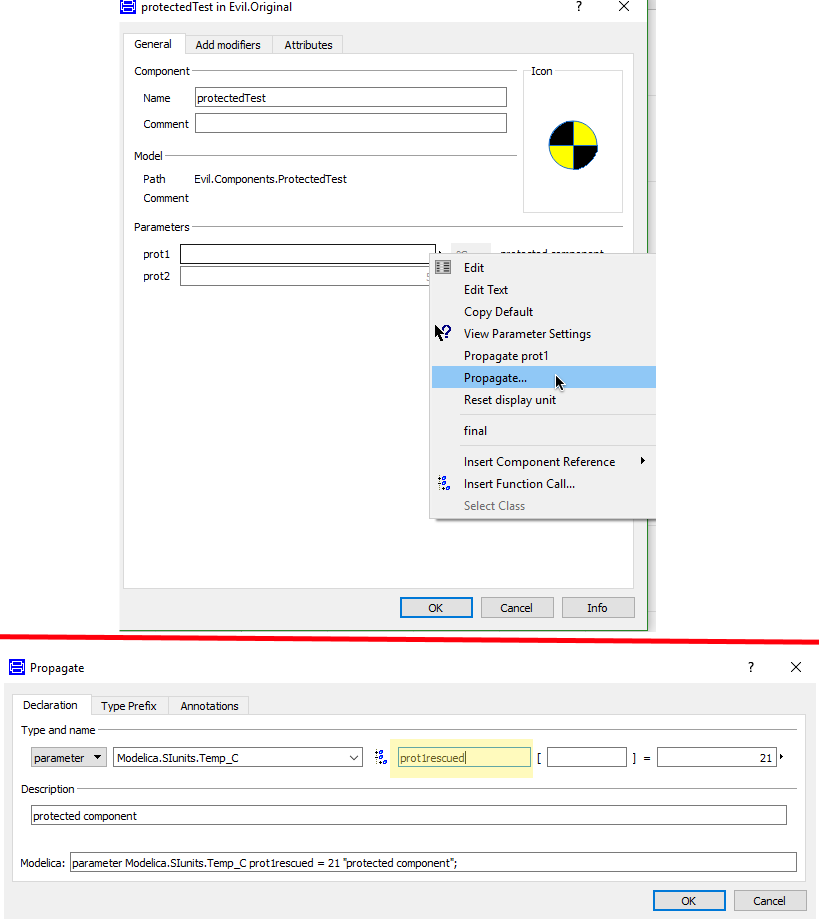
\*In orange, parameters that cannot be changed. The parameters in blue have disappeared\*

***Propagate parameters***

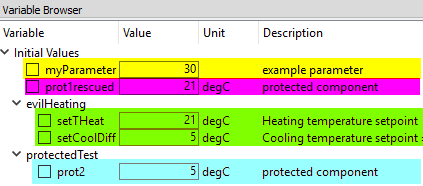
Sometimes it is not possible unprotect a component or there is an inner parameter we want to vary but it we cannot add a start value. We will find this problems with component that are in libraries. A solution could be to create a copy of the problematic component in out model. But that is a mistake because we break the link with the library. It could be possible to directly modify the component in the library, but that is comparable to have sex with your mother or father. It is possible to do it, but it is not an ethic option for the society. With libraries, the same –unless you are a developer of it-. Libraries have to be used as they are given.

To solve this problem it is possible to *propagate* those desired parameters from the inner element to the top level –not sure if it is necessary to go to the top level-. To do that, right click on the problematic component –diagram view- and select *Parameters*. Right click on the desired parameter and select *Propagate…*. A new window will appear. There, it is possible to create a new parameter at the upper level –top if we were at the top level-. Just clicking *OK* a parameter with the same name will be declared in the upper level and the inner parameter will be linked automatically to the new one. The solution is, in fact, to create a new parameter that is linked to the desired one. The name does not need to be the same as the original parameter, but it is highly recommended to keep it.

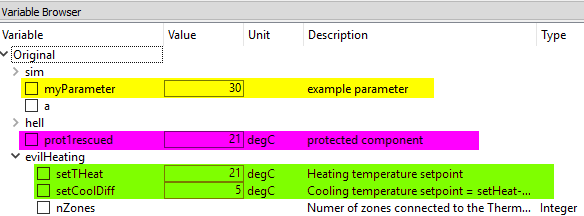
The process of propagation can be done several times to propagate a parameter through many levels. Sometimes there are parameters from libraries that are deep inside protected components. For those situations, propagation can be done on *extended* models –not explained here-.



\*Propagation of the parameter *prot1m* which is inside a protected component\*



\*After translation, *prot1* does not appear. *prot1rescued* will give the value to *prot1*\*



\*After simulation, the parameter *prot2* has disappeared,

but *prot1* can be changed thanks to the propagated one\*

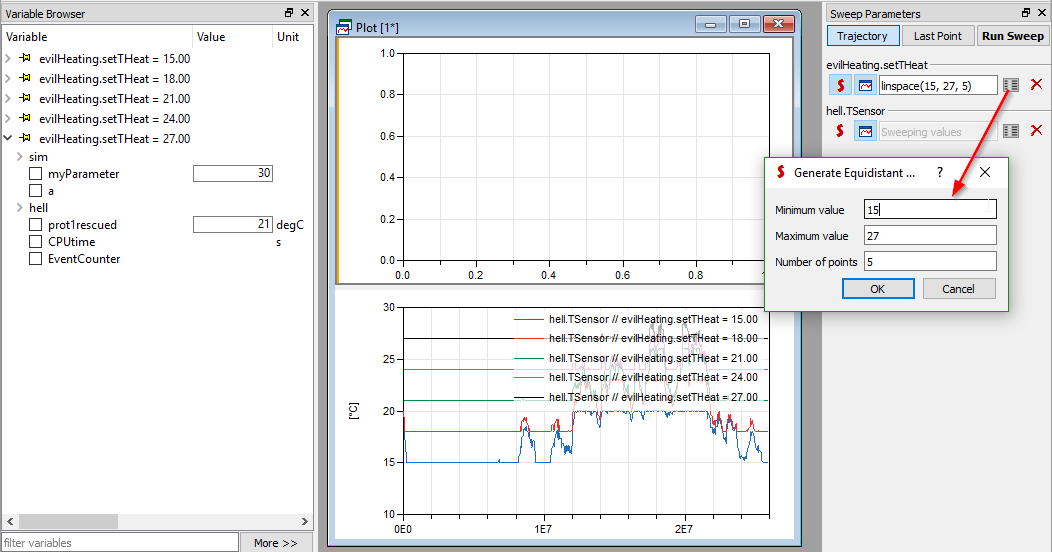
► NEW! *Sweep Parameters*

*Dymola 2019 FD01* includes a new function called *Sweep Parameters.* It lets you to change **one** changeable parameter, even if it is inside a protected component.



\*The red S\*

To use it, *translate* a model. The changeable parameters will appear. Right click on them and select *Add to Sweep Parameters*. You will be able to choose the protected parameters. Then, simulate the model once. The rest of variables will appear, and the protected parameters will disappear. It is possible to include more variables to the *Sweep Parameters*, but for the purpose of plotting results or changing the parameter to sweep easily.

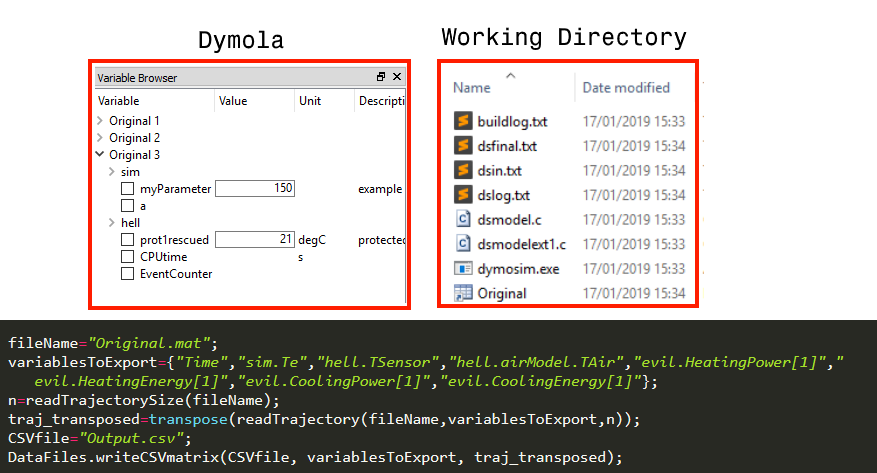


The variations of the parameters can be set on the list button as an array (example: *{15, 18, 22, 34})* or using the button on the right side of the text box. The parameter to change is selected by clicking on the red S. After that, by clicking on *Run Sweep* some simulations will take place and at the end a chart will appear with all selected parameters –button next to the red S-. The protected parameters will not appear on those plots.

► Save results of simulations in *Comma Separated Values* (*csv*) files

After any simulation, the results are displayed in the *Variable Browser*. To save them just right click on them and select the convenient option in *Export Result*. That results are in format *.mat* but sometimes we want to have some parameters in *.csv* to analyse them with other tools, such as *Excel*.

The function to do that is *writeCSVmatrix*, included in the library *DataFiles* -to load it: *File→Libraries→DataFiles*-. The code for exporting is shown in the following figure:



The parts of the code are:

* Variable *fileName:* it is the name of the file which contains all the results. If we go to the working directory after a simulation we will see a file called with the same name as the result and the simulated model. If we simulate the model several times we will have as many results as simulations we ran, but just one single file. The results will have numbers to easily identify them, but each new simulation will overwrite the file located in the working directory. The function *writeCSVmatrix* works with the file, not with the content of *Variable Browser*. So we can only export the results of the last simulation of a certain model –there are ways to avoid that by specifying the name of the result, check the attached script-.
* Array *variablesToExport*: It includes all variables we want to export. To refer to a variable we use pints (.) after the containing objects. It is possible to get the name and the path of a variable by right clicking on it on the *Variable Browser* and selecting *Copy Path*.
* Variable *n*: it is the total number of the results in every variable in *fileName*.
* Array *traj\_transposed*: the function *readTrajectory* reads the file *fileName*, looks for the variables addressed by *variablesToExport* and takes the *n* first values of them. As *n* refers to the whole simulation, it will take all the results. The result is a matrix that ned to be transposed to have the expected shape as a normal *.csv* file –one column by variables and n rows-. *DymolaCommands.Trajectories.readTrajectory* is the location of the main function.
* Variable *CSVfile*: The name of our output *csv* file.
* Function *DataFiles.writeCSVmatrix*: It saves the file. The first argument is the name of the file, the second one are the headers of the *csv* file and the last one is the matrix that contains all the data. The file will be saved in the working directory.

The code can be copied and pasted directly in the command window, or it can be executed as a script.

► Example of a script #1

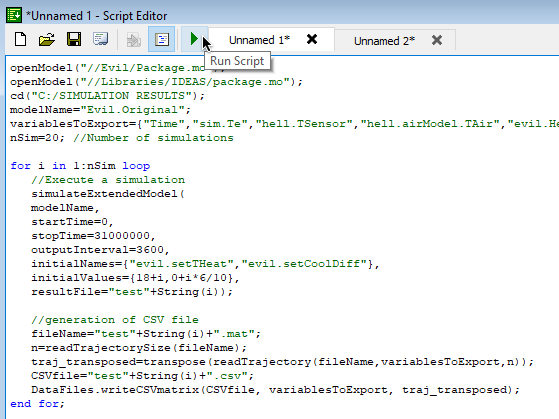
This script:

* Opens the model *Evil* -relative path-. The model is saved in one file -It is not recommended, but this is just an example-.
* Loads the library *IDEAS* – relative path-.
* Changes the *working directory* to C:/SIMULATION RESULTS
* Selects the model *Evil.Original* to be simulated.
* Exports 8 variables –including *time*- referred to the model *Evil.Original*
* Sets 20 simulations to be executed.
* For loop (20 loops)
  + Simulates the model every time with the same conditions.
    - 2 variables - *evil.setTHeat*, *evil.setCoolDiff*- have new values in every loop
    - *initialValues* defines the new values for each loop.
  + After every simulation, a *csv* file is generated and saved in the *Working Directory*

The script can be executed in *Dymola*, just change the paths of the model, *IDEAS* library, and the *Working* *Directory*. To open the script, go to *Simulation → Open script…* or click on this button:



Then, change the values and run it:



The code can be copied and pasted in the *Commands* window. The effect is the same.

