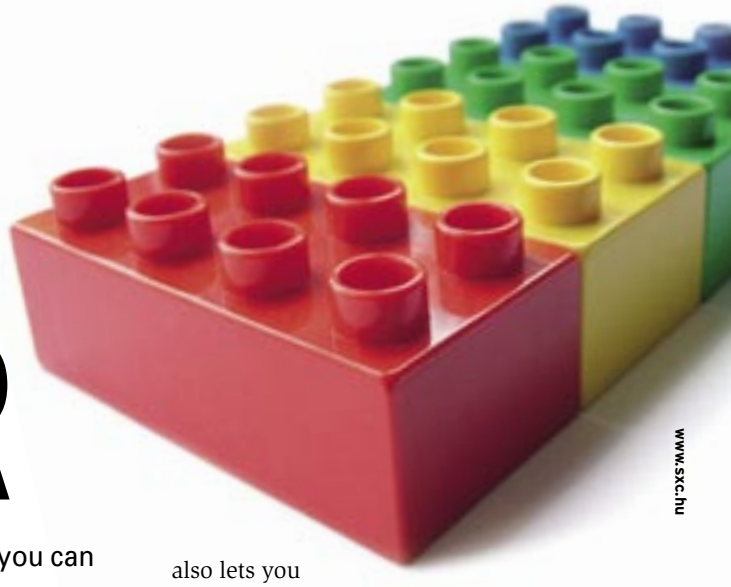
Creating a SuperKaramba theme

WIDGET BUILDER



If you can't find the SuperKaramba theme you're looking for, you can always build your own. **BY HAGEN HÖPFNER**

Pretty wallpaper is fine as far as it goes, but if you want desktop frills that actually do work for you, such as displaying weather reports or monitoring system data, you'll need a tool like SuperKaramba. SuperKaramba [1] is a KDE-based tool that lets you create helpful widgets for your desktop. The SuperKaramba website refers to these interactive widgets as "custom eye candy," but SuperKaramba applets are more commonly called *themes*. You'll find many pre-built SuperKaramba themes on the Internet, but you can also create your own themes. A SuperKaramba theme can take the form of a game, a system monitor, a desktop music directory, or even a custom toolbar.

Previous articles in Linux Magazine described how to get started with SuperKaramba. For additional information, see [2]. (These articles are currently available through Linux Magazine's convenient online archive.) This article is a brief workshop on how to create your own SuperKaramba theme.

Finding the Software

The current SuperKaramba version is number 0.37. If you prefer not to build SuperKaramba yourself, you will find RPMs and Debian archives at [3] and [4]. Running *superkaramba* opens the theme dialog (Figure 1) and drops a control icon into the KDE kicker.

SuperKaramba needs themes to bring the desktop to life. Designs are available for many typical applications. Pressing the *New Themes…* button opens a selection dialog, which offers you a number of the themes listed at [5]. When you select an entry, information about the functionality and a screenshot give you a first glimpse at your options. SuperKaramba downloads the required files and installs them in the *~/.kde/share/apps/superkaramba/themes* directory. When you close the theme dialog, the installed themes appear in the SuperKaramba window, where you can launch them.

You can run multiple parallel themes on a single desktop, starting and removing the themes individually as required. The right-click menu also lets you specify if you want to keep the ability to move a theme. Some themes have their own configuration dialogs. For example, you can configure the weather report to match your current location.

Creating Your Own Themes

To create your own theme, you need to create a file, say *my_theme.theme*, which you can open in SuperKaramba as a local file. As graphics and Python scripts can be added to the layout file, you will probably want to store all of these components in a single directory.

The *theme* file comprises three components: general commands specify the theme geometry and define interactive areas. Sensors help to read system parameters, such as the current CPU load. Finally, meters display the measured values. Listing 1 illustrates the makeup of a *theme* file.

The first line in Listing 1 positions the theme in the lower left corner of the desktop, sets a width of 200 pixels and a height of 400 pixels, specifies that the

Figure 1: Use SuperKaramba's theme dialog to install new themes.

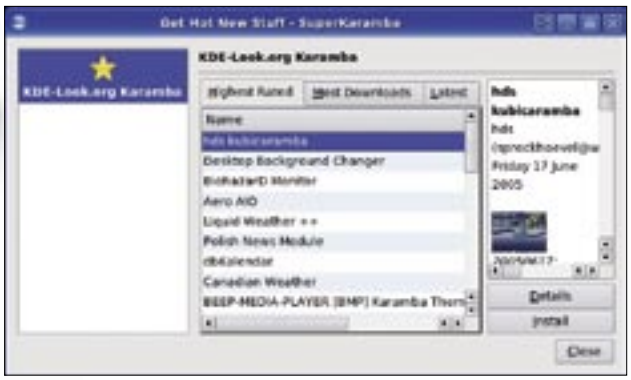


Figure 2: Before you start building your own SuperKaramba theme, you might like to check out some of the themes available on the Internet.

theme can be moved, and sets a refresh time of 1000 milliseconds. You could set the position for the theme by specifying *x* and *y* offsets from the top left corner of the desktop.

The *BOTTOM = true* parameter overrides the horizontal position. In a similar fashion, the *RIGHT = true* option positions the theme on the right margin. *ONTOP = true* prevents other windows hiding the theme. *TOPBAR = true* and *BOTTOMBAR = true* dock the theme at the top or bottom of the screen, just like the KDE kicker, without allowing maximized windows to hide it.

The second line in Listing 1 sets the default font type, size, and color for all text elements associated with the theme. The color code defines a combination of values for red, green, and blue with a maximum value of 255. The example in Listing 1 uses *255, 255, 255*, which is the code that represents white.

Sensors and Meters

Lines 5 and 6 in Listing 1 create two text meters that derive their content from the time sensor (*sensor = time*). Line 5 outputs the time and overrides the default font size, which was defined previously. It is a good idea to use *< GROUP >* tags to group similar elements; this improves readability and lets you specify a position for the group.

The first group in our sample theme is located at *x = 10* and *y = 10*; besides the time and date meter, it contains a definition for the interactive area. Line 7 tells SuperKaramba to call the KDE time setting command when the 120 by 34 pixel area is double-clicked. The *preview = true* parameter draws a frame round the theme to indicate that it is still undergoing testing.

The *format* parameter defines the output content and format of a field depending on the sensor you query. The "*hh:mm:ss*" format parameter tells the time sensor to give you hour:minute:second formatted time output, whereas *format = "ddd dd.MM.yyyy"* gives you the weekday, the day of the month, the month and the year. Besides the time sensor, SuperKaramba has the following sensors:

- *cpu*: System load
- *disk*: Disk usage for mounted file systems
- *memory*: Free and used main memory
- *network*: Incoming and outgoing network traffic
- *noatun*: Information from a running noatun process
- *program*: Standard output from any program

Figure 3: Thanks to the Python interface, programming themes with extended functionality is quick and easy.

- *sensor*: Processes the LM sensor [10] output
- *textfile*: Continually reads a text file
- *uptime*: System uptime
- *xmms*: Information from an XMMS process

The second group in Listing 1 illustrates the use of some of these sensors in combination with various output devices. Line 12 reads you the free memory in Megabytes without buffer and cache memory. Line 14 uses the *bar.png* image to create a bar diagram that indicates the current CPU load.

Table 1: SuperKaramba Callback Functions

Function	Trigger	Parameters passed
initWidget(widget)	Create the SuperKaramba widget	
widgetUpdated(widget)	Update the theme	Update interval from the *.theme* file
widgetClicked(widget, x, y, button)	Mouse click on theme	*x* and *y*: coordinates for click relative to theme; *button*: mouse button used
widgetMouseMoved(widget, x, y, button)	Mouse movement within the theme	*x* and *y*: coordinates for click relative to theme; *button*: mouse button used
menuItemClicked(widget, menu, id)	Click on menu item	*menu*: Menu handle; *id*: Menu item handle
menuOptionChanged(widget, key, value) (*true* or *false*)	Configuration menu item called in theme	*key*: Menu item handle; *value*: new value for menu item
meterClicked(widget, meter, button)	Click on meter device	*meter*: Meter device handle; *button*: mouse button used
commandOutput(widget, pid, output)	Programm called via	*executeInteractive()*, if output goes to *stdout pid*: Process ID for program; *output*: Output text
itemDropped(widget, dropText)	Objects dropped on theme in drag & drop operatons	*dropText*: Text for object (e.g. URL).
startupAdded(widget, startup)	KDE starts an application	After launching completes, the *startupRemoved()* and *taskAdded()* signals follow
startupRemoved(widget, startup)	See *startupAdded()*.	
taskAdded(widget, task)	See *startupAdded()*.	
taskRemoved(widget, task)	End of application program	
activeTaskChanged(widget, task)	Moves an application to the foreground	

Listing 1: my_theme.theme

```
01 KARAMBA x=0 BOTTOM=true w=200 h=400 LOCKED=false INTERVAL=1000
02 DEFAULT font="Sans" fontsize=10 shadow=2 color=255,255,255
03
04 <GROUP> x=10 y=10
05   TEXT x=12 y=0 sensor=time fontsize=12 format="hh:mm:ss"
06   TEXT x=12 y=15 sensor=time  format="ddd dd.MM.yyyy"
07   CLICKAREA x=0 y=0 w=120 h=34 onclick="kdesu kcmshell clock"
08 </GROUP>
09
10 <GROUP> x=10 y=50
11   TEXT  x=12 y=0  value="MEM"
12   TEXT  x=45 y=0  sensor=memory format="%fmb MB"
13   TEXT  x=12 y=15 value="CPU"
14   BAR   x=45 y=15 sensor=cpu  path="bar.png"
15   TEXT  x=12 y=30 value="IN"
16   GRAPH x=45 y=30 h=12 w=70 color=255,255,255 points=100
   sensor=network device="eth0" format="%in"
17   IMAGE x=0  y=50 path="background.png"
18 </GROUP>
```

A graph is used to evaluate incoming network traffic in Line 16. The graph is 12 x 70 pixels and white. The last line in this group uses the *IMAGE* output device to display an image. The *PATH* optionally accepts a URL, which allows you to embed an image from the Internet. Documentation on the formatting options for the various output devices is at [9].

SuperKaramba and Python

The Python interface [11] helps make a theme more flexible. Callback functions add the ability to react to events triggered on the KDE desktop (Table 1). To allow a theme to access a Python file, the file must reside in the same directory as the *theme* file; it also has to use the same basic name as the theme but with the *.py* extension. The template at [12] is a useful starting point for your own development, as it integrates a fair number of callback functions.

Listing 2 shows a Python script that allows the user to replace the image displayed in the SuperKaramba theme using drag & drop. It starts by importing additional modules: *import karamba* loads the SuperKaramba module and *import string* adds string manipulation functions for text variables to the script.

Listing 2: my_theme.py

```
01 import karamba # Imports the Karamba API
02 import string  # String manipulation functions
03 image=0
04
05 def initWidget(widget):
06     global image
07     karamba.acceptDrops(widget)
08     image=karamba.createImage(widget, 12, 102, "bar")
09     karamba.hideImage(widget,image)
10     karamba.redrawWidget(widget)
11
12 def itemDropped(widget, dropText):
13     global image
14     image_link=string.split(dropText, "file:" ,1)[1]
15     karamba.deleteImage(widget,image)
16     image=karamba.createImage(widget, 12, 102, image_link)
17     karamba.resizeImage(widget, image, 176, 286)
```

The *initWidget(widget)* callback function is automatically called when the theme window is generated. *karamba.acceptDrops(widget)* sets up the widget to accept images that users drag & drop to the window. At the same time, *karamba.createImage* creates an image that is used as a placeholder, which explains why it is hidden at first (*karamba.hideImage(widget,image)*) until *karamba.redrawWidget(widget)* refreshes the display.

The second callback function in Listing 2 is *itemDropped(widget, dropText)*; it is called when the user drops an object on the widget. The *dropText* variable provides the URL for the object. The next line strips off the *file:* prefix and discovers the path to a local file. Then the function removes the original image from the theme window and displays the new image, using the last line to scale the image (Figure 3). ■

INFO

[1] SuperKaramba: *http://netdragon.sourceforge.net/ssuperkaramba.html*

[2] "KTools: Decorating the Desktop with SuperKaramba," by Stefanie Teufel, Linux Magazine, February 2005, p. 72. *http://www.linux-magazine.com/issue/51/Ktools_Superkaramba.pdf*

[3] Guru's RPM Site: *http://linux01.gwdg.de/~pbleser*

[4] SuperKaramba Debian packages: *http://archive.linux-peter.de/debian/pool/main/s/superkaramba*

[5] SuperKaramba themes: *http://www.kde-look.org*

[6] Howto on creating a theme: *http://netdragon.sourceforge.net/screate.html*

[7] Creating non-rectangular themes: *http://netdragon.sourceforge.net/smask.html*

[8] Sensor overview: *http://netdragon.sourceforge.net/ssensors.html*

[9] All support meters: *http://netdragon.sourceforge.net/smeters.html*

[10] LM sensors: *http://secure.netroedge.com/~lm78*

[11] SuperKaramba Python API: *http://netdragon.sourceforge.net/api.html*

[12] SuperKaramba Python template: *http://netdragon.sourceforge.net/template.py*