



Quintessential Media Player Skinning

Revision: 5.00-1-070219

Quinnware

Table of Contents

1. Overview	5
1.1. Goals.....	5
2. Introduction	6
2.1. The World's Simplest Skin	6
2.2. The World's Simplest Resizing Skin.....	7
2.3. World's Simplest Resizing Skin With An Extension	8
2.4. Now It's Up To You.....	8
3. Map Bitmaps.....	9
3.1. Internal Maps.....	9
4. Extensions	10
4.1. Extension Definition.....	10
4.1.1. Alternative Extension Definition	10
4.2. How Controls Belong to an Extension	10
4.3. Extensions Add Interactivity	11
4.4. Extensions In Depth	11
4.4.1. Extension Map Colors.....	11
4.4.2. Moveable and Fixed Extensions	11
4.4.3. The Body Extension.....	11
4.4.4. Extension Owners	12
4.4.5. Extension Z-Order	12
4.4.6. Extension Control Buttons	12
4.4.7. Extension Resizing	12
5. Break Map Bitmaps	13
5.1. Creating the Break Maps	13
5.1.1. Break Lines	13
5.1.2. Centering Rectangles	14
5.2. Determining which Extensions Break Lines Affect	14
5.2.1. Resize Controls	14
5.2.2. Edge Resize Controls.....	15
5.2.3. Translating Extensions	15
6. Modes	16
7. Skin Controls	17
7.1. Controls	17
7.2. Control States and Behaviors	17
7.2.1. The Four Button States	17
7.2.2. Simulating a two-state button	17
8. Control Set Bitmaps	18
8.1. Control Sets	18
8.1.1. Common Image Layout for Control Sets	18
8.1.2. Reproducible Controls	18
8.1.3. How Skin Bitmaps Affect Player Performance.....	19
8.1.4. <i>Body.bmp</i> – The Body of the Skin	19

8.1.5.	<i>ButtonSet.bmp</i> – The Playback Control Buttons	20
8.1.6.	<i>WindowSet.bmp</i> – Common Controls	20
8.1.7.	<i>ExtBtnSet.bmp</i> – Extension Controls	21
8.1.8.	<i>TrackSet.bmp</i> – The Playlist	22
8.1.9.	<i>EncoderSet.bmp</i> – The Encoder	26
8.1.10.	<i>TimerSet.bmp</i> – The Timer Display	26
8.1.11.	<i>SliderSet.bmp</i> – Sliders	28
8.1.12.	<i>EqSet.bmp</i> – Equalizer Controls	31
8.1.13.	<i>SkinCtrlSet.bmp</i> – Skin Control Buttons.....	31
8.1.14.	<i>ResizeSet.bmp</i> – Resize Control Buttons	32
8.2.	Other Skin Bitmaps	32
8.2.1.	Window Borders	32
8.2.2.	<i>MenuSet.bmp</i> – Menu Banners	34
8.3.	Non-Skinned Controls	34
8.3.1.	Title Bar Control.....	34
8.3.2.	Artwork Control.....	34
9.	Text Controls	35
9.1.	Text Control Types.....	35
9.1.1.	Current Media Info	35
9.1.2.	Current Audio Info.....	35
9.1.3.	Encoding Info	35
9.1.4.	Miscellaneous	35
10.	Fonts	37
10.1.	Windows Fonts.....	37
10.2.	Bitmap Fonts	37
10.2.1.	Creating a Bitmap Font.....	37
10.2.2.	Specifying Color for Text	37
10.2.3.	Specifying Text Shadows	38
10.2.4.	Specifying Text Alignment.....	38
10.3.	The <i>CharSet.ini</i> file	38
10.3.1.	Valid Font Settings	40
10.3.2.	Including Windows Fonts with your Skin	40
10.3.3.	Example: <i>Charset.ini</i>	40
11.	Skinnable Visualizations	42
11.1.	Skinnable Visualization Settings	42
11.2.	Skinnable Visualization Bitmaps.....	42
11.2.1.	Bitmap Format for VU and Peak Meter Visuals.....	43
11.2.2.	Bitmap Format for Resonators Visuals	43
11.2.3.	Bitmap Format For Waveform Visuals.....	43
12.	Skin Settings.....	44
12.1.	<i>SkinKid.ini</i> Sections.....	44
12.2.	<i>SkinFamily.ini</i> Sections	44
12.3.	Available Settings.....	44
12.3.1.	Settings By Section	44
12.3.2.	Settings By Functionality.....	44

12.4. Setting Descriptions.....45

13. Media Library Skinning51

 13.1.1. *MLSet.bmp* – Media Library51

14. Additional Skin Contents.....55

 14.1. Thumbnail.bmp55

 14.2. Comments.txt55

 14.3. SkinLayout.ini55

 14.4. SkinColor.ini55

 14.5. Custom Contents55

15. Testing a Skin56

16. Packaging Skins.....57

 16.1. Packaging a Kid Skin57

 16.2. Packaging a Family Skin.....57

 16.2.1. Method 1: Sharing common bitmaps between modes58

 16.2.2. Method 2: Combining kid skins.....58

1. Overview

The skinning system for the Quintessential Media Player (QMP) lets you change the look, the feel, the functionality, the focus, and to some extent even the purpose of the application. For this reason when designing a skin for the Quintessential Media Player, you are in fact doing graphical user interface (GUI) design. But for the purpose of this document, we will continue to refer to the task as *skinning*.

Skinning for the player involves understanding a simple system of bitmaps and colors. The system lets you create an arbitrary number of both modest and elaborate user interface scenarios.

On the simple end of the skinning spectrum are skins that can have as few as three bitmaps. And these bitmaps would consist of little more than images for the application itself.

On the complex end of the spectrum, a skin can consist of dozens of bitmaps, plus additional settings and font files, which can be stored across many folders. In addition the bitmaps can carry detailed nuances that enhance how the whole interface interacts with itself and with the user.

This broad spectrum lets even a novice skinner create great user interfaces, while allowing the experts room to explore and discover possibilities that haven't surfaced in previous skins, or even discover possibilities that aren't anticipated by this document.

1.1. Goals

The player's skinning system is designed to be as flexible as possible while remaining simple for skin creators. Its goal is to avoid the systems of configuration files and scripting that other complex skinning systems typically employ. For this reason, the skinning system may not be capable of doing *everything* a skin designer may have in mind. There are many rules and limitations that all skins must adhere to.

The most apparent constraint placed on the skin designer is that there is a pre-defined set of controls available for use on an interface. There is no mechanism for a skin designer to create a control outside of the pre-defined set. The set of controls available should cover most of the needs of media player and media manager application, which also limits the type of application that you can create by skinning alone. The flexibility of the system lies in the methods of customizing these controls and how they interact.

2. Introduction

Before we go into great detail, let's start by looking at a simple skin. Then we will progress with the addition of more complex but typical additions. This is to give first time designers the gist of skin creation.

2.1. The World's Simplest Skin

Here is a fully functioning media player skin:



Figure 1: “World’s Simplest Skin” interface

This skin consists of three bitmaps:

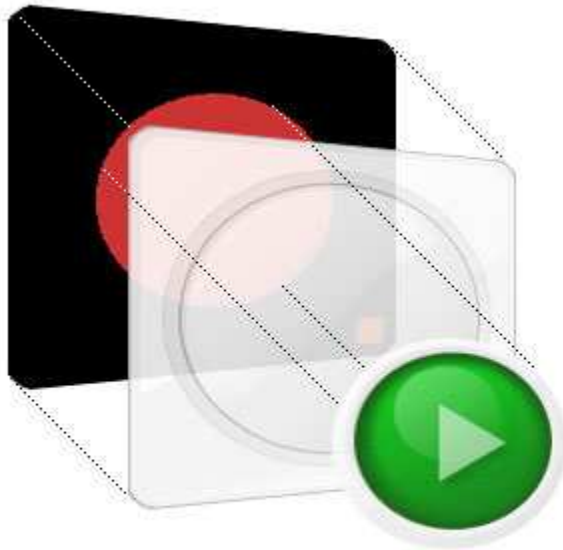
- The *Body Bitmap* that defines main player image;
- *Control bitmaps* for the Play control and its four states (controls normally have four states);
- The *Map Bitmap* that details the areas of the user interface.



Figures 2, 3, and 4: The three bitmaps for “World’s Simplest Skin”

The key concept of QMP skinning is the relationship between the Map Bitmap, the Body Bitmap, and Control Bitmaps. The Map Bitmap conveys the size, position, and relationship of the parts of the user interface to the player. The Body and Control bitmaps provide the images that the player uses to display the user interface.

It is important to notice how the Map Bitmap and the Body Bitmap align. The biggest trick to making skins is to get the Map Bitmap aligned with the image in the Body Bitmap. The Play button region on the Map Bitmap directly aligns to the Play button region on the Body Bitmap. And the Play buttons on the Control bitmap are exactly the same size as the region for the Play button on the Map Bitmap.



Hopefully from this example it is easy to see how to add other controls. By simply adding the area of the assigned color for the desired control to the Map Bitmap, and then creating the images for that control on the Control bitmap, you can add and position controls to your hearts content.

2.2. The World’s Simplest Resizing Skin

Let take our simple skin and add a rather complex feature to it: resizing.

Most skinning engines make short work of resizing because they are resizing rectangular windows and many just define a small graphic piece that it tiles as the window grows. But with QMP, a skin can be any shape and thus can have many areas in which simply tiling a single graphic is not enough. That this could make resizing the interface a rather complex feature, but the way QMP does this with bitmaps simplifies making it all work.



Figure 5: Resizing “World’s Simplest Skin”

To make the simple skin resize, we included a special Map Bitmap called a *Break Map*. The Break Map details where and how to divide the maps and images that make the skin, in order for the player to resize the whole skin.

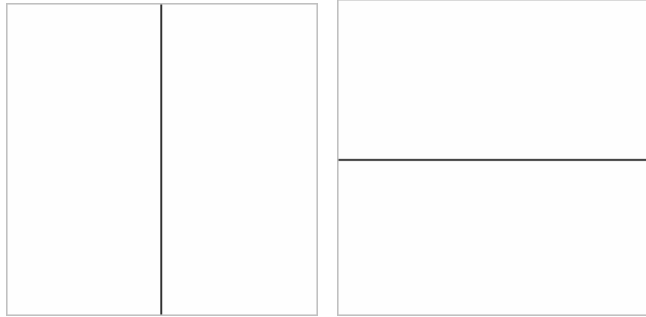


Figure 6, 7: Two Break Maps for resizing

The Break Map lines define where to divide and grow each control and extension defined on the Map Bitmap. The width of the lines on the Break Map defines what part of the image to tile to fill the resized area.

In this example the Break Map lines are centered horizontally and vertically, and are one pixel in width. Figure 5 shows how this Break Map causes the player interface to resize. You can see the Play button and background images have been tiled where the Break Map lines intersect those controls.

For further detail on Break Maps see Section 5.

2.3. World’s Simplest Resizing Skin With An Extension

Let take our simple skin one step further and add an *extension* to it. Extensions are an important concept to master to really expand skinning the player. You can use extensions to create secondary player windows or secondary areas for controls. But they also offer grouping for controls that need to behave in unison. Alternatively, they can provide overlapping functionality, animation effects, and many other alternate user interface experiences. Extensions are covered in detail in Section 4.



Figure 8, 9: Map with Body and Extension, Body bitmap with extension

This example shows a simple *fixed extension* that opens from the main body. The extension jutting out from the right side will show and hide depending on whether it is open or closed.

2.4. Now It’s Up To You

The “World’s Simplest Skin” examples are literally just the beginning, but do give the foundation for skinning. Hopefully, it makes the rest of the document easier to comprehend.

Now take the time to learn about the various controls and the bitmap nuances that are provided to allow you to create the skin you are imagining.

3. Map Bitmaps

Map Bitmaps define the overall layout of the skin. They provide the shape, location, and existence for of all the defined controls. Every skin must have at least one Map Bitmap. You can use multiple Map Bitmaps to define multiple regions of a skin. The main region of a skin is called the *Body*, and the map that defines this main region is called the *Body Map*. All other regions are called *extensions* and the map for these regions are referred to as *Extension Maps*.

Map Bitmaps depend on specific colors to convey skin information, so precise colors are required when creating them. You must not use anti-aliasing or any other effects on a Map Bitmap, just pure, clear, flat colors. For this reason Map Bitmaps should also be created in full resolution (24 or 32 bits-per-pixel bitmaps).

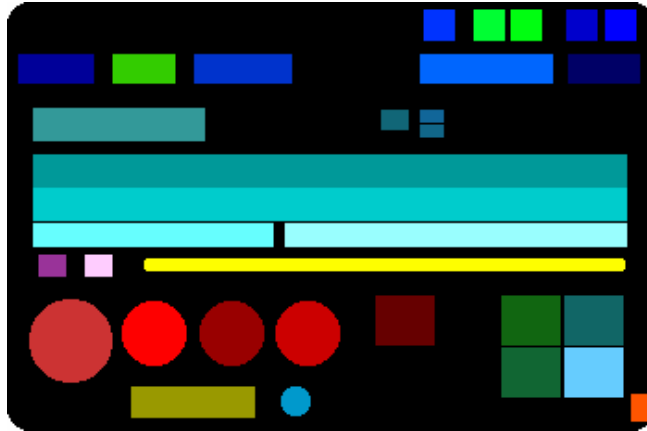


Figure 8: Example Map Bitmap

MapN.bmp bitmap files define the following:

- The Body and other extensions;
- The controls to display on the extensions;
- The overall geometry and layout of the skin;
- The shape of the individual controls.

The Body Extension's map color is always 0x000000 (black). Extensions are colored 0x112233 for Extension 0 (an odd color value due to legacy support), and 0x111111 through to 0x999999 for Extensions 1 to 9 respectively.

Colors that appear on the Map Bitmaps and are not defined will not be included as an area in the skin.

3.1. Internal Maps

Another concept similar to Map Bitmaps which you will encounter below when learning about [Control Set Bitmaps](#) are *Internal Maps*.

Internal Maps are the same as Map Bitmaps as they convey control information through their color, shape and size. However Internal Maps are located on the upper portion of some Control Set Bitmaps (see section 8 for more details on Control Set Bitmaps) rather than on their own bitmaps. The purpose of Internal Maps is to convey extra information for specific controls that cannot be conveyed on the Map Bitmap.

The content defined for each Internal Map is different for each Control Set. Internal Map specifications will be given in each of the Control Set Bitmap section where applicable.

4. Extensions

4.1. Extension Definition

Extensions in the simplest sense are regions of a skin interface. The main skin region is called the Body and all other regions are called extensions (but technically, the Body is an extension too).

Maps and extensions go hand in hand (so it is important to understand maps first). For each extension there must be a Map Bitmap that defines the extension and the controls that belong with that extension.

An extension by itself is merely a concept. You implement them using Body Bitmaps. Body Bitmaps are the images that will appear on the skin interface for the region defined by the extension. A Body Bitmap must share the same size and shape as the extension region, as it is drawn directly to the extension region.

For any skin there must be one Body and up to 10 extensions (thus 11 regions in total). Each extension is numbered from 0 to 9. The Map Bitmaps for the skin are named *map.bmp* for the Body Map Bitmap, and *map0.bmp ... map9.bmp* for the corresponding Extension Map Bitmaps. The corresponding Body Bitmaps are named *body.bmp* for the Body Bitmap, and *body0.bmp ... body9.bmp* for the corresponding Extension Body Bitmaps.

The position of the extension is defined by its position in the Map Bitmap. For example, if the upper left corner of the extension is at location 100,100 in its Map Bitmap, it will appear at location 100,100 in the player. Thus it is typical to have some unused space to the left and above the extension, as this space is just there to provide the proper offset to the extension.

The Body Image Bitmaps, which correspond to each of the Map Bitmaps, must be the exact size and shape of its corresponding Map Bitmap. However, the position of the Body image must always start at 0,0 in the bitmap. This makes positioning the Body image simple and cuts down on unused whitespace which makes the skin more efficient.

4.1.1. Alternative Extension Definition

If the skin has no overlapping extensions, this is an alternative method for defining extensions. This method is considered simpler since it uses less Map and Body Bitmaps and can potentially be just as efficient as the previous method. But there is no real reason not to use separate Map and Body Bitmaps for each extension.

Since there will be no overlapping extensions, you can define all extensions on the main Map Bitmap (i.e. *map.bmp*) and all Body images can be defined on the main Body Bitmap (i.e. *body.bmp*).

In this case, all extensions are defined on the one Map Bitmap at their destined locations, and all Body images are defined at their matching locations on the one Body Image Bitmap.

4.2. How Controls Belong to an Extension

Once you have the extension are defined, the next step is adding controls to the extension. The player will determine which extension a control belongs to by analyzing the position of the control relative to the extension around it.

A control “belongs” to (is owned by) an extension if:

- The extension color exists anywhere to the left of the control; or
- The extension color exists anywhere to the right of the control; or
- The extension color exists anywhere within the control.

If none of the above are satisfied, the control defaults to belonging to the Body. If two or more extensions satisfy these conditions, the first extension to meet the conditions (going from top to bottom, left to right) will be the owner extension.

There are scenarios where having the owner extension color just adjacent to the top or bottom of the control is not enough to establish ownership. If it is not possible to have the extension color to the left or right of the control, you will need to set at least one pixel somewhere in the region of the control to be that of the owner extension (usually somewhere on the outer bounds of the control).

4.3. Extensions Add Interactivity

Once you are familiar with creating extensions, the next step is to understand how to exploit them.

Extensions can open or close. They can resize or stretch. They can translate or slide. They can change z-order (the layering order on the screen—see Section 4.4.5). Extensions also have a multitude of settings that control their actions. Using these properties of extensions is where skins become really creative, interactive, and highly functional.

Extensions can be *fixed* or *moveable*. A skin's interface can consist of many regions that are independent of each other. These independent regions can move separately from one another and are defined by moveable extensions. Fixed extensions, on the other hand, are not independent. Instead they extend a moveable extension. (Note: the Body is always independent, or moveable). When the user moves a fixed extension, the moveable extension it extends moves with it.

Controls that belong to an extension will operate with the extension. They will show only when the extension shows, and the controls will maintain their relative positions to each other within the extension. To aid in exploiting extensions, instead of thinking of extensions as regions, think of them as control groupings. This way you can create extensions that show, hide, or move a group of controls.

Exploiting extensions is where the interesting aspects of creating a skin lie. There are more possibilities with extensions than this document can describe or anticipate. Explore them with intrigue and confidence.

4.4. Extensions In Depth

4.4.1. Extension Map Colors

Each mode of a skin can choose to utilize up to 10 other extensions. Each extension is numbered 0 through 9 and has map colors of 0x112233 for extension 0 (a perhaps odd choice of value due to legacy support), and 0x111111 through to 0x999999 for extensions 1 to 9 respectively. The user can press a hotkey for each extension. This key reflects the extension number (for example, Extension 1's hotkey will be **Shift+1**).

4.4.2. Moveable and Fixed Extensions

There are two types of extensions: *moveable* and *fixed*.

A *moveable extension* is one that the user can drag around the desktop and dock with other extensions and player windows. The Body is always a moveable extension.

A *fixed extension* is an extension that extends a moveable extension. (Note: the Body is always independent, or moveable.) When a fixed extension is moved by the user, the moveable extension it extends moves with it.

By default Extension 0 is moveable, and Extensions 1 to 9 are fixed (to provide backwards compatibility).

(See the [extdrag](#) setting to set moveable extensions.)

4.4.3. The Body Extension

By default, all skins have one extension that is called the Body. The Body is considered the main region of the player. Other moveable extensions that dock with the Body will move as one when the user drags the Body is dragged, and minimize when the Body minimizes.

(See the [extdragdocked](#) setting to give extensions similar docking properties.)

4.4.4. Extension Owners

Every extension except the Body has an *owner*. The owner of an extension is the extension that contains the button that opens the extension. If there is no such extension, the owner extension is the main Body. Also, if a skin manages to define any controls that do not reside on any specific extension, those controls will be assumed to belong to the main Body.

(See the [extNowner](#) setting to override an extension's default owner.)

Extensions can contain buttons to open other extensions, creating a branch effect. In the branching case, when the owner extension is closed, owned extensions close automatically.

(See the [extforced](#) setting to override this behavior).

4.4.5. Extension Z-Order

All extensions maintain a *z-ordering*. In other words, each extension knows what extensions it overlaps and what extensions overlaps it. When an extension opens, the player brings it to the top of the z-order and displays it above all other extensions (or more accurately, it will open the extension as close to the top of the z-order as possible as you can define other extensions to always be top-most). Moveable extensions will move to the top of the z-order when the user clicks on them using the mouse. Fixed extensions will not rise to the top of the z-order when clicked but it will raise its owner extension.

(See [extNafter](#) to set extensions that must remain above or below other extensions.)

4.4.6. Extension Control Buttons

Generally, extensions will have a control button that will open and close the extension. The extension's control button will appear pressed if the extension it controls is at its top-most position in the z-order. The control button will not appear pressed when the extension is not showing or is not at its top-most position in the z-order. This allows the extension control button to show or raise the extension when it is hidden or overlapped.

(See [extnotoggle](#) to set extensions that may not be closed by their control buttons.)

4.4.7. Extension Resizing

You can make extensions resizable using the Break Maps to define the section of an extension that can be stretched (see [Break Maps Bitmaps](#)). Both moveable and fixed extensions resize.

When extensions resize, by default all owned extensions that do not have their own resize controls will resize using the same break line that the owner uses to resize.

5. Break Map Bitmaps

Break Maps are Map Bitmaps that define areas where extensions and controls can be split in order to be resized. Resizing occurs by tiling the area of the skin that the lines on the Break Map overlap. You can achieve a variety of effects using Break Maps, including stretching, tiling, and translating (sliding). To simplify this section, we will use the term “resizing” to refer to all these effects.

To resize a part of a skin, the player divides the skin images at the points defined in the Break Maps. It moves apart the areas on either side of the break then fills in the area in between with the same area occupied by the break line tiled as many times as needed.

5.1. Creating the Break Maps

There are two Break Maps used for the entire skin. One defines *Horizontal Breaks* (*mapbreaksH.bmp*), and the other defines *Vertical Breaks* (*mapbreaksV.bmp*). All breaks for all extensions will be defined on these two bitmaps.

The Horizontal Break Map defines breaks for horizontal resizing. To facilitate a horizontal resize, a vertical *break line* is required. Don’t get confused between Horizontal Breaks and vertical break lines. Horizontal Breaks define horizontal resizing and go on *mapbreaksH.bmp*, but use vertical break lines.

Vertical Breaks define vertical resizing and go on the Vertical Break Map, *mapbreaksV.bmp*, but use horizontal break lines.

Each extension and control can have up to one horizontal and one vertical break. Multiple horizontal or vertical breaks per extension or control are not allowed.

5.1.1. Break Lines

A *break line* is much more than just a straight line dividing an area of the skin. There are a number of subtle properties of the break line that can produce quite different results on the final skin.

The color of the break line defines the type of break line it is. There are 10 types of horizontal and 10 types of vertical break lines. Each break line type is numbered from 0 – 9 and these are colored 0x000000 – 0x999999, respectively. There is no correlation between extension color and break line color. You can use any break line to define a break for any of the extensions.

A break line must be straight through each control or extension it passes, but it does not need to remain in the same straight line for *all* controls. Thus it can jump about dividing controls along different lines.

Break lines are not required to be contiguous and don’t have to go completely through a control. You can leave chunks out of a break line to allow for resizing of subsections of extensions or controls. Implementing non-contiguous break lines will produce odd effects in most scenarios, but you can do it. Using this property effectively will require extra testing.

All parts of a single break line must be the same defined color.

The width of a break line defines the width of the area to tile when the resize occurs. If your skin’s body has a pattern and you want the pattern to repeat at a certain width, make the break line align with the pattern, and match the pattern’s width. The width of the break line can be different for each control the break line intersects.

The break line’s position on the Break Map will determine where on the skin the break line will affect.

A single break line may go through multiple extensions.

5.1.2. Centering Rectangles

When resizing a control, you may not want to stretch the entire control. For example, in the case where a control has some text on it, it is likely that you want to leave the text un-stretched and centered on the control. To do this, define a *Centering Rectangle* on the Break Map. You can define Centering Rectangles on either of the Horizontal or Vertical Break Maps. There is no need to define the same Centering Rectangle on both.

The color of the Centering Rectangle must match the map color of the control it will apply to. The region specified by the Centering Rectangle must overlap an area of the control it applies to. The player will center the region of the control defined by the Centering Rectangle on the control as it resizes it. The player will not resize the centered region itself.

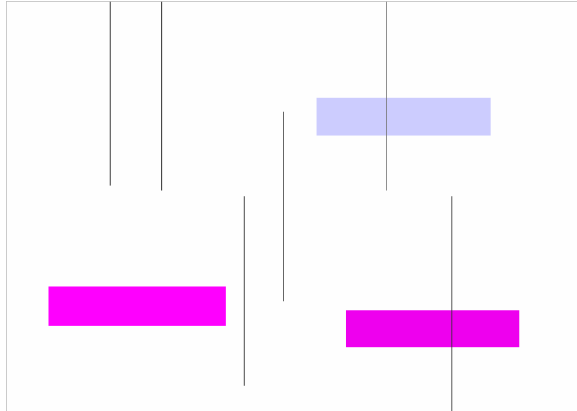


Figure 9: Example Break Map

This Horizontal Break Map defines six break lines to use to resize six different extensions (and the controls on those extensions). The two color regions are Centering Rectangles.

5.2. Determining which Extensions Break Lines Affect

To build the Break Map you need to visualize the break lines over top of the Map Bitmaps (the Break Map Bitmap should be the same size as the overall skin). What each break line affects is determined by its position on the Break Map. However whether or not a break actually resizes what is beneath it is determined by *resize controls*.

5.2.1. Resize Controls

There are 10 *resize controls* numbered 0 to 9 and are respectively colored 0xFF5500 - 0xFF5599. Each resize control is defined to control its corresponding numbered break line.

Extensions that own a resize control will resize based on the break line that the resize control operates. Extensions owned by the resizing extension will also resize based on the same break line, but only if they do not own a resize control themselves.

Aside from above, break lines will not affect extensions without resize controls.

A single break line can cross multiple extensions but defines the break for each extension independently. Thus you can have the same colored resize control on multiple extensions. In the scenario where a single break line affects multiple extensions, each resize control will only control the resize for its owner extension.

The position of the resize control relative to the break lines it controls will determine which direction the control resizes in. For example if the resize control is to the right of its horizontal break line and below its vertical break line it will resize to the right and down.

5.2.2. Edge Resize Controls

In addition to the resize controls described above, there are also *edge resize controls*, which are normally used for defining resize handles around the edge of a skin. Edge resize controls are only defined on Map Bitmaps and do not have any Control Set images. They follow the similar rules for normal resize controls above. You can use edge resize controls in addition to or in place of normal resize controls.

There are four types of edge resize controls defined, one for each edge of a rectangle: top, left, right, and bottom. Each type of edge resize control is numbered 0 to 9 and these correlate to the respectively numbered break lines.

- Left-edge resize controls are colored 0xEE1100 - 0xEE1199
- Right-edge resize controls are colored 0xEE2200 - 0xEE2299
- Top-edge resize controls are colored 0xEE3300 - 0xEE3399
- Bottom-edge resize controls are colored 0xEE4400 - 0xEE4499

Unlike normal resize controls, edge resize controls will only resize in their pre-determined directions.

5.2.3. Translating Extensions

For resizing, when a break line intersects an extension it is defined to resize, the resizing occurs by tiling the section under the break line and moving everything right of the break line to the right (in the case of resizing to the right). That is, everything outside the break line gets translated in the direction of the resize.

You can use this property to create an extension where the entire area translates, by putting the break line outside of the extension. For example, putting a horizontal break line to the left of the extension will cause the Resize Control on that extension to translate the entire extension to the right – thereby creating a ‘sliding’ extension.

See section 4 for more details on extensions

6. Modes

A *mode* is an “appearance” of skin. A skin can choose to have more than one appearance by implementing multiple modes. Each appearance can be subtly or dramatically different from the others.

For example, a skin can choose to have a full-featured interface and a separate more compact interface. Implementing each of these interfaces as a mode allows each interface to use all available control and extensions, and to have both be contained in a single skin.

Another example is a skin where two or more modes are strikingly similar, but differ slightly in the functionality they make available. Controls exist to switch seamlessly between modes such that the change is almost imperceptible, thus making Modes a powerful way to continue the interface experience across different interface scenarios.

A skin can have up to nine modes. Each mode can contain any and all features available to the skins.

There is a default ordering of modes in a skin, from 1 to 9. Controls can be added to a skin to allow it to the next mode, or to any specific mode. There are specific skin controls to switch to one of the specific mode numbers. (See [SkinCtrlSet.bmp](#) for skin control definitions). Or you can use the ‘next mode’ skin control to switch to the next mode in order. (See the [nextmode](#) setting to override which mode is considered next.)

7. Skin Controls

This section describes in detail the various controls available for skinning, how they are defined, and where to create the images for the controls.

7.1. Controls

Controls are the various interface components available to you as a skin designer. These include components such as the Play and the Stop buttons, the playlist, timers, scroll bars, volume controls, etc... Basically, everything you can interact with on a skin is a control.

Each control has a specific color assigned to it that you use to define the control's position and shape on the Map Bitmap. Some controls require further definition than what is provided by the Map Bitmap, by including details in an *Internal Map* on the Control Set Bitmap.

Many controls are reproducible. That is, they can exist more than once on a skin. Reproducible controls can only exist once on any extension, but by using multiple extensions, you can use a control many times.

Of course there are exceptions. Some controls do not require images that reside on a Control Set Bitmap. Controls that have such exceptions are described in detail in the Control Set Bitmaps sections.

See Map Bitmaps (Section 3) for details on Maps and Internal Maps.

See Extensions (Section 4) for details on extensions.

See Control Set Bitmaps (Section 8) for more details on controls and reproducible controls.

7.2. Control States and Behaviors

7.2.1. The Four Button States

Many of the controls available to skins have common interactions with the user. They respond when the mouse is over them and the user can click or press them. To cover these interactions, the skin requires four states to be created for these controls.

Each Button, regardless of behavior, uses these four states to display interaction with a control. The states provide important cues to the user about the function of the button and what will happen if the user clicks or releases the mouse button at that moment. You can make several of these appearances identical, but you must provide an appearance for all four states. All appearances of a button control must be the same size and shape.

The control set bitmap for each button includes the appearance for the following four states:

- **Normal:** The default state of the control. It is not pressed and the mouse is not over it.
- **Lit:** The mouse is over the control and the control is not pressed.
- **Pressed and unlit:** The control has been pressed but the mouse has moved off the control. This often occurs for toggle buttons (such as Repeat or Shuffle) where clicking the button leaves it in a pressed state. This can occur for a control button if the user presses the button then moves the pointer off the control before letting go of the mouse button.
- **Pressed and lit.** The control has been pressed and the mouse is over the control. This occurs for toggle buttons, as the *rollover* state when the button is locked in the pressed state. This can occur for normal buttons when the user presses a button down and has not released it yet. It serves as a cue that, when the user releases the mouse button, the control's function will execute.

7.2.2. Simulating a two-state button

If you give the Normal and Lit states the same appearance, and do the same for the two pressed states, you effectively create a two-state button rather than a four-state one.

8. Control Set Bitmaps

Control Set Bitmaps contain the images for each control. Most controls have four images to reflect the various states of the control. These four images are tiled in a row in the Control Set Bitmap. Each control is assigned a specific row in a specific Control Set Bitmap. So for each control you want to place on a skin, you must know its defined map color, and the Control Set Bitmap on which the images for the control reside.

All Control Sets have a *row order* that defines where the images are laid out for a given control. Controls that are used in the skin must be laid out in this row order. Controls not used in the skin are left off of the Control Set Bitmap and their row is not counted.

The row order defines the ordering of the types of controls. For Control Sets that have reproduced controls, you should place all reproduced controls in consecutive rows in order of the number of the extension in which the control resides.

See section 4 for details on extensions.

8.1. Control Sets

This section describes each of the defined Control Set Bitmaps. It is important to note what controls are defined on each Control Set and what the row order is for the controls.

The images that define the appearance of the various controls are laid out in Control Set Bitmaps. Each Control Set has specific controls that it contains and a specific layout for those controls. Some Control Sets also require an Internal Map to convey more control information. See the specific Control Set section for its layout specifics.

8.1.1. Common Image Layout for Control Sets

Most Control Sets share these common layout properties:

- Controls have four images (for the four states).
- All images for a control will be aligned in the same row of the bitmap.
- If a control is not defined on a map, you should not define it in its control set. Do not leave space for undefined controls. Instead, continue with the next row of images for the next defined control.
- Each of the four control images is separated by one pixel. Each row is separated by one pixel. This makes it easier to examine the final skin to find a misaligned control, since the bitmap background color will show when the skin is displayed.
- For Control Sets that require an Internal Map, the Internal Map will be located at the top of the file. The Internal Map conveys information specific to that bitmap using colored areas, much like the *map.bmp* bitmap. The position of colored areas in an Internal Map is not important. All Internal Maps must terminate with a horizontal gray line (color 0x777777, [119, 119, 119]) just before the control images begin. This is so the player knows where the Internal Map ends and the control images begin.

Exceptions to these common properties are noted in each Control Set section.

8.1.2. Reproducible Controls

It is possible to use the many controls more than once on a skin. Which controls can be reproduced are described in each Control Set section. When including images for a reproduced control on a Control Set Bitmap, you need to place all reproduced controls in consecutive rows in order of the extension number the control resides. Thus you may use a reproducible control only once per extension, but it may exist on any number of the extensions.

8.1.3. How Skin Bitmaps Affect Player Performance

All Control Set Bitmaps will reside in memory as 32 bits-per-pixel bitmaps after the player loads them. The bigger the Control Set Bitmaps, the more memory the skin will cause the player to use. Cropping your Control Set Bitmaps to be as small as possible will help minimize the player's memory footprint. Additionally for Control Sets that include Internal Maps, it is smart to organize the Internal Map contents so it occupies the smallest space possible, further reducing bitmap size.

The player loads map files and then discards them after scanning them, so they have no direct affect on memory used. However the larger maps will increase Body Bitmap sizes which increases memory and the time it takes to load the skin. (However skin-loading times are rarely an issue even for large skins.)

The color depth (bits per pixel) of each bitmap does not increase memory usage when the player is running, although it will increase the size of the skin file for download. These are the performance trade-offs to consider when making your skin.

8.1.4. *Body.bmp* – The Body of the Skin



The *Body.bmp* bitmaps provide the graphical background for all the controls on the main body of the skin, as well as for any extensions. Each Body Bitmap must have its corresponding Map Bitmap (that is, *body.bmp* corresponds to *map.bmp*; *body1.bmp* corresponds to *map1.bmp*, and so on).

The size and shape of each Body Bitmap must match those of the corresponding Map Bitmap perfectly. The position of the body image in *body.bmp* must match *map.bmp* exactly. For Extension Body Bitmaps the body image is always at position 0, 0.

(Also see [The Body Extension](#), Section 4.4.3.)

8.1.5. *ButtonSet.bmp* – The Playback Control Buttons

- Internal Map – NO
- Controls Reproducible – YES



- Row 1: Eject
- Row 2: Previous Track
- Row 3: Next Track
- Row 4: Stop
- Row 5: Pause
- Row 6: Play

The *ButtonSet.bmp* file defines the images for the basic playback controls. Each control must have a graphic element for each of the four states

8.1.5.1. Play/Pause and Play/Stop Combo Controls

If you create a Play/Pause combo button or Play/Stop combo button, the layout of this file remains the same, but the Play and Pause (or Play and Stop) images must be the exact same size and shape since they will occupy the same area on the map.

8.1.6. *WindowSet.bmp* – Common Controls

- Internal Map – NO
- Controls Reproducible – YES



- Row 1: Main Menu button
- Row 2: Minimize button
- Row 3: Maximize button
- Row 4: Close button
- Row 5: Help button
- Row 6: Always On Top
- Row 7: Preferences
- Row 8: QuickTrack Menu
- Row 9: Next Visualization
- Row 10: External Visualizations
- Row 11: Full screen Visualizations
- Row 12: External Video
- Row 13: Full screen Video
- Row 14: Edit Track Info
- Row 15: Plug-in menu
- Row 16: Open Music Browser
- Row 17: Open Library Window
- Row 18: Open the player website
- Row 19: Volume Up
- Row 20: Volume Down
- Row 21: Mute Volume

- Row 22: Browser Forward
- Row 23: Browser Back
- Row 24: Browser Stop
- Row 25: Browser Refresh

The *WindowSet.bmp* file defines the basic window controls (such as Close and Minimize) normally associated with a Windows application, plus a number of the general controls specific to the player.

8.1.7. *ExtBtnSet.bmp* – Extension Controls

- Internal Map – NO
- Controls Reproducible – YES (except *Close All* controls)



- Row 1: Open Extension 0
- Row 2: Close Extension 0
- Row 3: Open Extension 1
- Row 4: Close Extension 1
- ...
- Row 19: Open Extension 9
- Row 20: Close Extension 9
- Row 21: Close All Extensions 1
- Row 22: Close All Extensions 2

The *ExtBtnSet.bmp* file defines the controls for opening and closing extensions. There are separate open and close buttons for all extensions. However, unless the **extnotoggle** setting is used (see [extnotoggle](#)), the Open button will both open and close the extension.

There are also two *Close All Extensions* buttons. Each of these buttons will close all extensions that are owned by the extension that owns the Close All button. You can define the name of each of these controls to better define how they operate on your skin (see [extcloseallname](#)).

8.1.8. *TrackSet.bmp* – The Playlist

- Internal Map – YES
- Controls Reproducible – YES (controls on rows 6-18 only)

The *TrackSet.bmp* file defines the controls for the player’s *TrackBox*, an array of track buttons that list a set of tracks. The user can rearrange the track buttons to create a custom playlist. You can label each track button with the number of the track or with the text of the track title.

You can change the way *TrackSet.bmp* is laid out, depending on how you want the *TrackBox* to look and behave, and which Repeat control you decide to use.

8.1.8.1. Internal Map

The top of area of the *TrackSet.bmp* bitmap is the Internal Map. Within the Internal Map, the following color regions are defined:

0xFF00FF [255, 0, 255]	Required	Specifies the size and shape of a single track button.
0xFF99FF [255, 153, 255]	Optional	Specifies the size and shape of the text area within a playlist track region. (Note: the combined region formed by 0xFF99FF and 0xFF00FF make up the entire track region). Currently the text is only offset to the 0xFF99FF region. It is still clipped to the entire track region.
0xFF0000 [255, 0, 0]	Optional	Specifies the size of a track button <i>including</i> the space to the right and below the button. Note that shape is not a factor. The size is determined by the bounding rectangle. If you omit this region, the player tiles track buttons with no spacing within the <i>TrackBox</i> area. (See example 1.)
0x0000FF [0, 0, 255]	Required for numbered track buttons. Omit for text buttons.	Specifies the size of a digit of the track number. All digits must be the same size. The presence of this region tells the player to number the track buttons.
0x00FFFF [0, 255, 255]	Optional	Indicator that causes track buttons to follow the contours of the <i>TrackBox</i> boundaries instead of lining up vertically. The indicator’s size and shape are not important (1x1 pixel is sufficient).
0x777777 [119, 119, 119]	Required	Internal map delimiter

8.1.8.2. Examples

The examples below show *TrackSet.bmp* in two formats:

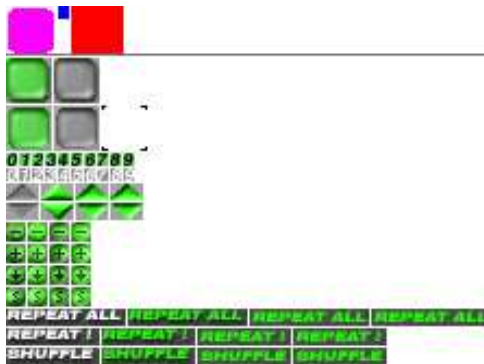
- Example 1: For a player that uses track titles to label the track buttons and the Repeat Combo control.
- Example 2: For a player that labels the track buttons with the number of the track and the Repeat Toggle controls.

Example 1: Using track titles



Internal Map
 Row 1: Normal track bitmap
 Row 2: Track Lit bitmap
 Row 3: Track Blocked bitmap
 Row 4: Track Lit and Blocked bitmap
 Row 5: Track selection overlay bitmap
 Rows 6–18 (See below.)

Example 2: Using track numbers



Internal Map
 Row 1: Normal track bitmap **and** Track Lit bitmap
 Row 2: Track Blocked **and** Lit & Blocked bitmap **and** Track selection overlay bitmap
 Row 3: Track number digits for normal track bitmap
 Row 4: Track number digits for lit track bitmap
 Row 5: Non existent
 Rows 6–18 (See below.)

Both Examples

Rows 6 to 18:

Row 6: Scroll Up button
 Row 7: Scroll Down button
 Row 8: Remove Menu button
 Row 9: Add Menu button
 Row 10: Save Playlist button
 Row 11: Sort Playlist button
 Row 12: Convert Menu button
 Row 13: Repeat 1/All Combo button
 Row 14: Repeat All button
 Row 15: Repeat Track button

Row 16:	Shuffle button
Row 17:	Auto-Prune button
Row 18:	Manual Advance button

8.1.8.3. Using Track Button Images

The 0xFF00FF (magenta) region of the *Map.bmp* file determines the size of the TrackBox in the player. The player tiles as many whole track buttons as it can fit across the width of the TrackBox and then starts a new row. It tiles as many whole rows as it can fit into the height of the TrackBox. If there are more tracks than can fit in the TrackBox area, the TrackBox Scroll Buttons can display the ones that don't fit.

The five bitmap images for the track buttons appear on either two or five lines, depending on whether you use numbers or text to label the track buttons.

- If you use numbers to label track buttons (that is, if you included dark blue pixels in the Internal Map), the bitmaps are on two lines.
- If you use the text of the track title to label track buttons, each of the five bitmaps is on its own line. Buttons labeled with text are much wider than buttons labeled with numbers, so this system saves space in the *TrackSet.bmp* file.

The first two track button bitmaps define the normal and mouse-over state of the track buttons. If the track is playing (and the mouse pointer is not over the TrackBox), the track button flashes between these two states.

The third and fourth track bitmaps define “Blocked” tracks. A user can block a track in the playlist to prevent the player from playing that track, while leaving it in the playlist. The player can also automatically block a track if the track is unplayable. These two bitmaps are for the blocked track's normal and mouse-over states.

The fifth bitmap provides a graphic that the player merges on top of a button, using a settable blend value (see [trackalphas](#)), to indicate that the track is selected.

8.1.8.4. Labeling Track Buttons with Text or Numbers

If you use track numbers to label the track buttons, the player constructs the numbers using digits centered within the button. The 0x0000FF (dark blue) region at the top of the *TrackSet.bmp* file determines the size of a digit.

To set where the text should be drawn, in *TrackSet.bmp*'s Internal Map, create a region (color 0xFF99FF) within the track region (color 0xFF00FF) that indicates where the text should be displayed. (Note: the combined region formed by the two colors mentioned make up the entire track region.) The text is only offset to the 0xFF99FF region. It is still clipped to the entire track region.

You should size the region and the digits so you can fit three digits in a track button. Track digits have a constant width. The bitmaps that define the digit characters should appear on two rows following the track button bitmaps (rows 3 and 4 in example 2, above). The two rows define the highlighted and un-highlighted versions of the digits, starting with digit 0, ending with digit 9. Separate each digit with a one-pixel gap.

If you are using the text of the track title to label the track buttons, omit the dark blue region in the Internal Map and the two rows of track digits. The player labels the button with the text of the track title, truncated if necessary to fit within the button width. The player formats the text using your specifications in the *Charset.ini* file (see [The CharSet.ini file](#)).

8.1.8.5. Creating TrackBox Controls

The remaining control images in *TrackSet.bmp* define the appearance of the other controls for the TrackBox. They all follow the standard button states and layout (see: [Control States and Behaviors](#)).

8.1.8.6. Repeat Toggle or Repeat Combo Controls

There are two ways to implement the Repeat All/Repeat Track buttons. The color of the repeat control defined on *Map.bmp* (Section 3) determines the type of repeat button. They are as follows:

- Implement two toggle buttons. One that toggles ‘Repeat All’ and the other that toggles ‘Repeat Track’. Row 14 will sport the 4 image states for the Repeat All control, and Row 15 will sport the 4 images state for the Repeat Track control.

Or

- Implement one button that cycles through the 3 repeat states (Repeat Off, Repeat All, Repeat Track), as shown in example 1 of *TrackSet.bmp*. In this case, Row 13 will contain 6 images states - 2 for each of the respectively repeat states. Only the Normal and Lit states for each pair are required.

It should be noted that both Repeat control implementation can co-exist on a skin at the same time.

8.1.9. EncoderSet.bmp – The Encoder

- Internal Map – YES
- Controls Reproducible – YES (controls on rows 6-14 only)

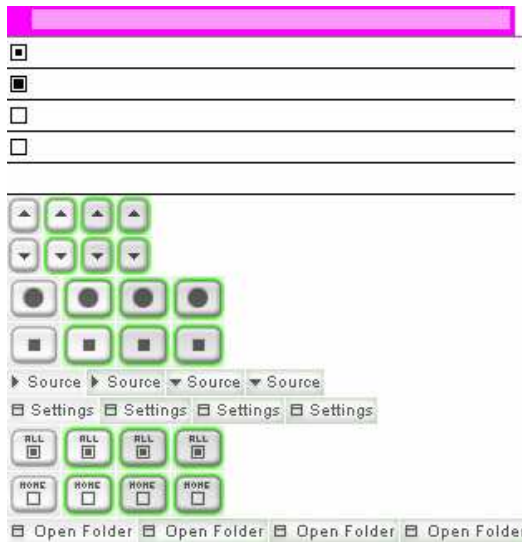
The Encoder Control Set is formatted similar to *TrackSet.bmp* with differences noted below.

8.1.9.1. Internal Map

The Internal Map is the same that in the *TrackSet.bmp* bitmap. See Section 8.1.8.1.


8.1.9.2. Example

Here is an example *EncoderSet.bmp* bitmap using track titles.

	Internal Map	See Section 8.1.8.1 for colors.
	Row 1:	Normal track bitmap
	Row 2:	Track Lit bitmap
	Row 3:	Track Deselected bitmap
	Row 4:	Track Lit and Deselected bitmap
	Row 5:	(not used, but must be present)
	Row 6:	Scroll Arrow Up
	Row 7:	Scroll Arrow Down
	Row 8:	Encode Start
	Row 9:	Encode Stop
	Row 10:	Encode Source
	Row 11:	Encode Settings
	Row 12:	Encode List Select All
	Row 13:	Encode List Select None
	Row 14:	Explore Encode Folder

8.1.10. TimerSet.bmp – The Timer Display

- Internal Map – YES
- Controls Reproducible – YES (controls on rows 2-7 only)

	Internal Map:	0xFF00FF - Timer digit size 0x777777 - Internal map delimiter
	Row 1:	Timer Digits
	Row 2:	Track time toggle
	Row 3:	Playlist time toggle
	Row 4:	Elapsed time toggle
	Row 5:	Remaining time toggle
	Row 6:	Timer play/track combo
	Row 7:	Timer elapsed/remaining combo

The *TimerSet.bmp* file defines the Playlist Timer display and controls. The size of the timer area is defined on the Map Bitmaps. The timer area should be at least as large as the placeholders for the minutes and seconds (e.g., **12:34**). If the timer area is not large enough for the time display, the player will truncate the left side of the number to the available space.

8.1.10.1. Using Bitmap Timer Digits

The top area of the *TimerSet.bmp* image is the Internal Map, consisting of an area of 0xFF00FF (magenta) pixels defining the size of one timer digit. This Internal Map is separated from the rest of *TimerSet.bmp* by a one-pixel line of color 0x777777 [119, 119, 119], followed by a one-pixel gap.

Below the Internal Map is a row defining the appearance of the timer digits. These are fixed-width digits 0 – 9, followed by a colon and a minus sign. They are not separated by a one-pixel boundary.

Tip: You can make the colon and minus sign narrower than the digits:

- The player calculates the width of the colon plus the minus sign as the width between the end of the **9** and the right side of the bitmap.
- The width of the colon and the minus sign individually are half that calculated width.
- The colon and minus are always of equal width and cannot be wider than the defined digit width.

8.1.10.2. Using Windows Fonts for Timer Digits

Instead of using bitmap digits for the timer, you can define a Windows font using *CharSet.ini* settings (see [The CharSet.ini file](#)). The existence of the TimerFont section in the *CharSet.ini* file will override the digits in *TimerSet.bmp*.

If you use this method, you can eliminate the Timer Digit size from the Internal Map and remove the bitmap digits, but the Internal Map delimiter must remain. If you want your skin to be compatible with old versions of the player, you must include the Internal Map and bitmap digits even if you are using the *CharSet.ini* method is used.

Alternatively there is also a Text Control available to display the playlist time (see [Text Controls](#)). This Text Control uses the font defined in *CharSet.ini* under the PLTimerFont section. Using this method has the benefit of being reproducible, which means you can place this Text Control on one or more extensions.

8.1.10.3. Timer Controls

Timer Controls allow for setting what the Timer displays. The Timer can be set to display the elapsed or remaining duration for the current track or all tracks in the play queue. That gives four possible states for the Timer:

- Elapsed time for current track
- Remaining time for current track
- Elapsed time for play queue
- Remaining time for play queue

There are two ways to implement the Timer Controls:

- Implement the timer controls as four two-state buttons (as shown with Rows 2-5 in the *TimerSet.bmp* example above). They all follow the standard button states and layout. It is common practice to group the Elapsed/Remaining Time controls, and the Track/Playlist Time controls together on a skin.

Or

- Implement the timer controls as two combo buttons. The track/ play queue combo is the 6th row; the elapsed/remaining control is the 7th row. For each combo control, the first two images in the row are for the first state; the second two images in the row are for the second state.

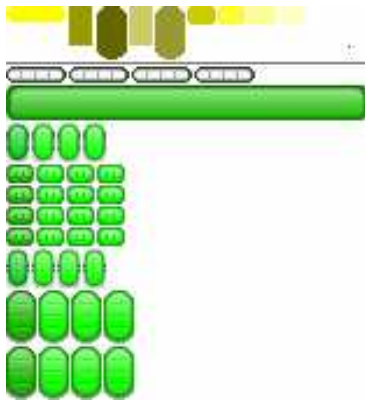
8.1.11. *SliderSet.bmp* – Sliders

- Internal Map – YES
- Controls Reproducible – NO

The *SliderSet.bmp* file defines the appearance of different types of *slider controls*. The eleven slider controls that are defined for this Control Set are listed in the Internal Map rows below. Other slider controls that are used for the Equalizer are defined on *EqSet.bmp* (see [EqSet.bmp - Equalizer Controls](#)).

At the top of the *SliderSet.bmp* is the Internal Map (colors described below). Each of the eleven slider controls has an assigned color code that you use to define the size and shape of the moving part of the control (the “slider”).

Below the Internal Map is a series of rows containing the bitmaps for the eleven slider controls. If you omit any slider control from the *Map.bmp* file, do not include any information for it on this Control Set.



Row	Control	Internal Map color code
Internal Map	(Internal Map delimiter color code:)	0x777777 [119, 119, 119]
Row 1:	Playback Progress Slider	0xFFFFF0 [255, 255, 0]
Row 2:	Playback Progress Slider (2 nd)	0xCCCC33 [204, 204, 51]
Row 3:	Encode Progress Slider	0x999966 [153, 153, 102]
Row 4:	Master Volume Slider	0x999900 [153, 153, 0]
Row 5:	Soundcard Volume	0xCCCC00 [204, 204, 0]
Row 6:	Bass Control	0xFFFF33 [255, 255, 51]
Row 7:	Treble Control	0xFFFF99 [255, 255, 153]
Row 8:	Balance Control	0xFFFFCC [255, 255, 204]
Row 9:	Visual Effects Level	0xCCCC66 [204, 204, 102]
Row 10:	Playlist Scrollbar Slider	0x666600 [102,102,0]
Row 11:	Encode List Scrollbar Slider	0x999933 [153, 153, 51]

8.1.11.1. Internal Map

For consistency, the assigned color code in the *SliderSet.bmp* Internal Map is the same color code as that used in *Map.bmp* (Section 3) for the same control. However, for convenience, the color assignments are shown above next to the corresponding row assignments for each control.

8.1.11.2. Slider Types

You can use any combination of these slider types to implement any of the slider controls.

- **Tracking Slider:** displays a *Slider* and a *Track* in which the slider moves. These can be horizontal, vertical, or diagonal (lower left to upper right).

- **Progressive Slider:** also called a “thermometer” slider, displays a graphic that fills in the Track as it progresses. These can be horizontal or vertical.
- **Rotational Slider:** displays a series of frames to represent an animation. These can simulate rotational knobs or other creative sliders.

8.1.11.3. Creating Tracking Sliders

A tracking slider consists of two parts: a movable *Slider* and a *Track* in which the slider appears to move.

To implement:

1. In the *Map.bmp* file, define the track size by specifying a region in which to draw the Track image.

Use pixels of the appropriate color code for the control you are defining. (For example, for the Bass control, use 0xFFFF33.)
2. In the *Body.bmp* file, include the Track image at the same position and size defined on *map.bmp*.
3. In the Internal Map area of the *SliderSet.bmp* file, define the Slider size using the appropriate color code for the control you are defining.

The dimension of the Slider relative to the Track size determines if the Slider moves horizontally, vertically or diagonally. The Slider will move from the lower left area of the track to the upper right area of the track.

- If the Slider is the same height as the Track but is shorter in width, the Slider will move horizontally.
 - If the Slider is the same width as the Track but a shorter height, the Slider will move vertically.
 - If the Slider is both smaller in width and height, the Slider will move diagonally.
 - If the Slider is larger than the Track in height or width, than the Slider as a *Rotational Slider*.
4. In the appropriate row of the *SliderSet.bmp* file, add the image for the Slider. Make the each Slider image the same pixel dimensions as the Slider size defined in the Internal Map.
 5. By default there is only one image for the handle bitmap. To create a Slider that has uses the four interaction states, add the color red (0xFF0000) to the internal map section. This color is a flag, so size and shape are not a factor (1x1 pixel will suffice). With this flag present, all tracking Sliders must implement the four image states (see [Control States and Behaviors](#)).

8.1.11.4. Creating Progressive Sliders

A Progressive (Thermometer) Slider is a single bitmap that fills in the Track area.

To implement:

1. In the *Map.bmp* file, define the region for the progressive slider.

Use pixels of the appropriate shade of yellow for the slider control you are defining. For example, for the Playback Progress control, use yellow 0xFFFF00.
2. In the Internal Map area of the *SliderSet.bmp* file, do not include any pixels in the color corresponding to the control you are defining.

For example, if you are defining the Playback Progress control as a progressive slider, omit any pixels of color 0xFFFF00 in the Internal Map. This is how the player knows to implement this as a Progressive Slider rather than a Tracking Slider.

Note: If you are using only Progressive Sliders, so that the Internal Map is empty, you still must draw the one-pixel gray line at the top of the file indicating the end of the Internal Map.

3. In the appropriate row of the *SliderSet.bmp* file, add the image that you want to grow into the designated region on *Map.bmp*.

The matching *Body.bmp* area defines the background of the Progressive Slider.

Progressive Sliders draw by revealing the Slider bitmap within the designated area of *Map.bmp*. When the control is in the zero position, the Slider bitmap is hidden. As the control progresses, or as the user drags within the designated area, more of the bitmap is revealed, from left to right and from bottom to top. In the maximum position, the slider is completely displayed within the designated area.

8.1.11.5. Creating Rotational Sliders

A Rotational Slider uses a series of bitmaps representing frames that show the control at various stages, from zero position to maximum position.

You can use Rotational Sliders to create special effects, such as a simulating a Tracking Slider that changes color. The user sees each frame as a *detent*. For smoother animation, you need more frames. But this will also make your Control Set Bitmap larger.

Note: For controls where a center position is meaningful, such as a Balance knob, create an odd number of frames so that there is a center detent (the middle frame will be the center detent).

To implement:

1. In the *Map.bmp* file, define the Rotational Slider size by specifying a region in which to draw the Rotational slider image.

Use pixels of the appropriate color for the control you are defining. For example, for the Master Volume control, use dark yellow 0x999900.

2. In the Internal Map area of the *SliderSet.bmp* file, define the width of the collection of frames by drawing a one-pixel-wide line using the appropriate reserved color for the control you are defining.

Make the frame set wider than the designated control region in the *Map.bmp* file to define this as a Rotational slider. If you have several Rotational sliders, draw a different one-pixel-wide line for each (in its appropriate color) because each knob may have different sizes, a different number of frames, or both.

3. In the appropriate row of the *SliderSet.bmp* file, draw the individual frames of the Rotational slider, from its zero state at left to its maximum state at right.

Do not separate the frames using a one-pixel gap. (This makes it easier to determine the total width of the frames for drawing the line in the Internal Map.)

Each frame should have exactly the same pixel dimensions and shape as the designated region in the *Map.bmp* file.

Rotational Sliders draw by displaying one of the frames within the designated region. When the control is in the zero position, the first frame is displayed. When the control is in the maximum position, the last frame is displayed. All other frames are drawn appropriately for the intervening states.

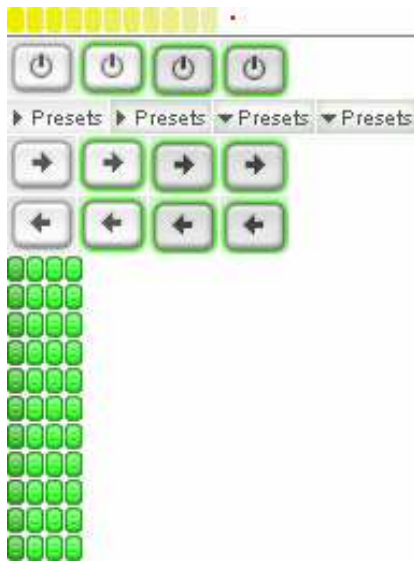
8.1.12. *EqSet.bmp* – Equalizer Controls

- Internal Map – YES
- Controls Reproducible – YES (controls on rows 1-5 only)

The *EqSet.bmp* file defines the equalizer controls. The equalizer has four buttons (On/Off, Presets, and two buttons for moving between presets), and ten sliders for the ten bands of the equalizer.

Use the same rules for the ten equalizer-band sliders in *EqSet.bmp* as for *SliderSet.bmp* (see [SliderSet.bmp – Sliders](#)). The color code in the *EqSet.bmp* Internal Map is the same as in *Map.bmp* for the same control (see the yellows in Section **Error! Reference source not found.**). As always, separate this Internal Map from the body of the *EqSet.bmp* file using a one-pixel gray (0x777777) line followed by a one-pixel gap.

The first four rows after the Internal Map are for the buttons, followed by rows for the ten sliders. Each slider can use a different slider type if desired. All buttons use the standard button states and layout (see [Button Appearance: The four button states](#)).



Internal Map (For colors, see section **Error! Reference source not found.**)

- Row 1: Equalizer Enable (On / Off)
- Row 2: Equalizer Presets Menu
- Row 3: Next Equalizer Preset
- Row 4: Previous Equalizer Preset
- Row 5: Mixer Channel button
- Row 6 - 15: Ten Equalizer bands (Sliders)
- Row 16: Pre-amp slider

8.1.13. *SkinCtrlSet.bmp* – Skin Control Buttons

- Internal Map – NO
- Controls Reproducible – YES












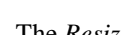
- Row 1: Skin Mode/Layout Menu
- Row 2: Next Skin Mode (or switch to custom mode)
- Row 3: Switch to skin mode 1
- Row 4: Switch to skin mode 2
- Row 5: Switch to skin mode 3
- Row 6: Switch to skin mode 4
- Row 7: Switch to skin mode 5
- Row 8: Switch to skin mode 6
- Row 9: Switch to skin mode 7
- Row 10: Switch to skin mode 8
- Row 11: Switch to skin mode 9

The *SkinCtrlSet.bmp* file defines the buttons that switch between skin modes. You can create a skin *family* consisting of up to nine skin *kids*, which the user experiences as different modes of the same skin (see [Packaging Skins](#)).

By default the Next Skin Mode control will change the skin to the next valid mode (in numbered order). You can override this behavior using the **nextmode** setting (see [Skin Settings: nextmode](#)).

8.1.14. *ResizeSet.bmp* – Resize Control Buttons

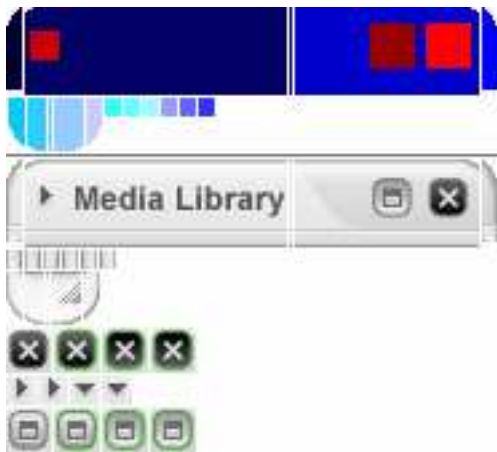
- Internal Map – NO
- Controls Reproducible – YES

	Row 1:	Resize control for break 0
	Row 2:	Resize control for break 1
	Row 3:	Resize control for break 2
	Row 4:	Resize control for break 3
	Row 5:	Resize control for break 4
	Row 6:	Resize control for break 5
	Row 7:	Resize control for break 6
	Row 8:	Resize control for break 7
	Row 9:	Resize control for break 8
	Row 10:	Resize control for break 9

The *ResizeSet.bmp* file defines the images for the controls that manipulate the resizing of the owner extension. Each resize control resizes its owner extension as defined by its corresponding break defined on the Break Maps (see [Using Break Maps](#)).

8.2. Other Skin Bitmaps

8.2.1. Window Borders



Top left corner	0x000033
Top left	0x000066
Top joiner	0x000099
Top right	0x0000CC
Top right corner	0x0000FF
Left top	0x33FFFF
Left joiner	0x66FFFF
Left bottom	0x99FFFF
Bottom right corner	0xCCCCFF
Bottom right	0x99CCFF
Bottom joiner	0x66CCFF
Bottom left	0x33CCFF
Bottom left corner	0x00CCFF
Right bottom	0x3333FF
Right joiner	0x6666FF
Right top	0x9999FF

Buttons

Close	0xFF0000
Menu	0xCC0000
Web - Forward	0xFF3300
Web - Back	0xFF3333
Web - Stop	0xFF3366
Web - Refresh	0xFF3399
Vis - Next	0xCC3300
Vis - List	0xCC6600

Vis - Fullscreen	0xCC6633
Video – Normal size	0xFF6600
Video – Double size	0xFF6633
Video – Fullscreen	0xFF6666

There are four skinned window borders utilized by the player:

- Music Browser A small web browser window with content provided by the Gracenote service.
- Skin preview window A small web browser with previews of available skins.
- External visualization area A visualization window detached from the player.
- External video window A window available to plug-ins to display video or other animations.

The border bitmap for the Music Browser and the Skin preview window is defined in *BorderWeb.bmp*. The External visualization area border bitmap is defined in *BorderVis.bmp*. The External video window border bitmap is defined in *BorderVideo.bmp*.

If one or more of the above border bitmap files are not included in the skin, the player will default to use *BordetSet.bmp* if available.

8.2.1.1. Border Bitmap Layout

The upper area is the Internal Map, which uses color codes to define the size of the bitmaps in the lower area. The Internal Map is separated from the bitmaps using a one-pixel gray line (color 0x777777, [119, 119, 119]).

The border is separated into 16 segments:

- Four corners
- Three segments each for the four sides.
 - Two of the three segments attach to the corners and do not scale.
 - The middle segment (called the *joiner*).

The player tiles the joiner bitmap either horizontally or vertically (depending on the edge) to fill the width or height of the window. Design the joiner bitmap to connect to the other two segments, and design the other two segments to connect to the corners.

The order and layout of the colored regions in the Internal Map is not important. They do not have to be top-aligned. (The player tells them apart by the color coding.) You can rearrange the regions to occupy less space and make the size of the border bitmap smaller.

The order and layout of the bitmaps in the lower bitmap area *is* important. This order is as follows:

- The bitmaps must be on three rows:
 - Row 1: The top border (left to right, including top corners)
 - Row 2: The left side (bottom, joiner, top), and the right side (top, joiner, bottom)
 - Row 3: The bottom border (left to right, including the bottom corners)
- The top of each bitmap in the lower area must align to the top of its row
- Within a row, each bitmap must be separated from other bitmaps by a one-pixel line.

8.2.1.2. Window Buttons

There are a number of four-state buttons defined for the skinned border windows. You can position any button within any border section in the Internal Map except for the joiner sections. Each button region must be enclosed entirely within one of the border sections.

If you include any Window button, the graphic elements for the buttons are aligned in rows 4 and above (see example in *BorderSet.bmp* file above). The rows of graphic elements for each button are ordered as displayed in the example above.

8.2.2. *MenuSet.bmp* – Menu Banners



The *MenuSet.bmp* file defines the banner image used as the title bar on the side of two player menus:

- The QuickTrack menu banner
- The Main Menu banner

The width of each banner is fixed at 23 pixels. The banners can be any reasonable height, but both must be the same height. If the menu becomes longer than the banner bitmap, the bottom 20 pixels will be tiled to fill the menu banner area. Thus it is important to make the bottom 20 pixels “tileable.”

8.3. Non-Skinned Controls

There are some controls available that can be defined on the Map Bitmaps but do not require any images on any Control Set Bitmaps. They either use the existing Body Bitmap image for their placement, or they are filled by other images generated dynamically.

8.3.1. Title Bar Control

The Title Bar, like in Windows, will Maximize and Minimize the player when double-clicked.

The Title Bar Control only needs to be defined on the Map Bitmap to indicate the location for the ‘Title Bar’ on the skin. It will use the Body Bitmap image respective to its location.

The Title Bar Control is Reproducible. The Map Bitmap color for the Title Bar Control is: 0x9900CC [153, 0, 204]

8.3.2. Artwork Control

The Artwork Control will display artwork for the currently playing media, if any is available.

The Artwork Control needs to be defined on the Map Bitmap to indicate the location for the control. It will use the Body Bitmap image respective to its location when no artwork is available.

The Artwork Control is Reproducible. The Map Bitmap color for the Artwork Control is 0x9900FF [153, 0, 255]

9. Text Controls

Text controls display textual information on the skin. Text Controls are also Non-Skinned Controls as they don't require Control Set images. As well they have several properties unique to themselves.

All text controls are reproducible (each control can exist on more than one extension).

All text controls are resizable horizontally, but not vertically (see [Using Break Maps](#)).

9.1. Text Control Types

9.1.1. Current Media Info

- Track Title
- Disc (Album) Title
- Artist Name

Extra Functionality: Clicking any of these controls will toggle auto-scrolling. Right-clicking any of these controls will display QuickTrack. Double-clicking any of these controls will launch the track info editor.

9.1.2. Current Audio Info

- Input Format (e.g. 'MP3')
- Channel Mode (e.g. 'Stereo')
- Samplerate (e.g. '44khz')
- Bitrate (e.g. '128kbps')

9.1.3. Encoding Info

- Encoding Rate (e.g. '10x')
- Encoding Elapsed Time (e.g. '1:23')

9.1.4. Miscellaneous

- System Messages (ToolTips and messages from player and plug-ins)
- System Messages Alternate (custom messages from plug-ins only)
- Music Browser Status (messages from navigating in the Music Browser)
- Volume Level Indicator
- Campaign Text (Browse current artist)
- Playlist Timer (alternative to *TimerSet.bmp*)

Extra Functionality: Clicking the Campaign Text control will launch the music browser. Clicking the Playlist Timer control will toggle time elapsed/time remaining. Right-clicking the Playlist Timer Control will display the Timer Menu.

You do not have to display all text controls. Of the Media Info controls, the most useful is the track title, followed by the disc title. If you do not have an artist name control, the player adds the artist to the beginning of the disc text. If no disc text control is present, the artist text is added to the beginning of the track text (separated by a delimiter).

Within each text control, the player centers the text vertically within the text area and left-justifies the text horizontally by default. If the text is too long to display in the area you designate, the player scrolls the text from right to left across the area. The user can also drag the text to scroll it left or right.

The player reserves some space at either end of the text control area to fade the text as it moves off the display. By default this fade area is about as wide as half the height of the control. You can override the fade area width using the [fadewidthleft](#) and [fadewidthright](#) settings.

10. Fonts

Fonts displayed in the skins are either Window's fonts (built in fonts such as True Type Fonts) or custom bitmap fonts.

10.1. Windows Fonts

Window's fonts are defined in *CharSet.ini* which sets the various properties for the fonts for each text area used on a skin. Using these fonts can make implementation easier, and are essential to support non-Latin languages. However there is the potential for these fonts to appear slightly different, or even be unavailable, on different systems depending on the font chosen.

10.2. Bitmap Fonts

Bitmap fonts are more customizable and guaranteed to be the exact across all systems, but do not support non-Latin languages.

The two default bitmap font sets are named *CharSet.bmp* and *SmallCharSet.bmp*. If *CharSet.ini* is not used, or a font section is omitted from it, the player will look for one of these bitmap font sets. Which one it defaults to depends on the text control being set. The Disc Title, Track Title, and Artist Name text controls will default to *CharSet.bmp*. All other text controls will default to *SmallCharSet.bmp*, or if that bitmap font is not available, will default to *CharSet.bmp*.

10.2.1. Creating a Bitmap Font



On a bitmap font you create each character individually with near-complete freedom of font design. (One of the only restrictions is that characters can't overlap.)

The top row of pixels in the bitmap is called the *boundary row*. This is a row of black and white pixels where each black pixel indicates the start of a new character below. The width of the character extends up to the start of the next character. Two black pixels together indicate a skipped character (a character of zero width).

The area on the bitmap below the boundary row is where each character is represented. The height of the bitmap (minus the boundary row) determines the general height of the font. The characters must be ordered on the bitmap from ASCII value 32 (space) to 255. You can skip characters only by creating a zero-width character. You can stop the character set short if none of the higher character values are required.

10.2.2. Specifying Color for Text

Each character in the bitmap font is a *luminance map* of the character, not the actual color of the character. Each character must be drawn in white, surrounded by black. The more white the pixel, the more opaque it will show on the skin. Absolute white (0xFFFFFFFF) displays a solid character. Absolute black (0x000000) is completely transparent. Values between absolute white and black result in a corresponding blend with the background.

You can set the color for each bitmap font in two ways.

- For skins that choose not to use *CharSet.ini* the color used when displaying the font is determined by the color of the pixel located at 0x0 in the bitmap (the first pixel in the boundary row). This pixel will not affect the width of the character below.
- For skins using *CharSet.ini* the color can be set for each text control individually. Color settings in *CharSet.ini* override the color indicated by the 0x0 pixel on the bitmap font.

10.2.3. Specifying Text Shadows

Shadows give text displays more depth by creating a shadow effect beneath the text. To enable text shadows, specify **ShadowLum=value** in the appropriate *CharSet.ini* section. The **ShadowLum=value** defines the brightness of the shadow. The **ShadowX=** and **ShadowY=** key/values define the length of the shadow up, down, left, and right of the text.

The color of the shadow is the same as the font color (this limitation may change in future versions).

Specifying shadows for the Search control is ignored.

10.2.4. Specifying Text Alignment

By default, most text controls align the text to the left side of the control. Exceptions are the Timer control, which aligns to the right and the Status Message and Volume Level controls that are centered by default. You can change these default alignments by specifying the **Align=value** setting in *CharSet.ini*. Valid values for this key are: **left**, **center**, **right**.

10.3. The *CharSet.ini* file

The *CharSet.ini* file is used to specify fonts to be used for each text control. It can specify settings for Windows Font and Bitmap Fonts.

The general format of *CharSet.ini* is as follows:

```
[<textcontrol font name>]
```

```
setting1=value1
```

```
setting2=value2
```

```
...
```

```
[<textcontrol font name>]
```

```
setting1=value1
```

```
setting2=value2
```

```
...
```

The [*<control font name>*] indicates the start of a 'section' that organizes settings for the font indicated by the section name. The available sections are as follows:

Section Name:	Sets Font for Control:
DiscFont	Disc (Album) Title
TrackFont	Track Title
ArtistFont	Artist Name
StatusFont	Status Message
StatusAltFont	Status Message (Alternate)
StatusBrowserFont	Music Browser Status Messages
CampaignFont	Campaign Text
SearchFont	Playlist Search Control
TimerFont	Main Timer Display (using <i>TimerSet.bmp</i>)
PLTimerFont	Playlist Timer (as text control)
AudioInfoFont	Audio Info text controls
EncInfoFont	Encoding Info text controls
VolLevelFont	Volume Level text control
TrackButtonFont	Playlist track font – default state
TrackCurrentFont	Playlist track font – highlight state
TrackSelectedFont	Playlist track font – selected state
TrackPressedFont	Playlist track font – pressed state
TrackCurrentSelFont	Playlist track font – highlighted and selected
TrackPressedSelFont	Playlist track font – pressed and selected
EncButtonFont	Encoder track font – default state
EncCurrentFont	Encoder track font – highlight state
EncSelectedFont	Encoder track font – selected state
EncPressedFont	Encoder track font – pressed state
EncCurrentFont	Encoder track font – highlighted and selected
EncPressedSelFont	Encoder track font – pressed and selected

Each section has a header in brackets (e.g., [**DiscFont**]), followed by key/value combinations describing the font for that section.

Since Text Controls are reproducible, each instance of a Text Control can also have a separate section for font settings. The section name for individual instances of each text control is `<sectionnameN>` where *sectionname* is the appropriate section from the list above, and *N* is the number of the extension on which the text control resides. Any text control that does not have an individual section will default to settings from the unnumbered section names.

Additional notes:

- **SearchFont** describes the font used in the Playlist Search control. If the search control exists on the skin, this section must be included since only Window's fonts will be used for the search control.
- **TimerFont** describes the font used for the Timer control. Without the TimerFont section, the timer control uses its bitmap skinning method (see [TimerSet.bmp – The Timer Display](#)).
- **TrackButtonFont** (up to 6 sections) describes the font used on the track buttons in the Playlist and Encode list controls (see [TrackSet.bmp – The Playlist](#)). To specify a different font for any of the different states a track button can have (Current, Selected, Pressed, Current+Selected, Blocked+Selected) include the appropriate sections from the following: **TrackCurrentFont**, **TrackSelectedFont**, **TrackPressedFont**, **TrackCurrentSelFont**, **TrackPressedSelFont**. Any section missing will default to the **TrackButtonFont** settings. (**Note:** the **TrackPressedFont** and **TrackPressedSelFont** remain from older player versions. They refer respectively to the Blocked and Blocked+Selected states).
- The notes above for **TrackButtonFont** also apply to **EncButtonFont**.

10.3.1. Valid Font Settings

This is a list of all valid key/value settings for each font section (exceptions noted within). All key values are for showing examples of valid settings only.

Height =10	Height of font
Width =8	Width font (0 for default width)
Weight =400	Weight of the font (0 – 1000)
Italic =0	Italic font if set to 1
Underline =0	Underline font if set to 1
Strikeout =0	Strikeout font if set to 1
Escapement =0	Specifies the angle, in tenths of degrees, between the escapement vector and the x-axis
Orientation =0	Specifies the angle, in tenths of degrees, between each character's base line and the x-axis
FaceName =Arial	String that specifies the typeface name of the font (e.g.: 'Arial', or 'Courier')
Color =255, 255, 255	Text color. Red, Green, Blue triple (valid values are from 0 – 255)
FontFile =filename	Filename of font file (e.g.: <i>arial.ttf</i>) included with skin to be used for this font
Bitmap =filename	Filename of bitmap font included with skin
BgColor =255, 255, 255	Background color. Red, Green, Blue triple (valid values are from 0 – 255) (valid for SearchFont section only)
Align =left	Alignment of text in control (valid values are: left, center, right) (Not valid for SearchFont section)
FadeWidthLeft =10	Width in pixels of fadeout at left edge of text
FadeWidthRight =10	Width in pixels of fadeout at right edge of text
Alpha =255	Opacity of font (0 for transparent, 255 for opaque). For bitmap fonts, this setting is applied in addition to the luminosity values of the bitmap font.
ShadowLum =128	Luminosity of shadow beneath text (Not valid for SearchFont section.)
ShadowX =2	Horizontal offset of text shadow (Not valid for SearchFont section.)
ShadowY =2	Vertical offset of text shadow (Not valid for SearchFont section.)

10.3.2. Including Windows Fonts with your Skin

If your skin uses a Windows font file that may not be commonly found on all systems, you can include it with the skin and have the player load it automatically. Use the **FontFile**= setting to reference the included font file.

10.3.3. Example: *Charset.ini*

The following demonstrates example *CharSet.ini* sections and formatting. This is not a complete *CharSet.ini* file. Comments are preceded with a semicolon (;).

```
; Describe a font for each section used on skin
;
```



```
; You can just supply the individual
; keys that matter (non zero) BTW: color is R,G,B
;

; this one is using the Window's 'Tahoma' Font
; and sets a shadow for the text

[TrackFont]
Height=17
Weight=900
FaceName=Tahoma
Color=0,0,255
ShadowLum=128
ShadowX=2
ShadowY=2

; this one is using the custom font bitmaps in 'mycharset.bmp'
; and center aligns it

[ArtistFont]
Bitmap=mycharset.bmp
Align=center
Color=0,0,255

; this one is using the Window's 'CoolFont' Font
; and the font file is included to be used incase
; it doesn't already exist on the system

[StatusFont]
Height=17
Weight=900
FaceName=CoolFont
FontFile=CoolFont.ttf
Color=0,0,255
```

11. Skinnable Visualizations

Skinnable visualizations are controls that show a graphical animation of the music. These visualizations are less complex than the default visualizations provided through plug-ins, but can be made to integrate seamlessly with your skin.

Up to four skinnable visualization controls can be used on a skin. Each of the visualizations can choose one of four music visualization methods: Peak Meters, VU Meters, Resonators, or Waveform.

11.1. Skinnable Visualization Settings

Implementing skinnable visualizations requires setting the following settings. 'N' is the number of the visualization (1 – 4).

visNmode=	For Waveform visuals, set visNmode= MonoTrigger, leftTrigger, or rightTrigger For VU Meter visuals, set visNmode= MonoVUMeter, leftVUMeter, or rightVUMeter For Peak Meter visuals, set visNmode= MonoPPMeter, leftPPMeter, or rightPPMeter Example: vis1Mode=MonoTrigger
	For a Resonator visual set visNmode= monoResonators, # or leftResonators, # or rightResonators, # Where # is the number of resonators (1..120). Example: vis1mode=monoResonators,80
visNbmp=	Bitmap filename to use for this visualizations (overrides defaults of <i>visN.bmp</i>) Example: vis1bmp=myvis.bmp
visNalpha=	Alpha transparency level for Skinnable Visual. (0..255) Example: vis1alpha=150
visNdrawdir=	Direction Skinnable Visual animates. For PP and VU visuals, use one of the following: left, right, up, down Example: vis1drawdir=up
	For Resonator visuals, use combinations of: up/down, left/right Example: vis1drawdir=up, right
visNcolor=	Waveform color of Skinnable Visualization. Use R,G,B values. Used only for Waveform Visuals. Example: vis1color=51,255,17

11.2. Skinnable Visualization Bitmaps

Each visual uses the background on the Body Bitmap when it is disabled. The bitmap used during animation is predefined as: *vis1.bmp*, *vis2.bmp*, and *vis3.bmp* for each visual respectively. You can override the bitmap file used with the '**visNbmp=**' setting defined above.

11.2.1. Bitmap Format for VU and Peak Meter Visuals

A single image the exact same dimensions as the visual region defined on the map, where a percentage of the bitmap is drawn for the visualization

OR

A number of images the exact same dimensions as the visual region defined on the map, where each image acts as a frame of the visual animation.

Example: If *VisI* is an area on the skin 50x80, and you want 10 frames of animation. Create a bitmap 500x80 with all ten frames next to each other.

11.2.2. Bitmap Format for Resonators Visuals

A single image the same dimensions as the visual region defined on the map. The bitmap is divided by the number of resonators set by *visNmode* to determine the width of each resonator. Each resonator uses its section of the bitmap determined by the resonator index and the width. This allows each resonator to be a different design or exactly the same depending on how you draw the bitmap.

Optionally you can choose to display peak level indicators for each resonator. To implement peak level indicators, increase the height of the resonator bitmap by the desired height of the peak level indicators. The top area of the bitmap defines the images for the peak level indicators.

11.2.3. Bitmap Format For Waveform Visuals

There is no bitmap for a Waveform visual. Set the color of the Waveform using *visNcolor* setting defined above.

12. Skin Settings

The skin setting file is used to define attributes of your skin that differ from defaults.

You can use the skin settings file to define names for each skin mode and each skin extension within a mode. You can also define intra-mode dependencies, extra extension behaviors, window backgrounds, changes to how the skin is laid out, and more.

Older versions of the player made a larger distinction between single-mode skins (skin ‘kids’) and multi-mode skins (skin ‘families’). This distinction has almost completely faded in the latest version but for the sake of reverse compatibility the terminology and idiosyncrasies remain.

For single-mode skins (*kid* skins) the settings file is named *SkinKid.ini*. For multi-mode skins (*family* skins) the settings file is named *SkinFamily.ini*.

12.1. *SkinKid.ini* Sections

Kid skins have only one mode thus *SkinKid.ini* has only one section [**settings**] where all settings appear below.

12.2. *SkinFamily.ini* Sections

A Family skin settings file has a common section called [**Family**] and a section for each mode called [**mode#**], where # is the mode number (1-9) pertaining to that section.

The settings are described in the next section.

12.3. Available Settings

12.3.1. Settings By Section

[family]

[name=](#)
[notes=](#)

[settings] (for kid skin)

[mode#] (for family skin)

[name=](#)
[file=](#)
[path=](#)
[chars=](#)
[common bitmap files](#)
[nextmode=](#)
[extnameN=](#)
[resizenameN=](#)
[extcloseall1name=](#)
[extcloseall2name=](#)
[extnotoggle=](#)
[extnofade=](#)
[extdrag=](#)
[extdragdocked=](#)
[extnohomedock=](#)
[extdefault=](#)
[extalwaysontop=](#)
[extontop=](#)
[extforced=](#)
[extforcedapply=](#)

12.3.2. Settings By Functionality

12.3.2.1. Naming

[name=](#)
[notes=](#)
[extnameN=](#)
[resizenameN=](#)
[extcloseall1name=](#)
[extcloseall2name=](#)

12.3.2.2. Skin Packaging

[file=](#)
[path=](#)
[chars=](#)
[common bitmap files](#)

12.3.2.3. Skin Elements

[trackalphas=](#)
[trackalphasenc=](#)
[visbgcolor=](#)
[visbmpalign=](#)
[visNmode=](#)
[visNbmp=](#)
[visNalpha=](#)
[visNdrawdir=](#)
[visNcolor=](#)

[extNclose=](#)
[extNopen=](#)
[bodyowner=](#)
[extNowner=](#)
[bodyafter=](#)
[extNafter=](#)
[bodydefaultsize=](#)
[extNdefaultsize=](#)
[bodylimitsize=](#)
[extNlimitsize=](#)
[bodyresizedep=](#)
[extNresizedep=](#)
[trackalphas=](#)
[trackalphasenc=](#)
[visbgcolor=](#)
[visbmpalign=](#)
[visNmode=](#)
[visNbmp=](#)
[visNalpha=](#)
[visNdrawdir=](#)
[visNcolor=](#)
[extButtonBrowser=](#)
[extButtonVideo=](#)
[extButtonLibrary=](#)
[extButtonPlaylist=](#)
[extButtonEncoder=](#)
[extButtonVisuals=](#)

12.3.2.4. Extension Behavior

[extnotoggle=](#)
[extnofade=](#)
[extdrag=](#)
[extdragdocked=](#)
[extdefault=](#)
[bodydefaultsize=](#)
[extNdefaultsize=](#)
[bodylimitsize=](#)
[extNlimitsize=](#)

12.3.2.5. Extension Z-Ordering

[extalwaysontop=](#)
[extontop=](#)
[bodyafter=](#)
[extNafter=](#)

12.3.2.6. Extension Dependants

[extforced=](#)
[extforcedapply=](#)
[extNclose=](#)
[extNopen=](#)
[bodyowner=](#)
[extNowner=](#)
[bodyresizedep=](#)
[extNresizedep=](#)

12.3.2.7. Extension Integration

[extButtonBrowser=](#)
[extButtonVideo=](#)
[extButtonLibrary=](#)
[extButtonPlaylist=](#)
[extButtonEncoder=](#)
[extButtonVisuals=](#)

12.4. Setting Descriptions

name= For kid skins, this is the name of the skin.
 For family skins, under the [**family**] section, this is the name of the skin.
 For [**mode#**] sections, this is the name of the skin mode.

notes= Includes a description of the skin, displayed in the player’s Skin Browser. The entire description must be on one continuous line. Use ‘\n’ to display line breaks in the text.

file= For Family skins that compose of multiple kid skins, this setting defines which kid skin file is the skin for this mode (see [Packaging Skins](#)).

path= Normally for Family skins without multiple intact kid skins (all the bitmaps

for a mode are in folders in the zipped archive), this setting defines which folder contains the bitmaps for this mode (see [Packaging Skins](#)).

chars=

Normally for Family skins without multiple intact kid skins, this setting defines which *CharSet.ini* file contains the text definitions for this mode (see [Packaging Skins](#)).

common bitmap files

If a bitmap exists in a skin that is common to two or more modes, instead of having multiple copies of the bitmap, you can define each mode to use the same instance of the one bitmap. Each setting defines the path and filename of the bitmap to use. Here are the keys for each skin bitmap:

```

extbtnset=      windowset=      trackset=
skinctrlset=   buttonset=        encoderset=

sliderset=     timerset=          menuset=
eqset=         resizeset=

borderset=     borderweb=
bordervis=     bordervideo=      borderlibrary=
viswndbg=     vis1..4bmp=
body=          body0..9=
map=           map0..9=
mapbreaksH=   mapBreaksV=
    
```

nextmode=

This setting defines which mode the Next Mode skin control button will switch to when pressed (see [SkinCtrlSet.bmp – Skin Control Buttons](#))

extnameN=

Sets the name of Extension *N* for current mode section. This will appear when the cursor is over the button for the extension (see [Understanding Extensions](#))

resizenameN=

Sets the name of resize control number *N* of this mode. (see [Resize Controls](#))

extcloseall1name=
extcloseall2name=

Sets the name of the *Close All Extensions* button. (see [ExtBtnSet.bmp – Extension Controls](#))

extnotoggle=

By default an extension button will open and close extensions. This setting defines extensions that will open but not close when clicking the extension button. Such an extension can only be brought to the top of other extensions.

Example: extnotoggle=2,4

Sets Extensions 2 and 4 cannot be toggled closed.

exnofade=

By default fixed extensions can fade-in when opened if the user has the option set. This setting defined extensions that must not fade-in when opened.

Example: exnofade=1,2

Set Extensions 1 and 2 to not fade-in on open

extdrag=

Set any extension to be moveable. Extensions not included will be set as Fixed. (see [Moveable and Fixed Extensions](#))

Example: extdrag=1,2,3

	Sets Extensions 1, 2 and 3 to be moveable. Extension 0 and 4-9 will be fixed in this case.
extdragdocked=	By default windows docked to the Body Extension will drag when the Body drags. This setting sets other extensions to have the same feature. Example: extdragdocked=1, 2, 3 Sets Extensions 1,2 and 3 will drag any extensions docked to them
extnohomedock=	By default all moveable extensions will dock to their original position in the skin. This setting will override that behavior. Example: extnohomedock=0, 1 Sets Extensions 0 and 1 to not dock to their original positions
extdefault=	Sets extensions that are open the first time the skin is loaded. Example: extdefault=1, 4, 0 Set Extensions 1, 4, and 0 to be open on first use
extalwaysontop=	Sets which moveable extensions will remain on top of all windows (player and other applications). Example: extalwaysontop=1, 2, 3
extontop=	Set which extensions will remain on top of other player extensions. Example: extontop=1, 2, 3
extforced=	By default, all extensions initially display in the last state used when the skin was in the same mode. Use this setting to define whether an extension should always be open, always be closed, or should remain in the same state as the previous mode when this mode is displayed. Specify the extension number, followed by one of the following: <ul style="list-style-type: none"> + Force the extension open. - Force the extension closed. = Force the extension to remain the same. Example: extforced=0=, 1-, 2+ On mode change to current mode section, sets Extension 0 to remain in its current state, Extension 1 to be closed, and Extension 2 to be open. Tip: For Family skins with similar-looking modes, set each common extension to remain the same so that extensions that look similar to the user remain in the same state. This is more intuitive for the user.
extforcedapply=	By default when extforced is used it is applied whenever that mode is selected. This setting allows you to control when extforced is applied. Example: extforcedapply=3, 4 Sets extforced to only be applied for this mode when switching from modes 3 or 4
extNclose= extNopen=	This setting sets dependant extensions that should open or close based on another extension opening or closing.

Example: ext2open=1-, 3+

Sets when Extension 2 opens, Extension 1 will close and Extension 3 will open.

Example: ext2close=4+, 3-

Sets when Extension 2 closes, Extension 4 will open and Extension 3 will close

Example: ext3open=2<, 5>

Sets when Extension 3 opens, Extension 2 will slide close and Extension 5 will slide open (provided Extensions 2 and 5 are capable of sliding).

extNowner=

By default, an extension's owner is the extension that contains its button to open or close itself. You can alter the owner by using this setting.

Example: ext3owner=2

Sets Extensions 3's owner to be Extension 2

bodyowner=

Same as above but applies to the main Body Extension

Example: bodyowner=2

Sets Body's owner to be Extension 2

extNafter=

This setting forced an extension to always be after (underneath) other specified extensions.

Example: ext4after=1, 2, 3, B

Sets Extension 4 to always be after Extensions 1,2,3 and the Body Extension ('B')

bodyafter=

Same as above but applies to the main Body Extension

Example: bodyafter=2

Sets Body to be after Extension 2

extNdefaultsize=

This setting allows you to set a resizable extension's initial size. Parameter order is left, top, right, bottom.

Example: ext4defaultsize=0, 0, 50, 50

Sets Extension 4's default resize state to be 50 to the right and 50 down.

bodydefaultsize=

Same as above but applies to the main Body Extension.

Example: bodydefaultsize=25, 50, 100, 100

Sets the Body Extension's default resize state to be 25 to the left, 50 up, 100 to the right and 100 down.

extNlimitsize=

This setting allows you to set a stretching limit to any extension. Extensions that are limited will also be able to resize automatically when the resize button is clicked. Parameters are left, top, right, bottom (0 indicates no limit).

Extensions that resize dependently with other extensions (see extNresizedep) can set a limit that is in addition to the amount the dependant extension has resized.

Examples:**ext1Limitsize=50,0,0,50**

Sets Extension 1 to have limited resize to the left and down of 50 pixels. The other directions are unlimited. The 0 values indicate the directions are unlimited.

ext1Limitsize=50,0,0,ext2+50

Sets Extension 1 to have limited resize to the left of 50 and down 50 pixels + whatever extension 2 has resized in that direction.

bodylimitsize= Same as above but applies to the main Body Extension.

Examples:**bodyLimitsize=0,0,100,100**

Sets the Body Extension to have limited resize to the right and down of 100 pixels. The 0 values indicate the directions are unlimited.

bodyLimitsize=0,0,ext1+100,100

Sets the Body Extension to have limited resize to the right of 50 and down 100 pixels + whatever extension 1 has resized in that direction.

extNresizedep= This setting allows you to set other extensions to resize together with extension N. Thus if extension N resizes to the right, the set extensions will also resize to the right by the same amount (extNlimitsize settings still apply). You can set the dependency to occur on either vertical, horizontal, or both, resizes.

Example: ext1resizedep=4x,6y

Sets extensions 1 to resize in the x-direction (horizontally) when 4 resizes in the x-direction. And to resize in the y-direction (vertically) whenever extension 6 resizes vertically.

bodyresizedep= Same as above but applies to the main Body Extension

Example: bodyresizedep=7x

Sets the Body Extension to resize whenever the extension 7 resizes horizontally.

visbgcolor= This setting defines the color for the background of the External Visualization window. The value should be an R, G, B triple (each value from 0 – 255).

Example: visbgcolor=128,128,128

visbmpalign= This setting defines how *viswndbg.bmp* displays in the External Visualization window. Valid values for this settings are **stretch**, **center**, or **tile**.

Examples:**visbmpalign=stretch**

Fits the bitmap to the entire visualization window

visbmpalign=center

Positions the bitmap in the center of the visualization window

visbmpalign=tile

Tiles the bitmap as many times as necessary to cover the visualization

window

trackalphas=

By default, each track button on the playlist is opaque. This setting defines the transparency level of each state of the Playlist's track buttons so you can create a playlist that is entirely or partially transparent. Define one alpha value for each track-button-state bitmap in a comma-delimited list. Since the fifth track button bitmap is merged over the top of the others, the fifth value of this setting defines the level of blending to use. The other four values define the level of transparency for the track button.

Example: `trackalphas=128,224,164,164,128`

trackalphasenc=

Same as above but applies to Encode list tracks.

visNmode=

See [Skinnable Visualization Settings](#)

visNalpha=

visNdrawdir=

visNcolor=

visNbmp=

extButtonBrowser=

extButtonVideo=

extButtonLibrary=

extButtonPlaylist=

extButtonEncoder=

extButtonVisuals=

These settings allow the skin to declare which extensions provide specific functionality. That way, the play can know which extension to open when that functionality is required.

The value set is the number of the extension that provides the functionality that matches the setting name.

Example: `extButtonPlaylist=4`

Sets Extension 4 to be the extension that the player should open when the Playlist needs to be shown.

13. Media Library Skinning

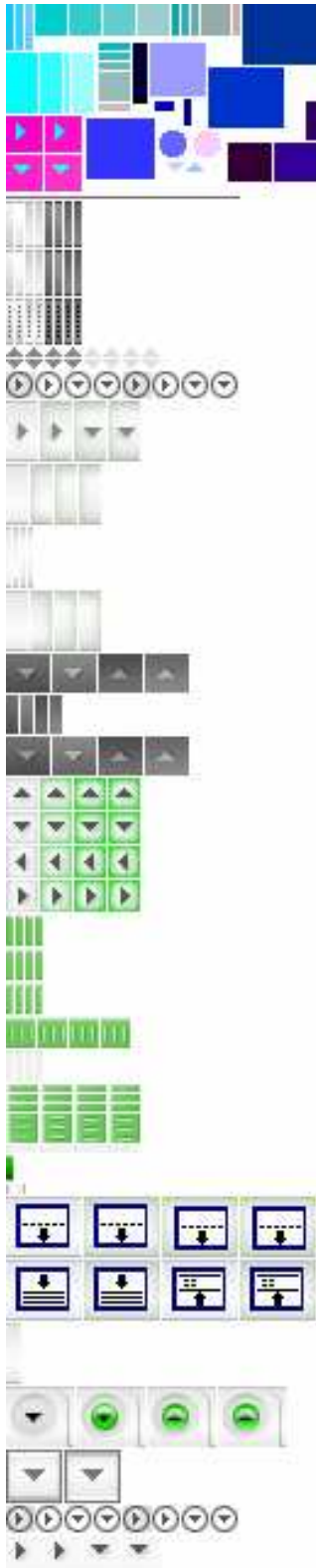
The Media Library is a unique component of QMP that requires its own skin control set and skin settings file to be skinned completely. This section describes the required elements.

The *MLSet.bmp* file defines the bitmaps used to skin the Media Library. The Media Library has a number of GUI components (scroll bars, buttons, etc...) and all images are provided on this bitmap. There is also an *MLSet.ini* file that partners with the bitmap to provide color and font information for parts of the Media Library that are not skinned with bitmaps.

13.1.1. *MLSet.bmp* – Media Library

- Internal Map – YES
- Controls Reproducible – NO

The *MLSet.bmp* file defines the bitmaps used to skin the Media Library. The Media Library has a number of GUI components (scroll bars, buttons, etc...) and all images are provided on this bitmap. There is also an *MLSet.ini* file that partners with the bitmap to provide color and font information for parts of the Media Library that are not skinned with bitmaps.



Each row below has a region in the Internal Map to define its size
 The color next to the row is the corresponding Internal Map color
 Rows contain the same four button states (normal, mouse-over, pressed-no-highlight, pressed-with-highlight)

Row 1: Column Header (left side)	0x00CCFF
Row 2: Column Header (center – tiled)	0x33CCFF
Row 3: Column Header (right side)	0x66CCFF
Row 4: Column Header Sort Ascending	0x99CCFF
Row 5: Column Header Sort Descending	0xCCCCFF
Row 6: Column Header Button	0xFFCCFF
Row 7: Menu Button (left side)	0x00FFFF
Row 8: Button (left side)	0x33FFFF
Row 9: Button (center – tiled)	0x66FFFF
Row 10: Button (right side)	0x99FFFF
Row 11 Menu Button (right side)	0xCCFFFF
Row 12: Device Separator (left side)	0x330033
Row 13: Device Separator (center – tiled)	0x330066
Row 14: Device Separator (right side)	0x330099
Row 15: ScrollBar Arrow (up)	0x00CCCC
Row 16: ScrollBar Arrow (down)	0x33CCCC
Row 17: ScrollBar Arrow (left)	0x66CCCC
Row 18: ScrollBar Arrow (right)	0x99CCCC
Row 19: ScrollBar Horizontal Thumb (left)	0x00AAAA
Row 20: ScrollBar Horizontal Thumb (center - tiled)	0x33AAAA
Row 21: ScrollBar Horizontal Thumb (right)	0x66AAAA
Row 22: ScrollBar Horizontal Thumb (center img)	0x99AAAA
Row 23: ScrollBar Horizontal Track (tiled)	0xCCAAAA
Row 24: ScrollBar Vertical Thumb (top)	0x00BBBB
Row 25: ScrollBar Vertical Thumb (center - tiled)	0x33BBBB
Row 26: ScrollBar Vertical Thumb (bottom)	0x66BBBB
Row 27: ScrollBar Vertical Thumb (centered img)	0x99BBBB
Row 28: ScrollBar Vertical Track (tiled)	0xCCBBBB
Row 29: Splitter Vertical (horizontally tiled)	0x000099
Row 30: Splitter Horizontal (vertically tiled)	0x0000CC
Row 31: Splitter Close Devices button	0x003399
Row 32: Splitter View Devices button	0x0033CC
Row 33: Toolbar Background (tiled)	0x000033
Row 34: View Dropdown Arrow	0x9999FF
Row 35: List Menu Button	0x6666FF
Row 36: TreeView Parent Item	
Mask Colors:	0xFF00CC, 0xFF11CC, 0xFF22CC, 0xFF33CC
Control Colors:	0x00FFCC, 0x11FFCC, 0x22FFCC, 0x33FFCC

MLSet.ini

MLSet.ini sets the various colors of the non-skinned controls and text in the Media Library. Each of the 4 panes in the Library can have its own color settings. The sections and settings are as follows:

[Library]

```
BackGround=241,241,241
LassoBG=
LassoBGAlpha=
LassoFrame=
```

[Toolbar]

```
ButtonTextDefault=0,0,0
ButtonTextHighlight=90,90,90
ButtonTextDisabled=170,170,170
```

[ViewChooser]

```
DropListBG=255,0,0
DropListText=0,0,0
ChooserBG=224,224,224
ChooserBorder=79,79,79
ChooserText=0,0,0
```

[ViewPane]

[ViewTrackList]

[DevicePane]

[DeviceTrackList]

```
HeaderTextDefault=0,0,0
HeaderTextHighlight=0,0,0
```

```
StatusBG=255,255,255
StatusBorder=119,119,119
StatusText=90,90,90
```

```
ListViewText=60,60,60
ListViewTextSelOff=60,60,60
ListViewTextSelOn=0,0,0
ListViewTextError=168,168,168
```

```
ListViewRowEven=248,248,248
ListViewRowOdd=242,242,242
ListViewRowSelOn=42,255,0
```

ListViewRowSelOff=28,196,0

ListViewRowError=64,64,64

SeparatorBGOpen=248,248,248

SeparatorBGClosed=242,242,242

SeparatorFrameOpen=42,255,0

SeparatorFrameClosed=28,196,0

SeparatorTextOpen=64,64,64

SeparatorTextClosed=64,64,64

There are also font sections available to define the fonts used.

See CharSet.ini in the QCD skinning document for how to use these fonts sections

[ButtonFont]

[SearchFont]

[ChooserFont]

[DropListFont]

[StatusFont]

[ListViewFont]

[HeaderFont]

[SeparatorFont]

Media Library fonts cannot use Bitmap Fonts, only Windows Fonts.

For more information on setting fonts, see [Valid Font Settings](#).

14. Additional Skin Contents

In addition to skin bitmaps and settings files, these are other content you can add to enhance your skin.

14.1. Thumbnail.bmp

Add a file called *Thumbnail.bmp* to the root of your skin (not in a folder) and it will be used for display in the Skin Browser in QMP. The bitmap must be a 60x60 bitmap file.

14.2. Comments.txt

Add a *Comments.txt* file to the root of your skin (not in a folder) to include a description with your skin that will be displayed, along with your thumbnail, in the Skin Browser in QMP.

The **notes=** field in the skin settings file will override *Comments.txt*, but you may choose whichever one to use is easier.

Tip: email addresses and URLs in this file (or in the notes= setting) will appear active in the description in the Skin Browser and will be clickable by the user.

14.3. SkinLayout.ini

Add a *SkinLayout.ini* to the root of your skin (not in a folder) to include preset layout configurations for your skin. The *SkinLayout.ini* file is generated by saving layouts in QMP with your skin loaded. Once you have the layouts as you like, just add the *SkinLayout.ini* file to your skin.

Tip: trim the *SkinLayout.ini* to just the section that is named for your skin. Other sections will be ignored and are just taking up space.

14.4. SkinColor.ini

Add a *SkinColor.ini* to the root of your skin (not in a folder) to include preset color themes for your skins. The *SkinColor.ini* file is generated by saving color settings in QMP with your skin loaded. Once you have the colors as you like, just add the *SkinColor.ini* file to your skin.

Tip: trim the *SkinColor.ini* to just the section that is named for your skin. Other sections will be ignored and are just taking up space.

14.5. Custom Contents

Any other bitmaps, files, can be added to a skin. Custom settings not defined in this file may also be added to any of the settings files. These custom files and settings can be accessed by plug-ins that expect their existence and know their contents.

15. Testing a Skin

Once you begin creating all the Maps, Bodies, Control Sets and Border Sets that will compose your skin, testing the work in progress becomes a priority.

To test your skin:

1. Copy all of the bitmap files for a single skin into one folder.
If you are using the '[path](#)' setting, you can prepare the folder hierarchy as it will be used in the skin.
 2. Start the player and navigate to **Preferences – Skins – Installed Skins**.
If your skin folder is not listed in the selection window, set the correct skin folder under **Preferences – Skins – Skin Folders**. Add your skins parent folder to the list of skin folders.
 3. Select your new skin in the Skin Browser.
The name of your skin is taken from '[name](#)' setting in the skin *.ini* file, or if it is not available it will use the folder name for the skin.
 4. Verify that the user interface functions as you expect.
 5. If you find problems or inconsistencies, edit the *.bmp* files and repeat the testing process.
- To reload a skin without restarting the player, press **F5**.

16. Packaging Skins

When you are ready to publish your skin, package all its files into a single zip-compressed archive. If your skin has multiple modes, package the skin into a family skin. Otherwise, package your single-mode skin into a kid skin.

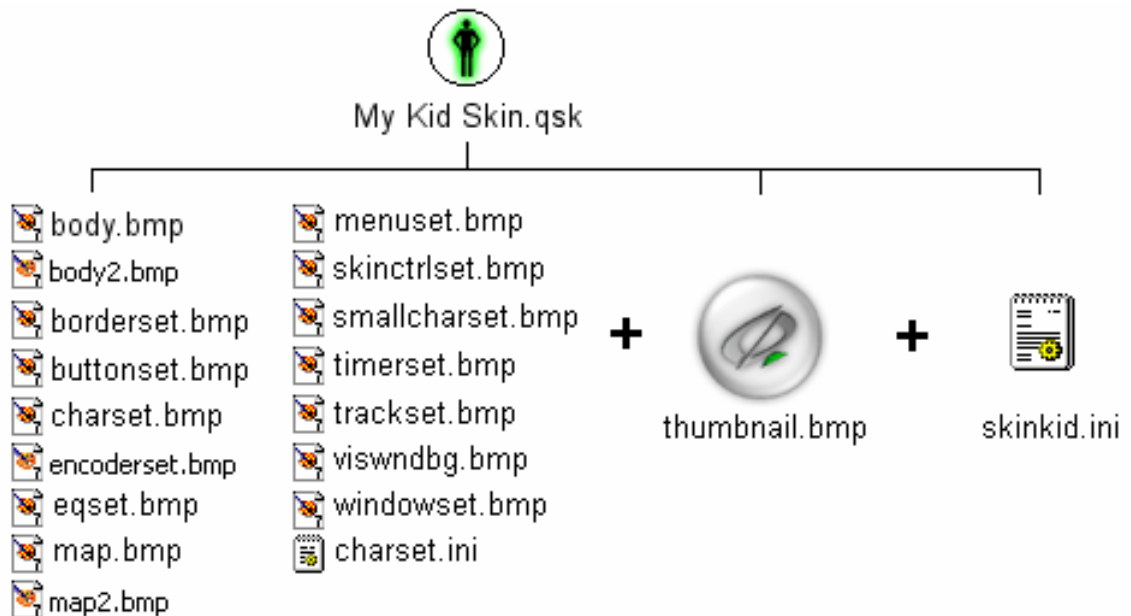


In either case, create a thumbnail image to represent your skin in the Skin Browser. This is a 60x60 pixel Windows Bitmap (.bmp) file that represents your skin.

16.1. Packaging a Kid Skin

To create a kid skin:

1. Use WinZip to combine all the skin bitmap files and settings files into a single *zip* archive.
2. Change the extension from *.zip* to *.qsk* (Quintessential Skin Kid).
3. If you have no settings files, but would like to include a description of your skin, add a *.txt* file to the zip archive (filename does not matter). The player will use the text file contents for the skin's description in the Skin Browser.



16.2. Packaging a Family Skin

There are two ways to package a family skin:

- Share common bitmaps between modes, potentially requiring less disk space and less memory for display. For this method, you must properly define the **path=**, **chars=**, and **common bitmap files** settings in the *SkinFamily.ini* (see: [SkinFamily.ini Sections](#))
- Combine kid skins. For this method, you must properly define the **file=** setting in the *SkinFamily.ini* (see [SkinFamily.ini Sections](#))

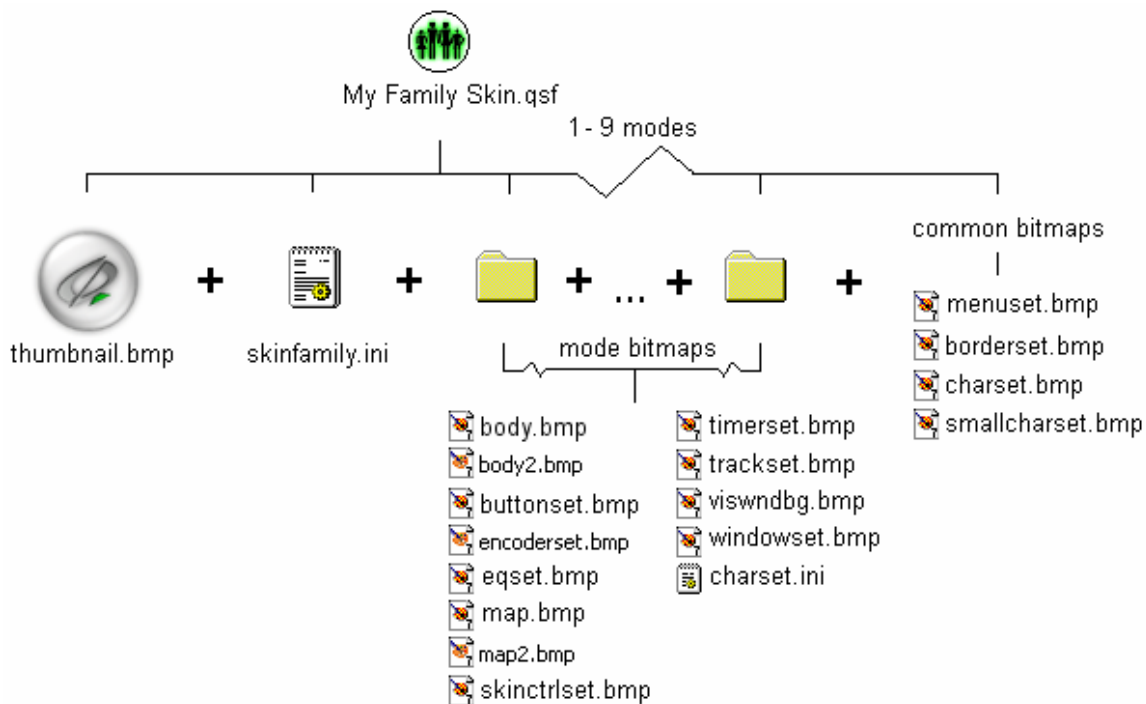
16.2.1. Method 1: Sharing common bitmaps between modes

To package a family skin by sharing common bitmaps between modes:

1. Keep all bitmaps for each mode in separate folders (see [path=](#) setting to implement).
2. Separate all bitmaps common to all skin into one location to be used by all modes.
3. Since the common bitmaps will be in a separate location you will need to reference each of the common bitmaps in the *.ini* file. The key for each bitmap is the same as the bitmap name, minus the extension.

(See: [common bitmap files](#). If the common bitmaps are in a subfolder, that subfolder must be included in the reference to the common bitmap)

4. Use WinZip to archive all the common bitmaps, the mode folders with the mode bitmaps, the *SkinFamily.ini* and the *thumbnail.bmp* file.
5. Change the extension from *.zip* to *.qsf* (Quintessential Skin Family).



16.2.2. Method 2: Combining kid skins

To package a family skin by combining kid skins:

1. Use WinZip to create individual kid skin (*.qsk*) files for each of the skin modes, as described above.

Do not include a *thumbnail.bmp* or *skinkid.ini* file in any of the kid skins as they will not be used.

2. Create a *SkinFamily.ini* with settings for the entire skin.
3. Use WinZip to create a zip archive containing all the individual *.qsk* files, the *thumbnail.bmp* file, and the *SkinFamily.ini* file.
4. Change the extension from *.zip* to *.qsf* (Quintessential Skin Family).

