MTexturedStyleEditor - User Guide

MeldaProduction software including all plugins uses a proprietary user interface systém, which allows redefining its appearance, size and sometimes even behaviour. This makes the plugins way more adaptive and user friendly compared to other plugins on the market. This guide is designed for those of you, who want to create their own styles for the plugins.

A style is a file containing all the resources required to render each component of the user interface. It contains images, fonts, textures, graphical settings and more. MTexturedStyleEditor is an advanced tool for creating these styles. Originally we developed it only for our own use, so it is a little "hi-tech", but some of you have already shown that it is quite usable ;).

The good news is, that once you create a style, it will work with all plugins and it will preserve all the marvelous features the plugins have. And even better, if your style is good enough, we shall include it in further version (or share it somehow) and you will get a compensation of course ;).

The bad news is that creating styles is pretty complicated. It's far from creating simple skins containing a background image and a few knob images. The styles must be able to adapt to any plugin, any layout, any size, any control inside another control... But, well, it's not that hard after all, you will see.

Sometimes the Editor will come up with some weird message, which would probably mean nothing to you, but that's just how it is with hi-tech tools :).

Installation

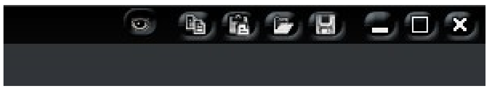
The Style Editor has no installer, instead you simply download an archive from our website and extract it somewhere. Before you start running anything, copy the supplied fonts from "Fonts" subdirectory into your system fonts directory:
Windows: C:/Windows/Fonts
Mac: /Library/Fonts

Then you can run the Editor itself:
Windows: MTexturedStyleEditor.exe
Mac: MTexturedStyleEditor.app

What you see now is probably pretty scary, so maybe you should close the Editor and continue reading instead :).

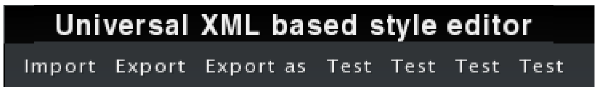
The archive also contains a set of "mtexturedstyle" files. Despite their names these are not the styles, these are so-called style definitions. They contain all resources of the style, the Editor can load them, but they are too big and impractical for use by the plugins. It's like when you edit a text in Word, you get the DOC file, which almost nothing understands, so at the end you convert it to PDF. Similarly, you edit your styles in MTexturedStyleEditor, your temporary results will be called "mtexturedstyle" and when you are ready, you export it into the resulting "mrf" file, which is the file type used by the plugins. But we will get to that later.

For now, just run the Editor. Look at the top right corner:



Over on the right-hand side of the Main toolbar are 8 buttons:
Eye, Copy, Paste, Load, Save, Minimise, Full-screen / Restore, Exit

Many of these buttons are used throughout the Style Editor and their general functions are described later. For now, click on the "Load" button and select any of the "mtexturedstyle" files. The Editor will probably look quite different now. So let's skip any editing and just look at the style itself. First we need to Export it – look at the main toolbar:



The only button that will work at this moment will be "Export as". It will let you choose any path on your hard drive and name of the style. Then it will create quite a few files in that directory, perform some housekeeping actions and voila the "mrf" is there, the style has been compiled. After that moment, every time you click the "Export" button, the same action will happen, but the Editor will not ask you for the style path anymore, just overwrite the previous one. To be safe, always click "Export as" the first time after you load an "mtexturedstyle" file; after that, click "Export".

It may be useful to choose the predefined style paths the plugins are using, so that you can check the styles immediately in the plugins. Hence you may want to export the style to this path:

- Windows Vista/7/8: C:\Users\{username}\AppData\Roaming\MeldaProduction\MTexturedStyles
- Windows XP: C:\Documents and Settings\{username}\Application Data\MeldaProduction\MTexturedStyles
- Mac: ~/Library/Application support/MeldaProduction/MTexturedStyles

Now click any of the "Test" buttons. Each will open a testing window showing the style in action. Each is different and are defined by the "window0.xml".. files in the Editor directory. If you known XML a little, you could probably create your own test windows easily ;). If you used the predefined style paths, you can actually open any plugin now and select your new style. If you exit the Editor and start it again, it will remember the state and loaded style, making it easy-to-use.

One more button in the main toolbar remains – "Import". This one actually opens the "mrf" file. This is not always possible, so it's not really an ideal approach, but if you get your hands on someone else's style, you can use this to open it, but always assume that it may not be completely loaded. Always try to use the style definitions (the "mtexturedstyle" file) instead.

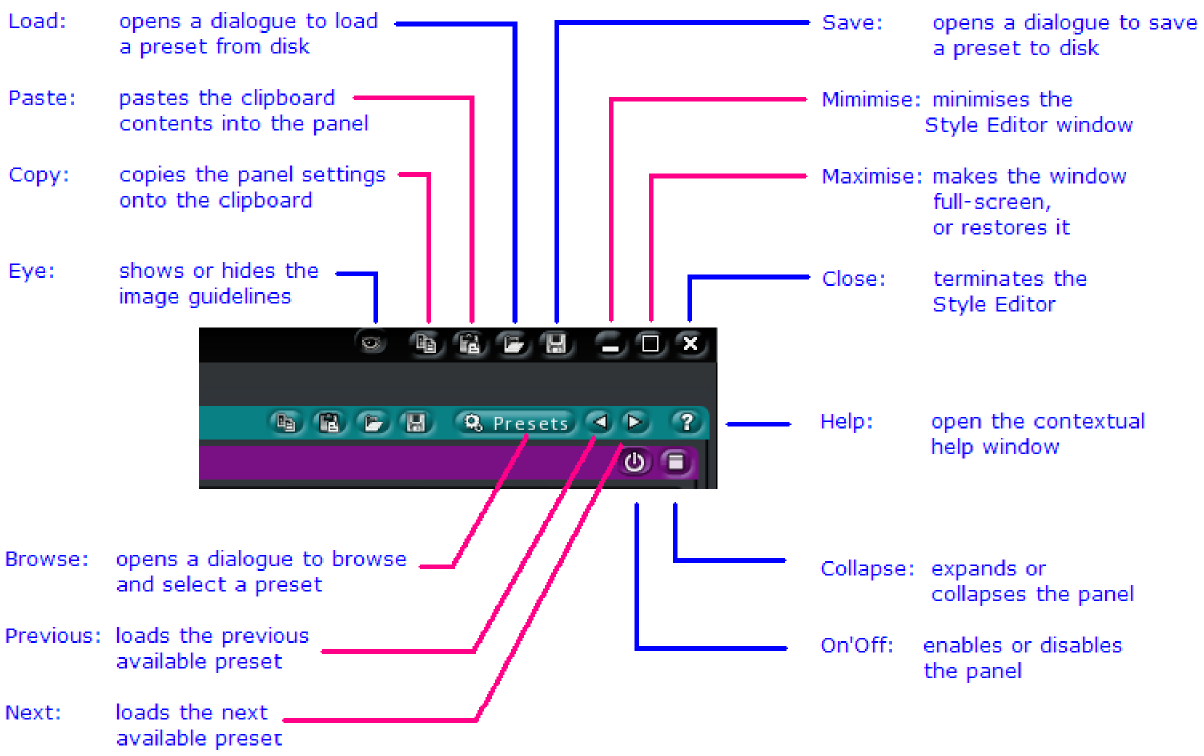
Well, now you know how to load/save a style definition, how to export the resulting mrf style from the Editor and to see it in action. It's time to edit your new style!

Getting started

The GUI engine essentially defines several component types, such as a knob, button or window. If you look at the left side of the Editor, you can see a huge, almost never ending tree, of these components. Such a tree may look be scary at first, the good news however is that these are almost the same. For example, a window works more or less in the same way as a button, however odd that may sound. In the end you can create a style, where everything will look like a button, not that it would be a good idea probably :).

The Browser, on the left-hand side, lists all the elements of the style that can be edited, in a simple expandable / collapsible hierarchy. Click on the [+] to expand an item or on the [-] to collapse it. The first section in the Browser is the Global Style settings. This section is a bit different, it's the only one which doesn't specify any controls and rather contains some global parameters and resources.

Before we get to it, let's check the titles with caption buttons, as these are just everywhere:



Load: opens a dialogue to load a preset from disk
Paste: pastes the clipboard contents into the panel
Copy: copies the panel settings onto the clipboard
Eye: shows or hides the image guidelines
Save: opens a dialogue to save a preset to disk
Mimimise: minimises the Style Editor window
Maximise: makes the window full-screen, or restores it
Close: terminates the Style Editor
Presets
Help: open the contextual help window
Browse: opens a dialogue to browse and select a preset
Previous: loads the previous available preset
Next: loads the next available preset
Collapse: expands or collapses the panel
On'Off: enables or disables the panel

The Editor is hierarchical and the good news is that you usually can save/load any section as a preset/file, use clipboard for it etc. So, you can easily export the whole knob definition from one style and load it to another. Or just one image of the knob...

It is worth remembering that you can point mouse to any place and click the "F1" button to get extended help. The Editor isn't documented very much though, so it will be useful only for specific places – e.g. a mouse above a slider shows some advanced editing techniques:

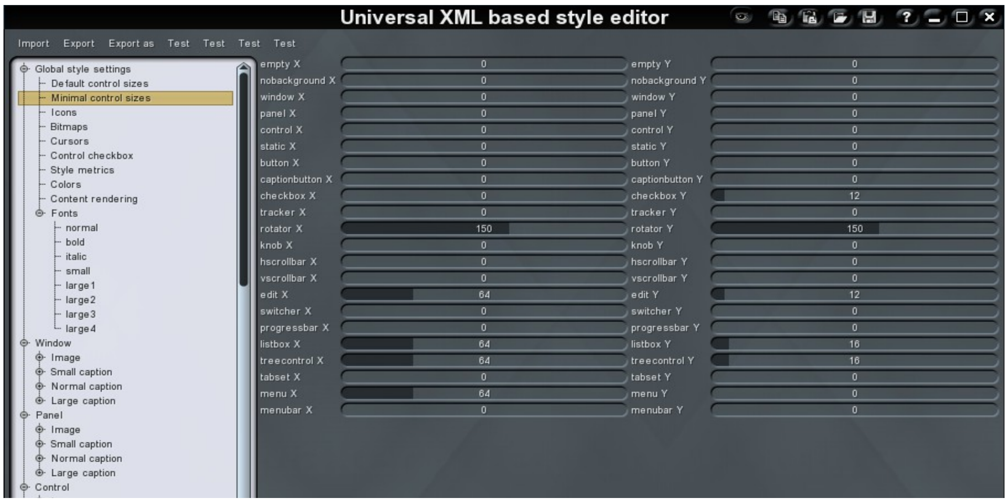
- Click/drag using left mouse button to change the value,
- Right-click selects the default value,
- Mouse wheel, arrow keys and vertical drag using middle mouse button or using left mouse

button while holding Ctrl modifies the value more accurately,
- Home key configures the minimal possible value,
- End key sets up the maximal one.
- Shift + left mouse button pops up a dialogue to let you type in the value, enter it by clicking the displayed keyboard or edit the value as text.
- Shift + Tab / Tab selects the previous or next field.

Global style settings item

As I explained before, this one is a very special item, which doesn't specify any component, but only some global properties and resources. So let's explain what these mean.

Default control sizes, Minimal control sizes



empty X	0	empty Y	0
nobackground X	0	nobackground Y	0
window X	0	window Y	0
panel X	0	panel Y	0
control X	0	control Y	0
static X	0	static Y	0
button X	0	button Y	0
captionbutton X	0	captionbutton Y	0
checkbox X	0	checkbox Y	12
tracker X	0	tracker Y	0
rotator X	150	rotator Y	150
knob X	0	knob Y	0
hscrollbar X	0	hscrollbar Y	0
vscrollbar X	0	vscrollbar Y	0
edit X	64	edit Y	12
switcher X	0	switcher Y	0
progressbar X	0	progressbar Y	0
listbox X	64	listbox Y	16
treecontrol X	64	treecontrol Y	16
tabset X	0	tabset Y	0
menu X	64	menu Y	0
menubar X	0	menubar Y	0

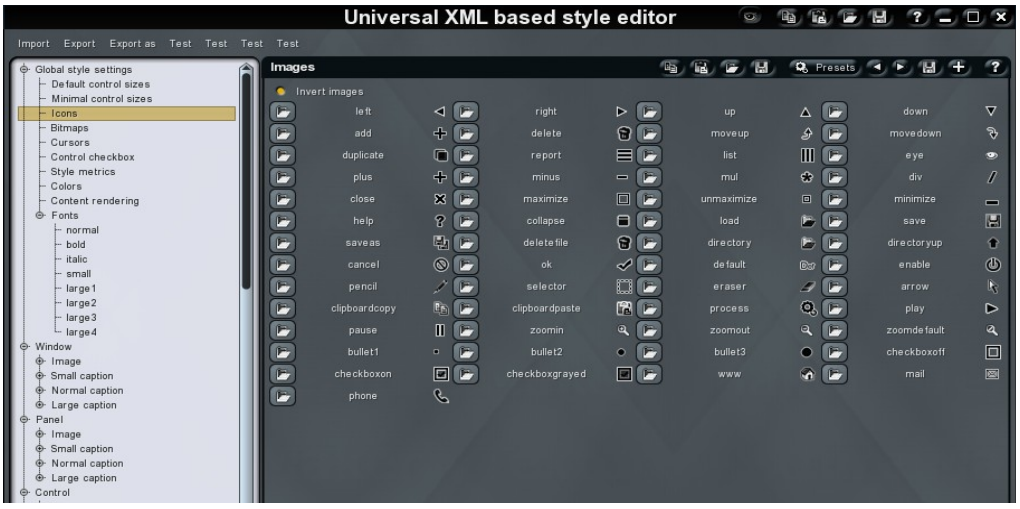
The style engine is pretty smart – it can determine minimum sizes based on how each component is painted, what is inside it, even its surroundings. And each plugin can restrict the sizes as well. However the style may want to say something about the component sizes too ;).

Default sizes are just, well, defaults. In most cases it will be 0, meaning that "you don't care" and default size should be as small as possible as long as everything fits inside it.

Minimal size is really a minimum and the component can never be smaller. Well, there are circumstances, where everything fails and it must fit, but that's not relevant here. Minimal size of zero means that the component size is restricted only by the minimal size on the element’s images and contents. But sometimes you want each component to be bigger, intentionally.
For example, for an "Edit" (text input field) painted using a rectangle with rounded corners with radius 10 pixels, the control must be at least 20 x 20. In the example above, Edit X is 64 – that's simply because an "edit", text input field, looks weird when too thin, so this ensures it will always be at least 64 units wide. Note that it doesn't specify the vertical minimum, because an edit usually contains a text, which will set the minimum height.

I said units, right? Yes, these are not pixels. Well, they are, but only if the style is not resized. As you know, the user can change colors and size of the style. Normally it is 100%, in which case these are pixels indeed. However when a user wants bigger fonts and just everything bigger, he sets 200% and everything will be twice as big, in this case Edit X will become 128. Always remember the resizing, it doesn't apply only here.

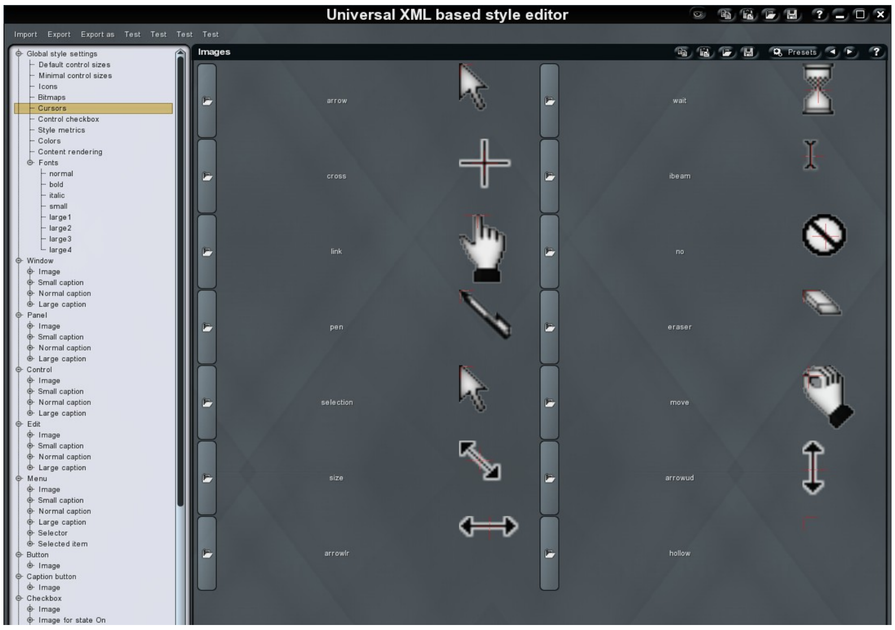
Icons, Bitmaps, Control checkbox



The plugins often need some standard images, "?" for help buttons, "X" for close buttons and so on. The style needs to provide these. You may just leave the images from the predefined styles, these are all the same. Currently these are colored but usually use only black and white. Not ideal, but has the advantage that it fits almost everything – you can see the icon on black background, white background, even pink background if you want...

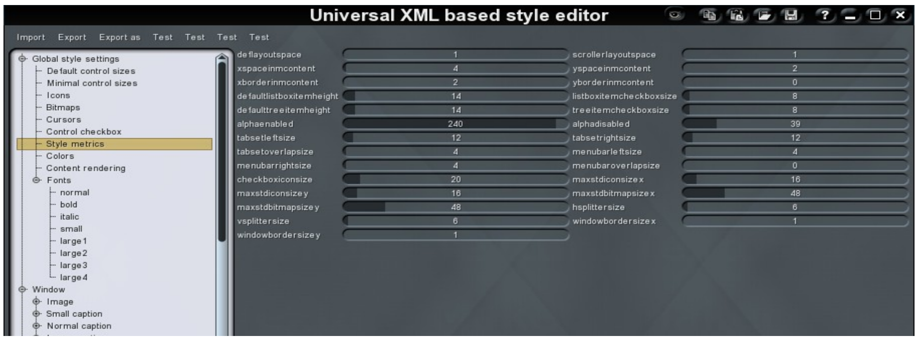
To load a different image, click on the Folder button which opens a Load Image dialogue. .jpg. .png or .tga image files can be used.
Icons are usually very small as they are used inside components. If they were bigger, the component would need to be bigger as well. So in most cases these are just 10x10, 12x12 or 14x14.
Bitmaps on the other hand are usually used in alert popups and are usually 64x64.
Control checkbox is used in listbox/tree views with checkboxes in each item. These should be fairly small, they must fit each item in the list, so by default these are 8x8.

Cursors



These are very similar to the icons and bitmaps, but the purpose is obviously very different. The principles are exactly the same, but the images are shown magnified and there is a little red cross inside each of them, the so called "hot spot". This marks the place the cursor is actually pointing to. You can change the position simply by clicking into the image. Cursors are usually 32x32 pixels square and we should rather keep it that way as the behaviour is a little complicated due to different implementation on each operating system.

Style Metrics



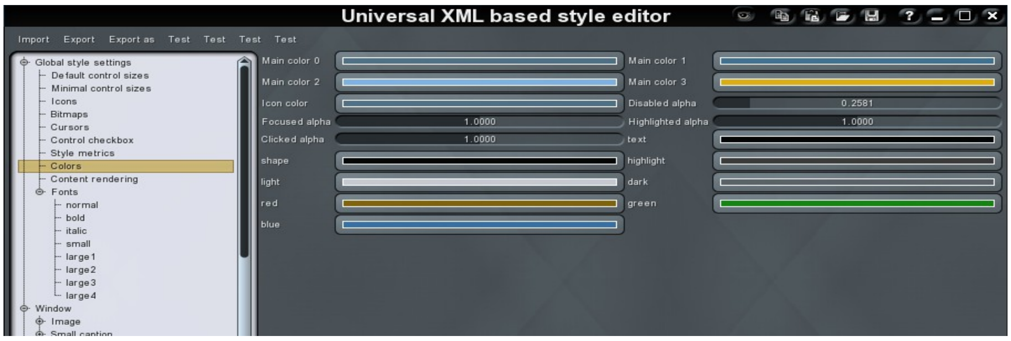
deflayoutspace	1	scrollerlayoutspace	1
xspaceinmcontent	4	yspaceinmcontent	2
xborderinmcontent	2	yborderinmcontent	0
defaultlistboxitemheight	14	listboxitemcheckboxsize	8
defaulttreeitemheight	14	treeitemcheckboxsize	8
alphaenabled	240	alphadisabled	39
tabsetleftsize	12	tabsetrightsize	12
tabsetoverlapsize	4	menubarleftsize	4
menubarrightsize	4	menubaroverlapsize	0
checkboxiconsize	20	maxstdiconsizex	16
maxstdiconsizey	16	maxstdbitmapsizex	48
maxstdbitmapsizey	48	hsplittersize	6
vsplittersize	6	windowbordersizex	1
windowbordersizey	1		

These values do affect various aspects of the style, but you will rarely need to change them. It will probably be the most mysterious part of the Editor :).

Anyway most of these values are pretty self explanatory. But what may be useful the most for you are these:

- deflayoutspace – space between controls
- defaultlistboxitemheight, defaulttreeitemheight – default size of items in list/tree views
- listboxitemcheckboxsize, treeitemcheckboxsize – sizes of the checkbox in list/tree views, which should most likely correspond to the images in "Control checkbox" section.
- alphaenabled, alphadisabled – opacity of text/graphics when a particular control is in enabled or disabled state

Colours



Main color 0		Main color 1	
Main color 2		Main color 3	
Icon color		Disabled alpha	0.2581
Focused alpha	1.0000	Highlighted alpha	1.0000
Clicked alpha	1.0000	text	
shape		highlight	
light		dark	
red		green	
blue			

As you know, by clicking "Settings" on any of the plugins and selecting "Style" the user can not only change style dimensions, he can also change 4 colors, called "**style colors**". You will use these colors later for example to alter the colors of knobs according to one these style colors. For now, here you can set the default 4 colors.

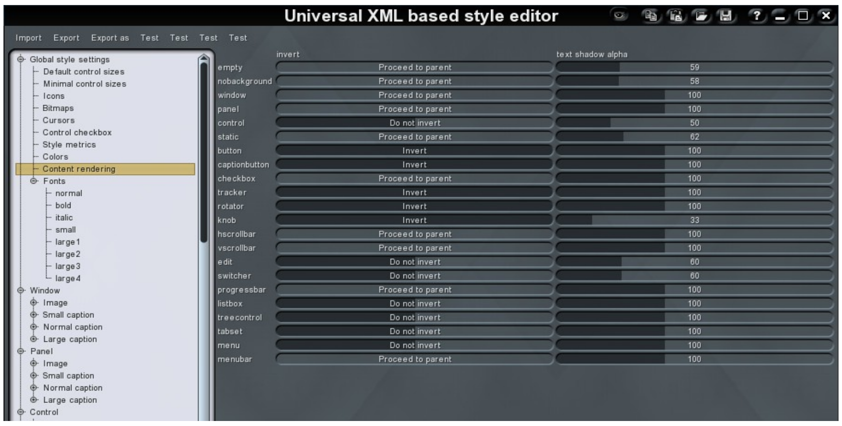
Icon color is a special color, which can alter colors of the "icons". For black/white icons it does nothing. But imagine the icons are colored, mostly green, and you design a style, which is mostly blue. Then you can either accept the fact that your icons will be green on blue, or use this color to "colorize" the icons to blue as well.

Disabled/Focused/Highlighed/Clicked alpha – controls opacity of shapes painted using the colors in different states of each component. For example, if a button is disabled, it is usually "less visible" meaning it has lower opacity. But then the contents of the button, the text inside it for example, should be less visible as well, right? So you just use the Disabled alpha property.

Text/Shape/Highlight/Light/Dark – despite our original design only Text & Shape colors are currently used and they specify the default colors of text and graphical objects. However it is recommended to keep the others default, just in case we need to use them.

Red/Green/Blue – these are special colors used in very specific areas. For example, you may design a very colorful style based mostly on orange color. Then there will be a particular graphical Editor, e.g. spectrum analyzer, requiring to show something "red". The true red color may look quite ugly on the orange background. So instead we defined 3 colors, which you can adjust to ensure they look good inside your style and the plugins use these instead of true red/green/blue, which just may not look good in the style.

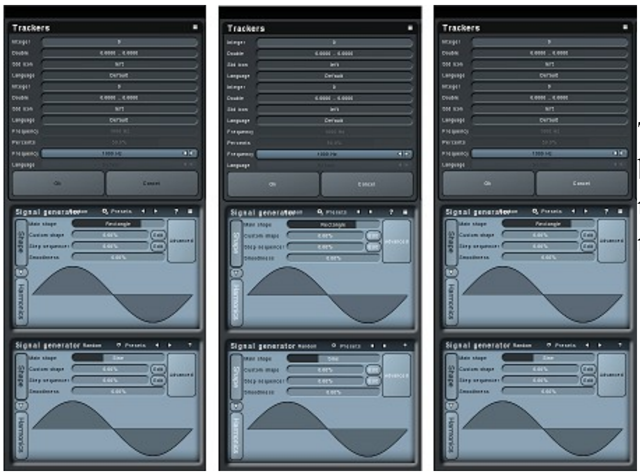
Content Rendering



	invert	text shadow alpha
empty	Proceed to parent	59
nobackground	Proceed to parent	58
window	Proceed to parent	100
panel	Proceed to parent	100
control	Do not invert	50
static	Proceed to parent	62
button	Invert	100
captionbutton	Invert	100
checkbox	Proceed to parent	100
tracker	Invert	100
rotator	Invert	100
knob	Invert	33
hscrollbar	Proceed to parent	100
vscrollbar	Proceed to parent	100
edit	Do not invert	60
switcher	Do not invert	60
progressbar	Proceed to parent	100
listbox	Do not invert	100
treecontrol	Do not invert	100
tabset	Do not invert	100
menu	Do not invert	100
menubar	Proceed to parent	100

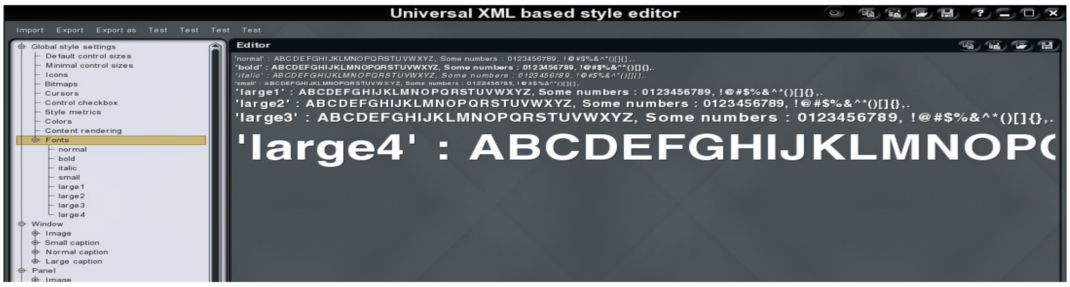
This element controls how contents (text/graphics) of each component will be rendered. Text shadow alpha is pretty self-explanatory. Invert parameter is less obvious, but it is very important.

Imagine you have a control, which has a background completely whitened. Then if the "shape" colour is black, you would not want it inverted (text and shapes should be black, so that it is visible). If the background is black, you would want the text/shapes inverted to white, so that they are visible. But if the control doesn't have a background (that is, it is transparent), you would not know which to choose! So set the value to "Proceed to parent", and the Style Editor will check the control underneath to ascertain whether the colour of the text/shapes should be inverted.



These screenshots show the effect on the button text of setting the ‘button’ value to ‘Do not invert’, ‘Invert’ and ‘Proceed to parent’ respectively.

Fonts



'normal' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'bold' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'italic' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'small' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'large1' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'large2' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'large3' : ABCDEFGHIJKLMNOPQRSTUVWXYZ, Some numbers : 0123456789, !@#$%&^*()[]{},.
'large4' : ABCDEFGHIJKLMNOP

The plugins take all resources from the style file, thus no system fonts are used directly. This ensures the style will look the same everywhere and you can choose fonts that will look good in the style. Fonts are one of the most important aspects of the style when it comes to "how good does it look".

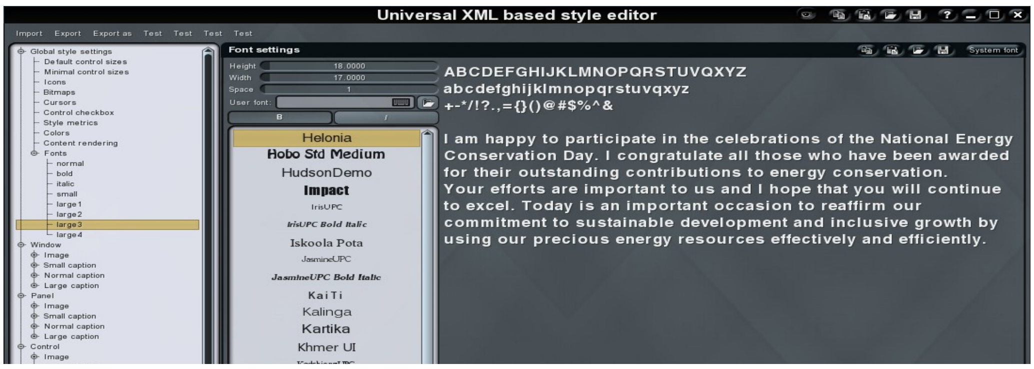
But there are other important aspects. First of all, the fonts are included in the style, so you need to check carefuly the file sizes of the font files. Although the fonts are usually vectorial (ttf fonts are recommended), they are also quite big. Sometimes the differences between fonts are huge, for example Arial is 750kB, and the supplied Helonia only 39kB! Moreover sometimes different faces are supplied – e.g. Arial, Arial Bold etc. When you have a style of 400kB, adding another 750kB of fonts is a huge waste of space. Not mentioning that Arial is so big mostly because it contains several different characters for other languages, but we can easily survive with just the basic ASCII set.

Secondly, the copyright rules – the font comes with the style, so one may think it is a violation of copyright rules unless the font is free. It is kind of a grey area, but still...

Ok, back to the Editor. The main "Fonts" item shows all of the fonts in the style as shown in the example above. There are 8 of them; normal, bold, italic, small, large1, large2, large3 and large4; these are used in other elements of the style. In some cases you may select one (e.g. you can choose a font used for the title of a window), in other cases the particular plugin selects the one it wants for particular places. Always remember – the plugins only use these 8 fonts, nothing else. This way the style can ensure it will always be good looking.

Note that although there are 8 fonts, they can all point to one single "ttf" font definition. That's the best option when it comes to size obviously. In most cases you will want all of them to be "Helonia" for example, but then you may need multiple faces – normal, bold and italic. The size is then irrelevant, because the fonts are vectorial, so they can be rendered in any size, therefore the ttf will be included just once.

Here's how it looks for one particular item, the "large3" definition:



Height 18.0000
Width 17.0000
Space 1
User font:

ABCDEFGHIJKLMNOPQRSTUVQXYZ
abcdefghijklmnopqrstuvqxyz
+-*/!?.,={}()@#$%^&

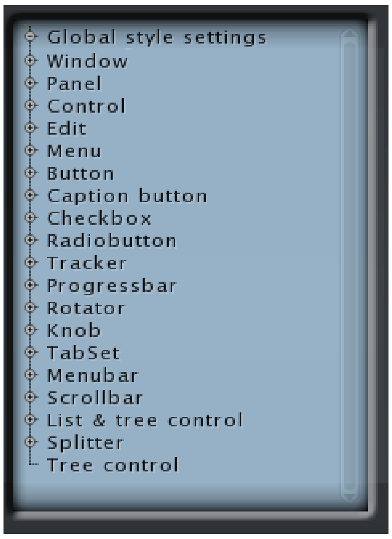
I am happy to participate in the celebrations of the National Energy Conservation Day. I congratulate all those who have been awarded for their outstanding contributions to energy conservation.
Your efforts are important to us and I hope that you will continue to excel. Today is an important occasion to reaffirm our commitment to sustainable development and inclusive growth by using our precious energy resources effectively and efficiently.

For each font definition you can specify:

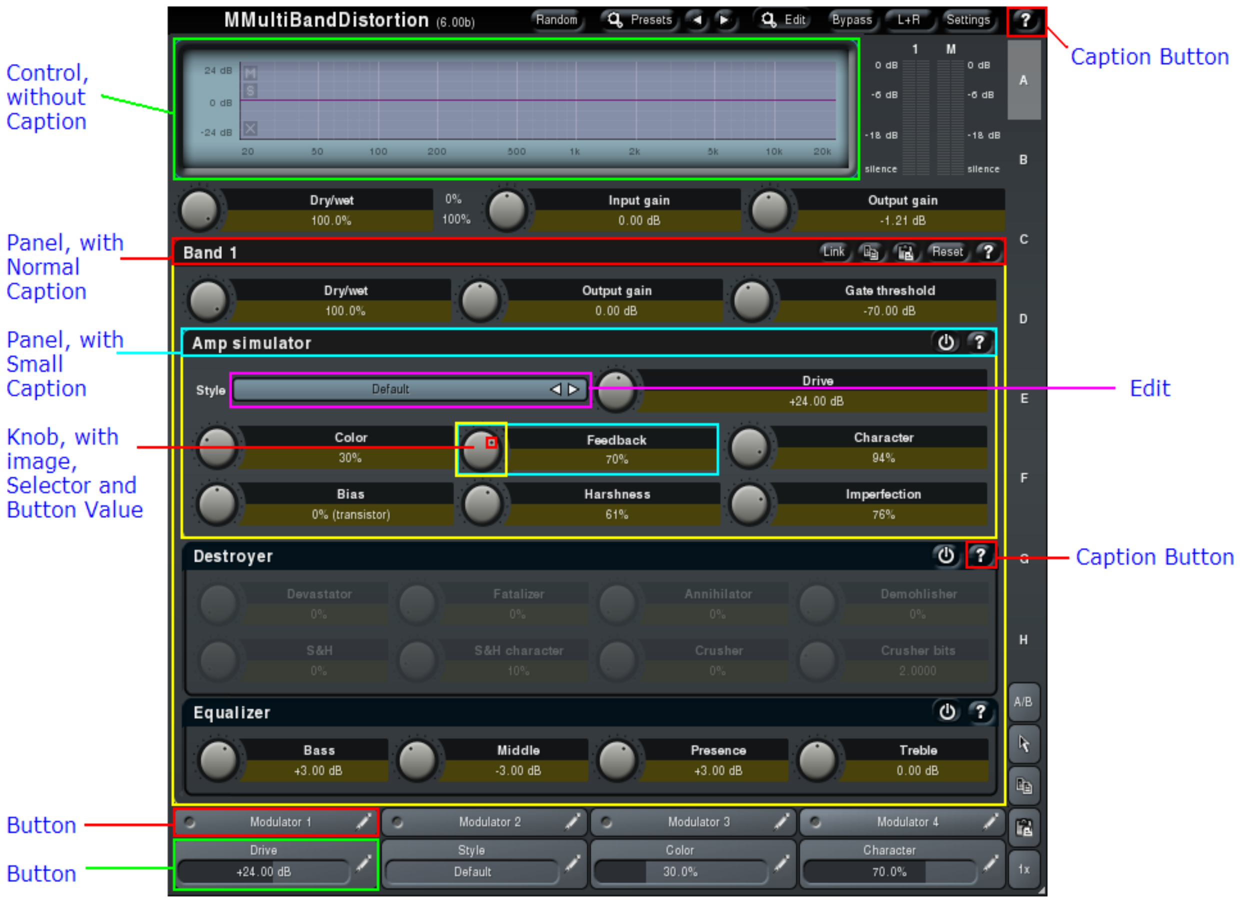
- the character width and height,
- the spacing between characters,
- a user font file (click the Folder button and browse to it to [Open] it,
- a Bold or Italic attribute by clicking the buttons,
- a system (shared) font from the scrollable list.

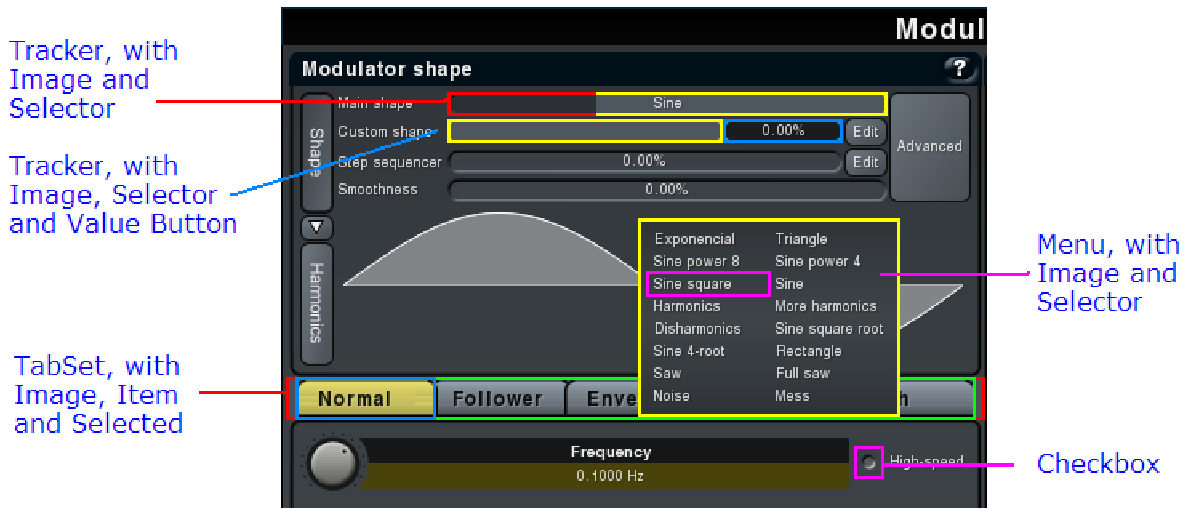
The "System font" switch is worth mentioning – if you enable it, the font will not be included in the style. The style will therefore be smaller and copyright laws won't matter, but you will be relying on the font being installed on the user’s particular computer. That doesn't seem a good idea, so this is not recommended.

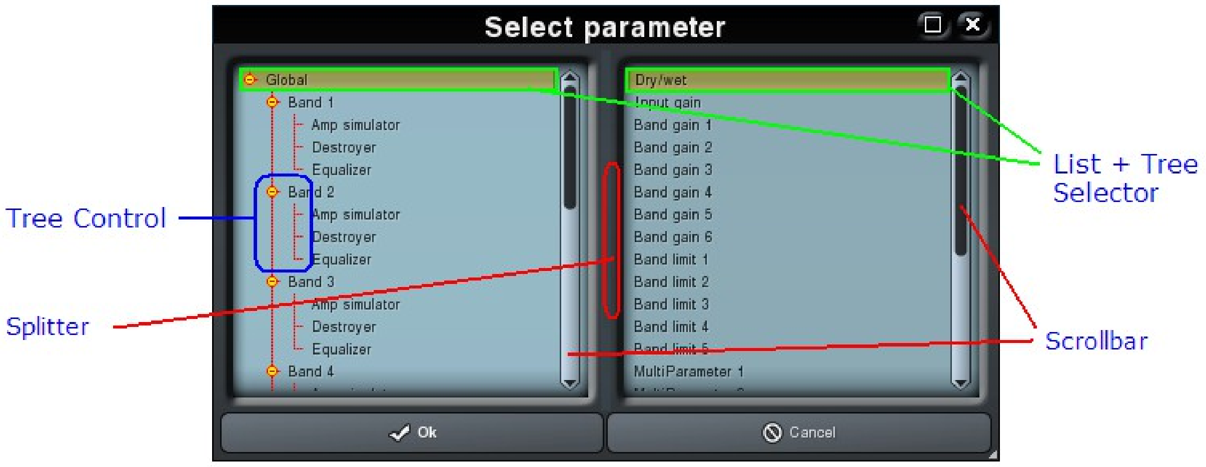
Components


Global style settings
Window
Panel
Control
Edit
Menu
Button
Caption button
Checkbox
Radiobutton
Tracker
Progressbar
Rotator
Knob
TabSet
Menubar
Scrollbar
List & tree control
Splitter
Tree control

There are 19 Component types in the Style Editor, together with a set of Global settings. The 19 Component Types are listed here, as seen in the Browser. Here are a few examples of a window containing different components:

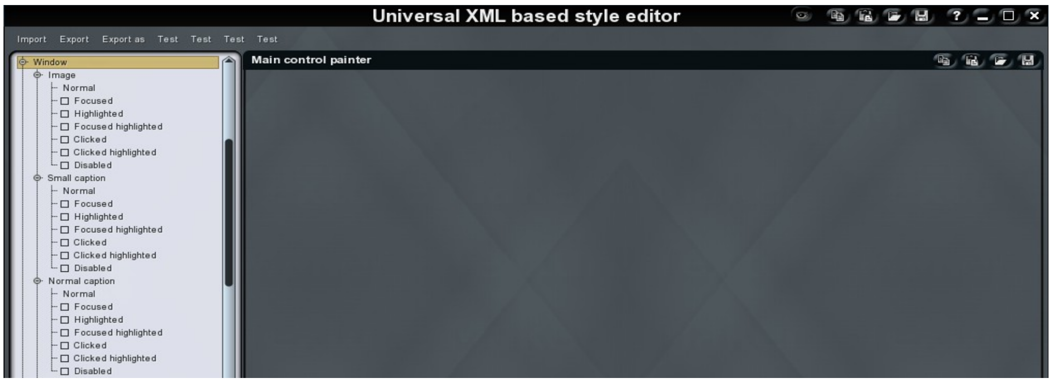






Each component looks different, yet they are rendered in a very similar way and have similar basic features. Each component contains one or more "Images", the main one is called Image, then you can have some additional ones, such as Large caption or Selector. But the interesting part is that the Image and Large caption definitions work in the same way.

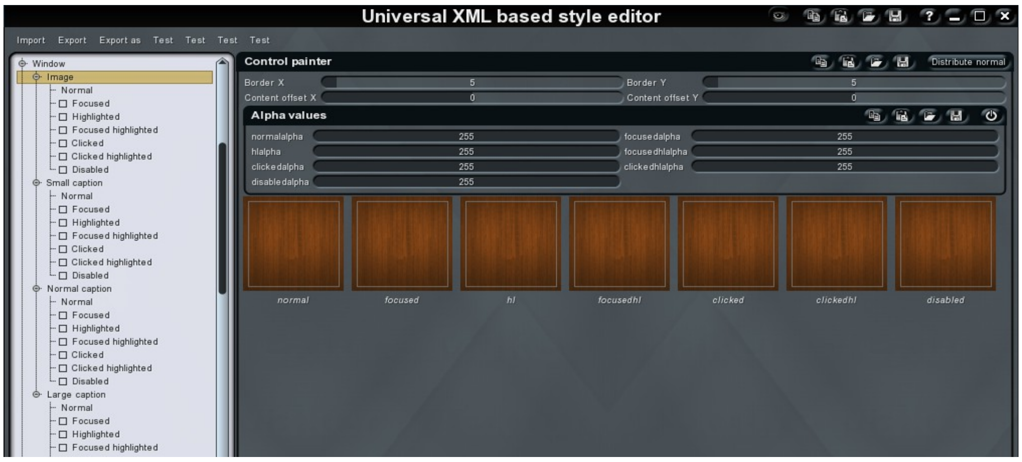
Now, in the Brower, select Window and expand the Caption elements if needed, it will look like this:



The title says "Main control painter" and you have 4 subimages: Image, Small caption, Normal caption, Large caption, all of them seem to have the same subcontents and they really do. Image is the main background of the window, captions are optional and really mean caption (or title).

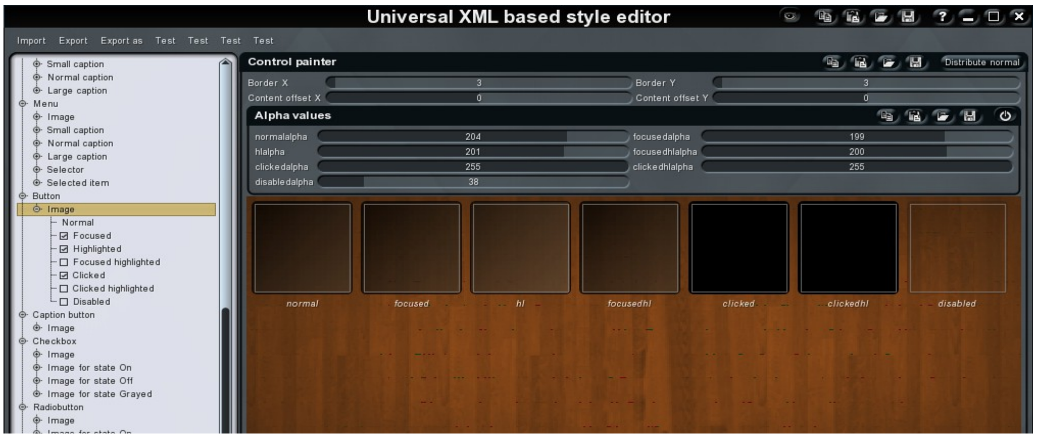
"Control painter" and component styles

Now select the Image and you will get to "Control painter" with this view:



Border X 5
Border Y 5
Content offset X 0
Content offset Y 0
normalalpha 255
focusedalpha 255
hlalpha 255
focusedhlalpha 255
clickedalpha 255
clickedhlalpha 255
disabledalpha 255

You can now see several values and 7 images called painters, and you may wonder what do these mean? These are the component states. They are not used for the Window element and it will be easier to describe them for a button instead, so select "Button" then "Image" in the Browser:



Border X 3
Border Y 3
Content offset X 0
Content offset Y 0
normalalpha 204
focusedalpha 199
hlalpha 201
focusedhlalpha 200
clickedalpha 255
clickedhlalpha 255
disabledalpha 38

First you may notice that this "Control painter" looks the same as in Window indeed. But back to business. The background used is that defined for the Window / Image element a "wood" image in this case. This is just to make things easier as all components can be displayed in a window, so it is usually good to picture all components on the Window background.

But while the control painter of a window contained 7 exactly same images, here it is not the case! Unlike a window which is simply a window, a container for other components, button can have different states. These states are listed as subitems of the "Image" item as well as under the separate images. So what do these mean?

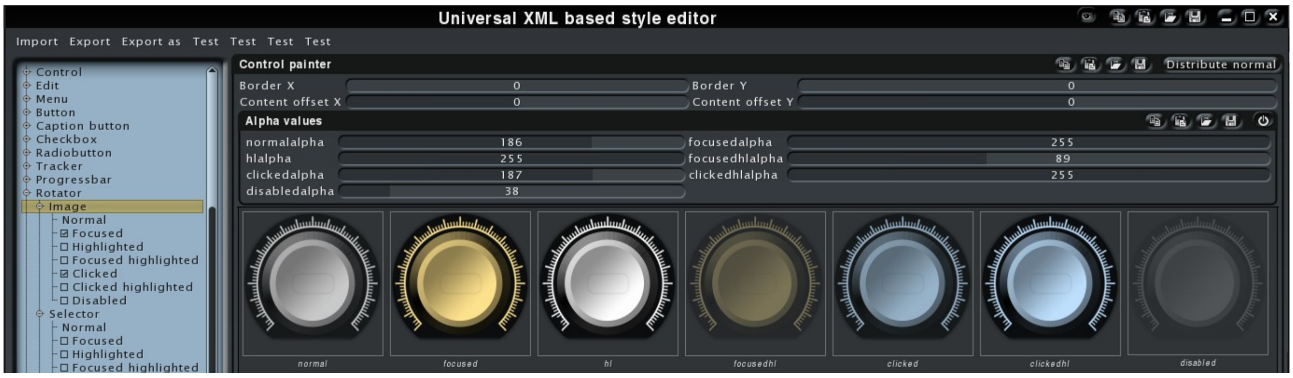
Normal means, well, normal, nothing is happening.
Focused means the component has a so-called keyboard focus – the user probably clicked on it and now it responds to keyboard events. This deserves some kind of highlighting, because otherwise the user wouldn't know which component has the keyboard focus!
Highlighted means the mouse cursor is pointing to the control or the user dragging it somehow. This doesn't necessarily require highlighting, but it looks good if the component looks different when the cursor goes over it, it kind of attracts user's attention and says that "this does something".

Clicked means not only the mouse cursor is pointing to the control, but a mouse button is actually pressed. It's pretty obvious to imagine what this means for a button for example.
Disabled means the component is disabled, it doesn't react to mouse nor keyboard. In most cases such control would be displayed with lower opacity, including the content. This has been discussed in the general style properties.

These are only 5 states, but they can be combined! For example, a component can have a keyboard focus and the mouse cursor may point to it. In most cases it doesn't matter, the keyboard focus is just more relevant, but you may want to reflect this.

Since there are 4 flags (focused, highlighted, clicked, disabled) you may think that there should be 16 states! But most of these are irrelevant – if a component is disabled, it cannot have keyboard focus and mouse state is irrelevant too. When a component is clicked, it is most probably focused too. But, you can click on a control, hold your mouse button and move the mouse elsewhere in which case it is clicked, but not highlighted!

The following example shows that you can really easily have different images for each state:



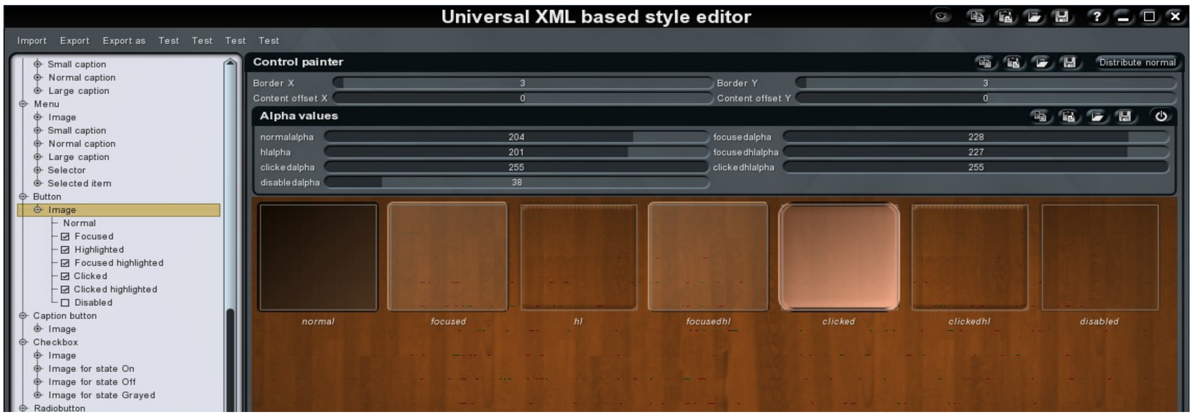
Border X 0, Border Y 0
Content offset X 0, Content offset Y 0

normalalpha 186, focusedalpha 255
hlalpha 255, focusedhlalpha 89
clickedalpha 187, clickedhlalpha 255
disabledalpha 38

So, how can I make the component look different in different states? By using 2 instruments:

1. **Use the Alpha values** – this will simply change the opacity when the component is in specific state. In most cases you will use this method at least for disabled state. In the example above you can see the "disabledalpha" is quite low compared to other states and so is the image of the disabled state (last one).

2. **Using different painters** – the "Normal" state is the only item in the list on the left, which doesn't have a checkbox. This is because there must always be at least one painter, for normal state. But if you enable another one, you can make the component look completely different in every state. Such as in this example:



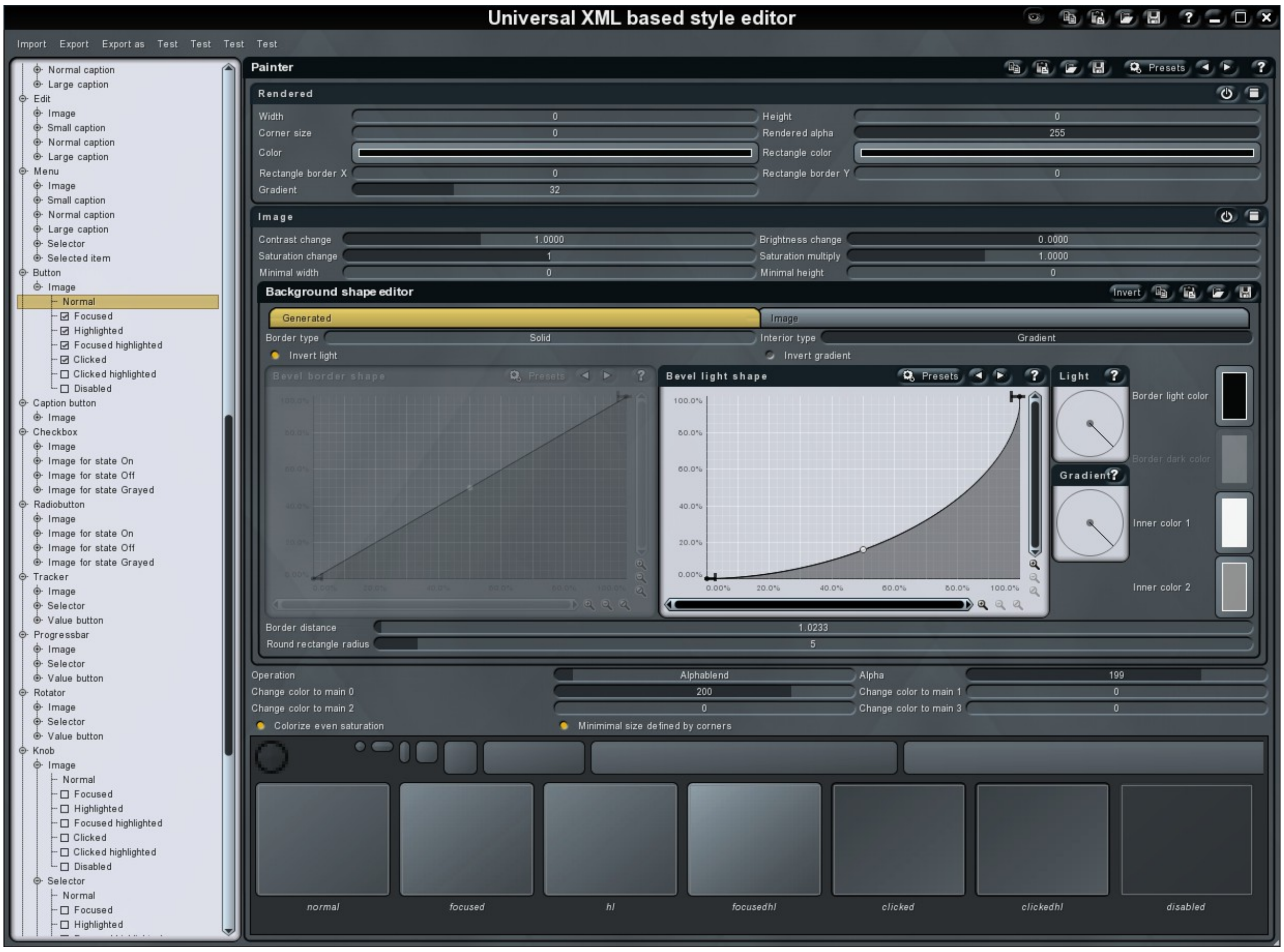
Border X 3, Border Y 3
Content offset X 0, Content offset Y 0

normalalpha 204, focusedalpha 228
hlalpha 201, focusedhlalpha 227
clickedalpha 255, clickedhlalpha 255
disabledalpha 38

So now what remains are the 4 values just below the "Control painter" title, **border and content offset**. These control the area "inside" the control, where the content will be positioned. This is the area where the text, graphics and subcomponents inside the component are placed and is indicated by the gray rectangles in the images below. You can turn off those guide lines using the "eye" icon in the Editor's title bar.

"Painter" and the rendering

Here we get to the bottom of things. We now know that each component is a compound of a few "Control painters", each of them defines how to paint the control according to the control state, but who is actually doing the painting? A painter! But how to edit it? Well, just select "Normal" in the list in the left for example. Now let me scare you with a screenshot first :)



Rendered

Width 0 | Height 0
Corner size 0 | Rendered alpha 255
Color | Rectangle color
Rectangle border X 0 | Rectangle border Y 0
Gradient 32

Image

Contrast change 1.0000 | Brightness change 0.0000
Saturation change 1 | Saturation multiply 1.0000
Minimal width 0 | Minimal height 0

Background shape editor

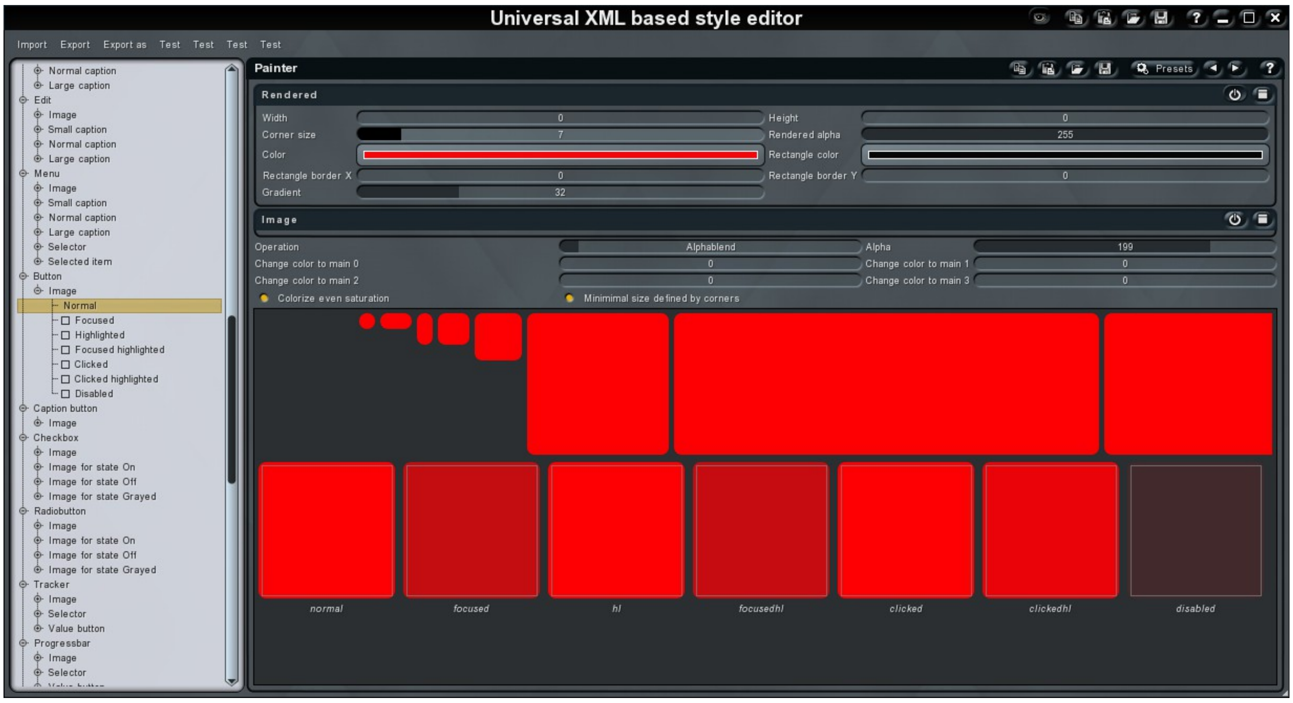
Generated | Image
Border type Solid | Interior type Gradient
● Invert light | ○ Invert gradient
Border light color
Border dark color
Inner color 1
Inner color 2
Border distance 1.0233
Round rectangle radius 5

Operation Alphablend | Alpha 199
Change color to main 0 200 | Change color to main 1 0
Change color to main 2 0 | Change color to main 3 0
● Colorize even saturation | ● Minimimal size defined by corners

Don't worry, it's not that bad. The painter contains 2 panels, "Rendered" and "Image", each has an enable button (the power switch almost at the end the panels title bar), which means you can use either method or combine them. So there are kind of 2 rendering engines, let's describe both.

Rendered

The Rendered panel contains the vectorial engine – well, it's very simple, it lets you render the component using a rectangle (with potentially rounded corners). That's it. Here's an example:



Rendered

Width 0 | Height 0
Corner size 7 | Rendered alpha 255
Color | Rectangle color
Rectangle border X 0 | Rectangle border Y 0
Gradient 32

Image

Operation Alphablend | Alpha 199
Change color to main 0: 0 | Change color to main 1: 0
Change color to main 2: 0 | Change color to main 3: 0
Colorize even saturation | Minimimal size defined by corners

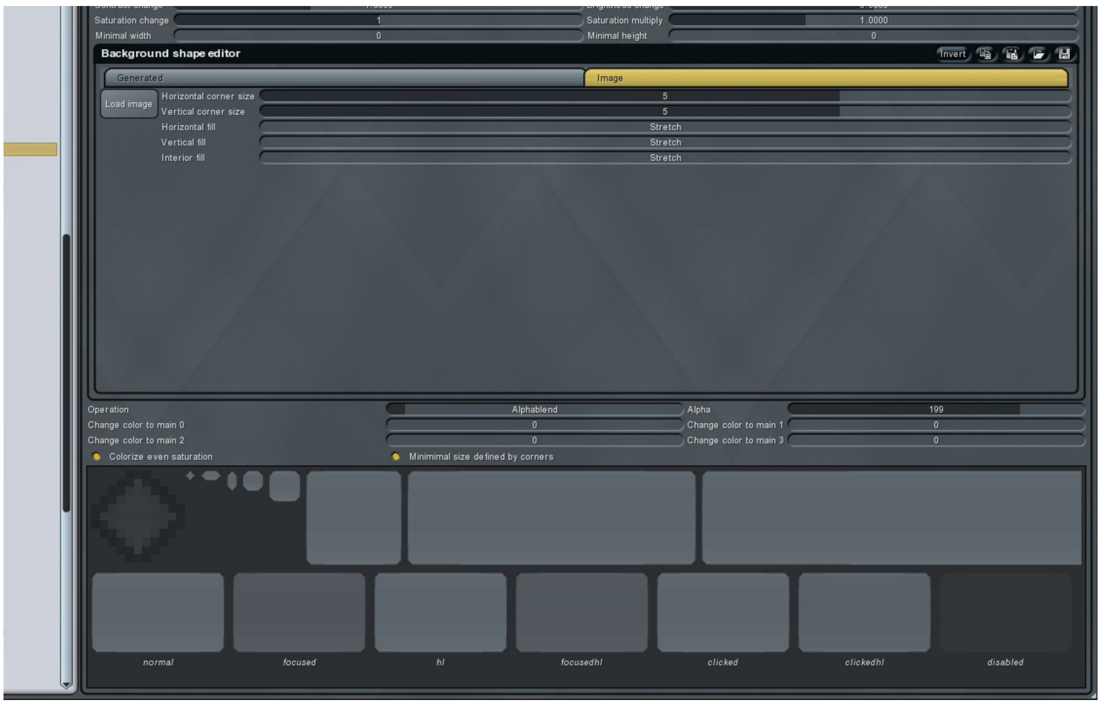
If your screen does not look like this, click the "Collapse" buttons at the end of the panels' title bars to collapse or expand the panels. In the lower part of the screenshot you can see the component with different sizes and different states. Note that although all painters for the other control states are disabled (unchecked in the listboxes in the Browser), each state still looks different because of the alpha values as described in the "control painter" section above.

Now to the parameters:

- Width & height – minimum size of the painter. This in effect controls minimum size of the component.
- Corner size, colors... - well, these are pretty obvious. But don't forget each color has "alpha" parameter, that one is very important here.
- Rectangle border – this may smaller than the rectangle. Why? Because sometimes you may not want the rectangle to be as big as the component.
- Gradient – it is not reflected in the Editor (technical stuff, excuses :)), but it makes the rectangle painted with a gradient, which usually looks better. But it also takes more CPU, so be careful!

Image

Image (or texture) based rendering is the kernel of the engine. You have 2 ways to setup a texture – you can draw it in an external editor (e.g. Photoshop or Gimp) and load it as JPG/PNG/TGA or you can generate the texture. To use an image file, select the "Image" tab like this:



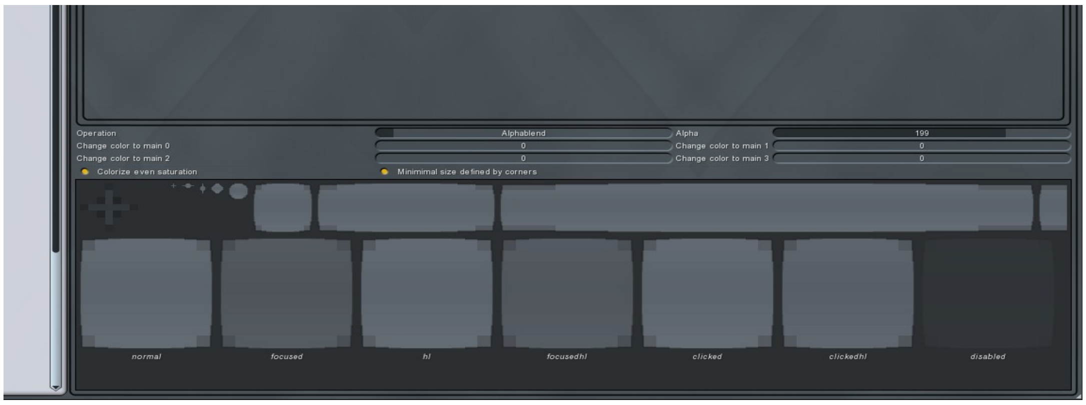
Saturation change 1 | Saturation multiply 1.0000
Minimal width 0 | Minimal height 0
Horizontal corner size 5
Vertical corner size 5
Horizontal fill Stretch
Vertical fill Stretch
Interior fill Stretch
Operation Alphablend | Alpha 199
Change color to main 0 0 | Change color to main 1 0
Change color to main 2 0 | Change color to main 3 0
● Colorize even saturation
● Minimimal size defined by corners

Here you just need to click Load image and select the image, which will then be a persistent part of the style. And here comes the interesting part – corners and stretching/patterning.

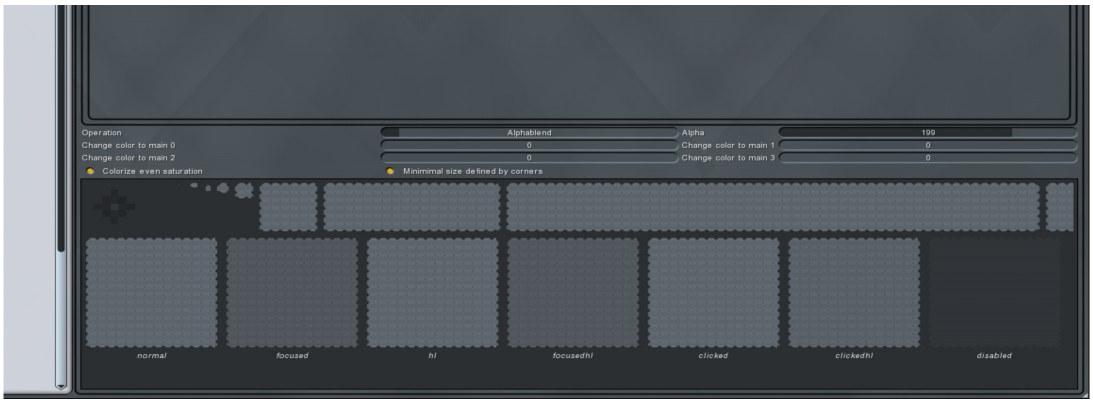
Imagine you create a rounded rectangle 40x40 with 10 pixels corner in your external editor, save it to PNG, load it in the Editor. How would you expect the engine to paint the component if it is 400x400? Resize it? But that would create an ugly rounded rectangle with 100 pixels corners!

So instead, the engine cuts off the corners, left/right/top/bottom part and renders each section separately. First it paints the corners without any resizing. Then it resizes left and right parts vertically and top and bottom parts horizontally. Finally the interior is resized in both directions.
And resizing means either stretching or patterning – stretching is obvious, patterning means the part of the image is repeated allover again until it fills the space.
This way the corners stay perfectly sharp, sides are resized only in one direction and only the interior is resized in both.

In our example we would need to specify corner size of 10x10 in the Editor. Look what happens if the corner size is too small:


Operation Alphablend
Alpha 199
Change color to main 0 0
Change color to main 1 0
Change color to main 2 0
Change color to main 3 0
Colorize even saturation
Minimimal size defined by corners

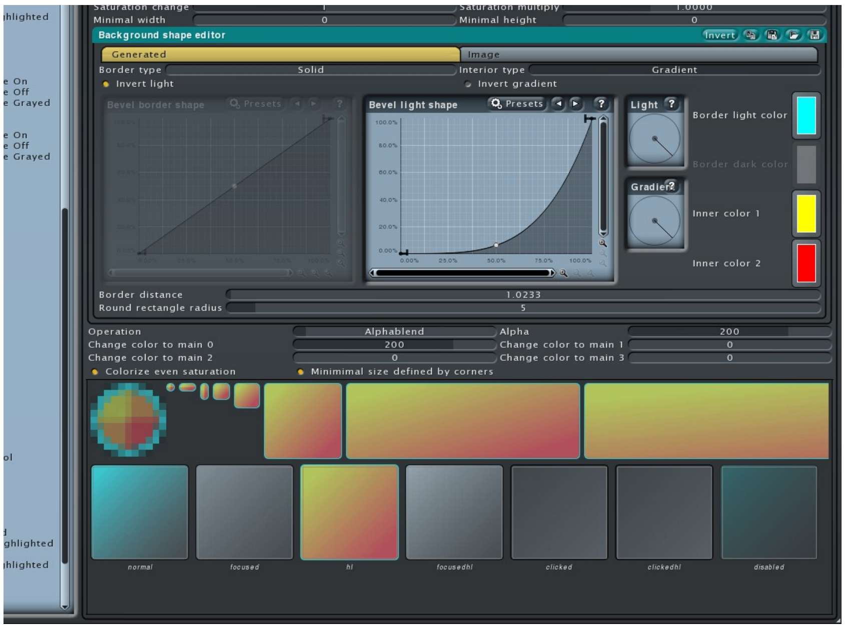
And then when we use patterns, it becomes even worse:


Operation Alphablend
Alpha 199
Change color to main 0 0
Change color to main 1 0
Change color to main 2 0
Change color to main 3 0
Colorize even saturation
Minimimal size defined by corners

Generated

The Editor contains an integrated generator, which creates the typical button-like images, but it has curves for almost anything, so you can get quite some crazy results from it. This way the Editor doesn't use the texture you supplied via a PNG file for example, but instead it creates an image on its own. To generate the texture, select the "Generated" tab.

You can play with gradients...


Saturation change 1 Saturation multiply 1.0000
Minimal width 0 Minimal height 0

Background shape editor

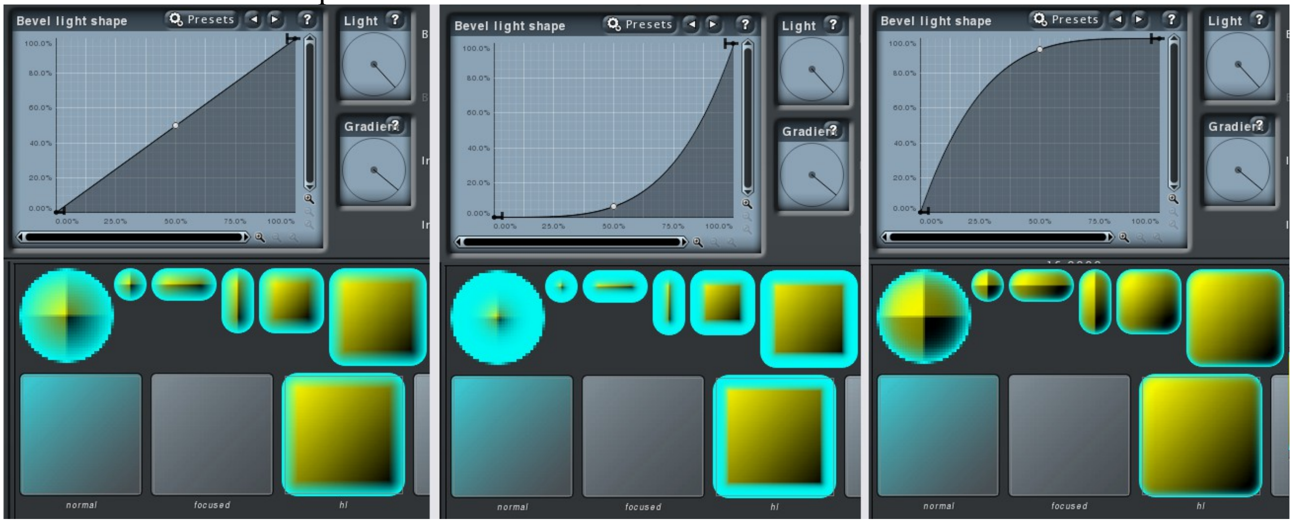
Generated | Image
Border type Solid Interior type Gradient
● Invert light ○ Invert gradient

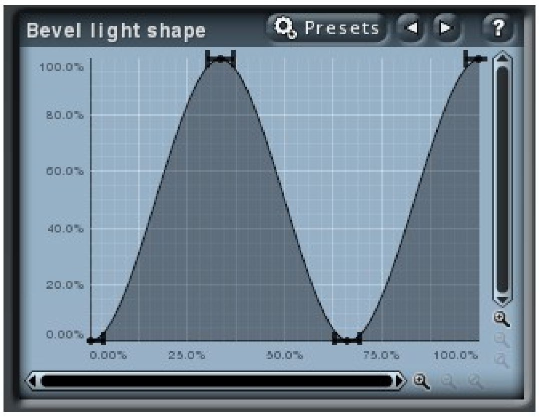
Border light color
Border dark color
Inner color 1
Inner color 2

Border distance 1.0233
Round rectangle radius 5

Operation Alphablend Alpha 200
Change color to main 0 200 Change color to main 1 0
Change color to main 2 0 Change color to main 3 0
● Colorize even saturation ● Minimimal size defined by corners

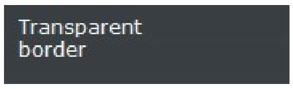
...border and interior shapes...

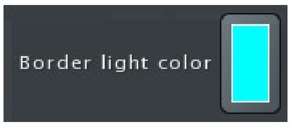


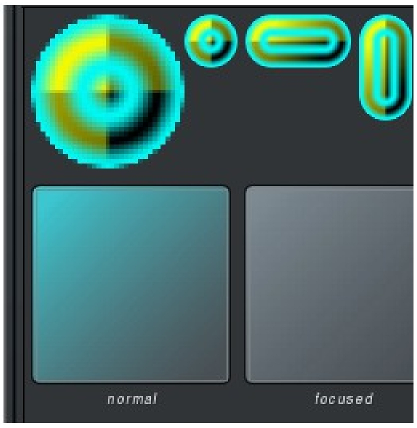


Outside
edge of border

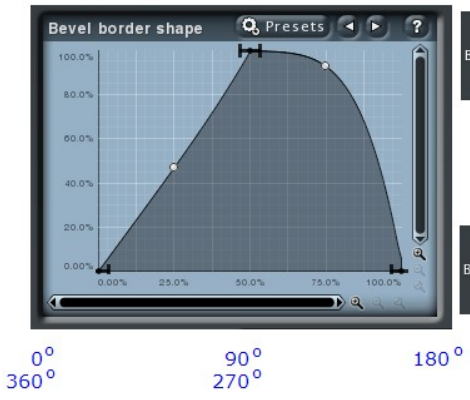
Inside
edge of border

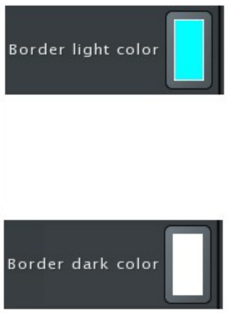

Transparent
border

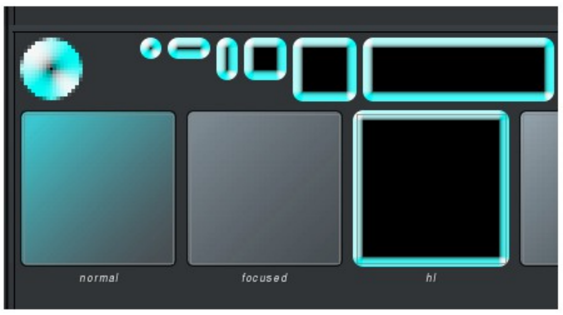

Border light color



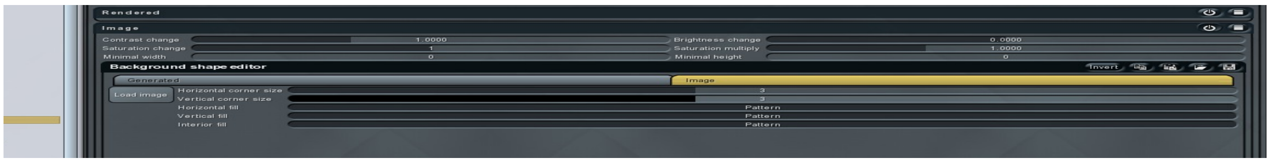
...light contours and more...







With both the image files and generated images you can postprocess the results using contrast, saturation and brightness modification:



Contrast change 1.0000
Brightness change 0.0000
Saturation change 1
Saturation multiply 1.0000
Minimal width 0
Minimal height 0

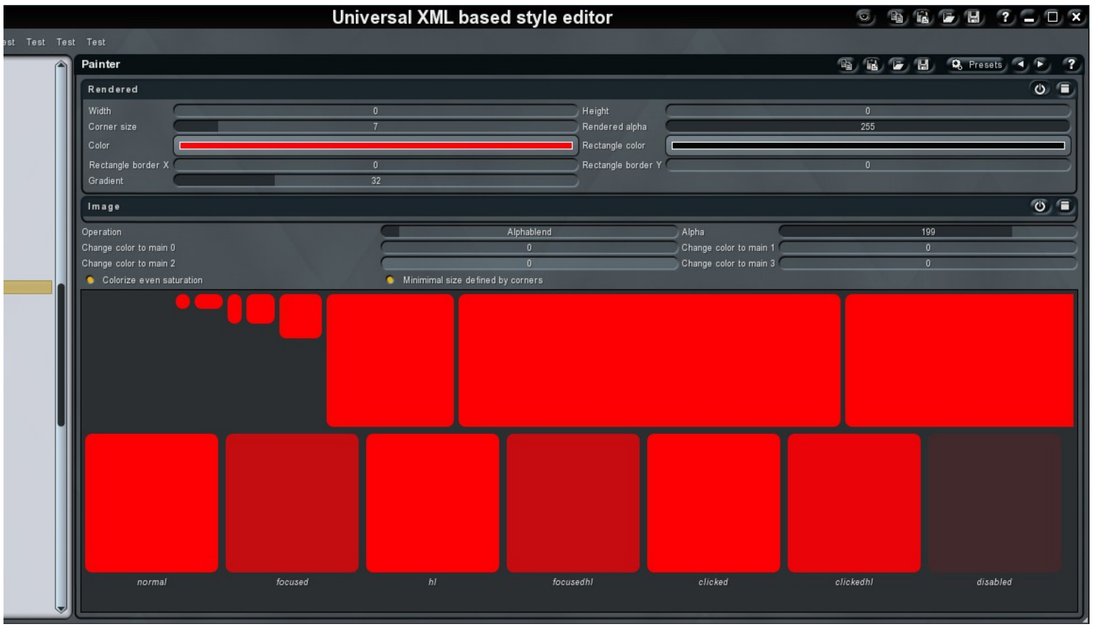
Horizontal corner size 3
Vertical corner size 3
Horizontal fill Pattern
Vertical fill Pattern
Interior fill Pattern

Possibilities are huge here, so I'll leave it to your experimenting.

Colorization and pixel operations

Remember the 4 colors users can change? Well, here we are with the explanation.

Imagine you paint a style, and a particular component will be really really red, because you like it that way:

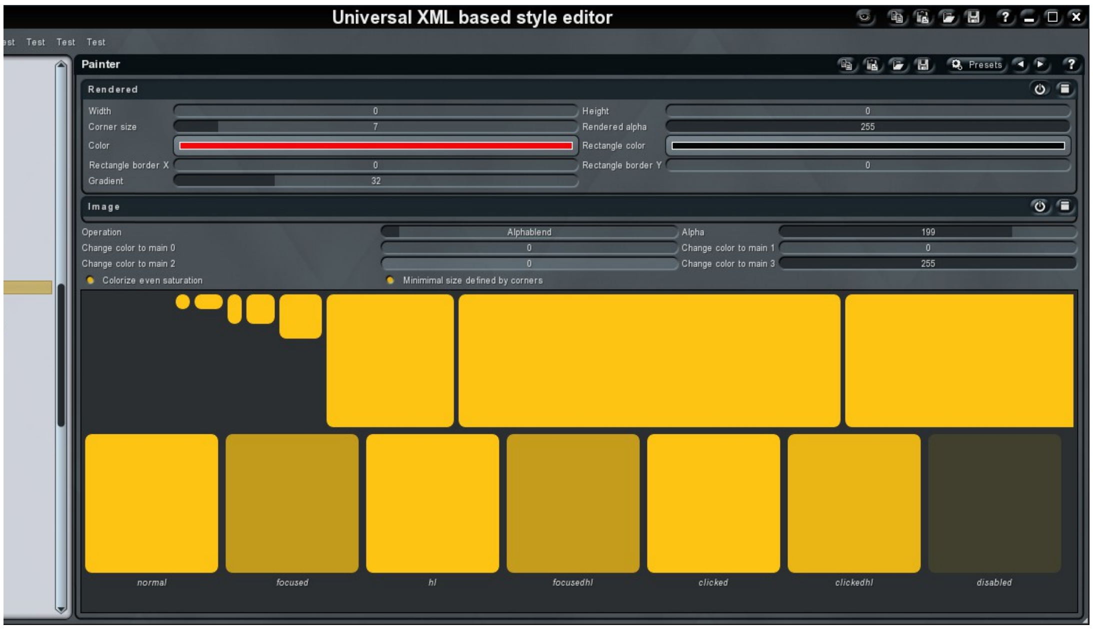


Width 0
Height 0
Corner size 7
Rendered alpha 255
Color
Rectangle color
Rectangle border X 0
Rectangle border Y 0
Gradient 32

Operation Alphablend
Alpha 199
Change color to main 0 0
Change color to main 1 0
Change color to main 2 0
Change color to main 3 0
● Colorize even saturation
● Minimimal size defined by corners

But some people don't like red so much, they want yellow! So you have several red components in your style and such person will just hate it, because it's red, it's ugly! :)

You can either change the style, which means a lot of work, or "colorize" it! The user can specify 4 colors, so since he likes yellow, he can set one of the four colors to yellow and all you need to do is ensure the style will change the color accordingly. And our engine makes this extremely simple – each painter has colorization sliders for all 4 colors, when 0, nothing happens, but if the value is higher, the painter gets colorized, like this:

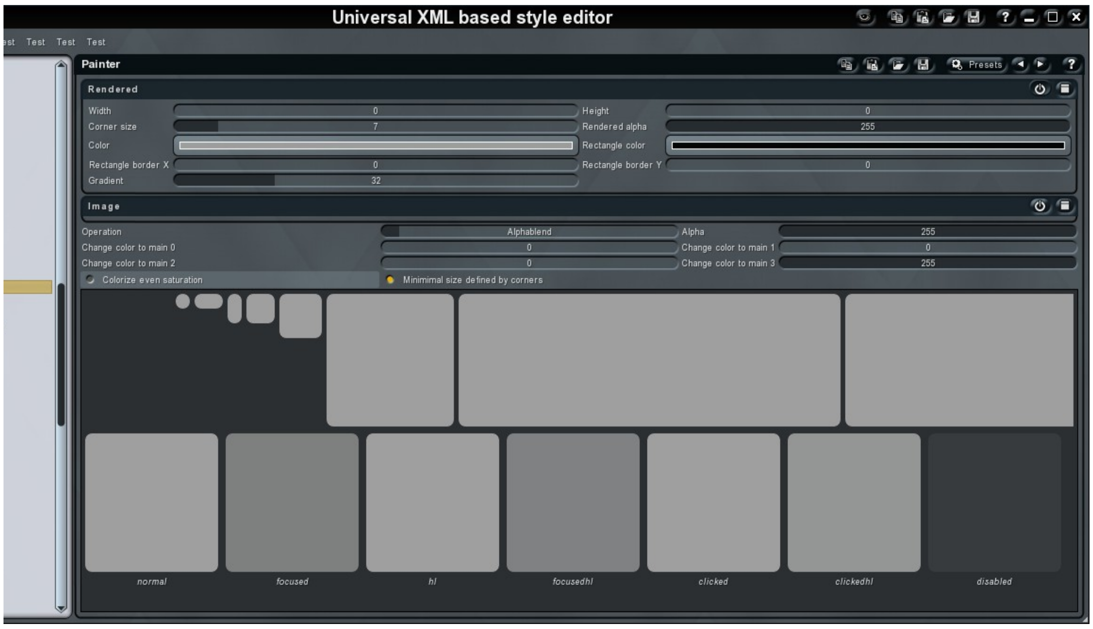


Width 0
Height 0
Corner size 7
Rendered alpha 255
Color
Rectangle color
Rectangle border X 0
Rectangle border Y 0
Gradient 32

Operation Alphablend
Alpha 199
Change color to main 0 0
Change color to main 1 0
Change color to main 2 0
Change color to main 3 255
Colorize even saturation
Minimimal size defined by corners

How does this work? The engine takes all colors used in the rendering and modifies them. In the image above the "Rendered" painter is used with black border and red interior. So before anything happens, the red is processed using yellow and in our case the "Change to Main3" slider is at maximum and the Main3 color is specified to be yellow (have a look back at the Colours section), so it just wins converting the yellow to red. With images it's more complicated and it needs to be performed for all pixels in the image, but the idea is the same.

Now what happens if the element's color wasn't red but grey? Same thing, yellow wins. Now note the little checkbox called "Colorize even saturation". Disable it and this is what you get:



Width 0
Height 0
Corner size 7
Rendered alpha 255
Color
Rectangle color
Rectangle border X 0
Rectangle border Y 0
Gradient 32

Operation Alphablend
Alpha 255
Change color to main 0 0
Change color to main 1 0
Change color to main 2 0
Change color to main 3 255
Colorize even saturation
Minimimal size defined by corners

It didn't work, because grey doesn't have any saturation. Simply put, a color can be defined by RGB, that's how our displays work, but for humans it is a little hard to grasp that concept. So there is an alternative colour representation called HSV (hue+saturation+value), which is way easier to use. Hue controls the color, red, yellow, or whatever. Value is the brightness, 0 makes it black, 255 white. Finally saturation controls amount of "color", with 0 the color is grey (or black..white), with 255 it is as "colorful" as possible.

So, if you have a grey rectangle, then all pixels have saturation of 0. The colorization changes the hue from grey to yellow, but the saturation is still 0, so it is still grey. But when "Colorize even saturation" is enabled, then the saturation is modified too and grey becomes yellow. The only thing left from the original color is its brightness and alpha.

It's necessary to say that you can colorize each painter using more than one color, but it takes CPU when the style is loading and it probably won't help you much.

The 4 colors

Now the question is, which color should you use to colorize particular element? It's up to you, but we suggest following the rule shown in the Style Configuration window that users use to change style, dimensions and colors:

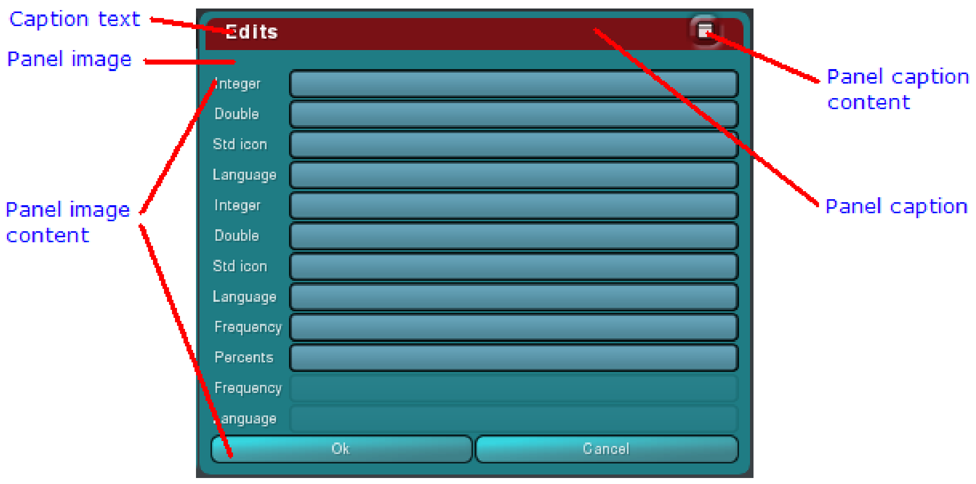
Color 0 : Background (window, menu...)
Color 1: Title background (window caption, menu caption...)
Color 2: Editors – background of control, edit, tracker, knob...
Color 3: Selectors – indicators in trackers, knobs, progressbars

Operation

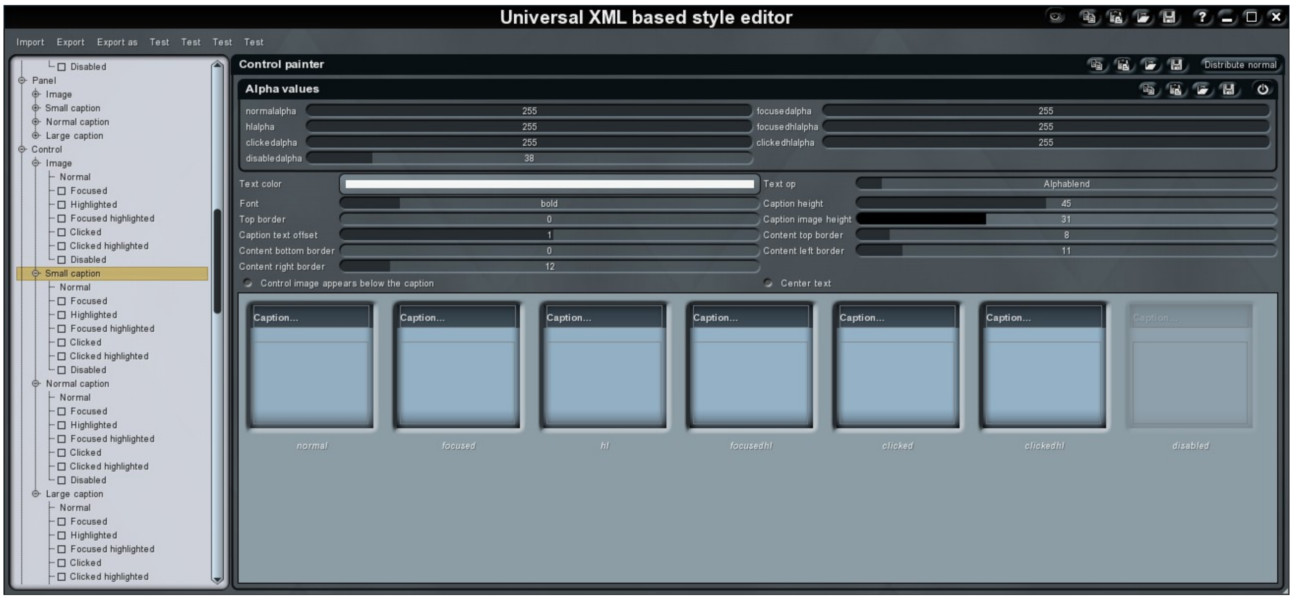
By default the rendered image is alpha-blended with the background. So it kind of overlays the background. You can use different pixel operations, but these will take more CPU and most likely won't be very useful.
It is a good idea to set "Copy" for windows and menues, or just anything that covers the whole rectangle leaving nothing under it, because copy operation is faster than alpha-blending.

Components with captions

We now know that a painter performs a rendering for a single object and the right painter is chosen in control painter according to component's state. Many components have more objects in them, the main Image and a Caption for example, so let's see how that works.



The Component Image has its own content. The Caption appears on top of the Image and typically contains some text (for example, the component name) and other content (caption buttons). Just select a small caption painter for example:



normalalpha 255
hlalpha 255
clickedalpha 255
disabledalpha 38
focusedalpha 255
focusedhlalpha 255
clickedhlalpha 255

Text color
Font bold
Top border 0
Caption text offset 1
Content bottom border 0
Content right border 12
Text op Alphablend
Caption height 45
Caption image height 31
Content top border 8
Content left border 11
Control image appears below the caption
Center text

The "Caption" object is very similar to "Image" object, it has all the alpha values, 7 control states, but it also has some more parameters – font, text color, parameters of the content position. And also it has 2 heights – caption height and caption image height. The idea is that the caption can actually be higher than what you see, so you can get the right bottom border.

In all cases a component (e.g. window) with caption is rendered like this:

1. The main Image is rendered covering the whole component.
2. The caption is painted on top overdrawing the background.

Note that such a control can have 3 different captions (small, normal or large) or none at all. The choice is made within the plugin’s code.

Additional compound components

We will now go through a few more components which has more than just an Image painter, just so that you have a head start, but really, it's all over the same thing again ;).

Menu

Menu is another component with a caption and in most cases these will look quite alike. It however has 2 more control painters, Selector and Selected item, which work exactly as any other painters, so no news here ;).

Checkbox

Checkbox has the default Image and Image for state On, Image for state Off and Image for state Grayed. The interesting part is that none of the additional painters are resized, they always have the same size. After all, these are just the checks on the left of the component. Another interesting part is that the main Image can actually be empty, or more likely empty for the normal control state. Why? Look at a checkbox, it's just text or some other content! It may need some highlight for mouse or for keyboard focus, but normally nothing is going on there really, unless you want to of course ;).

Tracker

Trackers, also called sliders, have 3 painters – Image as usual, but then there is Selector and Value button. Selector is interesting because it may not be completely painted, it depends on the current value of the slider. Value button may not even be present, if it is, it is painted on the right of the control and is clickable resulting in popup window for text input.

That's it!

Well, we are finished for the moment. Don't hesitate to ask at info@meldaproduction.com, but support for this thing may be a little limited. Anyway good luck and we are looking forward to seeing your styles!