USF skin format reference v1.8

1. USF skin format basics

The skins in all skinned RivaTuner technology based products (RivaTunerStatisticsServer, D3DOverrider, EVGA Precision or MSI Afterburner) are represented by *.USF files stored in the .\Skins subfolder in the application folder. USF skin files are compiled from separate bitmap images defining appearance of skin base, appearance of each skin control in all possible states (e.g. normal, focused (hot) and pressed button appearance) and special text header file, mapping these separate bitmap images to each control, defining positions of each control in the skin and so on. Each skinned application (i.e. RTSS.exe, D3DOverrider.exe, EVGAPrecision.exe or MSIAfterburner.exe) contains built-in command line based USF skin file compiler and decompiler. So you can always decompile and modify default skins or add your own skins to each of these products. It is recommended to use default skins as tutorials before trying to create your own skins. The skins can be decompiled using the following command line switch:

```
<application_name> -ds <skin_name>[,<skin_password>]
```

where ***<application_name>*** is the name of application with integrated skin decompiler, ***<skin_name>*** is the name of skin file to be decompiled and ***<skin_password>*** is a password, which can be optionally defined by skin author to protect it from decompilation. The following passwords are used for the following RivaTuner skinning technology based products:

RivaTunerStatisticsServer ***: jsRfsgTd***
D3DOverrider ***: ye6fyhmR***
EVGA Precision ***: jsRfsgTd***
MSI Afterburner ***: te6Xfjz3***

For example, default MSI Afterburner skin (***.\Skins\default.usf*** file in MSI Afterburner folder) is compiled with ***te6Xfjz3*** password, so it can be decompiled using the following command line switch:

```
MSIAfterburner.exe -ds default,te6Xfjz3
```

Similar to that, default EVGA Precision skin (***.\Skins\default.usf*** file in EVGA Precision folder) is compiled with ***jsRfsgTd*** password, so it can be decompiled using the following command line switch:

```
EVGAPrecision.exe -ds default,jsRfsgTd
```

After decompiling the skin decompiled source files will be stored in ***.\Skins\<skin_name>*** subfolder. You can modify them and compile modified skin using the following command line switch:

```
<application_name> -cs [-c] <skin_name>
```

where ***<application_name>*** is the name of application with integrated skin compiler, is and ***<skin_name>*** is the name of skin to be compiled. This command will create ***.\Skins\<skin_name>.usf*** file from the source files stored into ***.\Skins\<skin_name>*** folder. For example, to compile edited default MSI Afterburner or EVGA Precision skin use:

```
MSIAfterburner.exe -cs default
```
or
```
EVGAPrecision.exe -cs default
```

Similar approach can be used to create new skins, just use decompiled default skin as a template and copy ***.\Skins\default*** folder contents to your new skin folder (e.g. ***.\Skins\MySkin***). After doing so you may edit the header to specify new skin name, author, password and redefine control positions in the header if needed then modify bitmaps for the controls you are about to change. The next chapters will discuss skin source files format in details.

Please take a note that skin engine supports skin files compression, optional ***-c*** command line switch can be used to compress resulting skin during compilation. Compressed skin file size can be much smaller than uncompressed one (average compression ratio is 5x), however compressed skin usage may slow down application startup a bit due to runtime skin decompression. Also, if you are distributing your own skins collection containing multiple independent skins, then keeping your skin files in uncompressed format is probably a better idea, because multiple uncompressed files will compress much better in a solid archive comparing to multiple independently pre-compressed files.

2. USF images format

All source images representing different skin parts and controls must be stored in BMP or PNG format. Storing images in different format will result in skin compilation error.

3. USF header format

The header is ASCII text file defining all information related to the skin (e.g. skin name and author information, positions and styles of all controls in the skin and so on). The header must have "***;HDR***" signature in the very beginning, otherwise the skin won't work. The header format is derived from standard INI file format, so it is composed of multiple sections containing different fields. The list of sections is specific to the skin, but there are two common sections for all skins. Such common sections are [Global] and [Bitmaps] sections, the next chapter will discuss them in details.

3.1. [Global] section format

This section defines general information about the skin. This information includes skin file format identifier (current format is v1.3 and all information below will apply to this format only), skin host application name, skin name, optional author name, optional password and the list of skinned windows defined by the skin. For example:

```
[Global]
Format                  = 00010003h
Host                    = MSIAfterburner.exe
Name                    = MSI Afterburner skin
Author                  =
Password                = te6Xfjz3
Windows                 = WND_MAIN
```

Format field is required to inform the compiler and skinned application about the features supported by this skin. Format of some other fields and other sections depends on skin format defined with this field (for example, bitmap effects discussed in the chapter 3.6 are supported by format v1.1 and newer only).
Host field allows skin engine to filter skin files designed for different applications, for example if someone erroneously try to copy EVGA Precision skin files to MSI Afterburner skin folder, such files will not be visible to MSI Afterburner skin browser due to filtering the skins by this field.
Name and ***Author*** fields are optional and define the information you see in the skin browser.
Password field is also optional and it can protect your skin from being decompiled by others if you don't want to share some of source files (e.g. copyrighted raster font) with others.
Windows field list all skinned windows defined by the skin. In our example, there is only one skinned window identified by ***WND_MAIN*** ID.

3.2. [Bitmaps] section format

This section lists all bitmap images used in the skin and maps so called bitmap IDs to source bitmap images. All images in the skin are further referenced in the skin by IDs instead of explicit bitmap names. The skin compilation will fail if at least one ID is mapped to missing bitmap file or to a file having unsupported format (i.e. not BMP or PNG file). For example, this section definition may look like:

```
[Bitmaps]
BMP_PREVIEW                     = Preview.bmp
BMP_BASE                        = SkinBase.bmp
…
BMP_PROFILE1_HOT                = Profile1Button.bmp,Hot
…
```

In this example ***BMP_PREVIEW*** ID is mapped to ***Preview.bmp*** image and ***BMP_PROFILE1_HOT*** is mapped to ***Profile1Button.bmp*** postprocessed with ***Hot*** bitmap effect. Chapter 3.6 will discuss bitmap effects usage.

3.3. [Fonts] section format

This section lists all GDI fonts used in the skin and maps them to font IDs. All GDI fonts in the skin are further referenced in the skin by IDs. Each font defined in this section uses the following description format:

```
<font_id> = <font_path>,<font_typeface>,<font_height>,<font_weight>
```

where ***<font_path>*** is reserved for future versions for storing path to custom GDI font file redistributive, ***<font_typeface>*** is standard GDI font typeface name, ***<font_height>*** and ***<font_weight>*** are standard GDI font height and weight

3.4. Window description sections

As mentioned in the chapter 3.2, each USF skin file can define appearance of multiple independent skinned windows. The list of such skinned windows is defined by ***Windows*** field in ***[Global]*** section. Each skinned window has text ID associated with it, the ID is defined inside skinned application and must not be changed inside the skin. For example in MSI Afterburner, skinned application creates just one skinned window identified by ***WND_MAIN*** ID. This window must be listed in ***Windows*** value in ***[Global]*** section and also have a personal section defining the window properties. Window section name matches with window ID, so in for our example the header file must also contain ***[WND_MAIN]*** section having the following definition:

```
[WND_MAIN]
Base                = 0,0,,BMP_BASE,BMP_BASE_ALPHA
Preview             = BMP_PREVIEW
```

Base field defines shape, size and appearance of skinned window base (background). All these things are indirectly defined by base bitmap image referenced in this field. ***Base*** field uses the following format:

Base = [<color_key_x>],[<color_key_y>],[<flags>],<base_bitmap_id>,<base_alpha_bitmap_id>

where ***<color_key_x>*** and ***<color_key_y>*** are coordinates of pixel, which color have to be treated as color key into bitmap ***<base_bitmap_id>***. Please take a note that color key usage allows defining non-rectangular window shapes, but if color key pixel coordinates are not specified then color key based transparency technology is not used. You may also define non-rectangular window shapes via alpha channel based transparency using ***<base_alpha_bitmap_id>.*** Please keep in mind that skinned application can be configured to ignore alpha channel based transparency, so always design the skin base to support both color key and alpha channel modes if you use specify alpha channel bitmap ID for base image. In our example skin engine defines window base (background) using ***BMP_BASE*** and ***BMP_BASE_ALPHA*** bitmaps and defines pixel in the top left corner (i.e. pixel with [0,0] coordinates) as a color key pixel.

<flags> is an item reserved for future usage.

<base_bitmap_id> and ***<base_alpha_bitmap_id>*** are IDs of image and alpha channel bitmaps defining window base appearance. Size of base bitmap defines size of window. Skin engine will fail to load the skin if there is no bitmap with ***<base_bitmap_id>*** ID in ***[Bitmaps]*** section. Please keep in mind that bitmaps ID validation is always performed during skin load, but not during skin compilation. So successfully compiled skin is not always a sign of successfully working skin, always test the skins you're creating on the real skinned application.

Preview field defines ID of bitmap containing preview image of your skin, this image is visible to end user in the skin browser dialog, so you can add some additional things like your logos or watermarks to the bitmap mapped to this ID.

3.5. Window controls description sections

Each skinned window also has the set of controls associated with it. The controls are grouped by control types and defined in ***[<window_id>.<control_type>]*** sections. For example, the following section defines buttons for ***WND_MAIN*** window:

```
[WND_MAIN.Buttons]
…
BUTTON_CLOSE = 503,033,,,,BMP_CPT_CLOSE,BMP_CPT_ALPHA,BMP_CPT_CLOSE_HOT,BMP_CPT_ALPHA,BMP_CPT_CLOSE_PRESSED,BMP_CPT_ALPHA_PRESSED,BMP_CPT_CLOSE_PRESSED,BMP_CPT_ALPHA_PRESSED
…
```

Format of control description fields is specific to the control types, however all controls types have some common items in definition. Common control definition has the following format:

<control_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<control_specific_items>

where ***<control_id>*** is text ID of the control (defined by skinned application, cannot be changed in the skin).

<x> and ***<y>*** are coordinates in pixels of the control in reference to top left corner of the skin base (or in reference to top left corner of site if specified). The coordinates define top-left, top-right, bottom-left, bottom-right corner of the center of the control depending on the alignment flags specified in ***<flags_override>*** item.

<flags_override> is optional bitmask item allowing the skin to redefine some control flags (e.g. toggle control visibility to hide some unwanted button or select horizontal or vertical slider alignment). Please take a note that the mask of flags which you're allowed to override is defined by skinned application, so it can protect some flags from override. It is up to skinned application creator to decide which flags you're allowed to override via the skin for each control. If this item is not specified, all flags are specified by skinned application. The following flags / bitmasks are common to all control types:

0x00000001 – control is visible
0x00000002 – control is enabled
0x00000010 – control is aligned by right edge

0x00000020 – control is aligned by bottom edge
0x00000040 – control is centered horizontally
0x00000080 – control is centered vertically

<z_order> is optional field defining control Z-order, which can be used to control layering of overlapping controls.
<site> is optional field defining parent site for the control. Please refer to the next subchapter to get more info about sites.
<control_specific_items> is the list of additional items specific to a control type.
The next chapters will discuss window controls description sections by control types.

3.5.1. Window site controls description section

Site controls are special controls which can be associated with group of child controls. Positions of site children are relative to the top left corner of the site and visibility of the children is controlled by the site visibility. The sites are also the only controls able to change shape of the window, the rest controls are clipped with the window shape during rendering. Skinned applications can use sites to represent different detachable panels, for example monitoring panel in MSI Afterburner or profiles panel in EVGA Precision are implemented as sites. Site controls descriptions are stored in ***[<window_id>.Sites]*** section, which has the following format:

<site_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<color_key_x>,<color_key_y>,<site_bitmap_id>,<site_alpha_bitmap_id>

where the first 5 items are common USF control description items.
<color_key_x> and ***<color_key_y>*** are coordinates of pixel, which color have to be treated as color key for bitmap ***<site_bitmap_id>***. If color key pixel coordinates are not specified, alpha channel based transparency will be used instead of color key based transparency.
<site_bitmap_id> and ***<site_alpha_bitmap_id>*** are IDs of bitmaps defining site image and site alpha channel. Alpha channel bitmap ID is optional and can be ignored if color key based transparency is used. Please keep in mind that all controls inside skinned window support true alpha blending, however final non-rectangular skinned window can be blended with desktop via color key based transparency only. So avoid using alpha channel for sites changing window shape and overlapping with window color key areas.

3.5.2. Window button controls description section

This type of controls represents traditional buttons as well as checkboxes (i.e. buttons staying in pressed state until the next button press) and spin buttons. Button controls descriptions are stored in ***[<window_id>.Buttons]*** section, which has the following format:

<button_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<button_bitmap_id>,<button_alpha_bitmap_id>,<button_hot_bitmap_id>,<button_hot_alpha_bitmap_id>,<button_pressed_bitmap_id>,<button_pressed_alpha_bitmap_id>,<button_pressed_hot_bitmap_id>,<button_pressed_hot_alpha_bitmap_id>[,<button_animation_matrix_x>,<button_animation_matrix_y>,<button_animation_matrix_frametime>]

where the first 5 items are common USF control description items. In addition to standard control flags, ***<flags_override>*** field for buttons can specify combination of the following button specific flags if it is allowed by skinned application:

0x00010000 - button is a checkbox button
0x00040000 - button is associated with command line action
0x00080000 - button is a spin button

<button_bitmap_id>,<button_alpha_bitmap_id>,<button_hot_bitmap_id>,<button_hot_alpha_bitmap_id>,<button_pressed_bitmap_id>,<button_pressed_alpha_bitmap_id>,<button_pressed_hot_bitmap_id> and ***<button_pressed_hot_alpha_bitmap_id>*** are IDs of bitmaps defining button image and button alpha channel for normal button state, focused (hot) state, pressed state and focused (hot) pressed state.
<button_animation_matrix_x>,<button_animation_matrix_y> and ***<button_animation_matrix_frametime>*** are optional items allowing to animate button and button pressing process. When button animation is in use, the button bitmaps are treated as the sprite matrices instead of single images and ***<button_animation_matrix_x>*** and ***<button_animation_matrix_y>*** items define horizontal and vertical matrix sizes (in sprite units). ***<button_animation_matrix_frametime>*** item defines animation frame time in milliseconds. Animation matrix rows define passive animation frames loop and matrix columns define animation frames for button pressing animation process. For example, iPhone OS styled animated on/off buttons in default RivaTunerStatisticsServer skin are implemented with this technology.

3.5.3. Window slider controls description section

Slider controls are similar to native Windows slider controls. Slider controls descriptions are stored in ***[<window_id>.Sliders]*** section, which has the following format:

<slider_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<slider_bitmap_id>,<slider_alpha_bitmap_id>,<slider_hot_bitmap_id>,<slider_hot_alpha_bitmap id>,<slider_bgnd_bitmap_id>,<slider_bgnd_alpha_bitmap_id>,<slider_selected_bgnd_bitmap_id>,<slider_selected_bgnd_alpha_bitmap_id>

where the first 5 items are common USF control description items. In addition to standard control flags, ***<flags_override>*** field for sliders can specify combination of the following slider specific flags if it is allowed by skinned application:

0x00010000 – slider is rendered horizontally
0x00020000 – slider thumb/knob path is defined by alpha channel

<slider_bitmap_id>,<slider_alpha_bitmap_id>,<slider_hot_bitmap_id>,<slider_hot_alpha_bitmapid>,<slider_bgnd_bitmap_id>,<slider_bgnd_alpha_bitmap_id>,<slider_selected_bgnd_bitmap_id> and ***<slider_selected_bgnd_alpha_bitmap_id>*** are IDs of bitmaps defining slider knob and background images and alpha channels. Please take a note that slider background images are provided independently for normal and so called selected states. Selected slider area is background area starting from the left edge of slider till knob position for horizontally aligned slider and background area starting from bottom edge of the slider till knob position for vertically aligned slider. Specifying different background images for normal and selected areas allows implementing different equalizer-styled filling effects on the slider. For example, green neon light effect filling selected area on the sliders in default MSI Afterburner skin is implemented with this technology.

3.5.4. Window text controls description section

Text controls display text messages and allow user to edit the text. Text controls descriptions are stored in ***[<window_id>.Texts]*** section, which has the following format:

<text_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<font_bitmap_id>,<font_alpha_bitmap_id>,<font_base_char>,<font_char_w>,<font_char_h>,<font_remapping_table>

where the first 5 items are common USF control description items. In addition to standard control flags, ***<flags_override>*** field for texts can specify combination of the following text specific flags if it is allowed by skinned application:

0x00010000 – text is rendered using GDI font (see notes below)

<font_bitmap_id> and ***<font_alpha_bitmap_id>*** are IDs of raster font image and alpha channel bitmaps. The font is represented as vertically aligned matrix of sprites representing independent characters. Width and height of each character in this matrix are explicitly specified in description, so the number of characters in the font is calculated via dividing font bitmap height by character height.
<font_base_char> is font base character, which can be specified in both hexadecimal ASCII code format as well as in plain text format. Base character is required to inform skin engine about the first (base) character in font image if we use font with incomplete characters set.
<font_char_w> and ***<font_char_h>*** specify font character width and height in pixels. Font character width item can additionally control characters spacing or force monochrome font width. If ***<font_char_w>*** is positive and less or equal to font bitmap width then each character width is calculated on per-pixel basis using the font bitmap clipped to ***<font_char_w>*** width. If ***<font_char_w>*** is greater than font bitmap width then each character width is also calculated on per-pixel basis using the font bitmap, and difference between ***<font_char_w>*** and font bitmap width is used to adjust spacing between characters. If ***<font_char_w>*** is negative then the font is treated and monochrome width font and absolute ***<font_char_w>*** value defines font width.
<font_remapping_table> allows you to force some characters to be remapped to others during rendering, it can be useful when creating the font with incomplete characters set. The remapping table is a sequence of space separated character code pairs, defining source and destination character codes. For example, ***20h 5Fh*** remapping table forces the space character (ASCII code ***20h***) to be replaced with underline character (ASCII code ***5Fh***).
Please take a note that for GDI font mode text control description uses slightly different format:

<text_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<w>,<h>,<font_id>,<initial_text>,<text_color>,<bgnd_color>

where ***<w>*** and ***<h>*** are width and height of control in pixels, ***<font_id>*** is ID of font, ***<initial_text>*** is a text used to initialize the control, ***<text_color>*** and ***<bgnd_color>*** define the text color scheme

3.5.5. Window placeholder controls description section

Placeholders are fake invisible controls, reserving place for some other non-skinable controls. For example, area for monitoring window in monitoring panel in default MSI Afterburner or EVGA Precision skin is defined with placeholder control. Placeholder controls descriptions are stored in ***[<window_id>.Placeholders]*** section, which has the following format:

<placeholder_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<w >,<h>, <placeholder_specific_items>

where the first 5 items are common USF control description items.

<w> and ***<h>*** items define width and height (in pixels) of area reserved by placeholder.
<placeholder_specific_items> is a list of additional items specific to placeholder ID. For example, monitoring window placeholder in default MSI Afterburner and EVGA Precision skin use additional items list to store information about color scheme of hardware monitoring window.

3.5.6. Window indicator controls description section

Indicators are the controls changing their appearance depending on some condition. Such conditions can be programmed inside skinned application (e.g. indicator can reflect temperature changes) or the indicator can be connected to some other control from the skin, e.g. to reflect some slider position or pressed/released state of some button. Green neon light effect, grouping core and shader clock sliders in linked overclocking mode in default MSI Afterburner skin, is implemented as an indicator control attached to the ***Link*** button. Indicator controls descriptions are stored in ***[<window_id>.Indicators]*** section, which has the following format:

```
<indicator_id> = <x>,<y>,<flags_override>,<z_order>,<site>,<indicator_animation_matrix_x>,<indicator_animation_matrix_y>,<indicator_animation_matrix_frametime>,<indicator_bitmap_id>,<indicator_bitmap_alpha_id>
```

where the first 5 items are common USF control description items. In addition to standard control flags, ***<flags_override>*** field for indicators can specify combination of the following indicator specific flags if it is allowed by skinned application:

0x00010000 – indicator is inverted
0x00020000 – indicator is rendered using blind effect
0x00040000 – indicator is rendered horizontally, can be used in conjunction with blind effect only
0x00080000 – indicator is smoothed, cannot be used in conjunction with blind effect

<indicator_animation_matrix_x>,<indicator_animation_matrix_y>,<indicator_animation_matrix_frametime>,<indicator_bitmap_id> and ***<indicator_bitmap_alpha_id>*** items define indicator animation parameters, animation matrix image and alpha channel bitmaps. ***<indicator_animation_matrix_x>*** and ***<indicator_animation_matrix_y>*** items define horizontal and vertical matrix sizes (in sprite units). ***<indicator_animation_matrix_frametime>*** item defines animation or smoothing frame time in milliseconds. Animation matrix rows define passive animation frames loop and animation matrix columns define animation frames for different states indication.
Please take a note that you can use fake indicators with a single animated row (i.e. indicators looking similar regardless or indicated parameter value) to add some passive animation to your skins.

3.5.7. Window additional button controls description section

As you probably noticed, by default skin is a passive thing that simply defines appearance of controls created by skinned application. However, skin engine supports some advanced features, allowing you to add your own buttons to your custom skin and program them to execute some external applications when it is clicked. In USF skins it can be implemented by defining so called additional buttons. Additional button controls descriptions are stored in ***[<window_id>.AdditionalButtons]*** section, which has the following format:

```
<additional_button_id> = <additional_button_command_line_action>
```

where ***<additional_button_id>*** is ID on additional button to be added to the skin and ***<additional_button_command_line_action>*** is a command line action to be executed on this button click.
For example, let's try to edit default MSI Afterburner skin and add one more help button (MSI Afterburner logo button in the top left corner of the window) and program this button to open www.guru3d.com website on clicking. To do it we'll need to add the button to ***[WND_MAIN.AdditionalButtons]*** as explained above:

```
[WND_MAIN.AdditionalButtons]
BUTTON_MYURL  = explorer.exe http://www.guru3d.com
```

However, it is not enough to get it working. We've just added new button template and linked it with action, however we still need to tell the skin engine where to display this button and how it should look like. To do it we'll need to define ***BUTTON_MYURL*** entry in ***[WND_MAIN.Buttons]*** similar to any other skinned button, for example:

```
[WND_MAIN.Buttons]
…
BUTTON_HELP = 56,37,,,,BMP_HELP,BMP_HELP_ALPHA,BMP_HELP_HOT,BMP_HELP_ALPHA,BMP_HELP_PRESSED,BMP_HELP_ALPHA,BMP_HELP_PRESSED,BMP_HELP_ALPHA
```

BUTTON_MYURL = 100,37,00040003h,,,BMP_HELP,BMP_HELP_ALPHA,BMP_HELP_HOT,BMP_HELP_ALPHA,BMP_HELP_PRESSED,BMP_HELP_ALPHA,BMP_HELP_PRESSED,BMP_HELP_ALPHA

As you see, we used BUTTON_HELP as a template and shifted it a bit to the left by adjusting X coordinate. Also it is important to pay attention to the fact that we inserted flags override item and set it to ***00040003h*** for new button to tell the skin engine that new button should be visible, enabled and associated with command line action.

3.5.8. Window additional indicator controls description section

Similar to additional buttons, discussed in the previous chapter, additional indicators serve for adding custom indicators or passive animations to your skins. As mentioned before, indicators are the controls that can be connected to any button or slider controls inside the skin and display its state. So we can connect additional indicators to any slider or button available in the program, for example connect additional indicator to fan speed slider and show some warning animation when fan speed slider position is set too low. Additional indicator controls descriptions are stored in ***[<window_id>.AdditionalIndicators]*** section, which has the following format:

<additional_indicator_id> = <source_control_id>

where ***<additional_indicator_id>*** is an ID of new indicator to be added and ***<source_control_id>*** is an ID of source control for indicator (i.e. the control, state of which is displayed by indicator). ***<source_control_id>*** ID must be valid control inside the skin, otherwise skin creation will fail. Similar to additional button, additional indicator entry for each ***<additional_indicator_id>*** must be also added to ***[<window_id>.Indicators]*** section similar to similar to any other skinned indicator.

3.6. Bitmap effects

Traditionally skin designer must prepare multiple bitmap images defining appearance of each control in the skin for different states (e.g. create images for each button when it is in normal state, when it is focused (hot) and when it is pressed). However, most of such images are traditionally represented by applying standard effects to original image, e.g. focused button state is often indicated by boosting normal button image brightness and pressed button state is often represented by scrolling normal button image down and to the right by a few pixels. So preparing such images in general is a routine task consuming a lot of time. To automate the process, skin format v1.1 and newer introduced so called bitmap effects support. The main idea behind bitmap effects usage is the idea of applying some pre-programmed effects to the source images on the fly during skin compilation instead of manually applying effects to input images. This approach reduces skin designer workload and reduces skin creation time drastically. The next chapters will guide you through bitmap effects usage.

3.6.1. Built-in and custom bitmap effects

There are two types of bitmap effects: built-in and custom bitmap effects. Built-in effects are pre-programmed inside the skin engine and can be applied to bitmaps without any additional effect definition. Any custom effects must be defined using effect description sections inside the skin.

3.6.2. Using built-in bitmap effects

There is only one built-in bitmap effect referenced by ***Alpha*** name: alpha channel extraction effect. You may apply this effect to input image file containing valid alpha channel information (e.g. 32-bit PNG file) to tell the skin engine to extract alpha channel from it instead of explicitly loading alpha channel image from independent bitmap file. You may also apply this effect to input image file containing no alpha channel information (e.g. 24-bit PNG file) to tell the skin engine to generate fully opaque alpha channel image.

3.6.3. Adding custom bitmap effects descriptions to the skin

All custom bitmap effect descriptions are represented by ***[BitmapEffect.<Effect_Name>]*** sections. Each effect can be composed from of the following atomar bitmap operators:

- Per-pixel bitmap color mixing
- Per-pixel bitmap brightness tuning
- Per-pixel bitmap contrast tuning

- Per-pixel bitmap gamma tuning
- Per-pixel bitmap hue tuning
- Per-pixel bitmap saturation tuning
- Per-pixel bitmap luminance tuning
- Bitmap scrolling

For example, in default MSI Afterburner skin focused (hot) button effect is represented as atomar brightness tuning operator:

```
[BitmapEffect.Hot]
Brightness          = 25
```

3.6.4. Atomar operators

Atomar bitmap operators are discussed in the next subchapters. It is important to note that atomar operators are always applied in the following order:

- Per-pixel bitmap color mixing
- Per-pixel bitmap brightness, contrast and gamma tuning in single pass
- Per-pixel bitmap hue, saturation and luminance tuning in single pass
- Bitmap scrolling

That is why the following effects descriptions are equal:

```
[BitmapEffect.Effect1]
Brightness  = 25
Gamma   = -25
```

is equal to

```
[BitmapEffect.Effect2]
Gamma   = -25
Brightness  = 25
```

3.6.4.1. Per-pixel color mixing atomar operator

Syntax:

ColorMix = <R_Weight_R>,<R_Weight_G>,<R_Weight_B>,<G_Weight_R>,<G_Weight_G>,<G_Weight_B>,<B_Weight_R>,<B_Weight_G>,<B_Weight_B>

Result:

Reconstructs each pixel RGB components using the following formulas:

NewR = OldR * <R_Weight_R> / 100 + OldG * <R_Weight_G> / 100 + OldB * <R_Weight_B> / 100
NewG = OldR * <G_Weight_R> / 100 + OldG * <G_Weight_G> / 100 + OldB * <G_Weight_B> / 100
NewB = OldR * <B_Weight_R> / 100 + OldG * <B_Weight_G> / 100 + OldB * <B_Weight_B> / 100

The operator can be used for various color mixing based effects, e.g. for converting color image to grayscale or sepia.

3.6.4.2. Per-pixel bitmap brightness tuning atomar operator

Syntax:

Brightness = <Delta_R>,<Delta_G>,<Delta_B>
or
Brightness = <Delta_RGB>

Result:

Changes each pixel color component brightness by specified delta. The operator is applied in single pass with contrast and gamma tuning via RGB gamma ramp LUT similar to display adapter's hardware brightness control.

3.6.4.3. Per-pixel bitmap contrast tuning atomar operator

Syntax:

Contrast = <Delta_R>,<Delta_G>,<Delta_B>
or
Contrast = <Delta_RGB>

Result:

Changes each pixel color component contrast by specified delta. The operator is applied in single pass with brightness and gamma tuning via RGB gamma ramp LUT similar to display adapter's hardware contrast control.

3.6.4.4. Per-pixel bitmap gamma tuning atomar operator

Syntax:

Gamma = <Delta_R>,<Delta_G>,<Delta_B>
or
Gamma = <Delta_RGB>

Result:

Changes each pixel color component gamma by specified delta. The operator is applied in single pass with brightness and contrast tuning via RGB gamma ramp LUT similar to display adapter's hardware gamma control.

3.6.4.5. Per-pixel bitmap hue tuning atomar operator

Syntax:

Hue = <Delta>

Result:

Converts each pixel RGB color to HSL, changes H component by delta and converts it back to RGB. The operator is applied in single pass with saturation and luminance tuning.

3.6.4.6. Per-pixel bitmap saturation tuning atomar operator

Syntax:

Saturation = <Delta>

Result:

Converts each pixel RGB color to HSL, changes S component by delta and converts it back to RGB. The operator is applied in single pass with hue and luminance tuning.

3.6.4.7. Per-pixel bitmap luminance tuning atomar operator

Syntax:

Luminance = <Delta>

Result:

Converts each pixel RGB color to HSL, changes L component by delta and converts it back to RGB. The operator is applied in single pass with hue and saturation tuning.

3.6.4.8. Bitmap scrolling atomar operator

Syntax:

Scroll = <X>,<Y>

Result:

Scrolls bitmap in both horizontal and vertical directions.

3.6.5. Applying defined bitmap effects

Once the effect descriptions are defined in the skin header as explained in the previous chapter, we can apply them to any bitmap used in the skin. As explained in chapter 3.2, all bitmaps used in the skin are listed in the ***[Bitmaps]*** section. It is really easy to add an effect to any of those bitmaps, just add a comma and bitmap effect name after any bitmap filename, for example ***Hot*** effect can be added to ***Profile1Button.bmp*** in the following way:

```
[Bitmaps]

BMP_PREVIEW                    = Preview.bmp
BMP_BASE                       = SkinBase.bmp
…
BMP_PROFILE1_HOT               = Profile1Button.bmp,Hot
…
```

Please take a note that you can apply as many effects to each bitmap as you wish, to do it just add more comma separated effect names to the bitmap description.

4. Common skin creation hints and tricks

1. Clickable area of skinned buttons and sliders is defined by associated alpha channel bitmap pixels, absolutely black (transparent) color identifies non-clickable areas. Knowing this fact, you can make any desired control area clickable if needed (even if the area is invisible) by setting alpha channel to close to but not equal to zero values. For example, ***Apply overclocking at system startup*** button in default MSI Afterburner skin adds button caption to clickable area using this trick.

2. You can use bitmap effects to easily create multiple different color scheme variations of the same skin. For example black, blue, dark, green and purple EVGA Precision skins are based upon the same default skin bitmaps combined with different ***Hue*** and ***Gamma*** tuning bitmap effects.

3. Keep in mind that skin engine precaches all bitmaps used in the skin when skinned application is running, so total size of all bitmaps in the skin directly affects runtime skin engine memory footprint. Remember that you can use the same bitmap for multiple controls at once or in combination with different bitmap effects. For example, ***Minimize*** and ***Close*** buttons in the caption of MSI afterburner window in default MSI Afterburner skin use different bitmaps for button images but the same bitmap for button alpha channel to reduce skin file size and runtime skin engine memory footprint:

   ```
   [WND_MAIN.Buttons]
   …
   BUTTON_MIN = 484,033,,,,BMP_CPT_MIN,BMP_CPT_ALPHA,BMP_CPT_MIN_HOT,BMP_CPT_ALPHA,BMP_CPT_MIN_PRESSED,BMP_CPT_ALPHA_PRESSED,BMP_CPT_MIN_PRESSED,BMP_CPT_ALPHA_PRESSED
   BUTTON_CLOSE = 503,033,,,,BMP_CPT_CLOSE,BMP_CPT_ALPHA,BMP_CPT_CLOSE_HOT,BMP_CPT_ALPHA,BMP_CPT_CLOSE_PRESSED,BMP_CPT_ALPHA_PRESSED,BMP_CPT_CLOSE_PRESSED,BMP_CPT_ALPHA_PRESSED
   ```

So always use existing bitmaps when possible instead of adding new ones.

4. During compilation of the skin with bitmap effects the compiler stores both original and postprocessed bitmaps (i.e. original bitmaps processed by bitmap effects processor) in the skin bitmap cache. Due to this reason, size of the skins using bitmap effects can be significantly bigger than size of traditional skins. The following rule is used to generate names of postprocessed bitmaps to the cache:

   <PostprocessedBimapName> = <OriginalName><EffectName1><EffectName2>...<EffectNameN>.bmp

   For example, if we have the following bitmap definition:

   ```
   BMP_APPLY_HOT = ApplyButton.bmp,Hot
   ```

   then both ***ApplyButton.bmp*** and ***ApplyButtonHot.bmp*** will be stored into the cache during skin compilation. One important limitation follows from this rule: you should avoid giving your bitmaps the names conflicting with automatic postprocessed bitmaps naming scheme. For example, the following bitmaps definition is invalid:

   ```
   BMP_APPLY_HOT1 = ApplyButton.bmp,Hot
   BMP_APPLY_HOT2 = ApplyButtonHot.bmp
   ```

   because ***BMP_APPLY_HOT1*** bitmap definition will add both ***ApplyButton.bmp*** and ***ApplyButtonHot.bmp*** to the cache and ***BMP_APPLY_HOT2*** definition will be ignored.

   Original bitmaps are added to the cache to ensure proper skin decompilation and modification. As mentioned above, it increases compiled skin size, so if you assume that your skin will not be decompiled and modified by others, you may significantly reduce compiled skin size by excluding original bitmaps from the cache. It can be done by ***-oc*** (optimize cache) command line switch specified during skin compilation, for example:

   ```
   MSIAfterburner.exe -cs -oc default
   ```

   You can also extract postprocessed bitmaps from the skin during decompilation using ***-dc*** (decompile cache) command line switch if needed. For example:

   ```
   MSIAfterburner.exe –ds -dc default,te6Xfjz3
   ```

   Please take a note that postprocessed bitmaps are decompiled regardless of ***-dc*** command line switch when decompiling the skin with optimized cache (i.e. the skin complied with ***-oc*** switch). Such skins can be re-compiled after decompilation, but you cannot edit bitmap effects because the skin doesn't contain the original bitmaps to be processed by the effect processor.