



**PLECS**

*Tutorial*

## **Creating Custom Components with the PLECS Subsystem Block**

**Learn about different masking concepts in PLECS and how to create your own custom component model**

Tutorial Version 1.0

[www.plexim.com](http://www.plexim.com)

- ▶ Request a PLECS trial license
- ▶ Check the PLECS documentation

# 1 Introduction

In this exercise you will create a custom component model of a photovoltaic (PV) string using a masked subsystem. Masking a subsystem allows you to define a custom interface for a Subsystem block that hides the underlying schematic, making it appear as a single component with its own icon and dialog box. Many of the components in the PLECS library are in fact masked subsystems, e.g., all the electrical machine models.

In this tutorial you will learn the following:

- How to create your custom component in PLECS.
- How to add a parameter dialog to your custom component.
- A brief introduction to the scripting language Lua used to define dynamic masks.

**Before you begin** Ensure you have the reference files that you can compare with your own models at each stage of the exercise.

## 2 Implement the PV Current Characteristic

The PV string model consists of an arbitrary number of series-connected PV modules with the same electrical characteristic. A PV module is described by the following module parameters:

- Short-circuit current  $I_{SC}$  in A
- Open-circuit voltage  $V_{OC}$  in V
- Current at the maximum power point  $I_{MPP}$  in A
- Voltage at the maximum power point  $V_{MPP}$  in V

The output current characteristic of a single PV module is a function of voltage and solar irradiance (sun strength). The current characteristic is pre-calculated at simulation start from the **Model initialization commands** based on the simplified model of a photovoltaic (PV) module presented in [1]. The PV string is modeled in PLECS as a non-linear current source. The voltage is an internal feedback signal from the PV string model itself and the solar irradiance is an external parameter specified by the user. The pre-calculated PV current characteristic is loaded into a look-up table.



### Your Task:

- 1 Open a new model and copy the code snippet below in the **Model initialization commands** field in the **Initialization** tab of the **Simulation Parameters (Ctrl + E)** window. This code will calculate the non-linear current characteristic of a PV module based on four values given in a typical PV panel datasheet, e.g. open-circuit voltage, short-circuit current, voltage at the maximum power point (MPP) and current at the MPP.

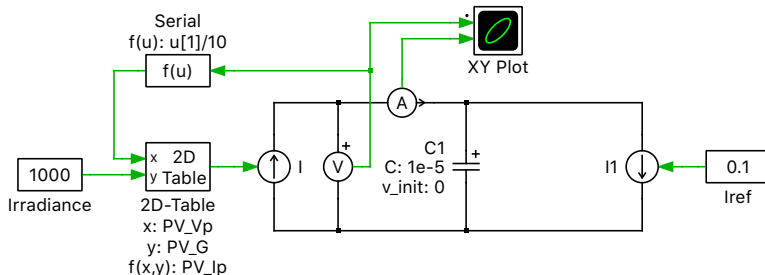
```
Voc = 40.7;
Isc = 10.04;
Vmpp = 33.2;
Impp = 9.49;
% Simplified Model of a Photovoltaic Module, A. Bellini et al.
PV_G = 100:100:1000;
PV_Vp = 0:Voc/100:Voc;
Isc = Isc*PV_G/1000;
Impp = Impp*PV_G/1000;
for j = 1:length(PV_G)
    C2 = ((Vmpp/Voc)-1)/log(1-Impp(j)/Isc(j));
    C1 = (1 - Impp(j)/Isc(j))*exp(-Vmpp/(C2*Voc));
    PV_Ip(:,j) = Isc(j).*(1-C1*(exp(PV_Vp/(C2*Voc))-1));
end
```

- 2 Next, build the model depicted in Fig. 1. The 2D Look-Up Table block needed to define the non-linear output characteristic is found in the PLECS Control component library under “Functions & Tables”.
- 3 To model the voltage characteristic of a 10-module PV string rather than a single PV module, place a Function block in the feedback loop between the voltmeter and look-up table and enter the expression  $u[1]/10$  to extend the voltage characteristic by a factor of 10.
- 4 A capacitor must be connected in parallel with the current source in order to eliminate the state dependency of the PV current on the load current. This would otherwise cause a simulation error. Set this capacitance to  $1e-5$  F.
- 5 Before you run the simulation, set the simulation parameters of the PLECS Solver to the following:
  - Stop time:  $1e-3$  s
  - Relative tolerance:  $1e-6$
  - Solver type: DOPRI (non-stiff)

When you run the simulation, you should see the output IV characteristic for an irradiance of  $1000 \text{ W/m}^2$ . Note that the output of a PV module is rated based on peak sun conditions, which is what this irradiance value represents. Values of irradiance between 0 and  $1000 \text{ W/m}^2$  should be used here.



At this stage, your model should be the same as the reference model, `custom_components_1.plecs`.



**Figure 1: PLECS implementation of PV string model as voltage and solar irradiance-controlled current source**

### 3 Create a Subsystem

The next step is to shift the PV model into a subsystem. Subsystems are useful for hierarchical modeling of circuits and for the creation of reusable custom components.



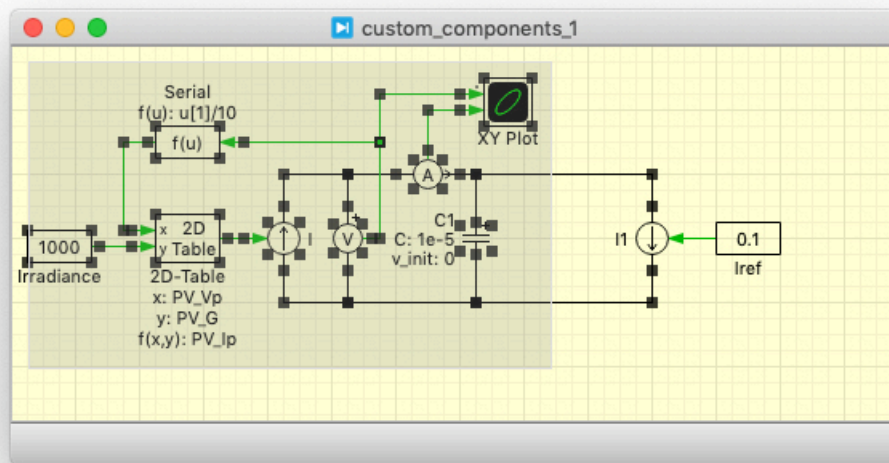
#### Your Task:

- 1 Select the components to be contained in the subsystem as shown in Fig. 2 and right-click and choose **Create Subsystem** or press **Ctrl + G**.
- 2 To reposition the subsystem terminals, hold down the **Shift** key and left-click on the connecting wire just outside the subsystem mask border. A hand symbol will appear, indicating that the terminal can be shifted. Reposition the terminals so that your PV subsystem looks similar to that shown in Fig. 3. You can then drag the subsystem name to the upper left corner.

- 3 To create a custom subsystem symbol, open the subsystem mask editor window by right-clicking and choosing **Subsystem + Create mask...** or press **Ctrl+M**. In the **Icon** tab, select the option **Hide terminal labels** to prevent the port labels from showing on the icon. In the **Language** selector choose Lua. Then enter the following in the **Drawing commands** window:

```
Icon:line({-10, 0, 10},{-20, -7, -20});
```

- 4 You may then need to adjust the size of the subsystem so the corners meet the endpoints of the line you have just drawn. To do this, you first have to unprotect the subsystem by right-clicking and choosing **Subsystem + Unprotect**. The final PV subsystem symbol should look like that shown in Fig. 4.



**Figure 2: Select components to be contained in the subsystem**



**Note:** If you define a mask icon for a Subsystem block, PLECS automatically protects the block and the underlying schematic. You can then no longer resize the Subsystem block or modify the sub-schematic (without first unprotecting it). The purpose of this protection is to prevent the user from making unintentional changes that might render the icon useless.

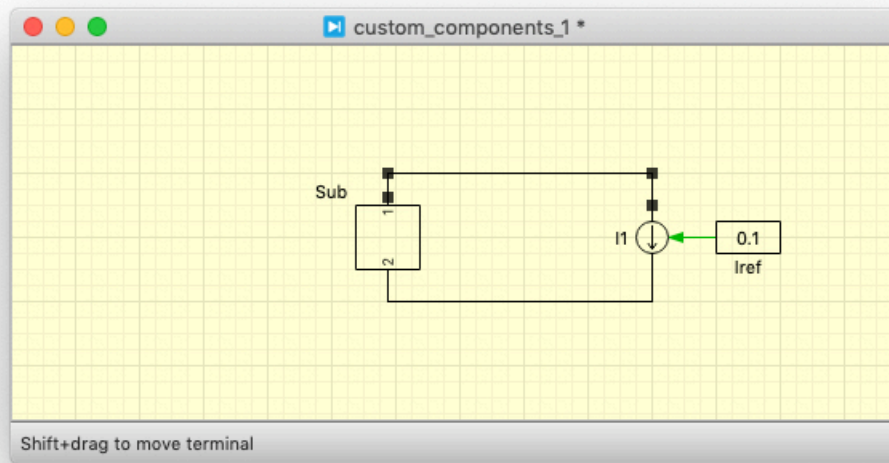
## 4 Add Probe Signals to the Subsystem Mask

Adding custom probe signals to a subsystem mask makes it possible to directly monitor the key values inside a subsystem using the PLECS Probe block. Any quantity that is measured or calculated inside the subsystem can be added to a custom probe signal list in the subsystem mask. In this example, the probe signals that will be added to the PV string subsystem mask are output voltage, current and power.

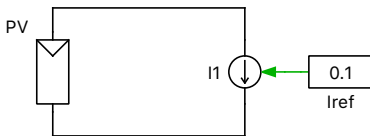


### Your Task:

- 1 Firstly, calculate the output power inside the subsystem so that this signal can be added to the probe signal list. Add a Signal Multiplexer block and combine the voltage and current signals to



**Figure 3: Reposition the terminals by holding down Shift and dragging the terminals with the left mouse button**

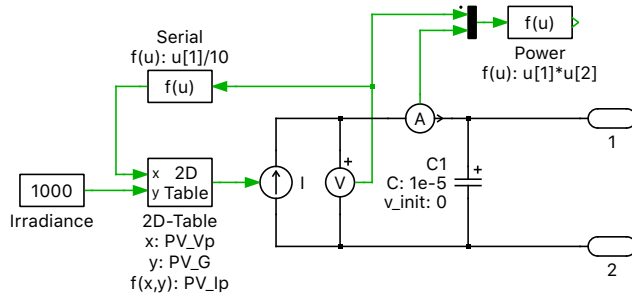


**Figure 4: Customize the subsystem mask using drawing commands**

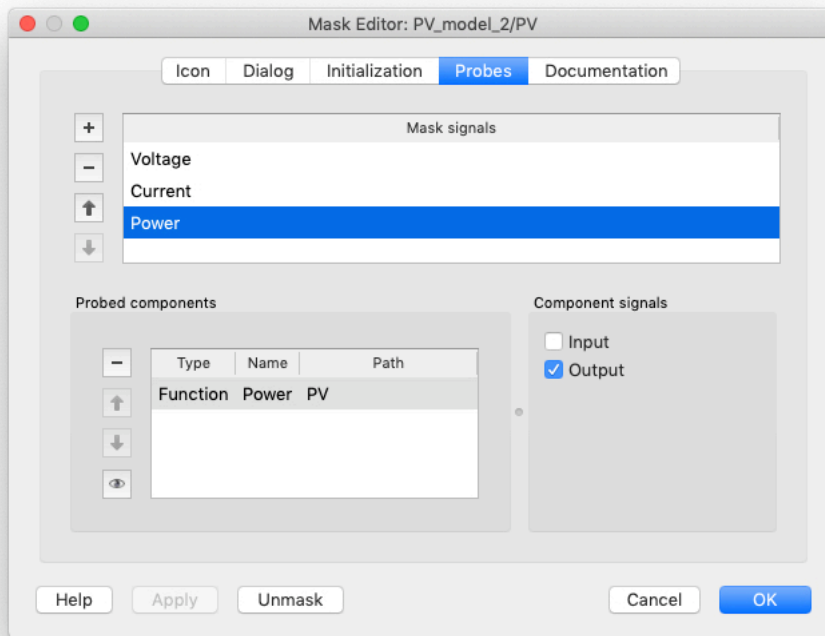
calculate the power using a Function block. The modified subsystem with the additional power calculation function is shown in Fig. 5.

- 2** To create the probe signal list, right-click on the PV subsystem icon and choose **Subsystem + Edit mask...** (this shortcut is also **Ctrl+M**) to open the Mask Editor window. Before you define any probe signals, select the **Documentation** tab and enter PV string in the **Mask type** box. This is the name that will appear in the probe list when the PV subsystem is dragged into a PLECS Probe block.
- 3** Under the **Probes** tab of the Mask Editor window, the custom probe signal list can be created. Click the **Add** button to define a new probe signal and name this Voltage. Then drag in the Voltmeter block, “Vm1”, from your subsystem into the box labeled **Probed components** and check the **Measured voltage** signal in the **Component signals** list. Repeat this procedure to add probe signals for Current and Power. When creating the power signal, you will need to drag the power calculation function block into the **Probed components** window and check the **Output** box. At this stage, your Mask Editor window should be similar to that shown in Fig. 6.

To test the probe signals you have created, place a Probe block onto the top level of the schematic and drag in the PV subsystem. You should see a list of three available signals that can be monitored: Voltage, Current and Power. Check the **Voltage** and **Current** signals. Use a Signal Demultiplexer block to connect the probed signals to an XY Plot block. When you rerun the simulation, the results should be the same as before.



**Figure 5: Add power calculation to the subsystem to allow output power to be added to the subsystem probe signals**



**Figure 6: Create probe signals using the Mask Editor**



At this stage, your model should be the same as the reference model, `custom_components_2.plecs`.

## 5 Add Parameters to the Subsystem Mask

Adding a parameter to a subsystem mask creates a user dialog parameter field that appears when you double-click the subsystem icon. Having mask parameters provides a convenient means of adjusting internal parameters without needing to open the subsystem. Initialization commands can also be included in a subsystem mask to allow automatic loading of subsystem parameters.

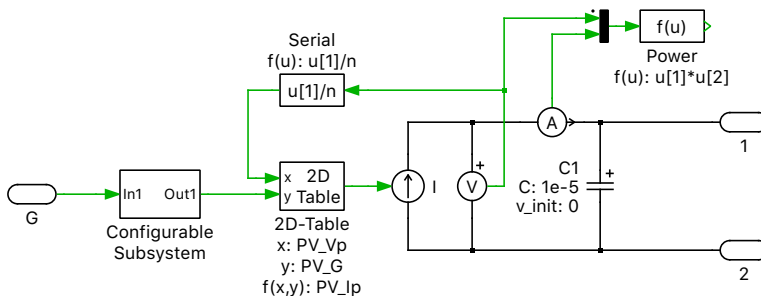
Adding parameters to a subsystem mask causes the components in the subsystem to lose visibility of global variables. Components in a *masked* subsystem only have visibility of variables defined in the

subsystem mask. This helps to maintain the modularity of a subsystem, since the subsystem is not dependent on external variables.



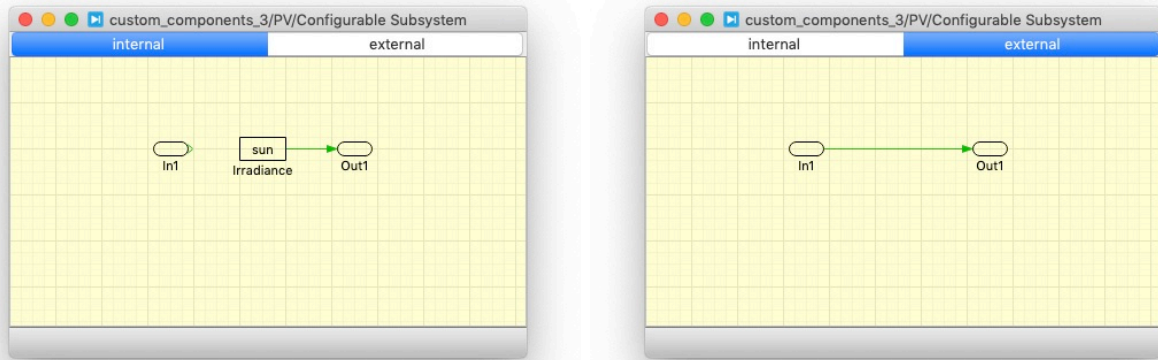
### Your Task:

- 1 To create a mask dialog parameter, open the Mask Editor window and under the **Dialog** tab, add a new parameter. In the **Prompt** box enter Number of modules and in the **Variable** box, give this parameter the variable name “n”. Now modify the PV string model by changing the expression in the Function block from  $u[1]/10$  to  $u[1]/n$ .
- 2 Add a second parameter called “Solar irradiance” and give it the variable name “sun”. Check the **Tunable** box to allow modifying the variable “sun” interactively during a simulation by entering a new value in the parameter dialog.
- 3 To complete the parameter mask, add the variable “Irradiance” and give it the name “sun\_combo”. The **Type** of this parameter is Combo Box. Define two **Combo box values** as internal and external on separate lines with no characters in between. The return value for a combo box parameter is a string containing the 1-based index of the chosen option. We will use the variable “sun\_combo” to change the active configuration in the Configurable Subsystem component.
- 4 Add a Subsystem component to the circuit model, and convert it to a Configurable Subsystem by right-clicking on the subsystem and choosing **Subsystem + Convert to configurable subsystem**. Then connect it as shown in Fig. 7. This modification will allow you to switch the signal source for the solar irradiance between the value given in the dialog mask or to use an external signal. Name the Signal Input block “G”. This input is used to bring the external irradiance signal into the masked subsystem.



**Figure 7: The Configurable Subsystem component allows changing the active configuration**

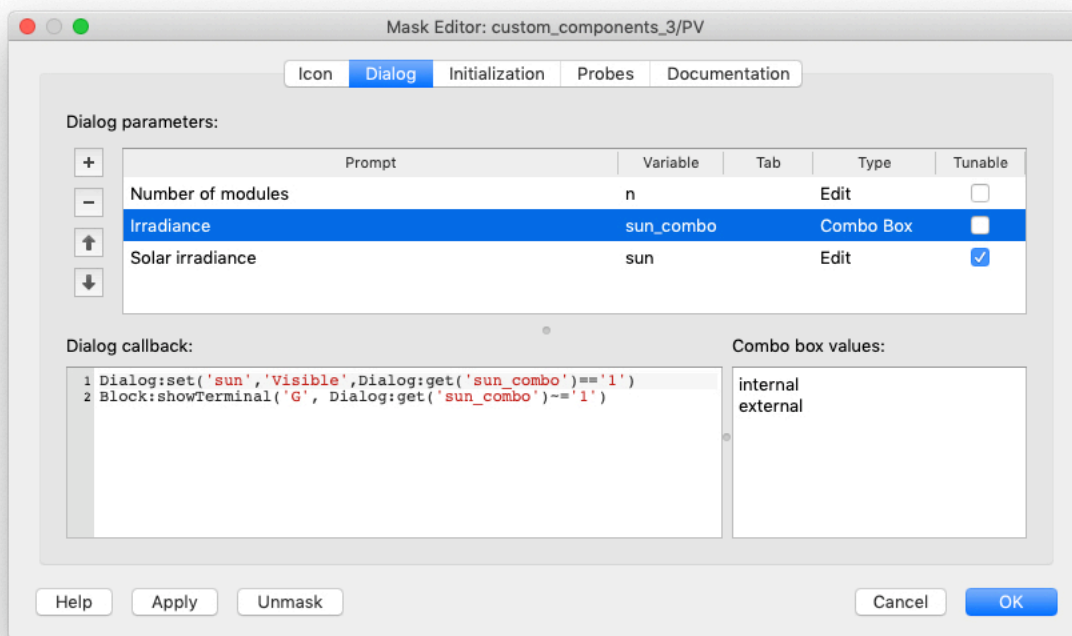
- 5 You can open the schematic view of the Configurable Subsystem block by right-clicking on the subsystem and choosing **Subsystem + Open subsystem** or pressing **Ctrl+U**. The schematic of each configuration can be accessed by the tabs on top of the schematic view. A double-click on the configuration tab allows for the corresponding configuration to be renamed, i.e. to internal. Add a second configuration via the context menu of the tab bar, accessible by a right-click. Name the second configuration external. The schematic diagram of each subsystem configuration is shown in Fig. 8.
- 6 Double-clicking the Configurable Subsystem block will bring up the **Block Parameters** window. Write sun\_combo in the **<reference>** option of the **Configuration** combo box field. This means the subsystem configuration is passed by reference instead of a hard-coded configuration in the drop down list. The active configuration of the subsystem now changes as a function of the mask variable “sun\_combo” defined in the mask parameters.
- 7 Add the two following lines of code into the **Dialog callback** section of the PV panel Mask Editor window. These Lua statements allow hiding the “Solar irradiance” parameter field when the **Irradiance** parameter is set to external and to hide the signal input terminal “G” when the **Irradiance** parameter is set to internal. Your **Dialog parameters** window should now look like what



**Figure 8: Schematic view of each subsystem configuration**

is shown in Fig. 9.

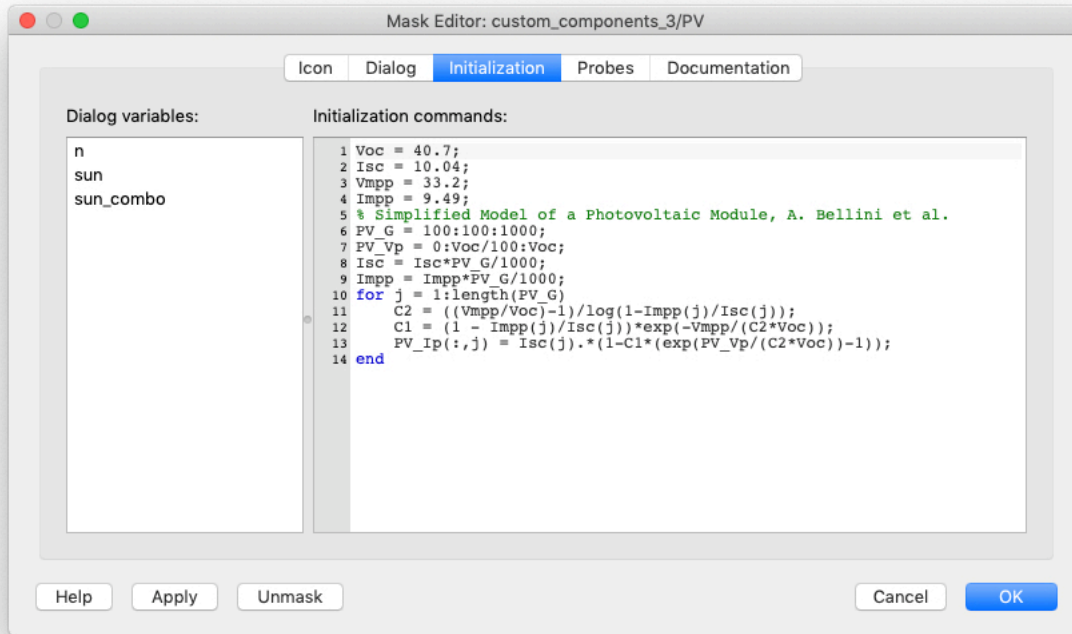
```
Dialog:set('sun','Visible',Dialog:get('sun_combo')== '1')
Block:showTerminal('G',Dialog:get('sun_combo')== '1')
```



**Figure 9: Creation of subsystem dialog parameters in the Mask Editor window**

- 8 The calculation of the current characteristic needs to be done within the subsystem mask since the subsystem only has visibility of mask variables. Move the code from the **Model initialization commands** window to the subsystem **Initialization commands** window. Your subsystem **Initialization Commands** window should now look like what is shown in Fig. 10.
- 9 Once you have applied parameters to the subsystem mask, double-clicking the subsystem will no longer show the underlying schematic but bring up the Block Parameters window. Enter a value of





**Figure 10: Initialization commands of the masked subsystem**

10 for the number of modules and 1000 for the solar irradiance value. When you change the **Irradiance** parameter to external, an additional terminal is visible on the top level of the schematic. You can connect any signal source from the PLECS library to this terminal, e.g., a Step block.



At this stage, your model should be the same as the reference model, `custom_components_3.plecs`.

## 6 Conclusion

In this exercise you have used the subsystem concept to create a custom component with a unique icon and mask parameters. Custom components support the top-down design approach and are easy to reuse and configure. Masking signals inside of a subsystem allows for easy measuring of signals as well.

This model of a PV cell component can be extended to account for variations in temperature and can be connected in parallel strings for use as the source of an inverter system.

## 7 Getting Started with Lua

Lua is a simple yet powerful open-source scripting language. This tutorial introduces you to the basic concepts that you are likely to need in order to create dynamic subsystem masks. For a full reference please visit the Lua website [2].

By default, Lua declares all variables as *global*. However, PLECS executes Lua code in a protected environment that forbids the creation or modification of global variables. Therefore, you must explicitly declare variables and functions as *local* using the `local` keyword, e.g.

```
local x = "a string"
```

## 7.1 Mask Icon Drawing Commands

Some drawing commands available in the Lua language are described below. If you enter more than one command, the graphic objects are drawn in the order in which the commands appear. In case an error occurs during evaluation of the commands, PLECS displays three question marks (???) in the mask icon.

### Text

The command

```
Icon:text(x, y, 'text')
```

displays *text* in the center of the icon or centered around the coordinates *x*, *y*. The text does not rotate with the icon; it is always displayed from left to right.

### Line

The command

```
Icon:line(xvec, yvec)
```

draws the line specified by the vectors *xvec* and *yvec*. Both vectors must have the same length. Note that vectors are entered using curly braces, e.g., {1, 2, 3}.

### Image

The command

```
Icon:image(xvec, yvec, 'filename')
```

reads an image from the file *filename* and display it on the mask icon. The parameter *filename* must be either an absolute filename (e.g., C:/images/myimage.png) or a relative filename that is appended to the model's directory (e.g., images/myimage.png). Supported image formats are BMP, GIF, JPG and PNG.

### Querying Parameters Values

The command

```
Dialog:get('variable')
```

returns the string value of the mask parameter associated with the variable *variable*. Note that the parameter values are not evaluated. With this command you can alter your mask icon as a function of the mask parameters.

## Functions

A function is defined as follows

```
local function drawTriangle(x, y)
    Icon:line(Vector{0, 8.66, -8.66, 0}+x, Vector{-10, 5, 5, -10}+y)
end
```

A function definition consists of the keyword `function`, a *name* (`drawTriangle`), a list of *parameters* (`x, y`), a *body*, i.e., a list of statements, and the terminator `end`. Parameters are local variables that are initialized with the values of the arguments passed in the function call. The example above defines a function that draws a triangle with the center point `x,y`. The function can be called as follows

```
drawTriangle(10, 0)
```

## 7.2 Dialog Callback Commands

### Setting Parameter Values

The command

```
Dialog:set('variable', 'property', value, ...)
```

changes one or more properties of the mask parameter associated with the variable *variable*. The properties `Enable` or `Visible` are possible. `Enable` specifies the enable state of the parameter. A disabled parameter is grayed out in the dialog and cannot be modified. `Visible` specifies the visibility of the parameter in the dialog.

### Hiding, Showing Terminals

The command

```
Block:showTerminal('name', flag)
```

shows or hides the terminal named *name* depending on the boolean value *flag*. The companion port of a hidden terminal acts in the same way as if the terminal was shown but unconnected.

### Moving Terminals

The command

```
Block:moveTerminal('name', x, y)
```

moves the terminal named *name* to the relative coordinates *x, y* with respect to the unrotated and unflipped block. Note that the terminal rotation is not changed.

## References

- [1] A. Bellini, V. Iacovone and C. Cornaro, *Simplified model of a photovoltaic module*, Applied Electronics, Pilsen, pp. 47-51, 2009.
- [2] Lua - the programming language, [Online]. Available: <http://www.lua.org>. [Accessed: Apr. 06, 2020].

## Revision History:

Tutorial Version 1.0      First release

## How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	info@plexim.com	Email
	<a href="http://www.plexim.com">http://www.plexim.com</a>	Web

## *PLECS Tutorial*

© 2002–2023 by Plexim GmbH

The software PLECS described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from Plexim GmbH.

PLECS is a registered trademark of Plexim GmbH. MATLAB, Simulink and Simulink Coder are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.