

# Simulation Visualization using KEV

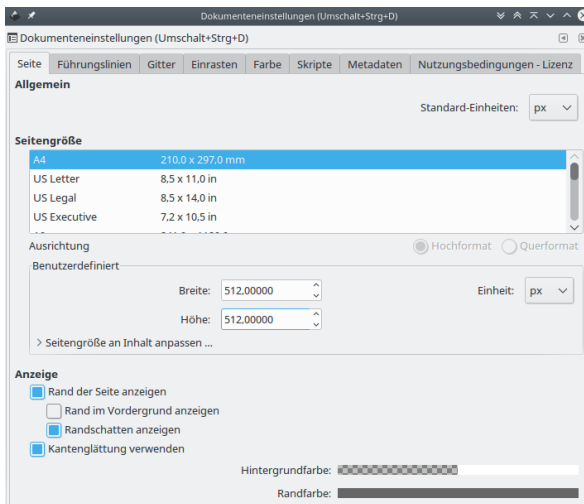
This tutorial uses the KIELER Environment Visualization (KEV) to display the state of a very simple traffic light controller. This process is split into the following steps:

1. creating an SVG image
2. creating a mapping file for this SVG image
3. creating a traffic light controller in SCCharts
4. simulating the controller, including visualization

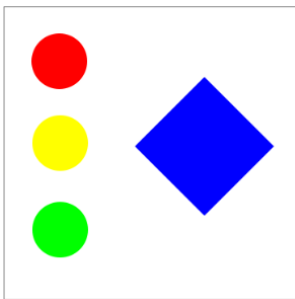
## 1. Creating an SVG Image

In this tutorial the open-source vector graphics program Inkscape is used to create an SVG image.

It is advisable to first set the document size. In Inkscape this is done under **File > Document Settings**. In the image below the size is set to 512x512 px. This setting defines the coordinate system of the file, which is useful when animating e.g. movement later in KEV. Anyhow, SVG images can be scaled to any size.

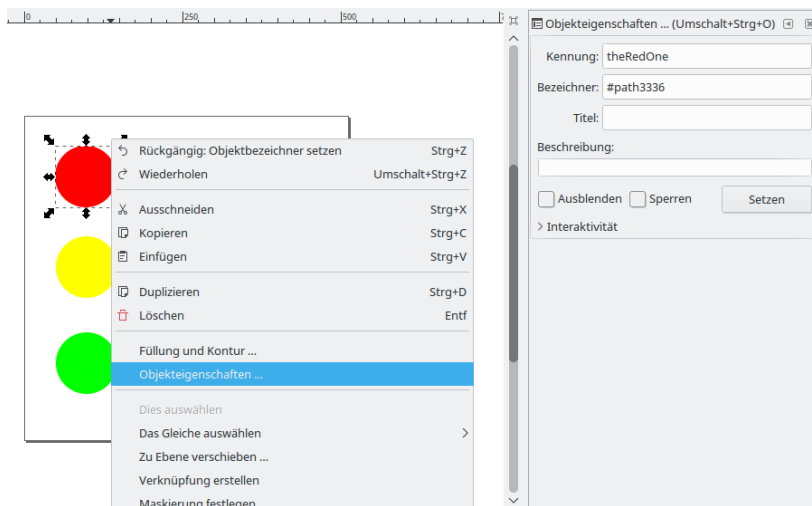


Afterwards three circles and a rectangle have to be created. The circles will simulate the red, green and yellow lights of a traffic light, and thus are colored accordingly. The rectangle will show the same information, but using different animations.



After the elements of the image have been created, it is necessary to give each element a unique name if it should be animated. This can be done in Inkscape by selecting the object and using **Right click > Object properties**. In the window that opens, an **id** can be set in the **first field** (the value without leading hash, labelled "Kennung" in german). Name the circles according to their color, **theRedOne**, **theYellowOne** and **theGreenOne**. The rectangle gets the id **theRect**.

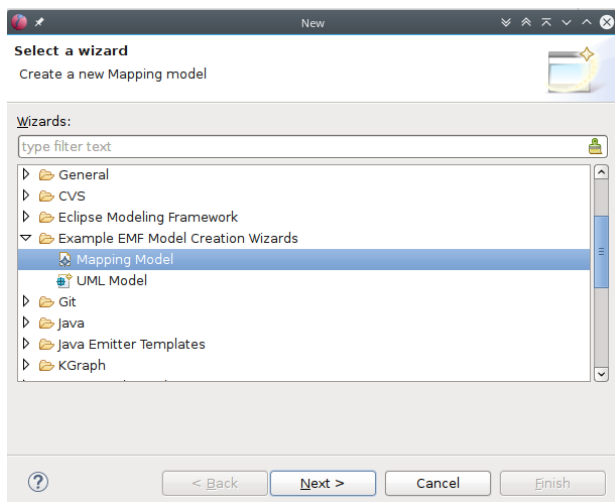
Note that a new id has to be applied using the button in the object properties window!



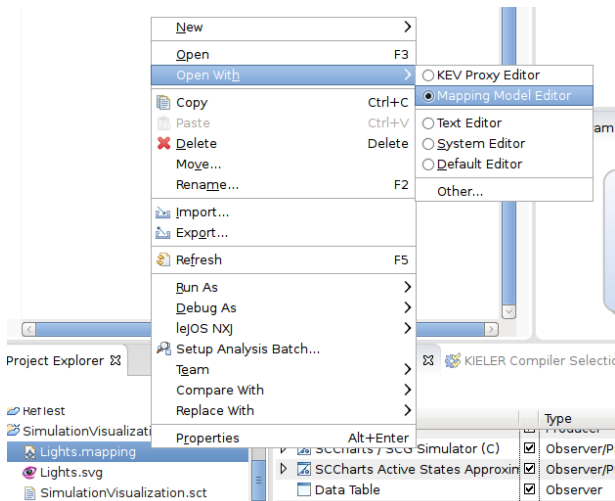
Finally, save the result as **Inkscape-SVG** (recommended) or **Normal-SVG** and name it **Lights.svg**.

## 2. Mapping animations to SVG elements

In the following a mapping of animations, domains and variables to elements in the SVG is created. Therefore one has to create a new mapping file in KIELER using **File > New > Other > Mapping Model**. Name the file **Lights.mapping** and select **SVG File** as **Model Object**.



After hitting **Finish**, the file has to be opened using **Open With > Mapping Model Editor**.



In the editor that opens, select the **SVG File** element. If there is not yet a **Properties View** open in your workbench, select **Window > Show View > Other > General > Properties**. This view works together with the Mapping Model Editor, such that the fields of elements can be manipulated.

Set the saved **Lights.svg** as property **Filename** of the **SVG File** element. Afterwards new children of the SVG File can be created. These represent the elements in the SVG file, which should be animated. Right click on the SVG File element and select **New Child > SVG Element**. Set the id of this element in the properties view to **theRedOne**.

An SVG Element can have any number of animations. In this tutorial, we will add an Opacity animation to theRedOne and configure it, such that it will be visible or invisible, depending on the value of a variable named **showLight**. Therefore we right click on the element, select **New Child > Opacity** and set its properties in the Property View to the following:

- Input: 0,1,2
- Key: showLight
- Opacity: 1,0,0

This will map the input values 0, 1 and 2 of a variable **showLight** to the opacity values 1, 0 and 0 of the element in the SVG image (showLight = 0 opacity = 1, showLight = 1 opacity = 0, and so on).

The animation for the yellow and green circle is created analogously:

Create SVG Elements and set the id to **theYellowOne** and **theGreenOne** respectively. Add an Opacity animation to theYellowOne and set the properties to the following:

- Input: 0,1,2
- Key: showLight
- Opacity: 0,1,0

Add an Opacity animation to theGreenOne and set the properties to the following:

- Input: 0,1,2
- Key: showLight
- Opacity: 0,0,1

This will animate the circles according to the value of the variable showLight.

Additionally, we add an SVG element with the id **theRect**, add a Colorize animation and set its properties to the following:

- Input: 0,1,2
- Key: showLight
- Fill Color: red,yellow,green

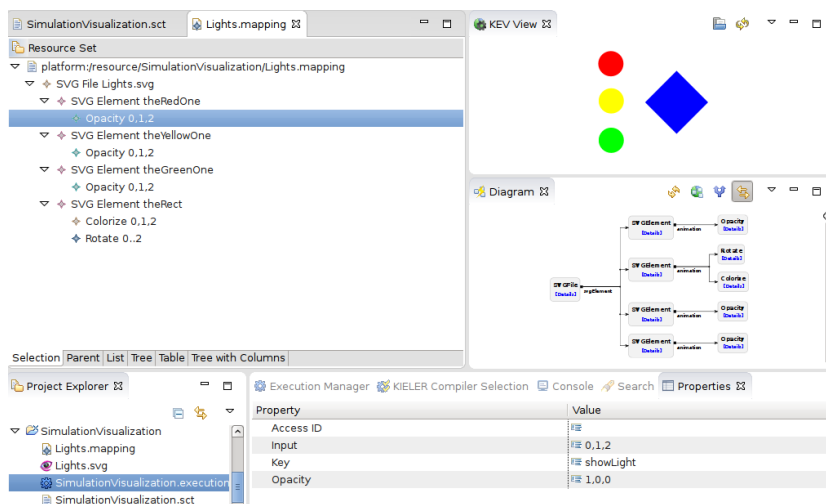
This will color the rect according to the value of showLight.

Finally we add another animation to theRect, namely the Rotate animation. Its properties are set to the following:

- Input: 0..2
- Key: showLight
- Angle Range: 0..90

This will map the input values from 0 to 2 linearly to the angles from 0 to 90 degree.

The final mapping structure should look similar to this:



### 3. Creating a Light Controller with SCChart

Create an empty SCChart and fill it with the following content:

#### Traffic light controller

```
scchart TrafficLight {  
  
    output int showLight;  
  
    initial state init  
    --> init with / showLight += 1; showLight = showLight % 3;  
}
```

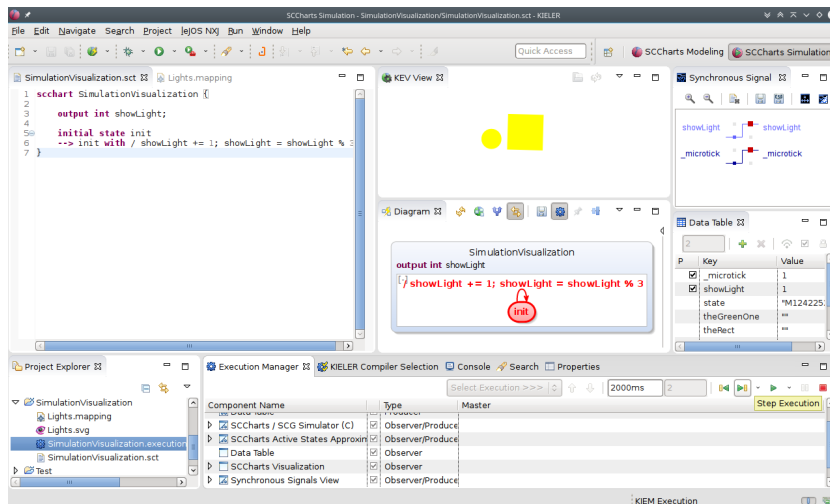
This model will set showLight successively to 0, 1 and 2.

### 4. Starting the simulation

Open the KEV View under **Window > Show View > Other > KIELER > KEV View**. In this new view, open the mapping file that has been created before. If everything is OK, the SVG graphic is displayed.

With the mapping file open in KEV, switch to the SCChart of the traffic light. Open the Execution Manager if it is not yet open (Window > Show View > Other > KIELER Simulation > Execution Manager). This view controls the simulation. However, to have the simulation visualized with the SVG image and mapping file, one has to add the KEV Data Observer to the list in the Execution Manager using **Right click > Add Component > KEV Data Observer**.

Afterwards you can run the simulation using Step Execution. After each step, the image in the KEV View should be updated to reflect the state of the controller. Congratulations!



### Problem Solving

Ensure that the id's in the SVG match the id's in the mapping file.

Ensure that the KEV Data Observer is added to the list in the Execution Manager (when opening an SCChart, this list might be lost).

Ensure that the KEV view has been (re-)loaded with the correct mapping file (saving the file on disk does not reload it in the KEV View automatically).

KEV is updating the SVG concurrently with the values from the execution. Thus it might occur that some states are skipped, if one steps through the execution very fast.