DUI Theming for Dumm1es

Authors: Tuomas Kuosmanen <tuomas.kuosmanen@nokia.com>

In Brief

Maemo “Harmattan” is built on top of the “Direct Ui” framework, and it uses a somewhat different approach to theming compared maemo5 and other earlier, gtk-toolkit based maemo releases.

The system provides a set of common building blocks for applications, called common components (programmers often call them “widgets” as well). Those common components are themed uniformly throughout the system, so all applications can share the same look and feel with no additional effort.

Applications can also create custom styling for certain elements when needed. This application-specific theming is explained in more detail later in this document.

Basic theming technology

The theme is composed of “theme bits” (graphic elements in SVG, PNG or JPG format, sound and vibra feedback files and possible other assets) and CSS stylesheets.

The CSS files tie our theme bits to component interaction states. For example, a piece of SVG called “duibutton-background-pressed” is used as “skin” to a component called “duibutton” when the user presses it. At the same time, a sound file called “blip.wav” is played and the vibrator makes a small bump effect. The CSS files also define common style parameters for components: margins, paddings, colors and fonts etc.

The SVG file format bears a loose resemblance to the W3C stylesheet standard, but it is both exending it and also only implements a subset of the syntax[1] It however tries to follow some web stylesheet conventions, hoping to be familiar to designers with a web design background. Therefore it is advisable to use the CSS files in the “base” theme as a reference, as those should reflect the supported parameters.

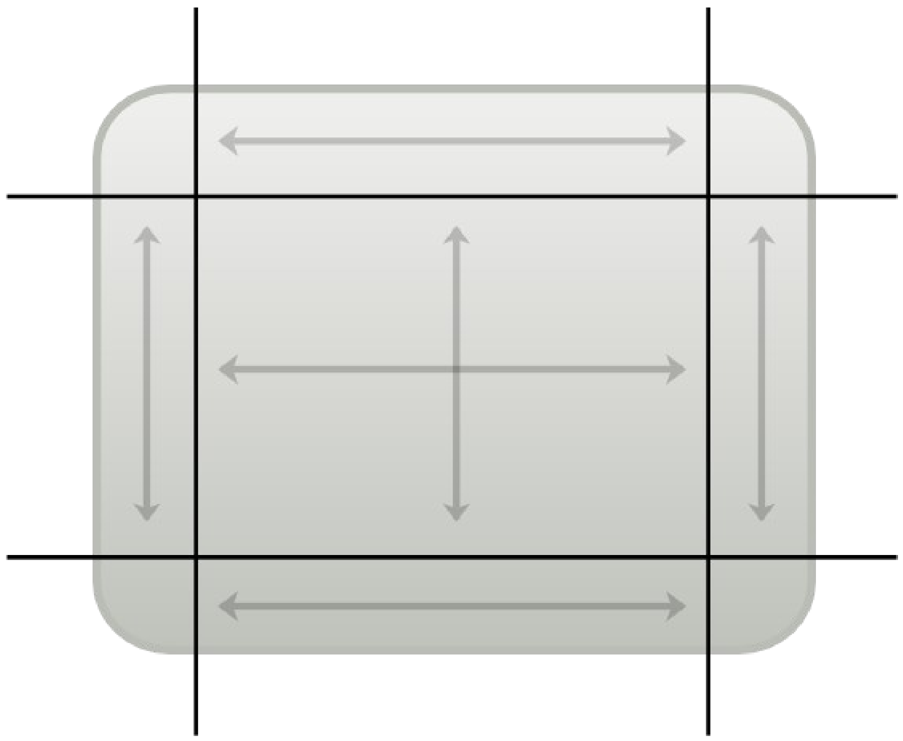
SVG files

Each SVG element is referenced by its svg group-id. Therefore it is possible to organize the graphics in a way that makes sense in the project – currently everything is split into individual files based on the component they belong to. There can be any number of SVG files in the theme, so you can either combine everything into one large canvas if you want, or in separate files – either way works.

Element sizes and stretch margins

SVG files are rendered as bitmaps in the graphics memory, and stretched to the desired sizes by using “border values”. Basically the graphical element is split into 9 pieces by defining left, right, top and bottom border sizes, explained best with the following graphic:

1 See Libdui documentation > Styling > Stylesheet syntax reference & Examples



DUI graphics scaling

At least I learn best by example, so here goes:

```
DuiButtonStyle {
      ...
      background-image: "duibutton-background" 10px 10px 10px 10px;
      preferred-size: 20mm 10mm;
      ...
}
```

In DUI everything is sized based on millimeters as units, so we can have a truly consistent physical size for UI elements regardless of pixel density of the actual screen. Since we work with pixels on screen thouhg, we define the border values as pixels, which is the native unit of SVG files. Therefore it is important to draw the svg graphics as “pixel perfect” as possible, even though they are vector graphics.

When the above button background image is stretched over the button, the following happens: The image is split into nine pieces, along those black lines. Now, lets say we have a screen with 254 pixels per inch – to make calculations easy – so the preferred size of our button would be 200x100 pixels. This is our “canvas” where the stretching happens. We take our 9 pieces and first stick the corners to each corner of the 200x100 pixel area. Each corner is 10x10 pixels in size. We then take the center piece of our graphic and stretch it so that it just touches the inner corners of the square bits: our centerpiece is scaled to 180x80 pixels size, and we have a X-shaped group of pixels. Finally the “sides” are stretched along those arrows – top and bottom sides are stretched to 180x10 and left and right sides to 10x80 to fill up the whole rectangle.

With this scheme it is possible to retain nicely rounded corners without distortion, but of course it causes its own limits. However, when constructed carefully, one can do pretty nice graphics. If the border values are kept as small as the radius of the round corners allows, the scaling can work quite well even with smooth gradients across the whole graphic.

Adobe Illustrator

Illustrator SVG export can be used to create DUI theme elements – simply draw your elements, combine each component state graphic into a group and name them in the layer dialog – the group name is saved as the group ID when exporting to SVG. I have had reports that Illustrator svg export has sometimes problems when using underscores in group names (for example duibutton_background_pressed) – this is why we are currently using dashes in the default duitheme (duibutton-background-pressed). However, if you have trouble with your graphics not showing up, please check your exported SVG for garbled id's, or use Inkscape. Latest versions of Illustrator seem to have much improved SVG export.

Inkscape (www.inkscape.org)

Inkscape is a free vector illustration program for Linux, Windows and Mac that can also be used to create DUI theme elements, thanks to its excellent svg support. On Inkscape, you also group elements together. However, there is a separate *Object Properties dialog* for setting a group ID, which is accessible through the *Object menu*. Set your ID there for each graphical element you intend to reference later.

Bitmap images

Bitmap images are referenced by their filename with the extension .svg or .png stripped off – for example mybackgroundimage.png can be referenced as “mybackgroundimage”. If you have several files using the same basename, with different extension (“meego.jpg”, “meego.jpeg” and “meego.png” and svg group called “meego”) bad things may happen, so better keep them unique by their basename. Also, as it is possible to have the same SVG group id in two separate files, it is important to stick to a naming scheme that avoids this. Duitheme is currently using *duiwidgetname-[additional_identifier]-part-state* naming convention.

Other things

To make it easier to maintain a consistent look and feel, themes define various “constants” like commonly used fonts and a colors in a file called “constants.ini”. These can be used in stylesheets to keep things consistent across the theme. This way you can later tweak those settings in one file, rather than having to edit all your stylesheets if you want to change something.

Themes can also contain vibra and sound feedbacks assigned to widget events, these can consist of a sound file (wav) or a vibra motor effect file, or both. *FIXME: these need to be explained in more detail.*

Theme folder structure

The system can have multiple themes and they are all located in /usr/share/themes/-path. Each theme has one root direcory, which will contain all necessary files for the theme. So if you are developing a theme, you should start by creating a new directory under /usr/share/themes.

```
System theme directory/ (/usr/share/themes)
  └ theme_x/ (directory for theme-specific files)
    ├ index.theme (theme description file)
    └ dui/ (directory for dui-specific theming files)
      ├ constants.ini (theme specific constants)
      ├ feedbacks/ (directory for common feedbacks)
      ├ images/ (directory for common pixmaps)
      ├ svg/ (directory for common svg files)
      ├ icons/ (directory for common icons)
      ├ lib_y/(directory for library-specific theme files)
      │ ├ lib_y.conf (library-specific view configuration file)
      │ └ style/ (library-specific stylesheets)
      │   └ lib_y.css (entry point stylesheet file for library)
      ├ app_z/ (directory for application-specific theme files)
      │ ├ app_z.conf (application-specific view configuration file)
      │ ├ feedbacks/ (directory for application-specific feedbacks)
      │ ├ images/ (directory for application-specific pixmaps)
      │ ├ svg/ (directory for application-specific svg files)
      │ ├ icons/ (directory for application-specific icons)
      │ └ style/ (application-specific stylesheets)
      │   └ app_z.css (entry point stylesheet for application)
      └ locale/(directory for different locales)
        └ en/ (directory for locale-specific theme files)
          ├ constants.ini (locale specific constants)
          └ icons/ (locale specific icons)
```

The common user interface components theme is in the subfolder “libdui”, and application specific theming is in a subfolder named after the application name. Otherwise the structure is the same on either case, and the subfolders are always as follows:

- **feedbacks** folder is used to define different vibra/sound event feedbacks. These can be bound into widget states and events through CSS. Sound files are wav format. *FIXME: Needs more information about how this works.*
- **icons** has all the icon files, one svg file per icon. Even though icons are SVG, the filename is used as an identifier for icons, and svg group id does not matter.
- **images** contains bitmap images, for example wallpaper files and application background graphics. Referenced by basename, (filename without the extension). Take care to keep the id's unique and separate from svg id's since they are read into the same namespace!
- **style** contains the stylesheets for theme, and there is one “master stylesheet” that is loaded, and other stylesheet can be included with the @import command. The master stylesheet name is the same as the parent folder name: libdui.css or myapplicationname.css, depending on whether it is part of common components or application specific theming. By using the import command you can split the CSS into logical parts, for example grouped by component.

- **svg** contains all of our SVG graphics. You can split the graphics across separate files if you like, or you can just have one huge SVG that contains all your graphics, it's up to you. The only important thing is that you need to make sure each referenced svg group ID is unique – so be careful with naming if you have separate files.
- **index.theme** file that defines the name of the theme etc.. See base/index.theme for more. *FIXME: explain better!*
- *FIXME: needs information about the other files like .conf and constants.ini etc...*

index.theme

The file “index.theme” describes the theme itself, contains information about theme inheritance etc. This file is common with for example gtk-toolkit themes, and follows the “windows .ini file” syntax with section names with brackets and basic key-value pairs. An example is below:

```
[Desktop Entry]
Type=X-DUI-Metatheme
Name=Very cool Theme
Encoding=UTF-8

[X-DUI-Metatheme]
X-Inherits=base
X-Icon = icon-theme-logo
X-Visible=true
```

Type for dui themes is naturally always X-DUI-Metatheme, and Name is the theme display name that should be shown to the user. X-Inherits is the important bit here – it defines which theme to inherit from. Commonly this should be “base”, but if you want to base your theme on some other theme, it is also possible to do that. X-Visible is used to hide the “base” theme from users, as it is not a complete theme by itself.

Theme Constants File (constants.ini)

This file contains constant definitions for this theme. The constant set is predefined and can be found from /usr/share/themes/base/constants.ini. Constants are used like in application code and in stylesheets to set common fonts and colors to be used in many places, so they can be defined in a central location.

You can use the constants like this in your CSS files:

```
DuiButtonStyle {
    font: $FONT_DEFAULT;

    background-image: "duibutton-background" 10px 10px 10px 10px;
    background-color: $COLOR_BACKGROUND;
    text-color: $COLOR_FOREGROUND;
    ...

}
```

Below is an example of constants.ini which changes the default colors for this theme.

```
[Palette]
COLOR_FOREGROUND            = #fafafa ;text color
COLOR_SECONDARY_FOREGROUND  = #7a7a7a ;secondary text
COLOR_BACKGROUND            = #ff0000 ;background

;reversed elements
COLOR_INVERTED_FOREGROUND           = #090909 ;text color
COLOR_INVERTED_SECONDARY_FOREGROUND = #595959 ;secondary text
COLOR_INVERTED_BACKGROUND           = #00ffff ;background
```

Creating your own theme

Finally the part you are looking for! :-) Probably the simplest way is to learn by example, so create a folder into /usr/share/themes/mytheme and copy the index.theme from “devel” theme there. Make sure X-Inherits=base exists, and try out Widgetsgallery's theme switcher in the main view's menu. Naturally your theme does not really look any different yet (and it has no icon) but it works.

Now take some assets from base and copy them into your own theme into the identical folder tree there, and modify them.

Whatever your theme is missing is used from the “base” theme, so you can change the theme step by step, and still have a working theme you can test.

Technical bits and troubleshooting

The important part is that since SVG elements are referenced by their group-ID, you need to keep those id's in sync with your CSS files – so if you draw new elements next to the old ones, be sure to rename the new one after deleting the old, or just work inside the “group” to begin with. If you are using Adobe Illustrator, the group ID is the same as the layer dialog's group name. In Inkscape you should set the ID from the Object Properties dialog (Object → Properties in the menu). On both applications doubleclicking the group “dives” into the group, so you can edit and replace elements, while keeping the group name intact. This is just a thing to remember, especially when you wonder why your new shiny graphic is not showing up... :-)

Installing a theme

(explain packaging here)