Roassal @ Pharo TechTalk

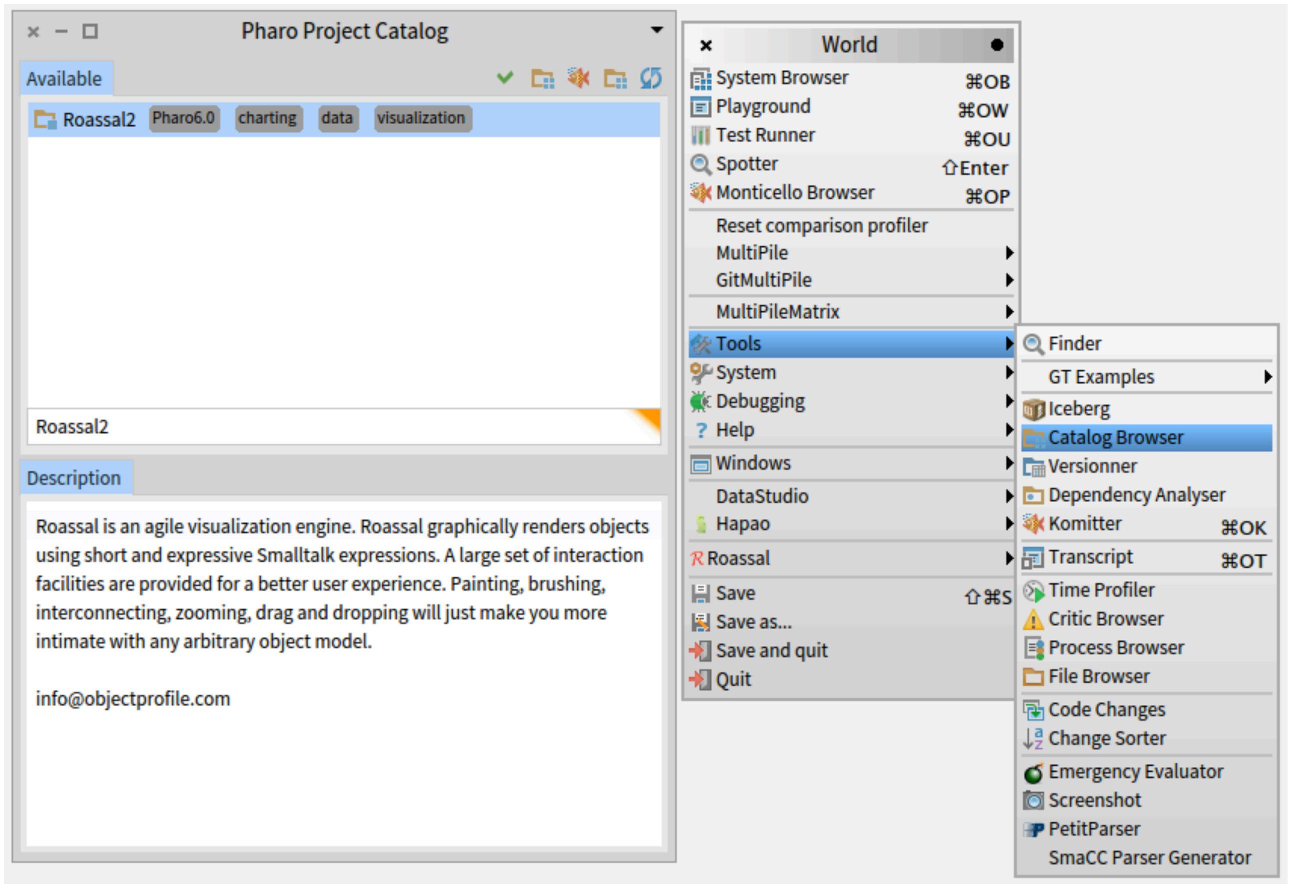
May 30, 2017

Content of the Roassal TechTalk:

1 - General introduction of Roassal
2 - Latest news in Roassal (Theme support, Scatterplot Matrix, Elastic boxes, ...)
3 - Applications based on Roassal (RTScatterplotMatrix, Test Coverage, Memory profiling)
4 - Ongoing effort and future work

Introduction

Roassal is a visualization engine for Pharo and VisualWorks. Roassal is available under the MIT license.
On Pharo, Roassal may be downloaded from the Pharo catalog,



Roassal is an agile visualization engine. Roassal graphically renders objects using short and expressive Smalltalk expressions. A large set of interaction facilities are provided for a better user experience. Painting, brushing, interconnecting, zooming, drag and dropping will just make you more intimate with any arbitrary object model.

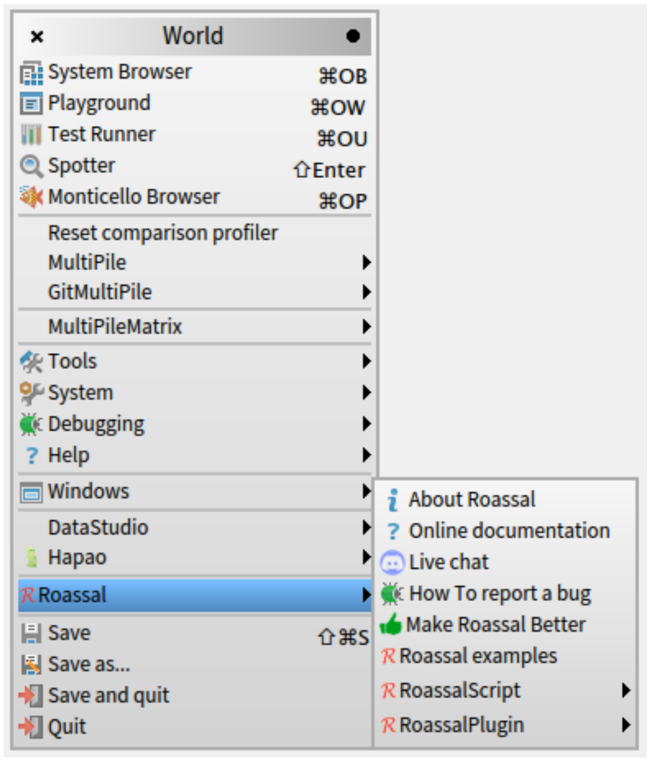
info@objectprofile.com

Alternatively, Roassal can be loaded using the following Gofer expression:

```
Gofer it
```

```
    smalltalkhubUser: 'ObjectProfile' project: 'Roassal2';
    configurationOf: 'Roassal2';
    loadDevelopment
```

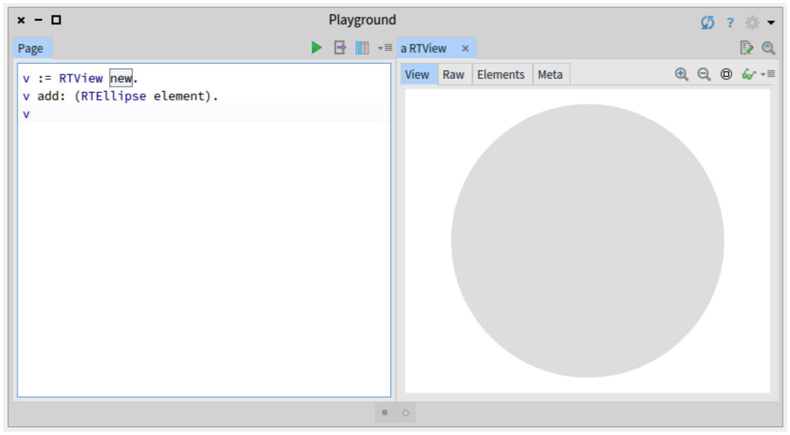
The World menu contains several entries to access the examples, to report bugs, to jump to the agile visualization book (http://AgileVisualization.com), and load addition plugins.



First example with Roassal

Open a Playground, and inspect the following expression:

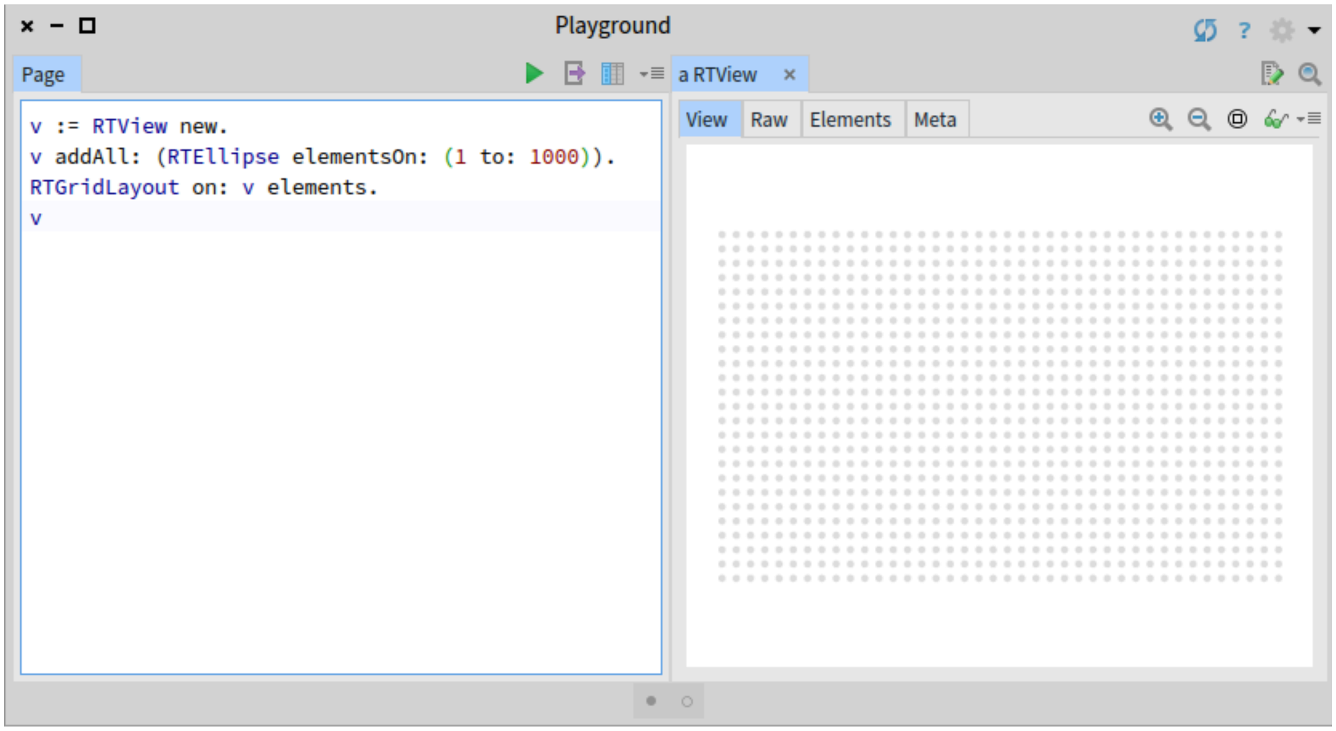
```
v := RTView new.
v add: (RTEllipse element).
v
```



```
v := RTView new.
v add: (RTEllipse element).
v
```

Many elements can be added:

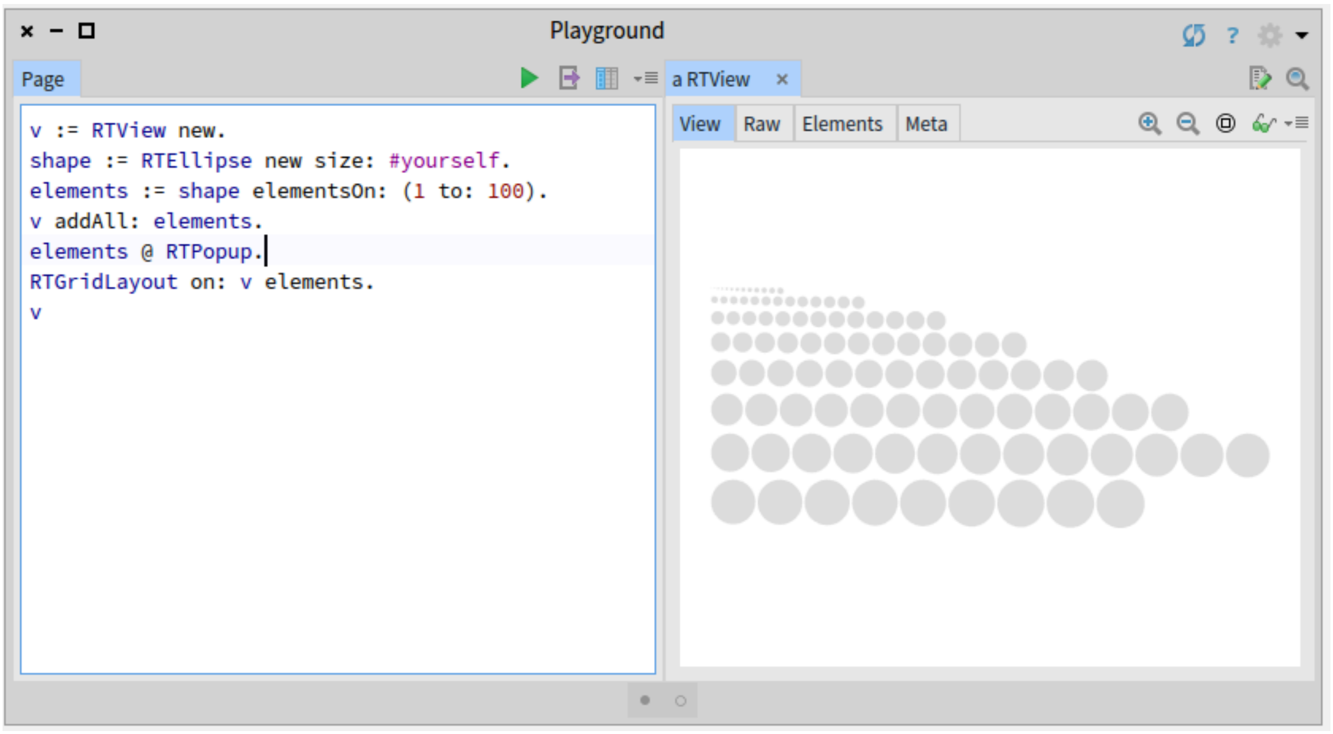
```
v := RTView new.
v addAll: (RTEllipse elementsOn: (1 to: 1000)).
RTGridLayout on: v elements.
v
```



```
v := RTView new.
v addAll: (RTEllipse elementsOn: (1 to: 1000)).
RTGridLayout on: v elements.
v
```

You can change the size of the circles:

```
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTGridLayout on: v elements.
v
```



```
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTGridLayout on: v elements.
v
```

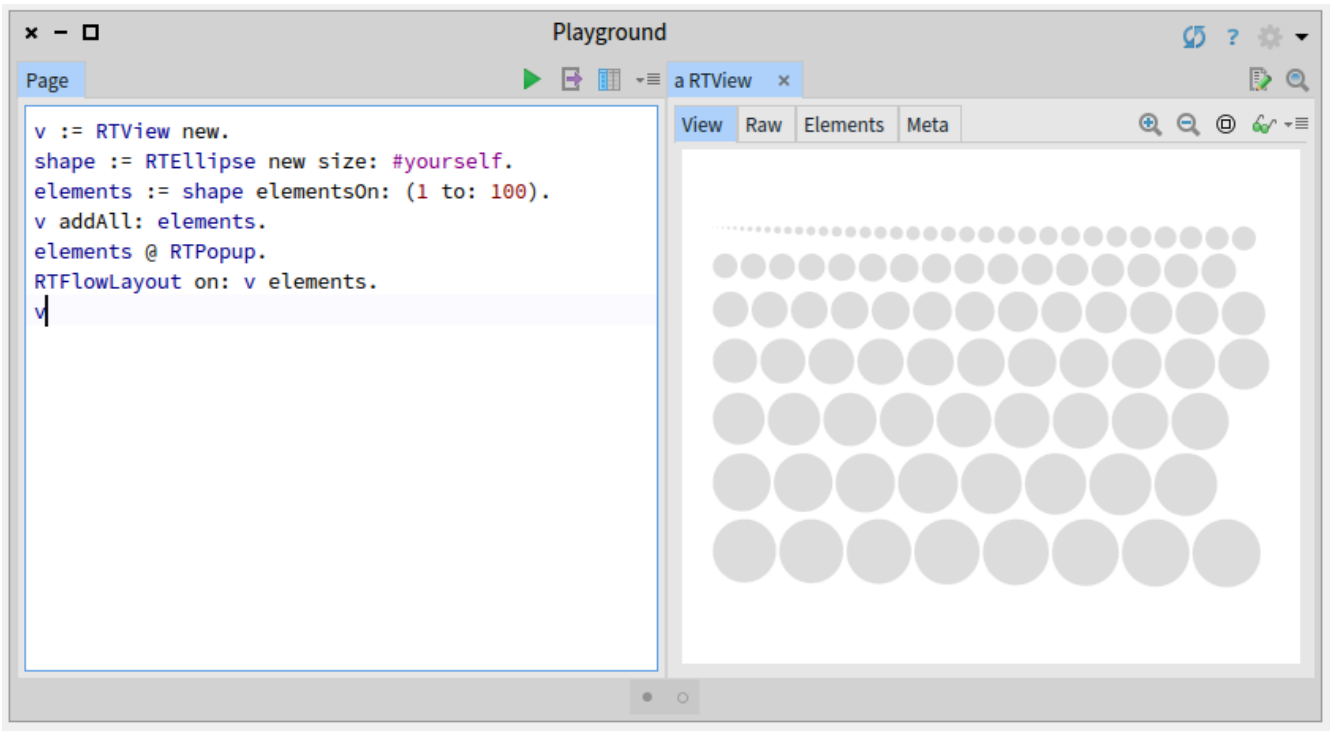
An element is an instance of the class RTElement, and represents a graphical element on the screen. An element points to an object, which could be any objects). A popup indicates which object is represented by an element.

A shape is a factory of elements. Shape may be configured in many different ways. On the example above, size of the element depends on the represented objects.

We have here used a grid layout to order the elements. Without the layout, all the elements will be on the exact same place, which is not really interesting in our case.

Another interesting layout is the flow layout:

```
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTFlowLayout on: v elements.
v
```



```
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTFlowLayout on: v elements.
v
```

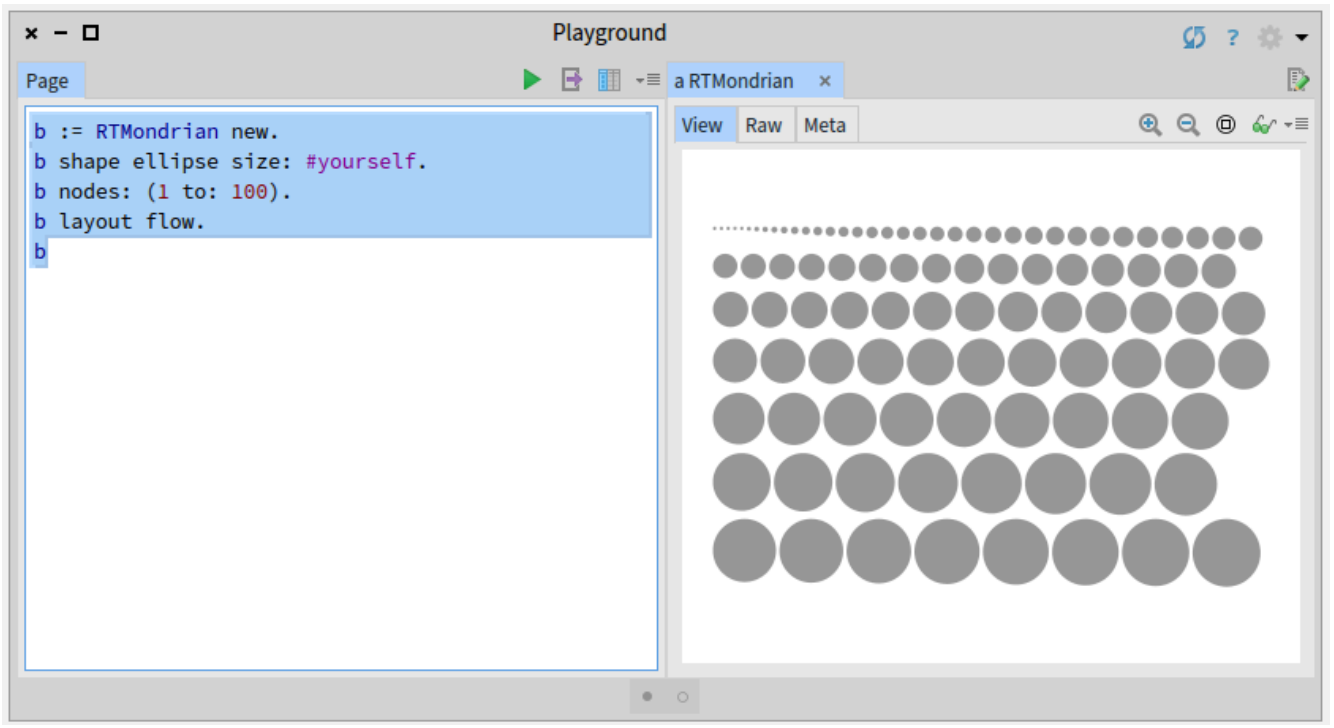
At that stage, Roassal appears to be very similar to any graphical libraries. One significant difference is the use of shapes as factories. A shape can be parametrized with metrics and properties that can be applied on the represented object.

Another significant differences are builders. A builder represent a dedicated visualization. The Builder infrastructure is described in a dedicated chapter:
https://dl.dropboxusercontent.com/u/31543901/AgileVisualization/Builder/0201-Builder.html

Mondrian

Mondrian is a Roassal builder to easily build what is called a *polymetric view*. The example given previous can be written as:

```
b := RTMondrian new.
b shape ellipse size: #yourself.
b nodes: (1 to: 100).
b layout flow.
b
```

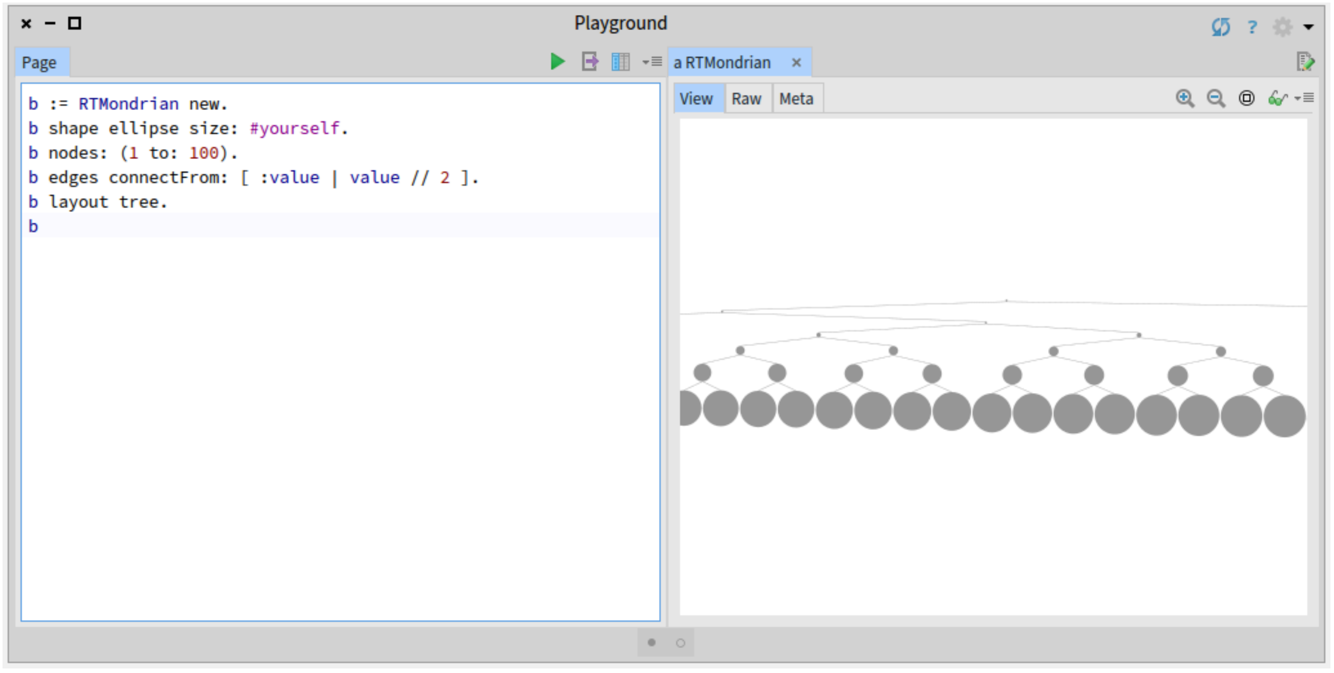


```
b := RTMondrian new.
b shape ellipse size: #yourself.
b nodes: (1 to: 100).
b layout flow.
b
```

As you can see, the code is slighter shorter using Mondrian. However, Mondrian comes with several large APIs. On the good side, hundreds of examples cover Mondrian.

Edges may be easily defined:

```
b := RTMondrian new.
b shape ellipse size: #yourself.
```



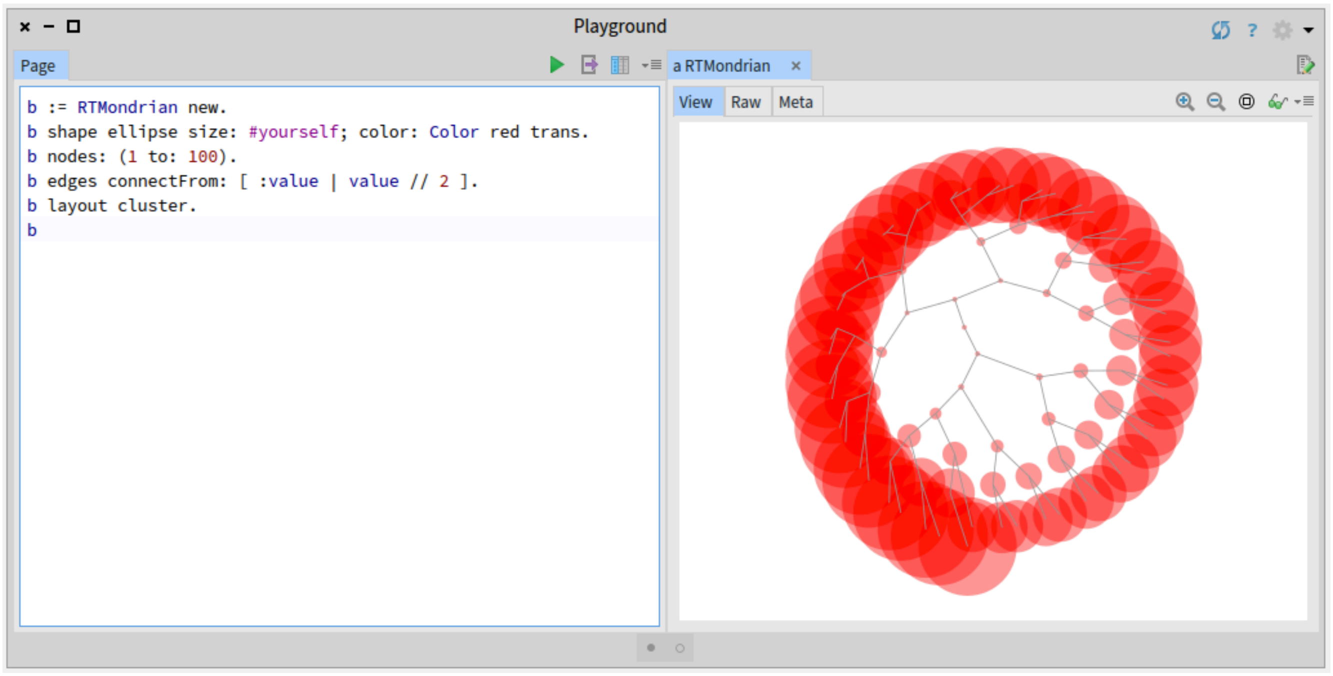
```
b := RTMondrian new.
b shape ellipse size: #yourself.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout tree.
b
```

```
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout tree.
b
```

The code given above uses the tree layout.

Here is an example using the cluster layout:

```
b := RTMondrian new.
b shape ellipse size: #yourself; color: Color red trans.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout cluster.
```

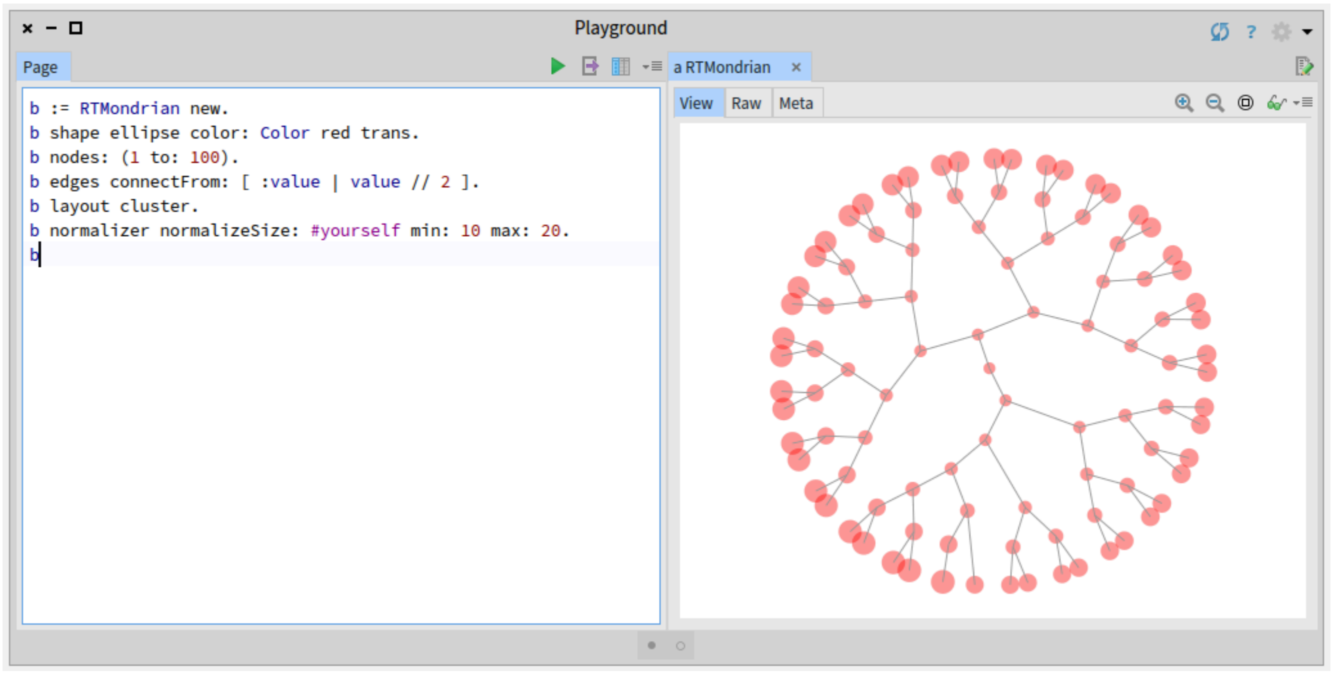


```
b := RTMondrian new.
b shape ellipse size: #yourself; color: Color red trans.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout cluster.
b
```

```
b
```

Normalization is an important aspect of Roassal and Mondrian. Instead of giving an absolute size, we can use a normalizer to set sizes. For example:

```
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout cluster.
b normalizer normalizeSize: #yourself min: 10 max: 20.
b
```



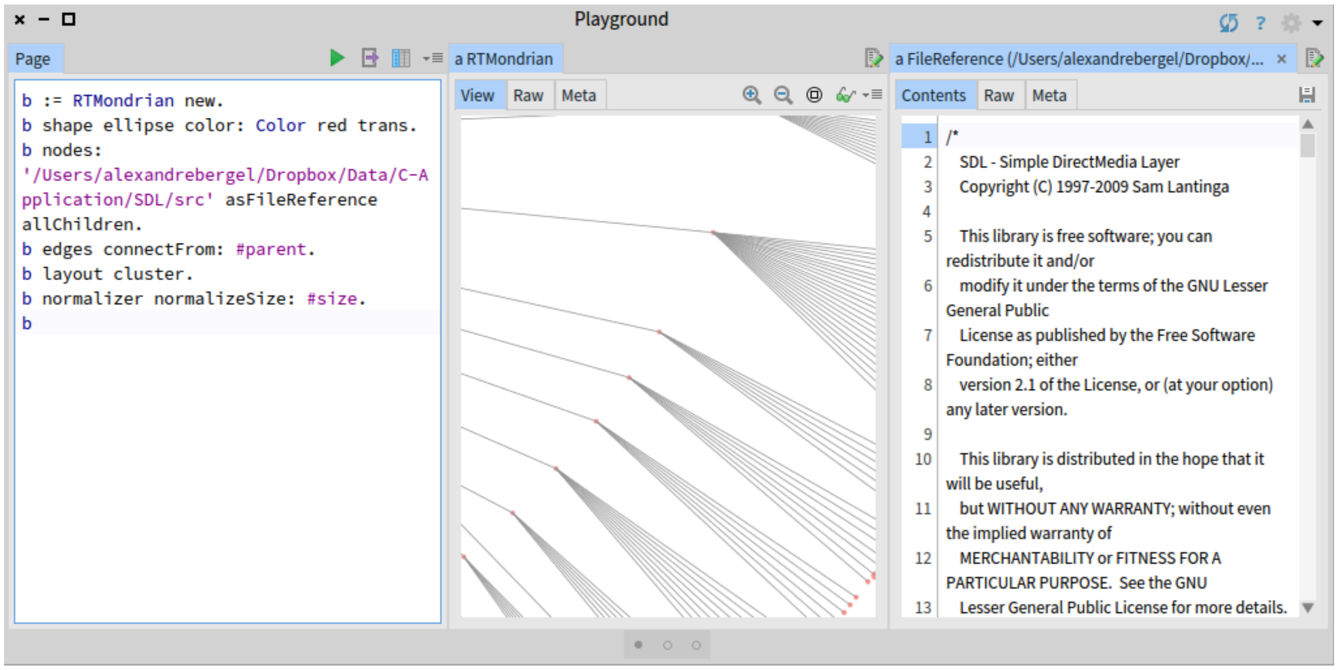
```
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout cluster.
b normalizer normalizeSize: #yourself min: 10 max: 20.
b
```

So far, we have used numerical values in our examples. We will visualize files instead:

```
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: '/Users/alexandrebergel/Dropbox/Data/C-Application'
asFileReference allChildren.
b edges connectFrom: #parent.
b layout cluster.
b
```

We can represent the size of the files:

```
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: '/Users/alexandrebergel/Dropbox/Data/C-Application/
SDL/src' asFileReference allChildren.
b edges connectFrom: #parent.
b layout cluster.
b normalizer normalizeSize: #size.
b
```



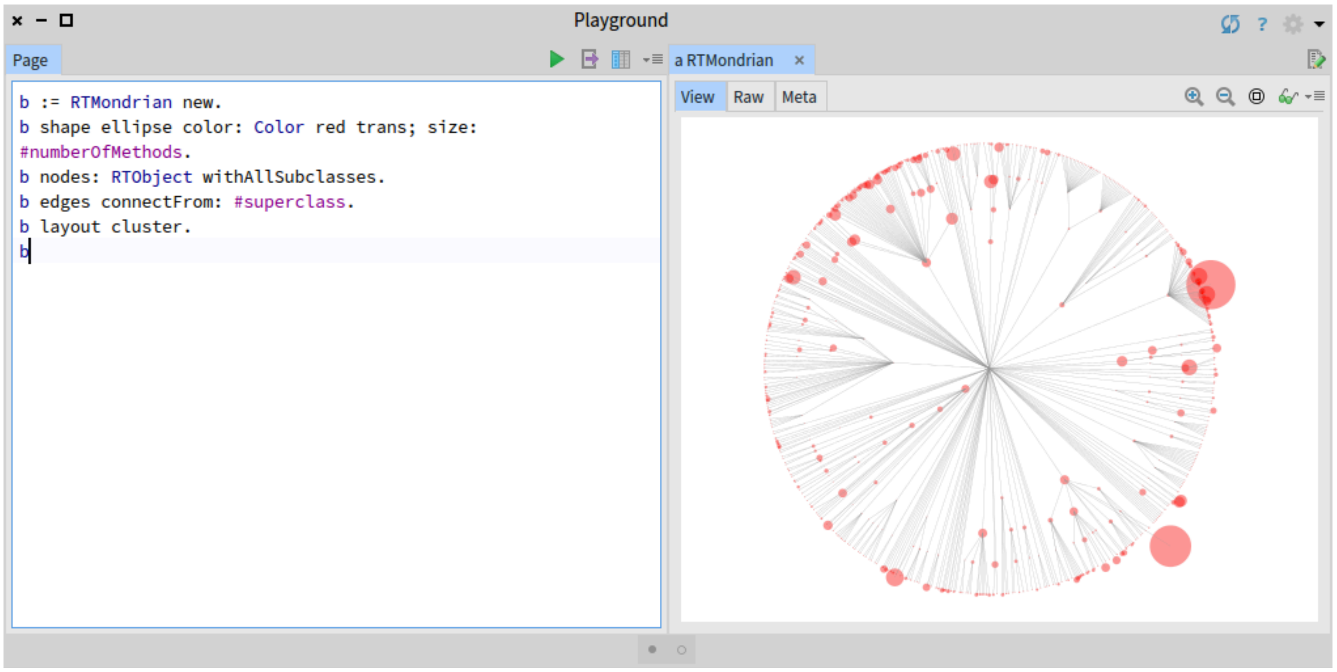
```
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes:
'/Users/alexandrebergel/Dropbox/Data/C-A
pplication/SDL/src' asFileReference
allChildren.
b edges connectFrom: #parent.
b layout cluster.
b normalizer normalizeSize: #size.
b
```

```
/*
    SDL - Simple DirectMedia Layer
    Copyright (C) 1997-2009 Sam Lantinga

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.
```

Another recurrent example is to visualize classes:

```
b := RTMondrian new.
b shape ellipse color: Color red trans; size:
#numberOfMethods.
b nodes: RTObject withAllSubclasses.
b edges connectFrom: #superclass.
```

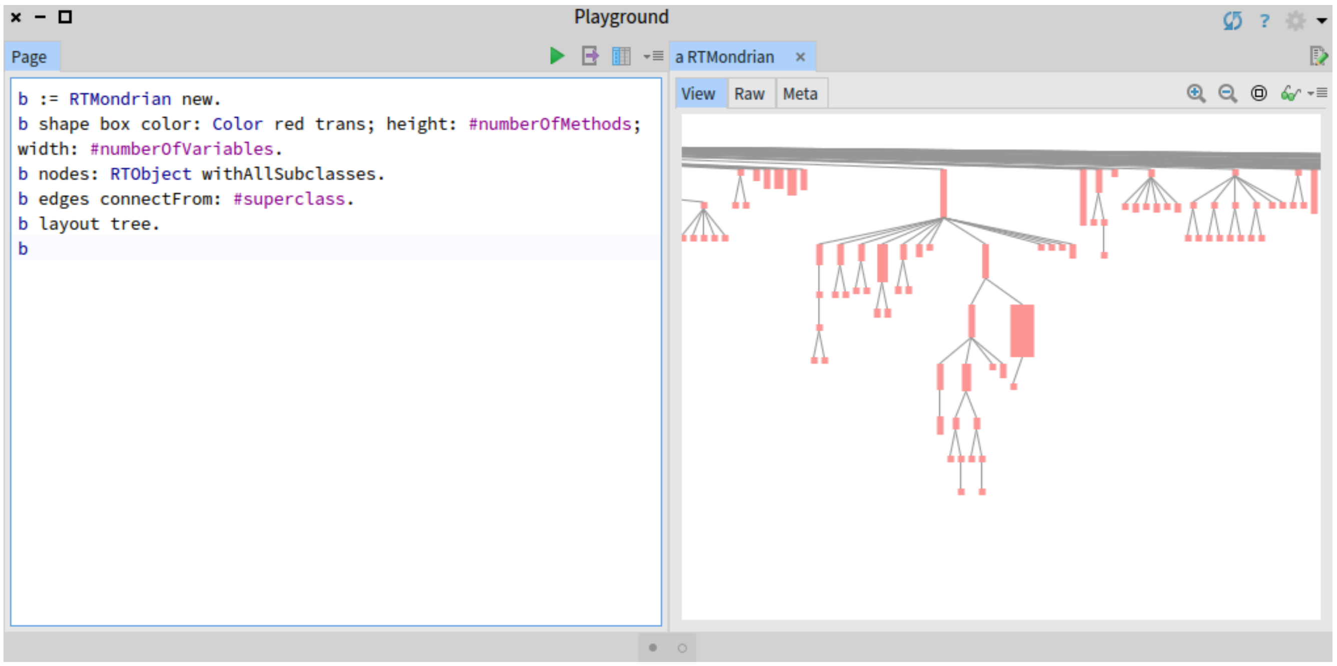
```
b layout cluster.
b
```



```
b := RTMondrian new.
b shape ellipse color: Color red trans; size:
#numberOfMethods.
b nodes: RTObject withAllSubclasses.
b edges connectFrom: #superclass.
b layout cluster.
b
```

The code above visualize all the classes of Roassal.
A tree layout works well in that case:

```
b := RTMondrian new.
```

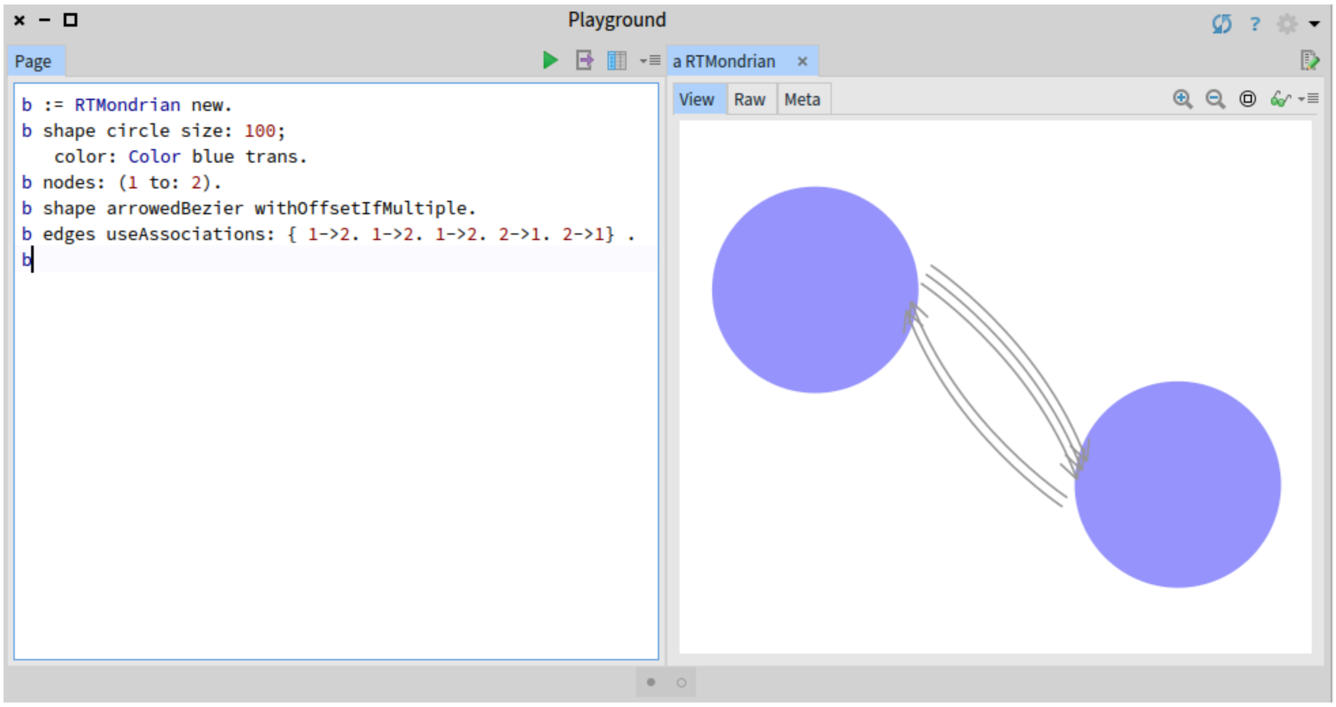


```
b := RTMondrian new.
b shape box color: Color red trans; height: #numberOfMethods;
width: #numberOfVariables.
b nodes: RTObject withAllSubclasses.
b edges connectFrom: #superclass.
b layout tree.
b
```

```
b shape box color: Color red trans; height:
#numberOfMethods; width: #numberOfVariables.
b nodes: RTObject withAllSubclasses.
b edges connectFrom: #superclass.
b layout tree.
b
```

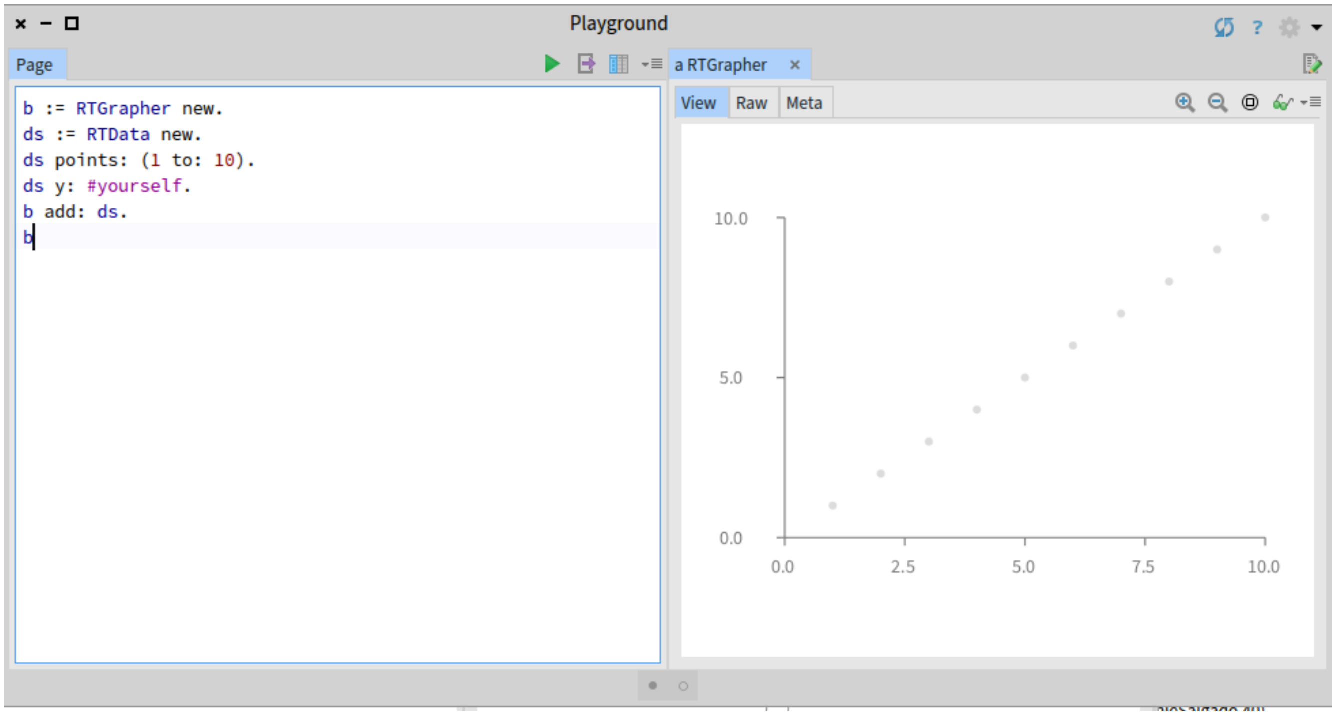
Mondrian allow for sophisticated configuration of edges. For example:

```
b := RTMondrian new.
b shape circle size: 100;
    color: Color blue trans.
b nodes: (1 to: 2).
b shape arrowedBezier withOffsetIfMultiple.
b edges useAssociations: { 1->2. 1->2. 1->2. 2->1. 2->1} .
b
```



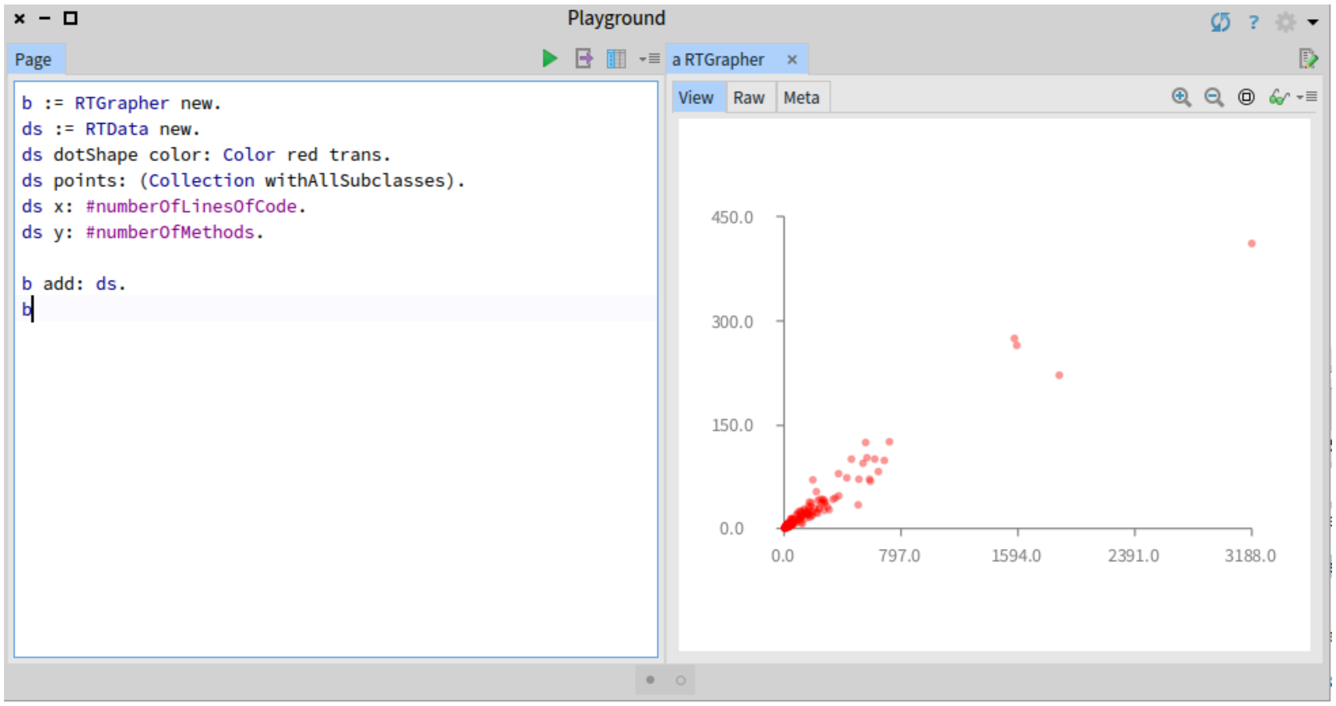
```
b := RTMondrian new.
b shape circle size: 100;
   color: Color blue trans.
b nodes: (1 to: 2).
b shape arrowedBezier withOffsetIfMultiple.
b edges useAssociations: { 1->2. 1->2. 1->2. 2->1. 2->1} .
b
```

Grapher



```
b := RTGrapher new.
ds := RTData new.
ds points: (1 to: 10).
ds y: #yourself.
b add: ds.
b
```

Grapher is another builder of Roassal to charts.
A very simple example is:
b := RTGrapher new.
ds := RTData new.
ds points: (1 to: 10).
ds y: #yourself.
b add: ds.
b



```
b := RTGrapher new.
ds := RTData new.
ds dotShape color: Color red trans.
ds points: (Collection withAllSubclasses).
ds x: #numberOfLinesOfCode.
ds y: #numberOfMethods.

b add: ds.
b
```

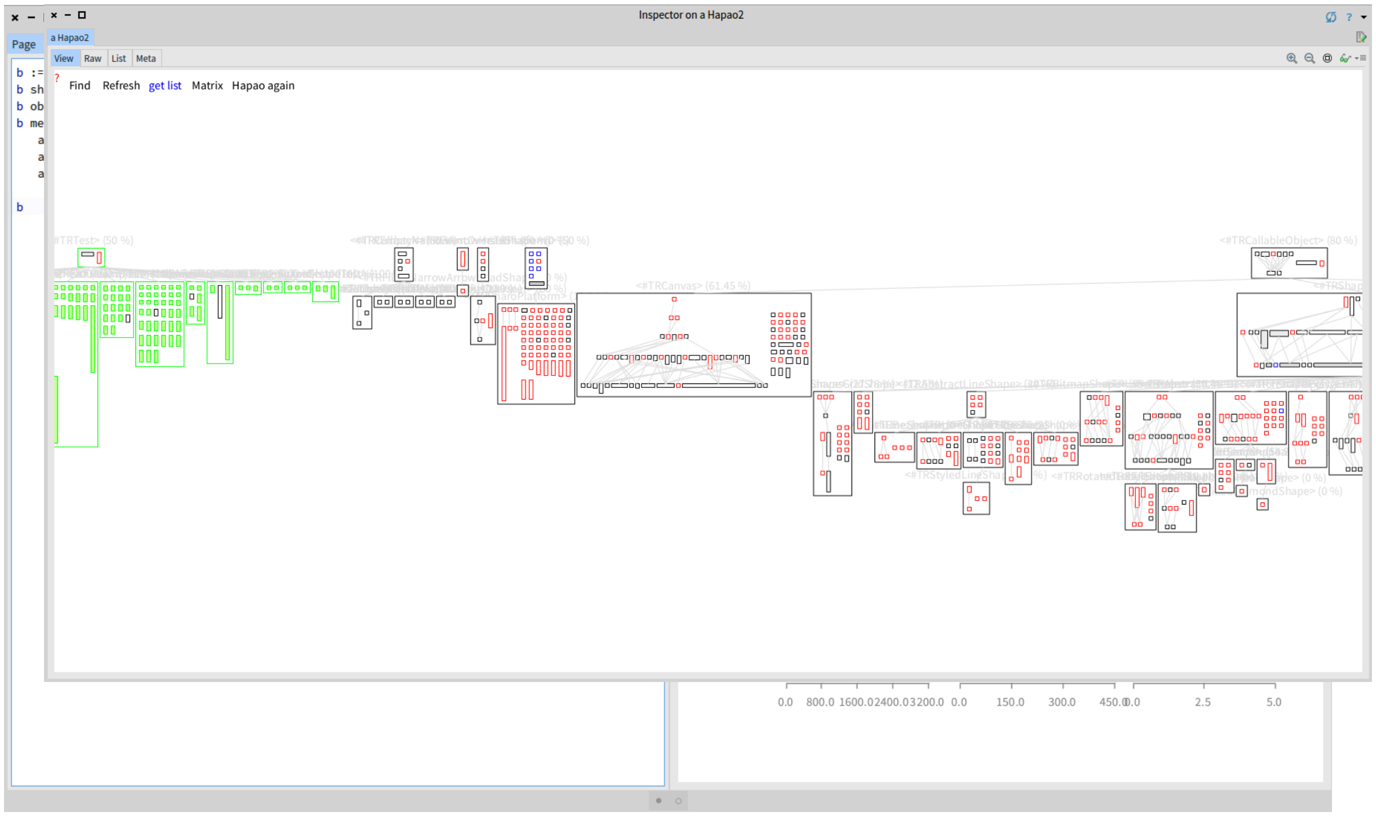
```
b := RTGrapher new.
ds := RTData new.
ds dotShape color: Color red trans.
ds points: (Collection withAllSubclasses).
ds x: #numberOfLinesOfCode.
ds y: #numberOfMethods.

b add: ds.
b
```

Applications

We recently developed RTScatterplotMatrix, a matrix of scatterplots:

```
b := RTScatterplotMatrix new.
b shape circle color: Color red trans.
b objects: Collection withAllSubclasses.
b metrics
    at: 'LOC' put: #numberOfLinesOfCode;
    at: 'NOM' put: #numberOfMethods;
    at: 'NOV' put: #numberOfVariables.

b
```

Hapao is a test code coverage, available under the Spy2 catalog entry. You can visualize the test coverage of the test execution or any execution.

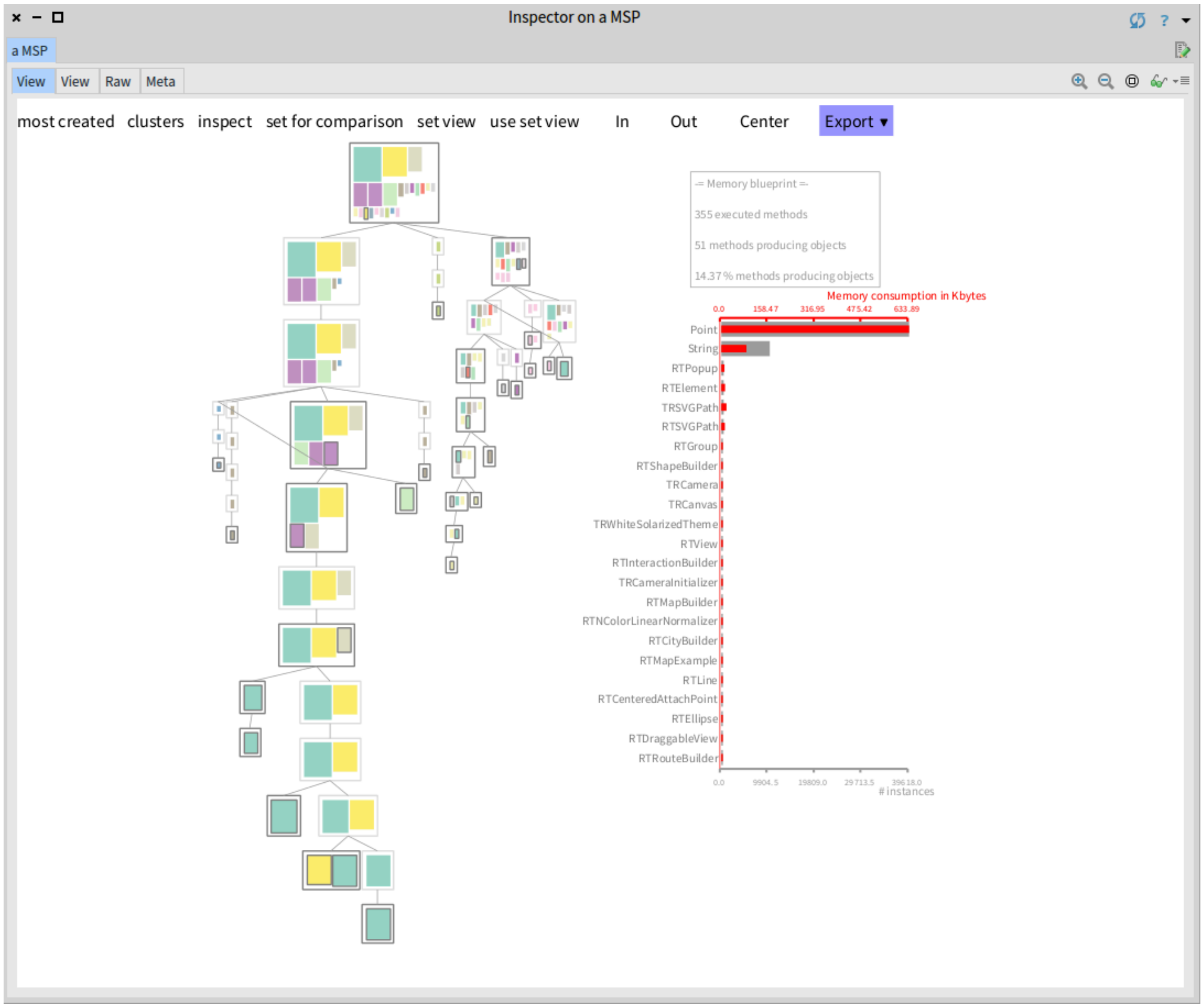


Visualizing the production of objects:

```
(MSP profile: [ RTMapExample new exampleWorldPopulation ]
onPackagesNamed: #('Roassal2' 'Trachel') ) inspect
```

Ongoing and Future work

We are aware of the complexity of the Roassal API. Data Studio is an effort to alleviate this complexity by providing an expressive graphical user interface to build visualizations.

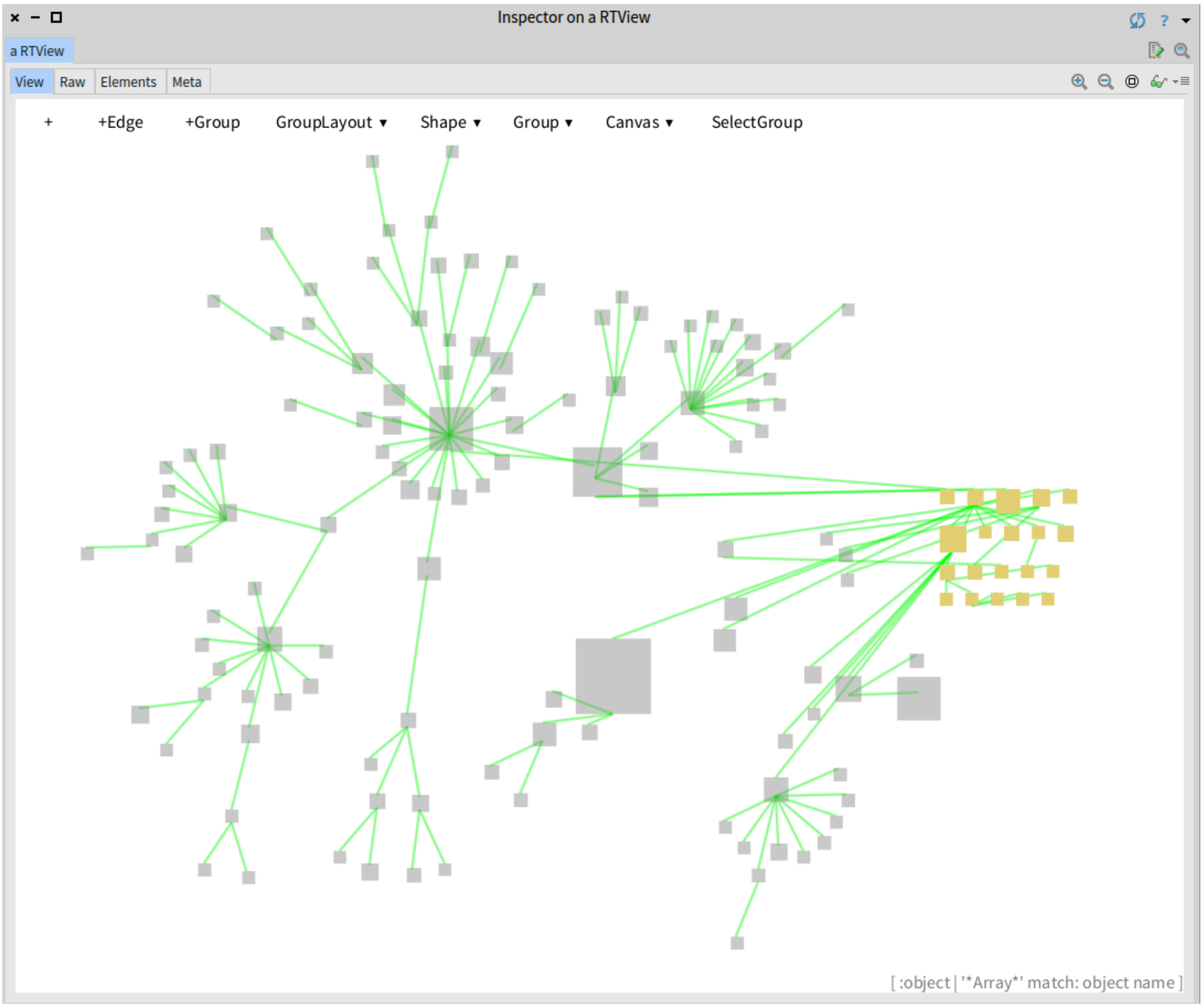


-= Memory blueprint =-

355 executed methods

51 methods producing objects

14.37 % methods producing objects



[:object | '*Array*' match: object name]