# ClassicPro::Flex

## Skinning Manual

## Table of Contents

# 1 Overview

Skinning for Winamp Modern can be very time consuming due to the need of XML and script knowledge. Especially people who are new to coding have big problems getting their great design concepts done as a working Winamp Modern Skin.

ClassicPro::Flex is trying to reduce coding for a freeform skin to an minimum so everyone can create a Winamp Modern skin straight out of a PSD file.

# 2 Getting Started

In order to create a new ClassicPro::Flex skin you will first need a design concept. I recommend using Adobe Photoshop for creating a skin design but if you do not want to buy this software GIMP is a very good free alternative.

It is also helpful to create your skin as XION skin first. XION's skin format bases on loading PSD files with tagged layers. So all you need is to give your flattened layer a name like 'play' if you want a button for starting playback. Goto http://xion.r2.com.au for more information about XION.

Back to topic – we want to create a ClassicPro::Flex skin. So goto your Winamp's skin directory (usually located in "C:/Program Files/Winamp/Skins/") and open the file "cProFlex – iFlex.wal" with a zip program (I recommend 7zip which can be downloaded from http://7zip.org). After this file is opened mark all files located within the archive using CTRL+A and click on extract. Extract the files to a new subfolder within the Winamp skin directory starting with cProFlex, like cProFlex – MySkin. If everything went fine you can open "C:/Program Files/Winamp/Skins/cProFlex – MySkin/" in Windows Explorer now.

## *2.1 skin.xml*

skin.xml is the most important file in this directory. It represents the entry point for the Winamp Modern skinning engine. It includes all author information and loads the ClassicPro::Flex engine. This is how skin.xml will most likely look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<WinampAbstractionLayer version="1.34">
    <skininfo>
        <version>0.1</version>
        <name>cPro::Flex iFlex</name>
        <author>Martin Pöhlmann</author>
        <comment>ClassicPro::Flex Base Skin</comment>
        <email>martin@skinconsortium.com</email>
        <screenshot>screenshot.png</screenshot>
        <homepage>http://www.skinconsortium.com/</homepage>
    </skininfo>

    <include file="colors.xml"/>
    <include file="@WINAMPPATH@/Plugins/classicPro/engine/load-flex.xml"/>

</WinampAbstractionLayer>
```

I will only explain the important things for you, all other stuff should be kept as is. The `<skininfo/>` Tag should hold your author information. Feel free to alter anything but `screenshot`.

### *2.2 flex.xml*

This is the second most important file and will be parsed by the ClassicPro::Flex engine. You can keep this file as it is for now. Details on how to use it can be found in chapter 4

### *2.3 Skinning Essentials*

Some nice to know stuff before we start creating the skin.

#### *2.3.1 Quick Skin Refresh*

You can easily refresh the current skin by hitting F5.

## 3 Bitmap Reference

Now that we have created a working directory for our skin design. So let's make the design go live. But first you need to know what all those funny symbols in the bitmap reference mean:

All files suffixed with a ***** are required files. Any other file can be deleted in order to remove its object from the layout

All files suffixed with a ~ are required if the previously explained file is found in the skin directory.

All files suffixed with a ^ are placed in another position in the image stack (see 3.1).

Some images are explained together. **$** is used within the filename to show that there are more images representing this kind of object (like cbutton-play$.png)

## *3.1 The Player Window*

The Player Window is the window holding all the playback controls and songinformation stuff. Basically there are two layouts: The 'normal' layout and the 'shade' layout.

The images for the normal layout are stored in YOUR_SKIN_DIR/player/normal/ and for the shade layout in YOUR_SKIN_DIR/player/shade/

Both layouts have the same core functionality so both directories hold the same files. Thus I only need to explain how you build the normal layout since the shade layout follows exactly the same rules except in bitmapID/gammaID every occurrence of normal is replaced by shade.

> ! *The order of how the following images are explained does also represent the order of how they are stacked in the layout if nothing else is mentioned. So background.png will be stacked below background-overlay1.png and so on.*

### *3.1.1 background.png **

```
BitmapID:   cpro.flex.player.normal.bg
GammaID:    player.normal.background
```

This file represents the background of your layout if no desktop alphablending [TODO screenshot] (smooth edges, shadows) is used and thus it defines the height and width of your layout. No other bitmaps are displayed in the area exceeding the bounding defined via this image!

Every time you save this file with a new dimension the layout will also get a new height/width if the skin is reloaded.

> ! *Keep in mind that buttons and other objects may 'loose' their correct placement if you resize this image!*

### *3.1.2 background-alpha.png **

```
BitmapID:   cpro.flex.player.normal.bg.alpha
GammaID:    player.normal.background
```

This file represents the background of your layout if desktop alphablending is enabled so it can include smoothed edges and a drop shadow.

> !! *File This file MUST have the same height/width and should have the same overall design as background.png*

### *3.1.3 background-snap.png*

```
BitmapID:   cpro.flex.player.normal.bg.snap
GammaID:    none
```

This bitmap should hold a rectangular area that defines the border where your layout will snap (dock) to other windows. If this file is not present the dimensions of background.png will be used for snapping.

> !! *File This file MUST have the same height/width as background.png*

### *3.1.4 background-overlay1.png*

```
BitmapID:   cpro.flex.player.normal.bg.overlay1
GammaID:    player.normal.background.overlay1
```

This file represents an overlay drawn above the background image and may be used to create sections with different colorthemes.

> ! *Keep in mind that all of the following files may not need to have the same height/width as background.png but I recommend to use the same size since it is easier just hiding the non-necessary layers in Photoshop and save the ones needed for a specific image (like a metal overlay).*

### *3.1.5 background-overlay2.png*

```
BitmapID:   cpro.flex.player.normal.bg.overlay2
GammaID:    player.normal.background.overlay2
```

This file represents an overlay drawn above the background image and background-overlay1 and may be used to create sections with different colorthemes.

> ! *For creating more themeable overlays you need to define them in the group player.normal.underlay. Please refer to [TODO]*

### *3.1.6 display.png*

```
BitmapID:   cpro.flex.player.normal.display
GammaID:    player.normal.display
```

Another overlay commonly used as the background of the display. Of course you can use this for anything else if you do not have a display in your design concept.

### *3.1.7 display-overlay.png*

```
BitmapID:   cpro.flex.player.normal.display.overlay
GammaID:    player.normal.display.overlay
```

This bitmap may be used to create structures above the display background. It has a different GammaID as the Background and thus it can be colored in a complete different way.

### *3.1.8 display-mask.png ^*

```
BitmapID:   cpro.flex.player.normal.display.mask
GammaID:    player.normal.display
```

Used to create half hidden areas in the display (see Songticker in iFlex skin). Has the same gammagroup as the display and is not clickable (ghosted).

> !! *This overlay is placed on top of the image stack below ghost.png so it can mask all elements like Songticker of Albumart.*

### *3.1.9 albumart-region.png*

```
BitmapID:   cpro.flex.player.normal.albumart.region
GammaID:    none
```

This image defines the bounding rect for an albumart object. Remove this file if you do not need such an object.

> *F.A.Q.*
> *How regions work*

### *3.1.10 classicvis-region.png*

```
BitmapID:   cpro.flex.player.normal.classicvis.region
GammaID:    none
```

This image defines the bounding rect for the classic visualizer. Remove this file if you do not want to display the visualizer.

> *F.A.Q.*
> *How regions work*

### *3.1.11 classicvis-colors.png ~*

```
BitmapID:   cpro.flex.player.normal.classicvis.colors
GammaID:    player.normal.classicvis
```

Defines the colors of the classic visualizer like in ClassicPro::One. The pixel at position (0,0) is the color of the analyzer peak and the pixels from (0,2) to (0,17) represent the analyzer band colors. The pixels from (2,0) to (2,4) define the oscilloscope colors.

> *This bitmap does need to have a minimum width of 3 pixels and a minimum height of 18 pixels, otherwise the visualizer will not be colored.*

### *3.1.12 cbutton-previous$.png*

```
BitmapID:   cpro.flex.player.normal.prev         (cbutton-previous.png)
            cpro.flex.player.normal.prev.hover   (cbutton-previous-hover.png)
            cpro.flex.player.normal.prev.down    (cbutton-previous-down.png)
            cpro.flex.player.normal.prev.glow    (cbutton-previous-glow.png)
GammaID:    player.normal.cbuttons
            player.normal.cbuttons.glow          (cbutton-previous-glow.png)
```

This button is used to play the previous track in your playlist. In order to make the button visible you will need to define at least the normal state. Down, hover and glow state are optional. (See 4.1.1 Button Glow for defining glow speed)

### *3.1.13 cbutton-play$.png*

```
BitmapID:   cpro.flex.player.normal.play         (cbutton-play.png)
            cpro.flex.player.normal.play.hover   (cbutton-play-hover.png)
            cpro.flex.player.normal.play.down    (cbutton-play-down.png)
            cpro.flex.player.normal.play.glow    (cbutton-play-glow.png)
GammaID:    player.normal.cbuttons
            player.normal.cbuttons.glow          (cbutton-play-glow.png)
```

This button is used to start playback. In order to make the button visible you will need to define at least the normal state. Down, hover and glow state are optional. (See 4.1.1 Button Glow for defining glow speed)

> *F.A.Q.*
> *How can i create a combined Play and Pause button?*

### *3.1.14 cbutton-pause$.png*

```
BitmapID:   cpro.flex.player.normal.pause       (cbutton-pause.png)
            cpro.flex.player.normal.pause.hover (cbutton-pause-hover.png)
            cpro.flex.player.normal.pause.down  (cbutton-pause-down.png)
            cpro.flex.player.normal.pause.glow  (cbutton-pause-glow.png)
GammaID:    player.normal.cbuttons
            player.normal.cbuttons.glow         (cbutton-pause-glow.png)
```

This button is used to pause playback. In order to make the button visible you will need to define at least the normal state. Down, hover and glow state are optional. (See 4.1.1 Button Glow for defining glow speed)

> *F.A.Q.*
> *How can i create a combined Play and Pause button?*

### *3.1.15 seek-fillbar.png*

```
BitmapID:   cpro.flex.player.normal.seek.fillbar
GammaID:    player.normal.seek.fillbar
```

Defines the full image of a fillable position bar. This means if the songposition is at 100% (255) the whole image will be shown. If the songposition is lower it is cut by seek-fillbar-map.png.

> *F.A.Q.*
> *How fillbars work*

### *3.1.16 seek-fillbar-map.png ~*

```
BitmapID:   cpro.flex.player.normal.seek.fillbar.map
GammaID:    player.normal.seek.fillbar
```

Represents a map file corresponding to seek-fillbar.png.

> *F.A.Q.:*
> *How fillbars work*

### *3.1.17 volume-fillbar.png*

```
BitmapID:   cpro.flex.player.normal.volume.fillbar
GammaID:    player.normal.volume.fillbar
```

Defines the full image of a fillable volume bar. This means if volume is at 100% (255) the whole image will be shown. If volume is lower it is cut by volume-fillbar-map.png.

> ? *F.A.Q.*
> *How fillbars work*

### *3.1.18 volume-fillbar-map.png ~*

```
BitmapID:   cpro.flex.player.normal.volume.fillbar.map
GammaID:    player.normal.volume.fillbar
```

Represents a map file corresponding to volume-fillbar.png.

> ? *F.A.Q.:*
> *How fillbars work*

### *3.1.19 ghost.png*

```
BitmapID:   cpro.flex.player.normal.ghost
GammaID:    player.normal.ghost
```

Commonly used to create a glass overlay above the display area. This image is not clickable.

# 4 flex.xml - Configuring your skin

Like stated in the introduction to the ClassicPro::Flex engine flex.xml is the second most important file needed in your skin. It defines all configurable stuff like font colors, the ClassicPro::Flex version your skin is defined for and even more. But let's go through this file step by step.

The all enclosing tag is <ClassicPro::Flex/>.

```
<ClassicPro::Flex version="1.20">
...
<ClassicPro::Flex/>
```

It has just "version" as attribute, that should be a floating point value representing the version number of your ClassicPro Plugin. The reason for this attribute is that you might develop on let's say ClassicPro 1.22 and if someone wants to open the skin with ClassicPro 1.20, he will be asked to upgrade to the newest ClassicPro version (due to mussing features in 1.20). The warning will NOT occur if you try to run it on ClassicPro 1.23 for example.

> ! *It is recommended to use the newest version of ClassicPro in this tag.*
> *Check http://cpro.skinconsortium.com for the latest version.*

## *4.1 Appearance Configuration*

All appearance stuff is enclosed by an <Appearance/> Tag

### *4.1.1 Button Glow*

Most of the buttons in ClassicPro::Flex can display a glowing layer if your mouse enters the button area. The glow fades in / out with a specific speed and type that can be defined here.

```
<ButtonGlow fadeInSpeed="0.3" fadeOutSpeed="0.5" type="flash"/>
```

**fadeInSpeed** – (*float*) amount of seconds till the glow is fully displayed

**fadeOutSpeed** - (*float*) amount of seconds till the glow is fully hidden

**type** - (hold|flash|bounce) whereas hold means that the glow will be displayed as long as the mouse is over the button, flash means that it shows on hovering the button and hides after the glow is full shown and using bounce causes the glow to fade in and out all the time the mouse is over the visible area of your button.

### *4.1.2 Combined Play / Pause Button*

Lots of the skins developed for Winamp have a combined Play Pause button. The PlayPauseButton tag can enable this behavior in your skin as well

```
<PlayPauseButton normal="1" shade="0"/>
```

**normal** – (*bool*) enables combined Play/Pause button in normal layout

**shade** - (*bool*) enables combined Play/Pause button in shade layout

# 5 Frequently Asked Questions

## *5.1 How regions work*

Lots objects (like the AlbumArt or the Songticker) are placed via regions. This means that you usually create a rectangular area without transparency in Photoshop which represents the bounding you would like to show (for example) the AlbumArt in. If you draw a circle the AlbumArt will be clipped to the circle's bounding but only the area masked by the circle itself will display the AlbumArt and the rest is hidden.

> ! *This non-rectangular clipping does not work for Text Objects yet. But I am working to fix this bug in the Winamp freeform engine itself.*

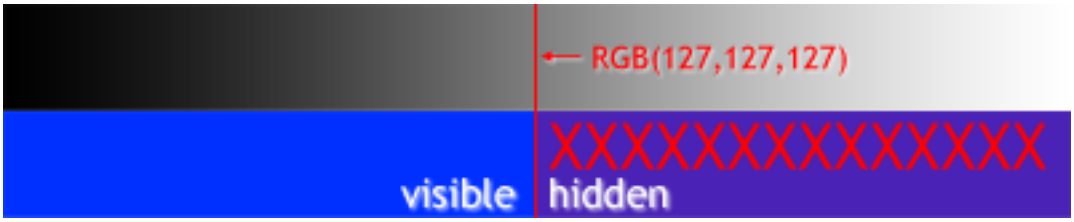
## *5.2 How fillbars work*

A fillbar usually consists out of two images. Let's take the volumebar as example: There is a volumebar-fillbar.png (fillbar) file and volumebar-fillbar-map.png (map) file in your layout folder.

As next thing you need to know that the Volume is internally stored in a integer-range from 0 to 255 with 0 being silent and 255 being the loudest.

If the volume is at 255 the complete fillbar should be shown and if volume is at 0 nothing should be shown. This would work nice if we would just have volume on or off, but the values in between (like 127) need to be handled as well. For this you need the nice map file, which should only contain greyscale values. The reason for this is that every volume value is mapped to a greyscale RGB value (0...255) on the map.

So if you have a volume of 127 the map is searched for all values with RGB color (127,127,127) or less. All those areas mask the visible part of your fillbar. All other areas of the fillbar that are not covered by this mask will be cut off (See picture).




*Picture 1: Cutting a fillbar with a map*

Just play a bit with the maps and you will see it is quite easy to use them.

A few additional notes: (a) The transparent areas of the map will always be cut off the fillbar. (b) Of course you can also make non-linear maps to stimulate a filled circle. (c) Making the map a little bit bigger than the actual fillbar and adding some extra RGB(0,0,0) and RGB(255,255,255) pixels helps to improve the handling of the fillbar for the users of your skin.