The Geyser Layout Manager for Mudlet

By guy

March 15, 2010

Abstract

Geyser is an object oriented framework for creating, updating and organizing GUI elements within Mudlet. This is an overview of the Geyser architecture and design.

1 Motivation

Mudlet makes the creation of label, miniconsoles and gauges a quick and easy thing. Mudlet provides a nice signal when window resize events happen. Mudlet does not provide a good framework for complex window management, which Geyser attempts to address. The name 'Geyser' was partly chosen to go with Crucible...sort of like if you were to pour some window glass into a hot crucible it might shoot up or something.

2 Main Geyser Features

Geyser is based on traditional GUI concepts and should feel similar to using Java's Swing. The biggest difference is in how positions are specified.

- All window[1] positions are specified relative to their container - nothing new there. However, window positions can also take on percentages and negative pixel and character values. For instance, a window could be constrained to have a height of 50% of its container and a width such that the window's right edge is always 20 characters from its container's right edge. See examples below and the demos/tests that come with Geyser.
- All windows under Geyser control are automatically resized as necessary when the main Mudlet window is resized. Once you create a window and assign it to a container, you never have to worry about positioning it again. Nonetheless, it's very easy to shift the container that a window is in, maintaining its constraints so that it resizes as needed according to the dimensions of the new container.
- Due to the container heirarchy, hiding a container window automatically hides all its contained windows too. The same for show. With clever construction and labels with callbacks, this lets one make complex GUI elements that are minimizable or maximizable or ...
- However, there is always some overhead for automation systems and Geyser is no exception. Fortunately, most of the overhead is during window creation and resize events - not things that happen frequently.

[1]Where *window* is a generic term and refers to any GUI element, be it a label, a container, a window, ...

Type	**Example**	**Meaning**
Raw Pixel	`width = “-20px”`	The right border of this window will always be 20 pixels from its container’s right border
Characters	`height = “10c”`	The window can hold up to 10 lines of text at the default font size
Percentage	`y = “25%”`	The top of this window will be 25% of the way down from its container’s top

Table 1: Mudlet Constraint Format

Because Geyser is OO, remember that all instance methods are invoked with the ‘`:`’ syntax, e.g.

```
l = Geyser.Label:new(...)
l:echo(“my message”)
l:hide()
```

3 Constraints format

Geyser position constraints are very simple, see Table 1. They are a string composed of a number and a format type. For example, “10px” means 10 pixels, either pixels from the origin (e.g. x or y) or a value for the width or height. A negative number indicates distance from a container’s right or bottom border depending on whether is a constraint for x/width or y/height, respectively. Percentages for width and height are in terms of the container’s dimensions and negative values are converted to the equivalent positive value. A 100% width subwindow with an x-coordinate of 10 will have part of itself displayed outside the confines of its container. There is no hard limit on maximum window size, just as with regular Mudlet windows. If a number, n, is passed as a contraint instead of a string, it is assumed to be equivalent to “$n$px”.

Any Lua table that contains entries for x, y, width and height in the proper format can be used for Geyser constructors and setting constraints. The following is a valid example:

```
{x = "20px", y = "-10c", width = "40%", height = 30}
```

4 Examples

See the Geyser→GeyserTests script. I recommend setting up an alias that matches on `^!(.*)` and executes `assert(loadstring(matches[2]))()` so that arbitrary Lua expressions (like running the tests) can be done right from the Mudlet input line by prefixing them with a ‘`!`’.

5 Geyser Class Heirarchy

See Figure 1 for a visual overview. As currently distributed, Geyser only includes the same basic window types that come with Mudlet. Using Geyser, more complex windows and containers can be created (e.g. a Chat window class).

Each subclass contains all the parameters (including defaults) and methods of its parent class, though they can of course be overridden. If shadowing occurs, the parent’s version can be accessed through the `Geyser.Class.parent` table element. To change a parameter from its default during creation, just include the value you'd like it to be in the constraints table. By default

For specifics, see the luadoc files.

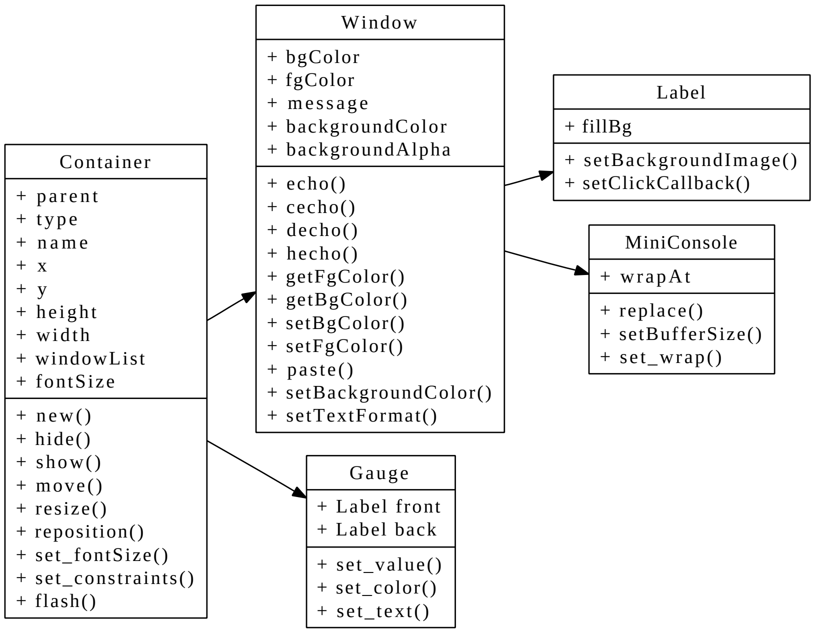

Figure 1: Geyser Class Heirarchy

5.1 Container

The `Container` class is the root type of window. The `Geyser` table/namespace has the functionality of a `Container` and represents the main Mudlet window and control of other windows therein. Notice that because `Container` is the root class, that all types of Geyser windows can act as a container - labels anchored within miniconsoles anchored within gauges are possible (but not necessarily desireable).

5.2 Window

Abstract class meant to be subclassed into Label and MiniConsole that contains functions common to both.

5.3 Label

Based on the primitive Mudlet label, nothing new.

5.4 MiniConsole

Based on the primitive Mudlet miniconsole, nothing new.

5.5 Gauge

This is a composite window. `Gauge` duplicates the functionality of the built in Mudlet gauges, but in a clean and easily extended way. Internally, a Geyser `Gauge` is a container holding two `Labels`, `front` and `back`, which are initially scaled to fill the entire `Gauge` container. Hence, a gauge, `g`, can be given callbacks with the `g.front:setClickCallback()` method.

The `backgroundColor` parameter initially sets the colors of the gauge, but of course the front and back components can be accessed individually as labels for high control over their looks. Gauges can be horizontal or vertical and decrease in value left to right, right to left, down up or up down depending on the value of the `orientation` parameter.

6 Limitations

- As of this writing, I haven't had the time to create OO wrappers for all the myriad window related functions provided by Mudlet. Basic functionality should be strong though.
- Similarly, I haven't done extensive testing that all combinations of classes/methods will work; the 3 simple tests work for me. Please lavish me with your bug reports.
- Function/Method names are inconsistent. Mudlet seems to use `camelCase`, and so I've kept the `camelCase` names for methods that are just wrappers. I've used `this_style` for getters and setters that are specific to Geyser. I'd love feedback on which style should be standardized and used throughout.