Roassal Tutorial (visualization engine)

The following tutorial is meant for the student to get used with basic concepts of Roassal. At the end of the tutorial, the student should be able to know how to use Roassal to visualize object's metrics and relationships by using shapes, a layout, color, size and interactions. He should also be able to know where to find more information to resolve doubts and to learn more about the engine. The tutorial is split in three parts: 1) Hand’s on session, 2) Examples of Roassal’s builders and 3) Pointers to find more information. Let’s start.

1. Installation
   I. Download the latest version of Moose
   II. If you want to use the Pharo image you can follow the instruction in the Roassal website

2. Hands-on (blue $\Rightarrow$ added code; red $\Rightarrow$ removed code)
   I. View, Shape
      Goal: Show a collection of elements with a grid layout.
      i) Create a RTView object
      ii) Depict RTObject subclasses using a RTEllipse shape
      iii) Apply a RTGridLayout

      ```
      view := RTView new.
      shape := RTEllipse new.
      els := shape elementsOn: RTObject withAllSubclasses.
      view addAll: els.
      RTGridLayout new gapSize:15; on: els.
      view open
      ```

   II. Shape, color and size
      Goal: Specify a different shape, color and size for elements.
      i) Change the shape to RTBox
      ii) Set height in 20 and width in 10
      iii) Set a random color for all/each nodes

      ```
      view := RTView new.
      shape := RTEllipse RTBox new height:20;
      width:10; color:[:e|Color random].
      els := shape elementsOn: RTObject withAllSubclasses.
      view addAll: els.
      RTGridLayout new gapSize:15; on: els.
      view open
      ```
III. Edges

*Goal: Show the class hierarchy through edges with a tree layout.*

i) Add edges from each class element to its subclasses

ii) Change the layout to a RTTreeLayout

```plaintext
view := RTView new.
shape := RTBox new height:20; width:10;
color:[:e|Color random].
els := shape elementsOn: RTObject withAllSubclasses.
view addAll: els.
RTEdge buildEdgesFromObjects: RTObject
withAllSubclasses from:#yourself toAll:#subclasses
RTGridLayout new gapSize:15; RTTreeLayout on: els.
els @ RTDraggable.
els @ RTPopup.
els @ (RTHighlightable new highlightedColor: Color red).
els @ (RTMenuActivable new action:#inspect).
view @ RTDraggableView.
view open
```

IV. Interactions

*Goal: Try interactions for elements and for the view.*

i) Set elements RTDraggable

ii) Add Popup to elements

iii) Set elements RTHighlightable (with red color)

iv) Add a RTMenuActivable to inspect elements

v) Set elements RTLabelled

vi) Specify the layout’s gapSize as 15

vii) Set view RTDraggableView

```plaintext
view := RTView new.
shape := RTBox new height:20; width:10;
color:[:e|Color random].
els := shape elementsOn: RTObject withAllSubclasses.
view addAll: els.
RTGridLayout new gapSize:15; on: els.
els @ RTDraggable.
els @ RTPopup.
els @ (RTHighlightable new highlightedColor: Color red).
els @ (RTMenuActivable new action:#inspect).
view @ RTDraggableView.
view open
```
V. Metrics

*Goal: Define and map metrics to visual dimensions.*

i) Define height as LOC (Number of Lines of Code)

ii) Define width as NOM (Number of Methods)

iii) Define color as NDC (Number of dependent classes)

iv) Show NOD in the Popup

```plaintext
view := RTView new.
shape := RTBox new height:#numberOfLinesOfCode; width:[:e| e methods size]; color:[:e|Color grayScale256: (256 - e dependentClasses size)].
els := shape elementsOn: RTObject withAllSubclasses.
view addAll: els.
RTEdge buildEdgesFromObjects: RTObject withAllSubclasses from:#yourself toAll:#subclasses using: RTLine inView: view.
RTTreeLayout on: els.
els @ RTDraggable.
els @ (RTPopup new textElement:[:e| e model dependentClasses size]).
els @ (RTHighlightable new highlightedColor: Color red).
els @ (RTMenuActivable new action:#inspect).
view @ RTDraggableView.
view open
```
VI. Normalization

*Goal: Normalize the color and height properties.*

i) Color using NDC metric from green to red

ii) Height using LOC metric with a square root function

```plaintext
view := RTView new.
shape := RTBox new
    height:#numberOfLinesOfCode
    width:[:e| e methods size]
    color:[:e| Color grayScale256: (256 - e dependentClasses size)].

els := shape elementsOn: RTOBJECT withAllSubclasses.
view addAll: els.
RTEdge buildEdgesFromObjects: RTOBJECT withAllSubclasses
    from:#yourself toAll:#subclasses using: RTLine inView: view.

RTMetricNormalizer new
    elements: els;
    normalizeHeight: #numberOfLinesOfCode
        min:5
        max:1000
    using:[:e| e sqrt];
    normalizeColor: [:e| e dependentClasses size ];
using:{Color green . Color red}.
RTTreeLayout on: els.
els @ RTDraggable.
elsWithEditMenu on: els.
elsWithEditMenu on: els.
elsWithEditMenu on: els.
view @ RTDraggableView.
view open
```
2. Builders

A builder encapsulates the logic of visualization and it maps a particular domain to visual elements. It eases the reuse of domain-specific visualizations. Developers can define custom builders by sub classing `RTBuilder`. The custom builder should define public methods for end-users to specify the objects to be visualized, layouts that can be applied and any other attribute specific to the domain. The custom builder should override the `renderIn:` method where it can specify how the visualization is built. The following examples (amongst others) of domain-specific builder are found in Roassal Examples.

I. UML  
II. RTCharterBuilder  
III. RTPieBuilder  
IV. RTSpectrograph  
V. RTSunburstBuilder  
VI. RTGraphBuilder  
VII. RTNameCloud  
VIII. RTMondrianViewBuilder  
IX. RTMapLocationBuilder

3. Resources

I. [Agilevisualization.com](http://www.agilevisualization.com) (book in progress)  
II. Roassal examples  
III. [Deep into Pharo book](http://www.deepintopharo.com)  
IV. Mailing lists Pharo / Moose

Leonel Merino  
PhD Student SCG  
[merino@scg.unibe.ch](mailto:merino@scg.unibe.ch)

5-nov-2014