



# New generation Qt control components for HLS

Giacomo Strangolino



Elettra Sincrotrone Trieste

# cumbia-qtcontrols-ng

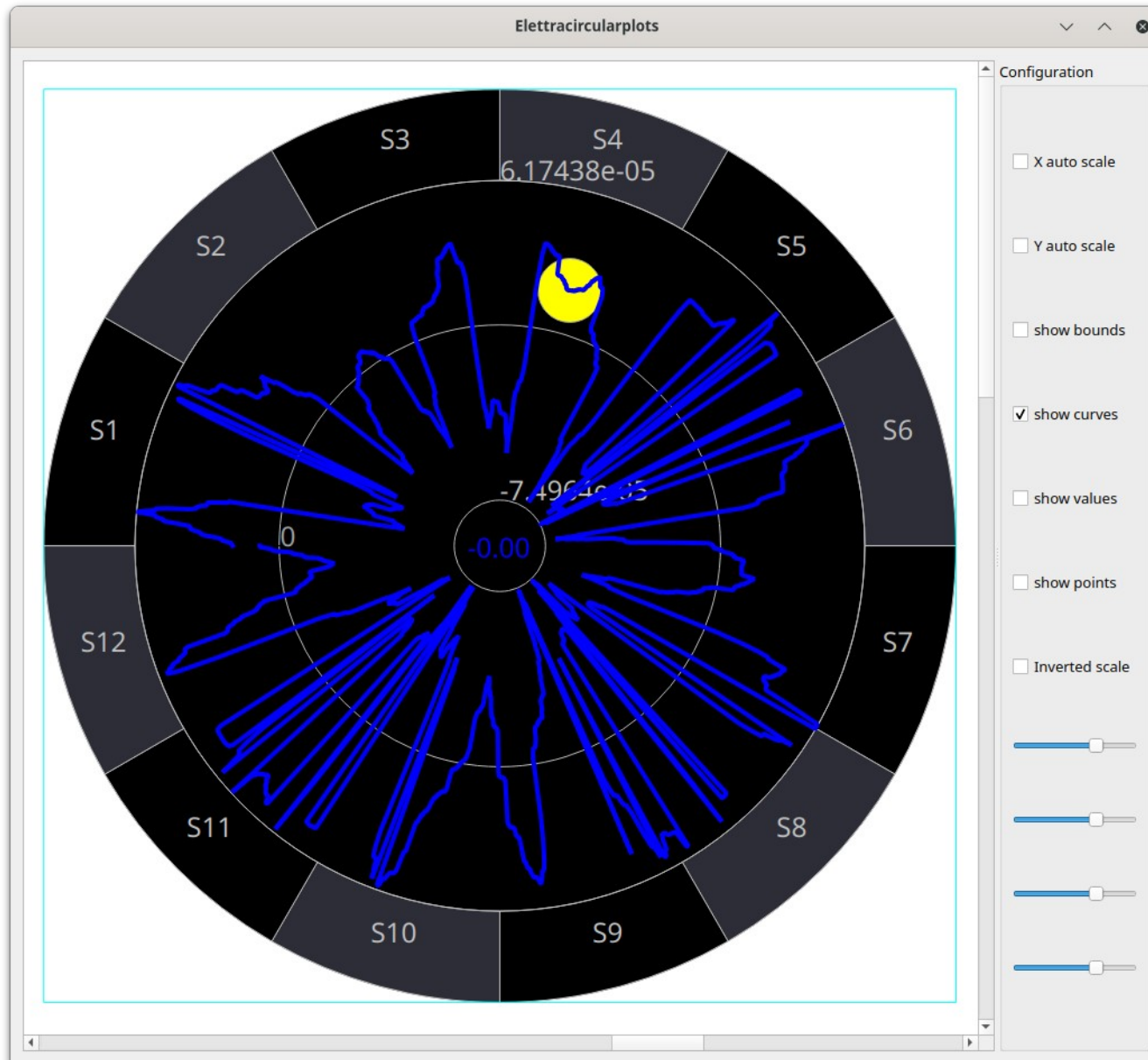
The new generation of cumbia controls focuses on the design of **high level software**, offering components for *custom-tailored data visualization* as well as for *synoptic applications*

Designed to be

- ✓ scalable
- ✓ Integrated into SVG
- ✓ Highly customizable
- ✓ Support for classical widget technology
- ✓ *GraphicItems* and widgets share *common* paint and configuration engine

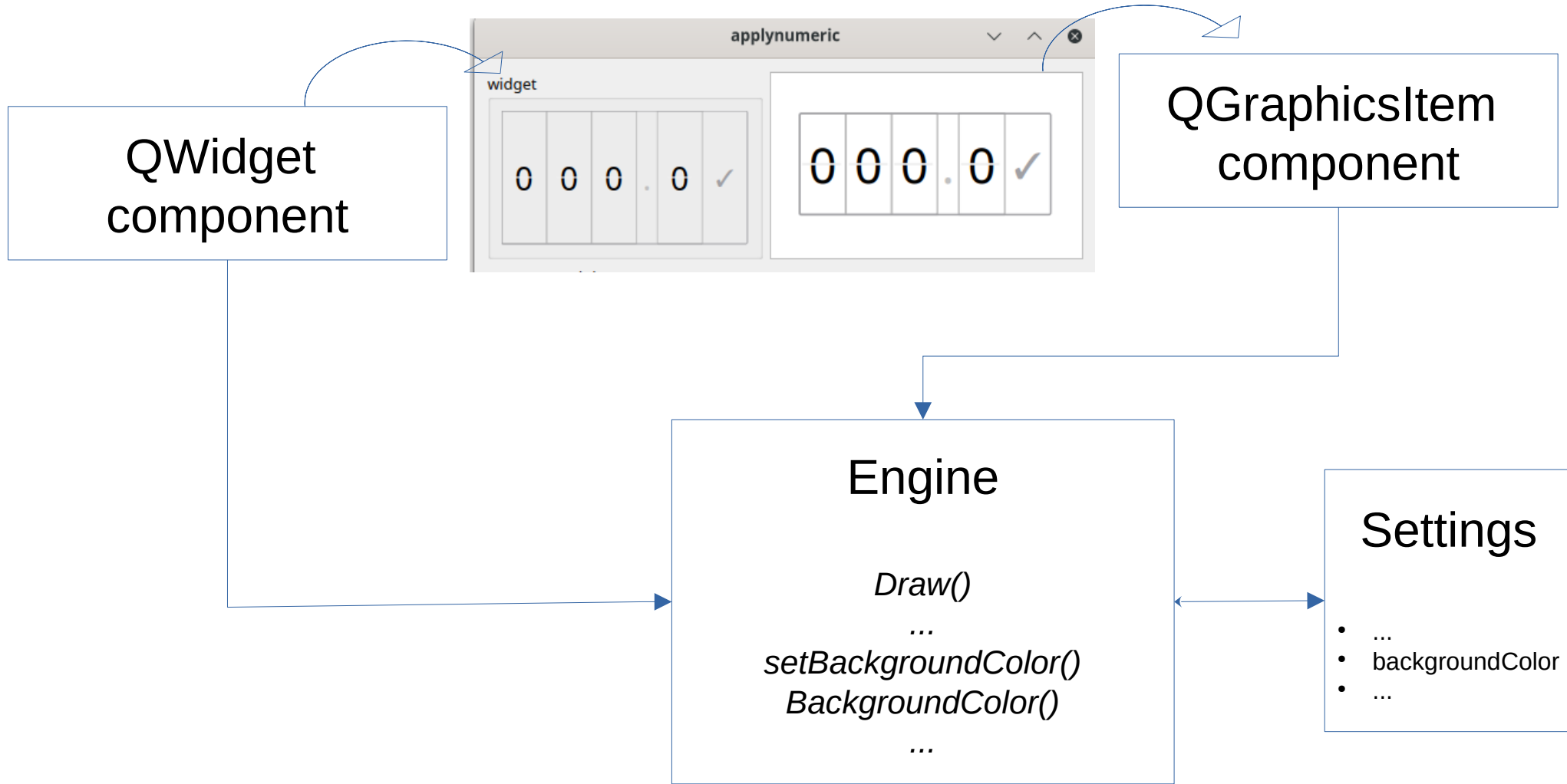


# cumbia-qtcontrols-ng



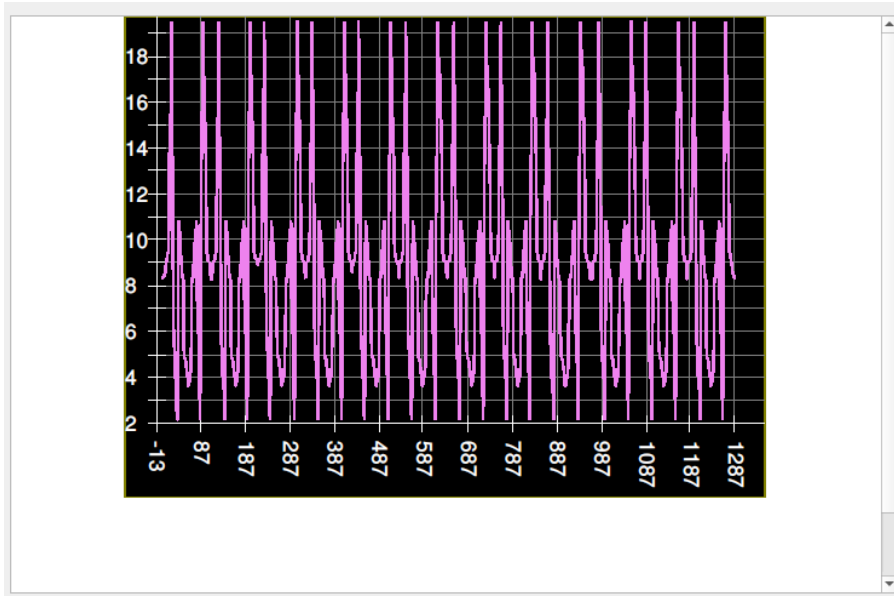


# cumbia-qtcontrols-ng

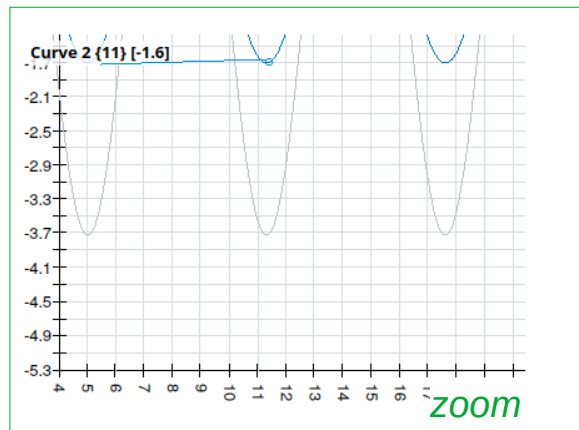
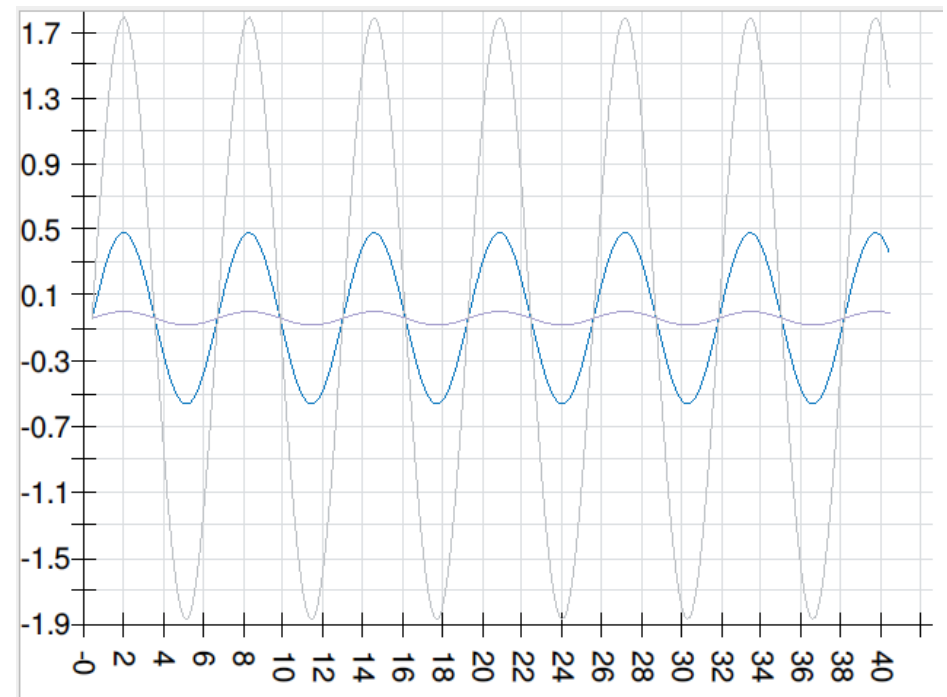




## Components – Cartesian plot



*QGraphicsplotItem*

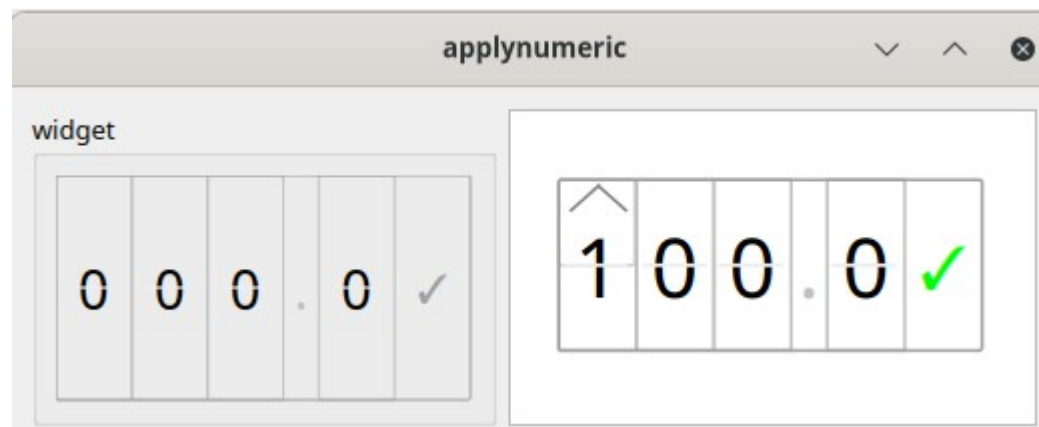


## Components – Apply numeric

*QuNumericWidget*

*QuNumericItem*

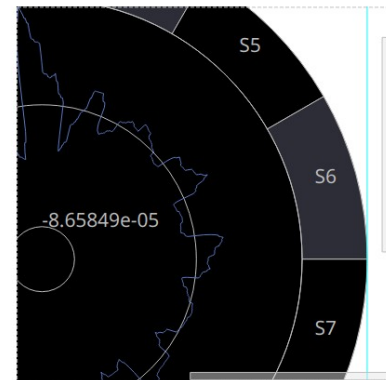
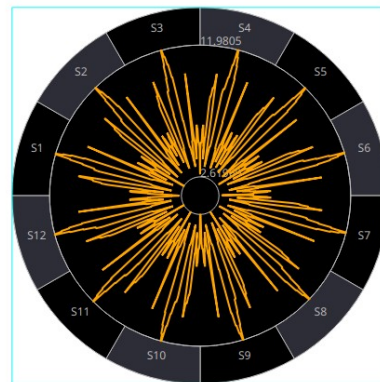
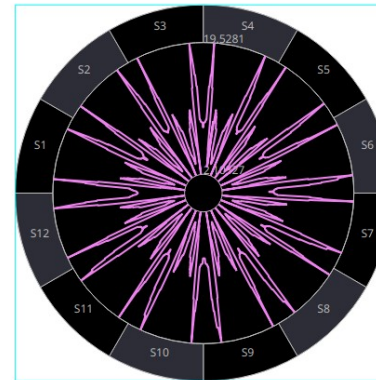
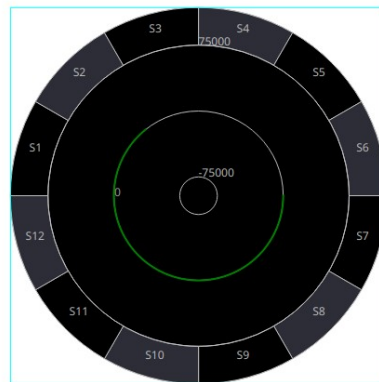
*Cumbia-qtcontrols “old”  
QuApplyNumeric*



- Touch
- Auto repeat

## Components – Circular plot

Represents values on a circumference (storage ring)



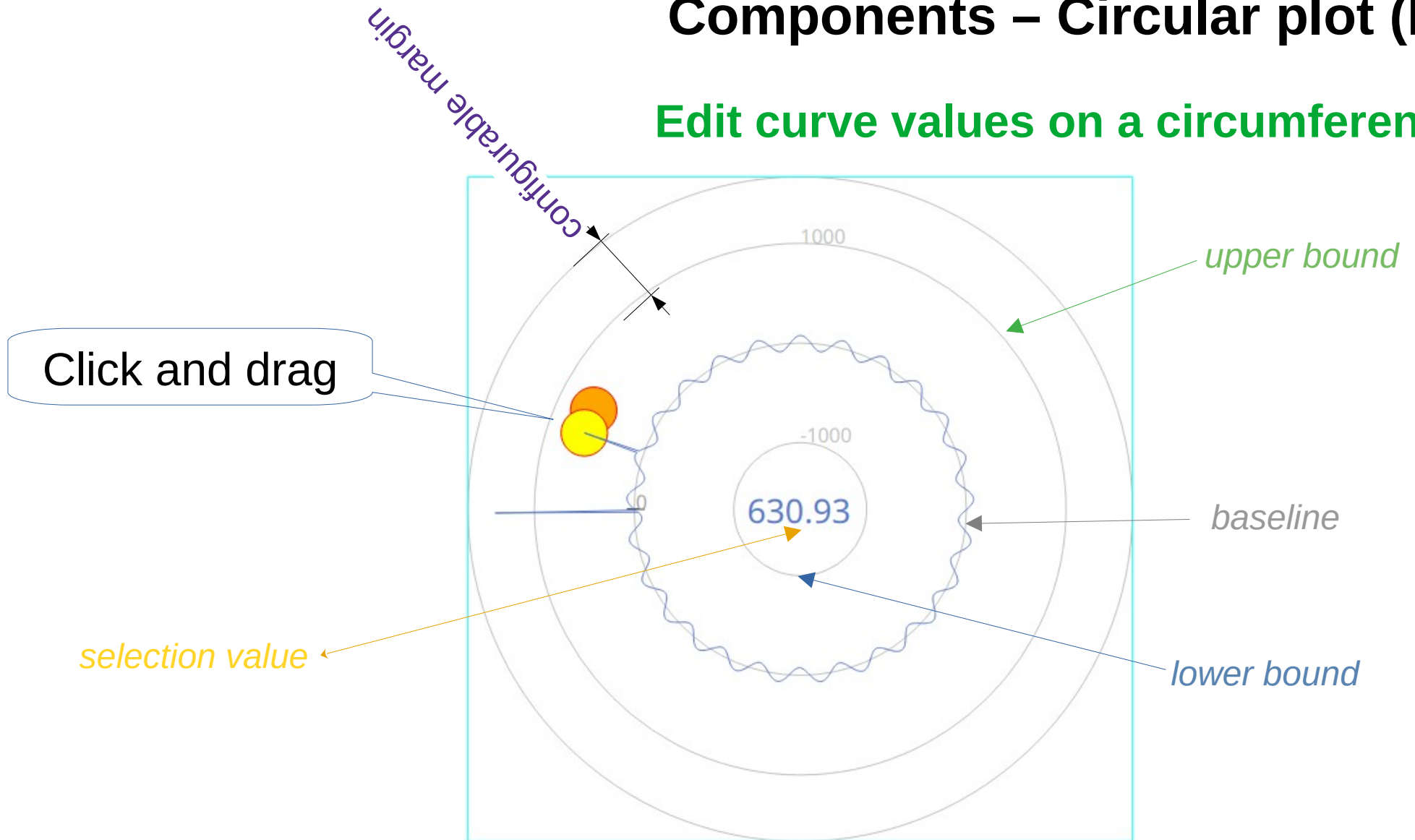




# cumbia-qtcontrols-ng

## Components – Circular plot (II)

Edit curve values on a circumference



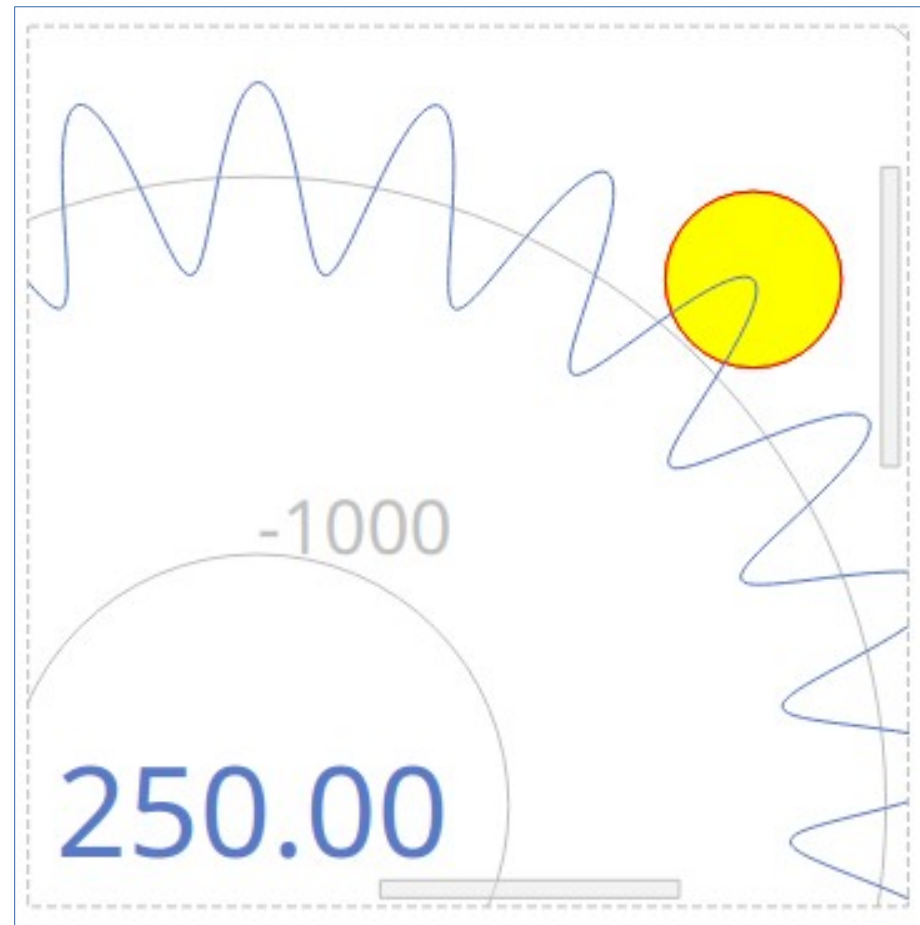
## Generic zoomer

Zooms everything (almost<sup>1</sup>) automatically

- ✓ Can be applied to any *QGraphicsItem* or *QWidget*
- ✓ Activated by selecting an area on the item or widget
- ✓ Movable
- ✓ Stackable

<sup>1</sup> a minimum impact on the object's "paint" code is an objective

## Generic zoomer

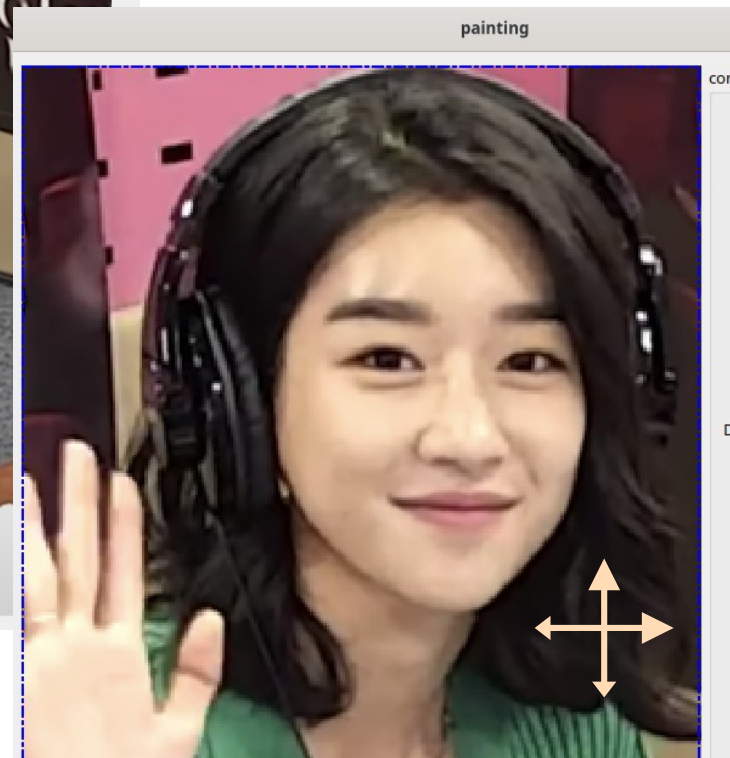


*graphics item*



# cumbia-qtcontrols-ng

## Generic zoomer



*widget*

# qumbia-svg

The **Qt SVG** C++ module provides functionality for handling *SVG* images.

The *cumbia* engine can be used to animate any *SVG* element, grouped in user defined *QGraphicsSVGItem* objects.

# qumbia-svg

## Connect with the Tango and Epics (and more...) control system software

SVG elements in the drawing can be connected to values obtained from the available cumbia engines and their properties changed accordingly.

In several cases, the connections defined and the type of attributes in the SVG elements allow for automatic changes in the representation of the object within the drawing.

In more complex scenarios, the programmer will map values from the engines to values of the attributes in the SVG DOM document.

# qumbia-svg

Whenever possible, SVG attributes changes reflect data variations automatically

```
<svg:g
  id="rect_group"
  item="true">
  <svg:rect
    y="16.730942"
    x="43.138882"
    height="53.787224"
    width="126"
    id="rect"
    style="fill:#00ff00;fill-rule:nonzero">
    <read
      attribute="style/fill"
      src="$1-&gt;State"
      hint="state_color" />
  </svg:rect>
  <svg:rect
    y="3.0934055"
    x="43.439999"
```

If we want a dedicated Qt SVG item

Source of data

Hint to change the color  
According to the Tango state



# qumbia-svg

Whenever possible, SVG attributes changes reflect data variations automatically

```
<svg:text  
  id="text6205"  
  y="51.245438"  
  x="70.177216"  
  style="font-style:normal;font-weight:normal;  
  xml:space="default">  
  <svg:tspan  
    style="stroke-width:0.26458332"  
    id="tspan6201"  
    y="51.245438"  
    x="69.090614">TEXT HERE</svg:tspan>  
  <link  
    src="$1/double_scalar"</>  
</svg:text>
```

We have a text element here

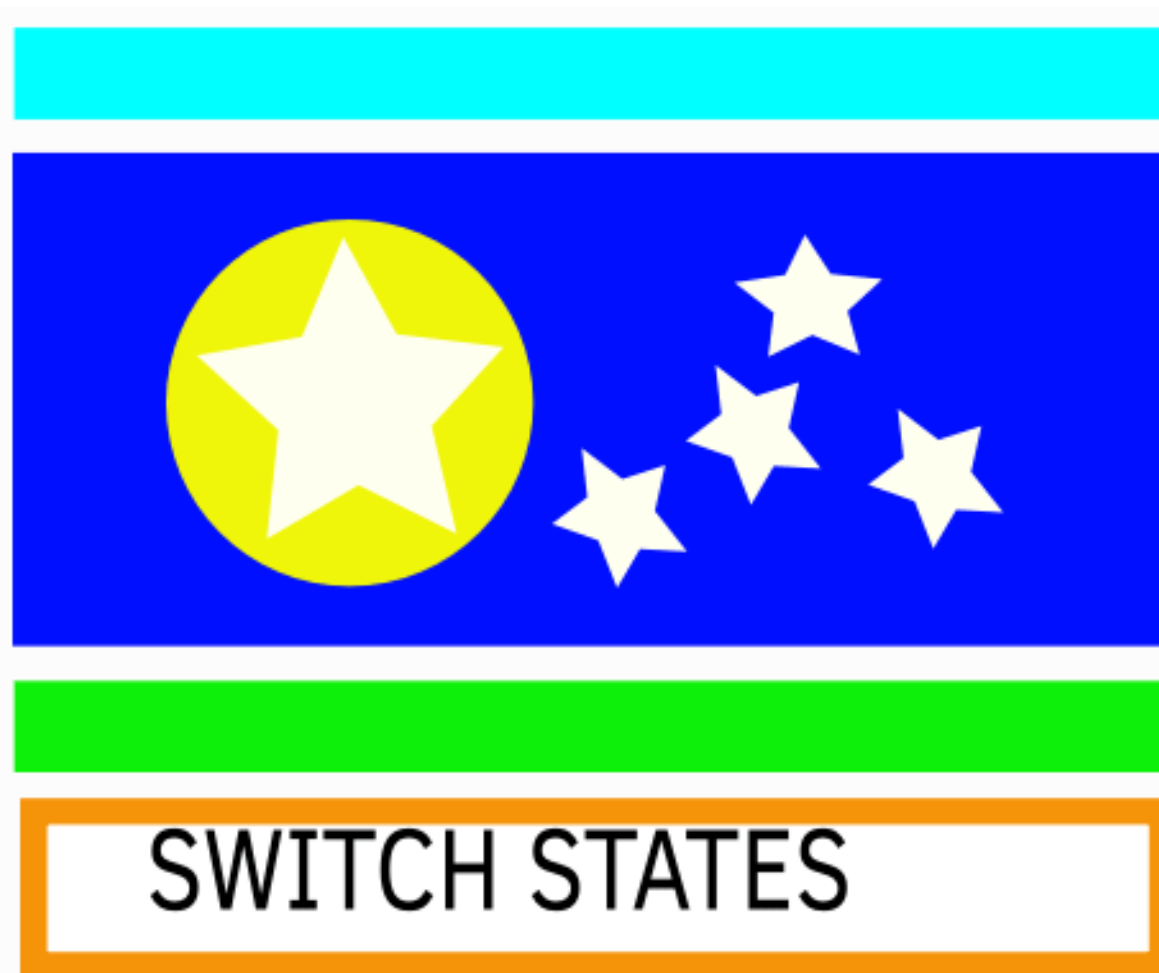
Source of data

The text element shows the current value of double\_scalar



# qumbia-svg

Whenever possible, SVG attributes changes reflect data variations automatically



## Helper application support

Helper applications can be defined by the *helper* attribute in any item. As an alternative, they can be deduced from the source connected to the element, if a single one is defined (and if the engine in use supports this feature, e.g. *Tango* does).

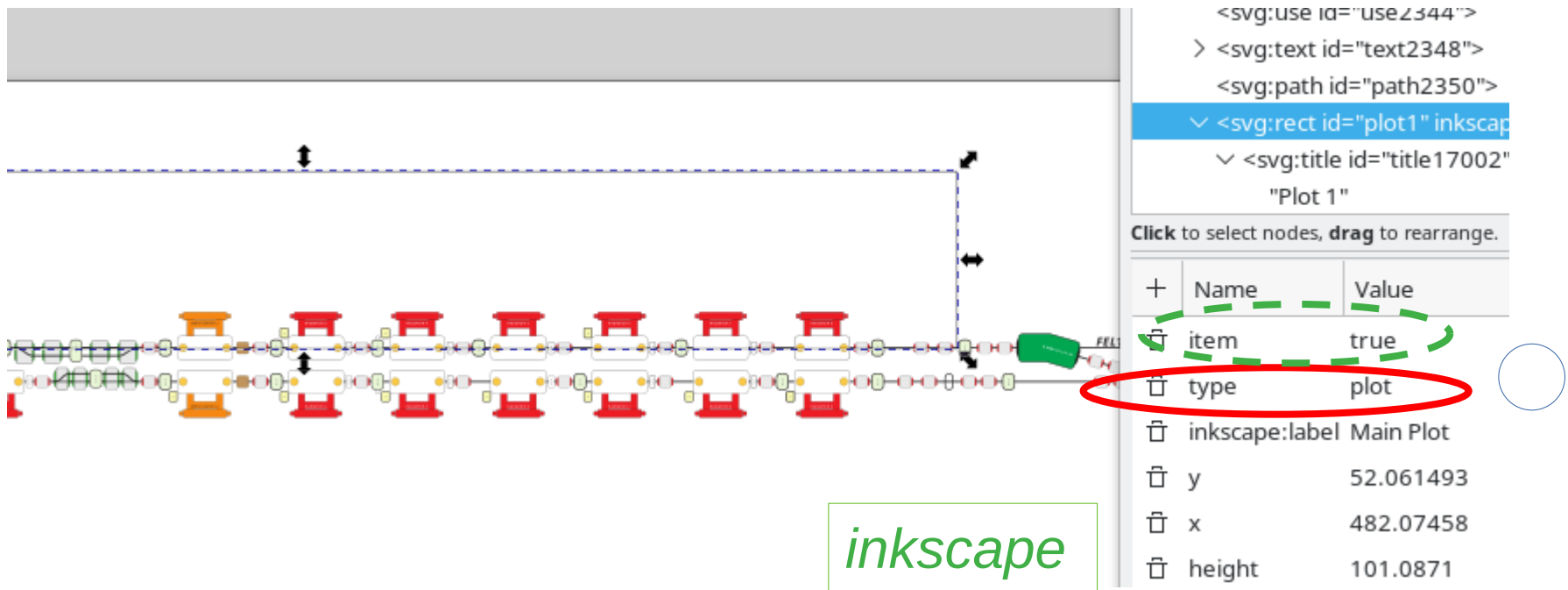
## Item event handling

Events on an item are notified by Qt signals. This includes clicks (left button and contextual menu events). Clicks can target a *write* operation, while *contextual menus* will by default make available helper applications (if defined) and pop up dialogs to perform more complex writings (for example, write a scalar number or change a text)

# qumbia-svg-extensions

Provides custom items and factories to be included in SVG

Place a *rect* on the SVG, set a *type* attribute and get a custom item of the given type at runtime



inkscape

```
<svg:use id="use2344">  
> <svg:text id="text2348">  
<svg:path id="path2350">  
v <svg:rect id="plot1" inkscape:  
v <svg:title id="title17002"  
"Plot 1"
```

	Name	Value
+	item	true
+	type	plot
+	inkscape:label	Main Plot
+	y	52.061493
+	x	482.07458
+	height	101.0871

# qumbia-svg-extensions

Provides custom items and factories to be included in SVG

```
svgview = new QuSvgView(this);  
m_qu_svg = new QuSvg(svgview);  
QuSvgItemEventHandler *item_event_han =  
    new QuSvgItemEventHandler(svgview);  
item_event_han->addActionProvider(new QuSvgHelperAppActionProvider(this, m_qu_svg));  
item_event_han->addActionProvider(new QuSvgWriteActionProvider(this, m_qu_svg));  
connect(item_event_han, SIGNAL(error(QString,QString)), this, SLOT(onItemError(QString,QString)));  
m_qu_svg->init(cu_pool, m_ctrl_factory_pool);  
lo->addWidget(svgview, 0,0, 5, 5 );
```

Handle mouse events

init qumbia-svg with engines

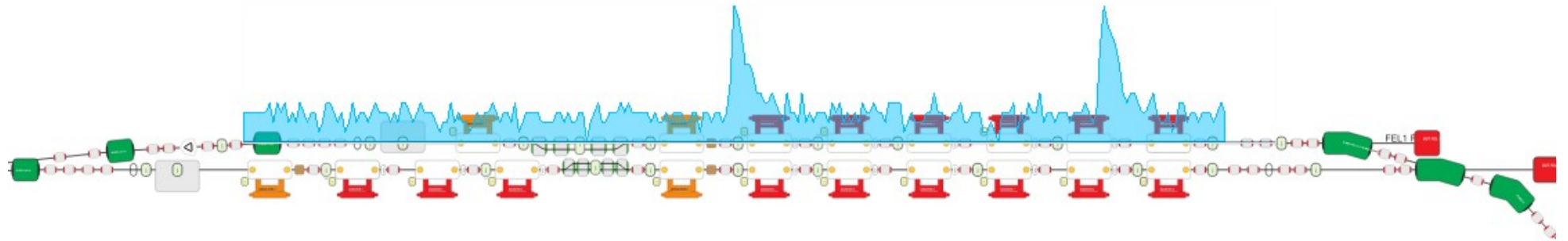
```
QuSvgItemPlotFactory *plot_factory = new QuSvgItemPlotFactory;  
svgview->extension_factory()->registerItemFactory("plot", plot_factory);
```

```
// mloader.modules() to get the list of loaded modules  
// cumbia  
bool ok = m_qu_svg->loadFile(":/uh-plain.svg");  
if(!ok) {
```

Matches type attribute

# qumbia-svg-extensions

## QuSvgItemPlot



*Application at runtime*

## Touch interaction

Touch events will enhance the user experience, through web interfaces and, on a smaller scale, Qt applications.

A 55 inches touch screen will flank smaller handheld devices in the control rooms. Ergonomic studies encompassing monitor positioning and angling, and optimal GUI controls size are under way.

# Roadmap (II)

## Implications (II)

- ✓ Classical atomic panels will be used occasionally for diagnostics
- ✓ Large quantities of data are expected to be visualized at a refresh rate of at least 10 or 20Hz.
- ✓ Touch will be used to edit curves and feedback is expected at the very same rate

## Implications (III)

- ✓ Powerful network and processing hardware
- ✓ Efficient hardware access (optimize ethernet based communication protocols between FPGA and the control system)
- ✓ Less *memcpy* (minimize copies in general)
- ✓ Low level data processing for lighter apps



- ✓ Interpreted languages only in specialized applications where the the low-level control system software lack the necessary computational libraries or algorithms.



# Roadmap (IV)

## New fancy controls

Classic *shades of gray* buttons and controls will be replaced by a new generation of more beautifully drawn and scalable components.





Elettra  
Sincrotrone  
Trieste

Thank you!





Elettra  
Sincrotrone  
Trieste



[www.elettra.eu](http://www.elettra.eu)