Extending the gridSVG package

Paul Murrell

March 9, 2023

Introduction

It is sometimes useful or necessary for the user/developer to define a new grob (or gTree) class. The gridSVG package only knows about the classes in the **grid** package (basic graphical primitives, gTrees, plus a few others like frames and cell grobs). This means that gridSVG cannot export, or animate, or garnish new grob classes without additional information. This document describes how to write methods for a new grob class to work with gridSVG.

```
> library(grid)
> library(gridSVG)
```

The assumption in this document is that a new class has been made because there is a need to have a special `drawDetails()` method (because what the class draws has to be determined at drawing time rather than at grob creation time).

There should not be any need for the information in this document if you have grobs or gTrees that are entirely defined at time of creation (so do not need any `drawDetails()` methods). Those cases should be handled by the gridSVG package already. Users wanting to garnish or animate pieces of your scene should be able to get at whatever components of the scene that they need using gPaths (assuming that you have named all your grobs properly).

Simple class example

A simple case is one where the grob being created (at drawing time) is just a simple graphical primitive. For example, the following (contrived) code defines a new class of (text) grob, a `timegrob` class, that writes out the time that its text is generated (see Figure 1).

```
> tg <- grob(name="tg", cl="timegrob")
```

Because we will be able to reuse it later, we will write a function to create the grob at drawing time.

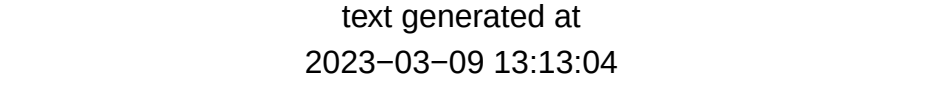

Figure 1: A simple new grob class that draws the time that its text was generated.

```
> timegrob <- function(x) {
+     textGrob(paste("text generated at", Sys.time(), sep="\n"),
+              gp=x$gp, name=x$name)
+ }
```

The `drawDetails()` method for this new class just calls the `timegrob()` function to create a text grob and then draws that grob.

```
> drawDetails.timegrob <- function(x, ...) {
+     grid.draw(timegrob(x))
+ }
```

Drawing the `timegrob` object calls the `drawDetails()` method above.

```
> grid.draw(tg)
```

The important thing about this class is that the grob that is being generated in the `drawDetails()` method is just a simple `text` grob.

Not simple class example

A slightly more complex case is when a new grob class actually generates a gTree in its `drawDetails()` method. For example, the following code defines a new class, called `boxedtext`, that creates a gTree containing text and a bounding box (at drawing time so that the box can be created from the current size of the text, in case the text has been edited; see Figure 2).

```
> bt <- grob(x=unit(.5, "npc"), y=unit(.5, "npc"),
+                   label="hi", name="bt", cl="boxedtext")
```

Again, it will be useful to have a function that creates the gTree.

```
> boxedtext <- function(x) {
+     tg <- textGrob(x$label, x$x, x$y,
+                    name=paste(x$name, "text", sep="."))
+     rg <- rectGrob(x$x, x$y,
+                    width=grobWidth(tg) + unit(2, "mm"),
```

Figure 2: A not simple new grob class that draws text with a bounding box.

```
+                     height=grobHeight(tg) + unit(2, "mm"),
+                     name=paste(x$name, "rect", sep="."))
+     gTree(children=gList(tg, rg), gp=x$gp, name=x$name)
+ }
```

The `drawDetails()` method is again very simple.

```
> drawDetails.boxedtext <- function(x, ...) {
+     grid.draw(boxedtext(x))
+ }
```

Drawing the `boxedtext` grob call the `drawDetails()` method whch creates a gTree, containing a text and bounding rectangle, and draws it.

```
> grid.draw(bt)
```

The important thing about this example is that it creates a gTree at drawing time.

Exporting a new grob class to SVG

If gridSVG knows nothing about a grob class then it will not export any SVG elements via `gridToSVG()`. The technique for telling gridSVG about a new class varies depending on how complex the new class is.

Exporting a simple class

The `primToDev()` generic function is the function that converts a simple grob into SVG. There are `primToDev()` methods for all of the standard **grid** graphical primitives, but for grob classes that gridSVG does not recognise, no SVG code will be generated.

If the new class only creates a simple grob at drawing time then all that is required is to define a new method for the `primToDev()` generic that generates a standard grob and calls `primToDev()` on that. For example, the following code creates a method for the simple `timegrob` class. This method simply creates the required text grob and then calls `primToDev()` on that, which exports the normal SVG code for a text grob.

```
> primToDev.timegrob <- function(x, dev) {
+     primToDev(timegrob(x), dev)
+ }
```

The `gridToSVG()` function will call this method to produce the appropriate SVG code.

```
> grid.newpage()
> grid.draw(tg)
> gridToSVG("simpleclass.svg")
```

The SVG code is shown below. One important feature is that the `id` attributes of the SVG elements are sensible. This has happened because the `timegrob()` function that we wrote sets the name of the text grob that it creates from the name of the `textgrob` object.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="simpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <g id="tg.1">
        <g id="tg.1.1" transform="translate(252, 252)" stroke-width="0.1">
          <g id="tg.1.1.scale" transform="scale(1, -1)">
            <text x="0" y="0" id="tg.1.1.text" text-anchor="middle" fill="rgb(0,0,0)" fill
              <tspan id="tg.1.1.tspan.1" dy="-4.33" x="0">text generated at</tspan>
              <tspan id="tg.1.1.tspan.2" dy="17.28" x="0">2023-03-09 13:13:04</tspan>
            </text>
          </g>
        </g>
      </g>
    </g>
```

```
  </g>
</svg>
```

Another feature of the `timegrob()` function is that it sets the `gp` slot of the text grob from the `gp` slot of the `textgrob` object. This is important to make sure that any graphical parameter settings on the `textgrob` are exported properly to SVG.

Exporting a not simple class

The solution for exporting a new class that creates a gTree at drawing time is very similar to the simple solution. We need to write a `primToDev()` method for the new class, the only difference being that this method should create a gTree (rather than just a standard graphical primitive grob).

For example, the following code creates a method for the `boxedtext` class. This just calls `boxedtext()` to create a gTree containing the appropriate text and bounding rectangle and then calls `primToDev()` on that gTree.

```
> primToDev.boxedtext <- function(x, dev) {
+     primToDev(boxedtext(x), dev)
+ }
```

The `gridToSVG()` function will now generate SVG output from a `boxedtext` object.

```
> grid.newpage()
> grid.draw(bt)
> gridToSVG("notsimpleclass.svg")
```

The SVG output is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="notsimpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
```

```
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <g id="bt.1">
        <g id="bt.text.1">
          <g id="bt.text.1.1" transform="translate(252, 252)" stroke-width="0.1">
            <g id="bt.text.1.1.scale" transform="scale(1, -1)">
              <text x="0" y="0" id="bt.text.1.1.text" text-anchor="middle" fill="rgb(0,0,0
                <tspan id="bt.text.1.1.tspan.1" dy="4.31" x="0">hi</tspan>
              </text>
            </g>
          </g>
        </g>
        <g id="bt.rect.1">
          <rect id="bt.rect.1.1" x="244.5" y="244.86" width="15.01" height="14.29" transfo
        </g>
      </g>
    </g>
  </g>
</svg>
```

Animating a new grob class

In addition to converting a **grid** scene to SVG code, the gridSVG package also provides a way to animate components of a **grid** scene. The `grid.animate()` function allows the user to select a grob by name and provide animated values for features of the grob (e.g., a set of x-values for a circle grob).

The `grid.animate()` function actually only attaches the animation information to a grob. The real action happens when `gridToSVG()` is called and that calls the `animate()` generic function. The purpose of that function is to generate `<animate>` elements in the SVG code. As with `primToDev()`, there are `animate()` methods that generate `<animate>` elements for all standard graphical primitives, but nothing will happen for a grob class that gridSVG is unaware of.

If we want a new (simple) grob class to be able to be animated, we need to write an `animate()` method for that grob class. For example, the following code defines an `animate()` method for the `timegrob` class. This is similar to the `primToDev()` method in that it creates a text grob and then calls `animate()` on that (to take advantage of the existing `animate()` method for text grobs). The only complication is that the animation information on the `timegrob` object must be transferred over to the new text grob (this is done in a brute force manner here; perhaps a nicer encapsulation will be provided in the future).

```
> animate.timegrob <- function(x, ...) {
```

```
+     tg <- timegrob(x)
+     tg$animationSets <- x$animationSets
+     tg$groupAnimationSets <- x$groupAnimationSets
+     animate(tg, ...)
+ }
```

With this `animate()` method in place, the following code draws a `timegrob` object and then animates it so that it will move left-to-right across the screen.

```
> grid.newpage()
> grid.draw(tg)
> grid.animate("tg", x=c(.3, .7))
> gridToSVG("animsimpleclass.svg")
```

The SVG code that is generated from this scene is shown below to show that the animation has been recorded in the SVG output (it's actually an `<animateTransform>` element rather than an `<animate>` element for text grobs).

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="animsimpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <animateTransform xlink:href="#tg.1.1" attributeName="transform" type="translate" be
      <g id="tg.1">
        <g id="tg.1.1" transform="translate(252, 252)" stroke-width="0.1">
          <g id="tg.1.1.scale" transform="scale(1, -1)">
            <text x="0" y="0" id="tg.1.1.text" text-anchor="middle" fill="rgb(0,0,0)" fill
              <tspan id="tg.1.1.tspan.1" dy="-4.33" x="0">text generated at</tspan>
              <tspan id="tg.1.1.tspan.2" dy="17.28" x="0">2023-03-09 13:13:04</tspan>
            </text>
          </g>
```

```
        </g>
      </g>
    </g>
  </g>
</svg>
```

For the case of a new gTree class, things are a little more complicated. Again, we need to write a new `animate()` method, but this function can be a bit more complicated because there may be animation information to apply to the gTree as a whole *and* animation information to apply to just the children of the gTree.

For example, the code below defines an `animate()` method for the `boexedtext` class. This creates a gTree containing a text grob and a bounding rect grob. The `groupAnimationSets` information is added to the gTree and then `animate()` is called on that to output an `<animate>` element for the entire gTree, *plus* each child of the gTree is extracted, `animationSets` information added, and then `animate()` called to output `<animate>` elements for each child of the gTree.

```
> animate.boxedtext <- function(x, ...) {
+     bt <- boxedtext(x)
+     bt$groupAnimationSets <- x$groupAnimationSets
+     animate(bt, ...)
+     # Animate the children of bt
+     btrect <- getGrob(bt, "bt.rect")
+     btrect$animationSets <- x$animationSets
+     animate(btrect, ...)
+     bttext <- getGrob(bt, "bt.text")
+     bttext$animationSets <- x$animationSets
+     animate(bttext, ...)
+ }
```

The following code makes use of this method to animate a `boxedtext` grob. The first call to `grid.animate()` makes both text and bounding rect move left-to-right across the screen. The second call to `grid.animate()` makes the whole gTree disappear after 1 second (once the text and rect have moved to the right of the screen).

```
> grid.newpage()
> grid.draw(bt)
> grid.animate("bt", x=c(.3, .7))
> grid.animate("bt", visibility=c("visible", "hidden"),
+              begin=1, duration=0.1, group=TRUE)
> gridToSVG("animnotsimpleclass.svg")
```

The resulting SVG code is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
```

```
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="animnotsimpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <animate xlink:href="#bt.1" attributeName="visibility" begin="1s" calcMode="linear"
      <animate xlink:href="#bt.rect.1.1" attributeName="x" begin="0s" calcMode="linear" du
      <animateTransform xlink:href="#bt.text.1.1" attributeName="transform" type="translat
      <g id="bt.1">
        <g id="bt.text.1">
          <g id="bt.text.1.1" transform="translate(252, 252)" stroke-width="0.1">
            <g id="bt.text.1.1.scale" transform="scale(1, -1)">
              <text x="0" y="0" id="bt.text.1.1.text" text-anchor="middle" fill="rgb(0,0,0
                <tspan id="bt.text.1.1.tspan.1" dy="4.31" x="0">hi</tspan>
              </text>
            </g>
          </g>
        </g>
        <g id="bt.rect.1">
          <rect id="bt.rect.1.1" x="244.5" y="244.86" width="15.01" height="14.29" transfo
        </g>
      </g>
    </g>
  </g>
</svg>
```

Garnishing a new grob class

The `grid.garnish()` function provides a way for users to add interactivity to a **grid** scene, by specifying (javascript) event handlers for components of the scene. More generally, the function allows **grid** grobs to be garnished with arbitrary SVG attributes.

Like `grid.animate()`, all `grid.garnish()` does is attach the garnishing infor-

mation to a grob. The appropriate SVG code is only generated when `gridToSVG()` is called. The crucial action happens in the generic function `garnish()`, which transfers the garnishing information to the SVG device that is writing out SVG code.

As should be familiar by now, there are `garnish()` methods for standard **grid** graphical primitives, but no svg attributes will be exported for grob classes that gridSVG does not know about.

Fortunately, `garnish()` methods are once again pretty easy to write. For example, the following code defines a method for the `timegrob` class. This is very much like an `animate()` method: a text grob is created, the garnishing information is added to the text grob, then `garnish()` is called on the text grob so that the method for text grobs takes care of transferring the SVG attributes to the SVG device.

```
> garnish.timegrob <- function(x, ...) {
+     tg <- timegrob(x)
+     tg$attributes <- x$attributes
+     tg$groupAttributes <- x$groupAttributes
+     garnish(tg, ...)
+ }
```

The following code shows the new method in action. A `timegrob` is drawn, then `grid.garnish()` is called to specify that a mouse click on the text should pop an alert dialog.

```
> grid.newpage()
> grid.draw(tg)
> grid.garnish("tg", onmousedown="alert('ouch')")
> gridToSVG("garnishsimpleclass.svg")
```

The resulting SVG code is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="garnishsimpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
```

```
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <g id="tg.1" onmousedown="alert('ouch')">
        <g id="tg.1.1" transform="translate(252, 252)" stroke-width="0.1">
          <g id="tg.1.1.scale" transform="scale(1, -1)">
            <text x="0" y="0" id="tg.1.1.text" text-anchor="middle" fill="rgb(0,0,0)" fill
              <tspan id="tg.1.1.tspan.1" dy="-4.33" x="0">text generated at</tspan>
              <tspan id="tg.1.1.tspan.2" dy="17.28" x="0">2023-03-09 13:13:04</tspan>
            </text>
          </g>
        </g>
      </g>
    </g>
  </g>
</svg>
```

The method is just as simple for a custom gTree class as well, as shown by the `garnish()` method for the `boxedtext` class below.

```
> garnish.boxedtext <- function(x, ...) {
+     bt <- boxedtext(x)
+     bt$attributes <- x$attributes
+     bt$groupAttributes <- x$groupAttributes
+     garnish(bt, ...)
+ }
```

The following code shows this method in action, with an additional demonstration of the difference between garnishing individual components of a **grid** scene and garnishing the overall parent component. A `boxedtext` grob is drawn, then the first call to `grid.garnish()` ensures that a mouse click on any part of the text or bounding box will produce an alert dialog. The second call to `grid.garnish()` sets up a different interaction on just the text so that moving the mouse over the text pops up a different dialog.

```
> grid.newpage()
> grid.draw(bt)
> grid.garnish("bt", onmousedown="alert('ouch')")
> grid.garnish("bt", onmouseover=c(bt.text="alert('watch it!')"),
+              group=FALSE)
> gridToSVG("garnishnotsimpleclass.svg")
```

The resulting SVG code is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="
```

```
  <metadata xmlns:gridsvg="http://www.stat.auckland.ac.nz/~paul/R/gridSVG/">
    <gridsvg:generator name="gridSVG" version="1.7-5" time="2023-03-09 13:13:04"/>
    <gridsvg:argument name="name" value="garnishnotsimpleclass.svg"/>
    <gridsvg:argument name="exportCoords" value="none"/>
    <gridsvg:argument name="exportMappings" value="none"/>
    <gridsvg:argument name="exportJS" value="none"/>
    <gridsvg:argument name="res" value="72"/>
    <gridsvg:argument name="prefix" value=""/>
    <gridsvg:argument name="addClasses" value="FALSE"/>
    <gridsvg:argument name="indent" value="TRUE"/>
    <gridsvg:argument name="htmlWrapper" value="FALSE"/>
    <gridsvg:argument name="usePaths" value="vpPaths"/>
    <gridsvg:argument name="uniqueNames" value="TRUE"/>
    <gridsvg:separator name="id.sep" value="."/>
    <gridsvg:separator name="gPath.sep" value="::"/>
    <gridsvg:separator name="vpPath.sep" value="::"/>
  </metadata>
  <g transform="translate(0, 504) scale(1, -1)">
    <g id="gridSVG" fill="none" stroke="rgb(0,0,0)" stroke-dasharray="none" stroke-width="
      <g id="bt.1" onmousedown="alert('ouch')">
        <g id="bt.text.1">
          <g id="bt.text.1.1" transform="translate(252, 252)" stroke-width="0.1">
            <g id="bt.text.1.1.scale" transform="scale(1, -1)">
              <text x="0" y="0" id="bt.text.1.1.text" text-anchor="middle" fill="rgb(0,0,0
                <tspan id="bt.text.1.1.tspan.1" dy="4.31" x="0">hi</tspan>
              </text>
            </g>
          </g>
        </g>
        <g id="bt.rect.1">
          <rect id="bt.rect.1.1" x="244.5" y="244.86" width="15.01" height="14.29" transfo
        </g>
      </g>
    </g>
  </g>
</svg>
```

Extending complex classes

A more complex case is one where a new grob class defines its own `preDrawDetails()` or `postDrawDetials()` methods. If that is the case, then it will be necessary to write a `grobToDev()` method for the class, just to get SVG exported correctly. See the existing methods for `grobToDev()` in the gridSVG source for some examples.