Emulation Station 2.x Theme Building Guide 1.0

This appeared daunting when I started looking at it, but it's actually a pretty simple and user-friendly process. I hope this sheds some light on any of the quarks in the process of creating themes. I'm going to try and make this as idiot-proof as possible, so apologies if it gets too basic at points. It's important to note that all this information is located on Aloshi's GitHub page.

The Design

I'm a multimedia designer by trade, so I naturally jumped into Photoshop for my initial design layouts. Obviously this can be done with any number of photo editing programs. A cool, free, online Photoshop-esque solution is Pixlr (http://apps.pixlr.com/editor/)

I don't know an incredible amount about XML, but I kind of know a few other forms of coding and was able to transfer that skillset into this project. I assume that this delves as far into XML as you want to go, but I took a preexisting theme (Simple 1.0) for my build and modified heavily. The "elements" in the "views" below were present in the other theme, but there may be more elements available to customize.

There are 3(4) main sections you will be designing for, each with specific pieces you can display. I say 3(4) because I am currently unaware of the grid-view functionality.

- System View
    - Function
        - Main screen where you slide through systems
    - Elements
        - Background (individual backgrounds for systems while scrolling)
        - Logo (system logo set in the individual system's xml file, but we'll get there later)
        - Helpsystem (bottom navigation button reference)
        - (side note – I would assume you can alter the background of the system logo bar, as well as the game count bar background and font, but I'm not sure how to do that…yet)
- Basic View
    - Function
        - System game list without metadata
    - Elements
        - Logo (header)
        - Background (background of specific system page)
        - Font
        - Gamelist (list of games)
            - Primary color (game title color)
            - Secondary color (folder title color)

            - Selected color (selected game title color)
            - Selector color (highlight of selected game title)
        - Helpsystem (bottom navigation button reference)
- Detailed View
    - Function
        - System game list with metadata
    - Elements
        - Logo (header)
        - Background (background of specific system page)
        - Font
        - Gamelist (list of games – see modifiers above)
        - Box art (image of game box art)
        - Metadata (title and value – value just means “who the publisher is” or “how many times played,” etc.)
            - Rating
            - Release date
            - Developer
            - Publisher
            - Genre
            - Number of players
            - Last time played
            - Play count
            - Game description
        - Helpsystem (bottom navigation button reference)
- Grid View
    - Function
        - Idk, but it sounds cool

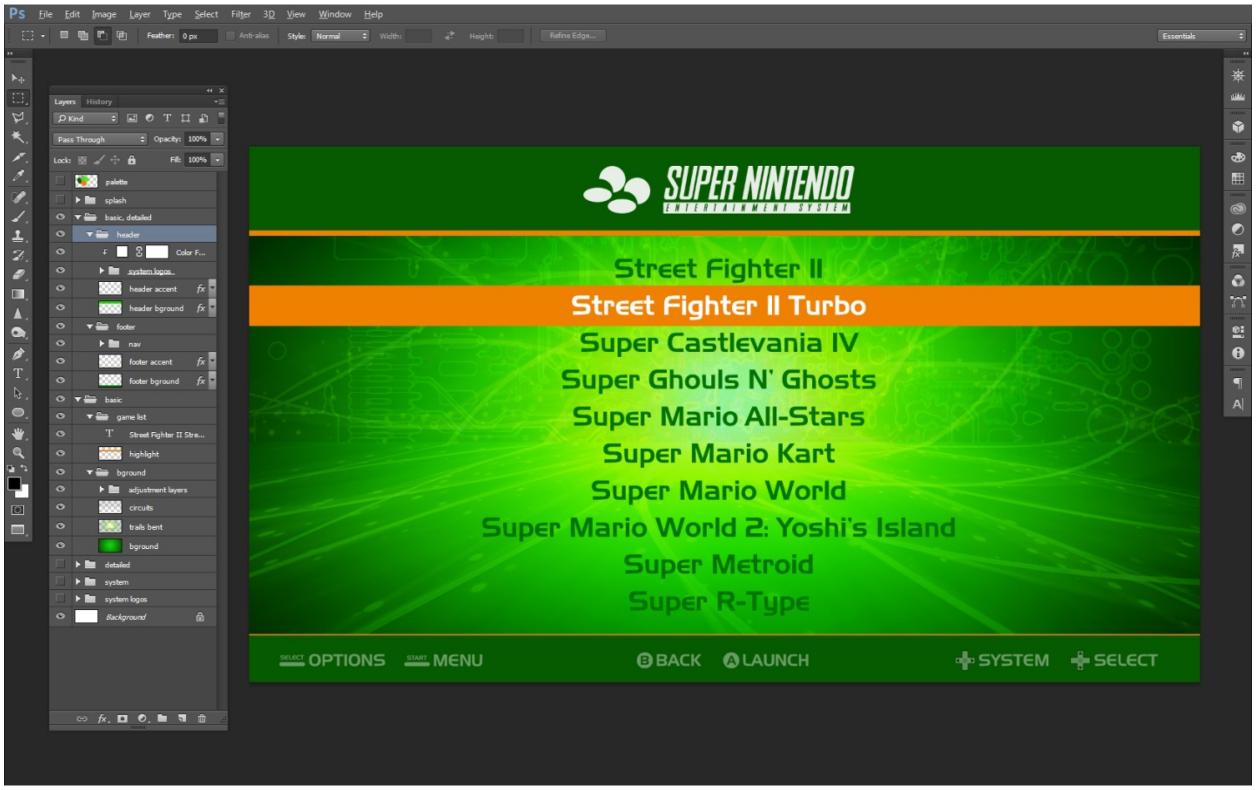
I built my theme at 1920x1080 to accommodate for high definition screens. If you use SVG files instead of JPG or PNGs, they will scale without issue. I don’t think there should be an issue scaling down from maximum resolution (1080p) JPGs, so I exported in JPG (except for the logos on the System screen, which I used SVGs.) I need to test my theme on a lower-res screen before I can comment further.

Any of these views can have additional elements added to them, like text or images, so don’t feel like you are stuck to just the elements listed above.

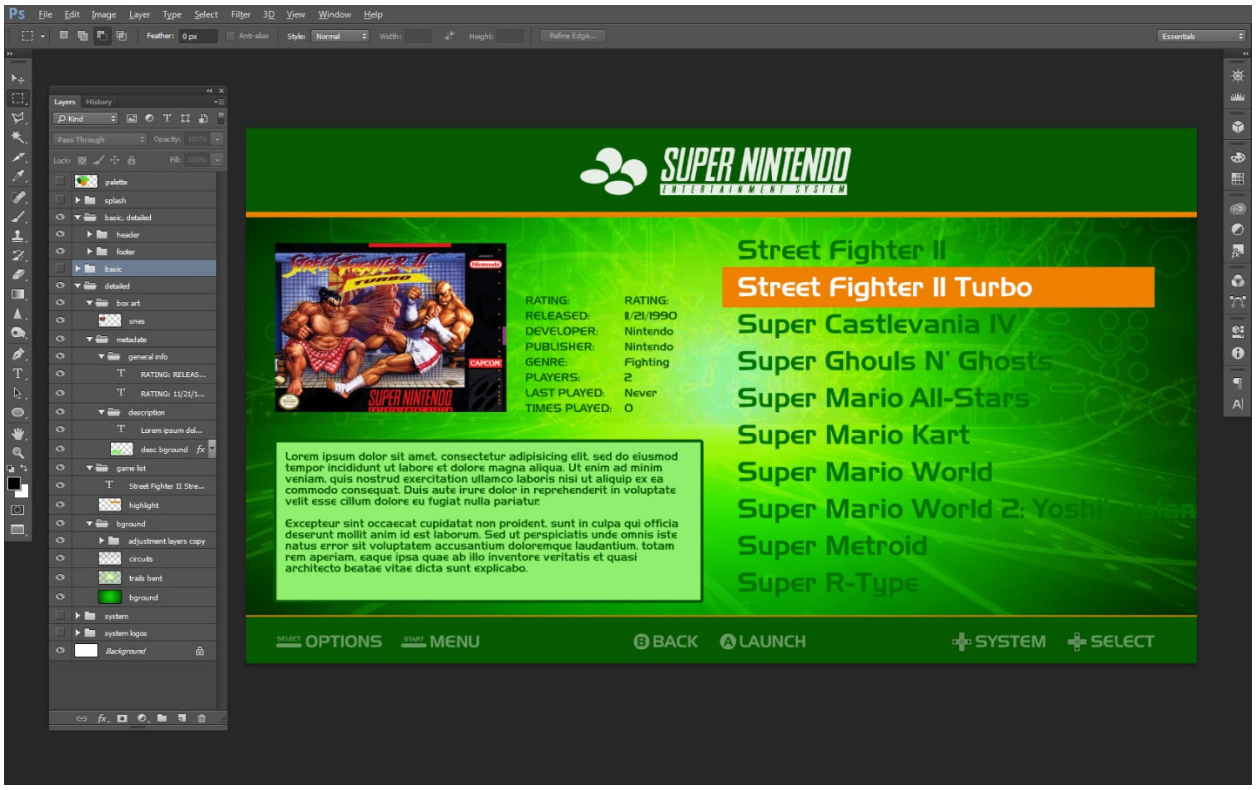
Fonts – choose free fonts with open licenses, as you will need to package the font file(s) with your final product. Don’t forget to give credit to the font creator!

Here are examples of my final screens in Photoshop:

Basic View



Detailed View

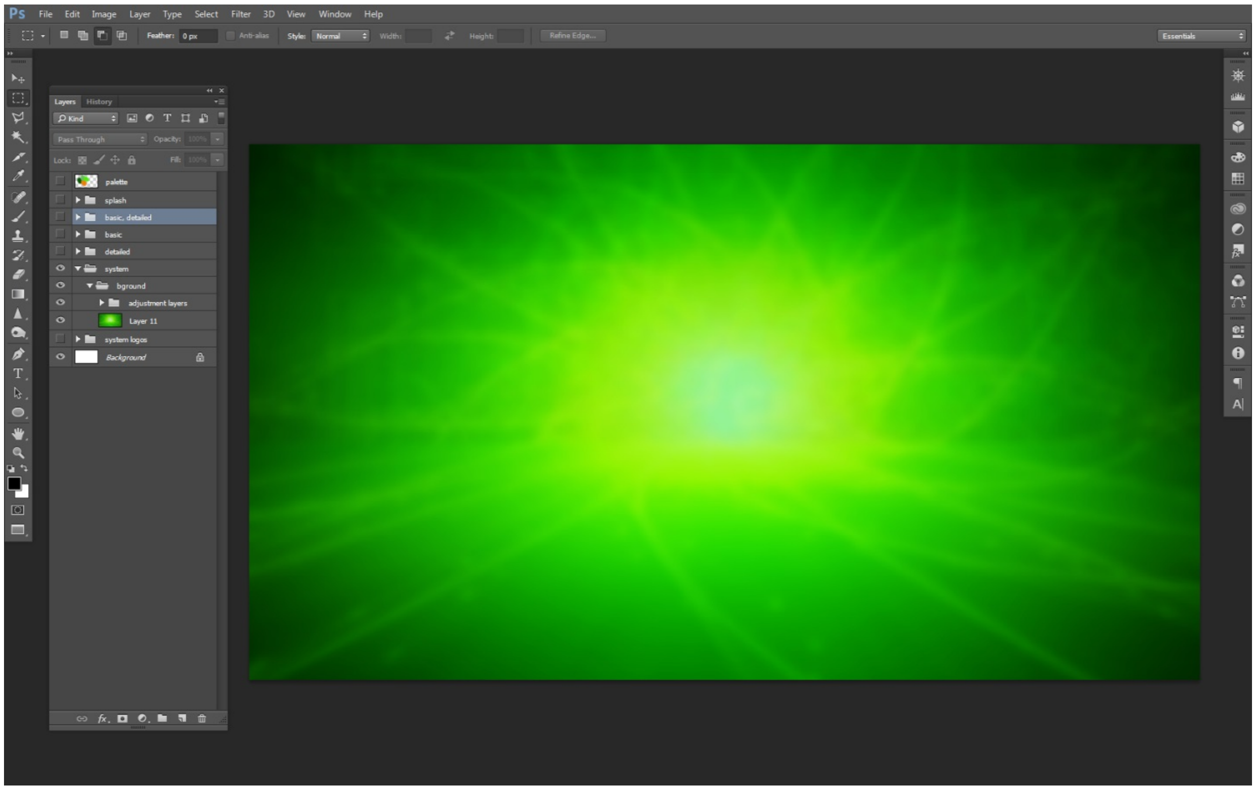


RATING:	RATING:
RELEASED:	11/21/1990
DEVELOPER:	Nintendo
PUBLISHER:	Nintendo
GENRE:	Fighting
PLAYERS:	2
LAST PLAYED:	Never
TIMES PLAYED:	0

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

System Background *(As stated, I'm not sure how to change the default system scroller, so I just made a universal background. You can make a unique background for each system!)*



Feather: 0 px
Style: Normal
Pass Through
Opacity: 100%
Fill: 100%

Collecting Assets:

Ok, so you have a super sweet theme designed…what now? Well, name it first! For the purpose of this guide, let's say you named your theme "Bitchin Theme." Now let's get that mutha out of Photoshop (or your design program of choice) and into pieces that Emulation Station can use!

Start by creating your theme's folder structure. You will have a root folder, an assets folder and individual folders for each system you are building a theme for. Individual system folders will need their own asset folder (for organizational convenience – you could definitely put all your assets in the main assets folder, but this way allows for quick reference)

Your folder structure should look something like this (where "." would be your "Bitchin" directory):

```
./
./assets
./gba
./../assets
./nes
./../assets
./snes
./../assets (etc.)
```

Now you need to assess what you need for each "view" or screen. (**super important note**: write down the pixel dimensions of each piece you export. You will need this info later!!) What are we exporting here? Just the graphics. Ignore all the text. Text will be coded in later, or auto-populated. Basically you will crop out, or slice out, or however you want to export, elements into individual graphic files.

System View

Let's see…the System View needs the background(s) that will be displayed while selecting your system, so save each one of them into their respective system asset folders, or if you are using one universal background in the root asset folder.

I.e. your NES background goes in ./nes/assets and your GBA background goes in ./gba/assets. If you are using a universal background it would go in ./assets

Are you using a custom footer in place of the Helpsystem? Crop out and export that bad boy to ./assets

Do you have custom system logos to be displayed in the system select carousel? Export them to their respective system asset folders. (Custom NE S logo goes in ./nes/assets, etc.)

Basic View

Basic View is, well, basic. At the minimum it should display your game list. There isn't much else to do here because as soon as metadata is added we switch to Detailed View. You may have some graphics you would like to float somewhere, or maybe another piece of text, but that all depends on your original design. For the most part, supplemental graphics can be added into the background. But I digress.

So let's get the background out and save it just like you did for the Systems View background(s.) Same dog-and-pony show; if you have custom backgrounds for individual systems, put them in that system's assets folder. If you are creating a universal background, put that in the global assets folder.

Header and Footer both go by the same process as the System View as well. You will most definitely have individual headers, so save them in their respective system assets folders. If you are using the logo from the System View in your header, don't worry about exporting twice – just call to it from one place while coding. If you have individual footers for each system, put them in that system's assets folder. If you have a global footer, it goes…that's right…in the global assets folder.

Export any additional elements you have for Basic View and put them in the appropriate assets folder.

<u>Detailed View</u>

This View is way cooler and impressive than Basic View. If Views were Baldwins, this would be the Alec of the bunch. Same as before, export your individual elements to their respective assets folders.

As stated in the beginning of this, skip all text. Just pull out the pieces you need, like header, footer, background, system logos (if not incorporated in your header,) and place them in their respective assets folders.

Ok. Assets are all chopped up, exported and placed nicely into their new folder homes. Give yourself a high-five; the first half of the project is done!! Now what? Well, now we need to write a few XML files so that Emulation Station knows what to do with all your badass graphics. Scared? Don't be!

Coding the Theme

Coding is best done in a coding program, but can be done in a Notepad or and txt editor! I personally like the flow of Dreamweaver, so I use that for my coding platform. Word? Word. Ok, let's get to it!

Global XML

You will need a Global XML file and Individual systems XML files. Think of them as CFG files in Linux. If you have a Global CFG file, it affects all of its children. However, these children can have their own CFG files that override the global CFG, thereby allowing you Global and Individual customization. It is good practice to Globalize as much as you can to keep individual files as clean as possible. No one likes sifting through messy code, and more than likely it will be you. Additionally, and more importantly, it keeps the possible bugs in one place opposed to spread out in many places. Is your font size wrong on your metadata? If you defined <fontSize> globally, you would only need to go to one document to investigate.

Let's start by making our Global XML file. Right off the bat, save your file into your theme's root directory:

./bitchin.xml

Ok, all this file (and Individual XML files) is (are) doing is telling Emulation Station what to do with your assets, as well as how to display the information it already has. All elements are defined within opening and closing bracket tags. For example:

```
 <something>

           <stuff> here is the stuff for something</stuff>

</something>
```

First, get wrap your head around the necessary sections of this file. You will have a <theme> that holds <view> sections. <theme> is the container of the whole shebang. <view> denotes one of the main 4 sections (system, basic, detailed, grid.) Within <view> you will put elements, represented by whatever kind of element they are <image> or <text> or <textlist>, etc. Within these elements, you can apply properties, like <color> or <size> or <pos>. Elements are defined with a pre-set naming convention. You can find the full list of these in the reference section on Aloshi's GitHub page.

Let’s show that visually:

```
<theme>
<formatVersion>3</formatVersion>
        <view  name=”basic”>
                <image name=”background”>
                        <pos> 0 0</pos>
                        <size>1 1</size>
                        <origin>0 0</origin>
                        <path>./assets/bground.jpg</path>
                </image>
        </view>
<theme>
```

Ok, what just happened there?

First, it’s important to note that you must define the formatVersion on EVERY XML file that you are using. Put it just under the Theme opening tag. Format 3 is used for Emulation Station 2.x.

But, anyway, what did we do here? We just told Emulation Station to display a file called background.jpg as the background to the Basic View, positioning it in the top left corner of the screen, sized to the full width and height of the screen. Coooool.

But why did that work?   Well, we told Emulation Station that this is a Theme file with <theme>, we defined the format version with <formatVersion>, we told ES that we are talking about the Basic View with <view name=”basic”>, we told ES that the element within Basic View it needs to pay attention to is the background image with <image name=”background”>, then we told ES how to display the image with <pos>, <size>, <origin> and <path>.

Quick definitions:

<pos> means “position,” and it very literally means that. What is the XY position of this element? Position is defined with a NORMALIZED_PAIR of coordinates without quotes and separated with a space. The top left of the screen is 0 0 and the bottom right is 1 1. “1” means move 100% of the total distance of the referenced element down the respective plane. The first number represents the X coordinate and the second number is the Y coordinate. So 0 1 would be the bottom left corner (move on the X plane 0 and move on the Y plane 1) and 1 0 would be the top right (move on the X plane 1 and move on the Y plane 0.) You will use fractions of 1 when defining position on the page unless you are talking about a corner of the canvas. The direct center of your canvas would be represented as 0.5 0.5 (move 50% down the x plane, move 50% down the y plane.)

Remember when we were exporting assets and we wrote down our pixel dimensions? When defining position of our graphics, we need to convert the pixel dimensions into decimals for reference and placement. So if your canvas was 1920x1080px and you had a rectangle that was 300x200px, it would be taking up 0.15w and 0.18h. Why? Because 300(the width of your graphic)/1920(the width of your canvas)=0.15, and 200(the height of your graphic)/1080(the height of your canvas)=0.18. Knowing the pixel dimensions in decimal form allows you to use the decimal values when defining position and setting size. It also allows you to measure areas in your graphic editing program and know how large gaps should be in your XML positioned view.

<origin> defines the center point of the element. Origin is defined with a NORMALIZED_PAIR of coordinates as well. For example, an element with an origin of 0 0 means the center point is the top left corner of the element. Telling ES that the <pos> is 0 0 when the <origin> is 0 0 means "take the top left corner of my element and position it in the top left corner of the screen." Let's say the origin is 0.5 0.5. This means that the center point is in the middle of the element (move on the X plane half of the total distance of the element in both the X and Y directions.) Now let's say <origin>0.5 0.5</origin> <pos>0.5 0.5</pos>. This would place the element directly in the center of the screen. Why? Because we told ES that the center point of the element is the middle of the element (with <origin>), then we told ES to place the center of the element in the center of the screen (with <pos>). Sweeeeeet.

<size> defines the size of the element. This is also defined with a NORMALIZED_PAIR of coordinates. When we tell ES that our background image is <size>1 1</size>, it means that our image spans the entire available width and height of the screen (100% down the X plane and 100% down the Y plane.) Saying our element is <size>1 0.25</size> would mean that our element is the full width of the screen, but only 1/4 the height of the screen.

<path> defines the path to referenced assets. Pretty cut and dry.

Ok, where were we? Ah, yes, the Global XML file. So you see how we defined the purpose of the XML file (<theme>), the format we are building in (<formatVersion>) and the page we are working on (<basic>)? That's as simple as everything is!! Now we just need to start incorporating our elements from our Bitchin Theme!

So what can we define globally? Fonts, global backgrounds, font size, general gamelist properties, headers, footers, maybe your Rating images if replacing the stars are all good things to throw in this overarching configuration file. Here is a chunk of code that we will pick apart:

```
<theme>
   <formatVersion>3</formatVersion>

   <view name="detailed">

                        <text name="md_lbl_rating, md_lbl_releasedate, md_lbl_developer, md_lbl_publisher,
md_lbl_genre, md_lbl_players, md_lbl_lastplayed, md_lbl_playcount">
                                    <color>393a3b</color>
                                    <forceUppercase>1</forceUppercase>
                                    <fontPath>./assets/NEWTOW__.TTF</fontPath>
                                    <fontSize>0.02</fontSize>
```

```
                <size>0.12 0.04</size>
            </text>

            <text name="md_developer, md_publisher, md_genre, md_players, md_playcount">
                <color>393a3b</color>
                <fontPath>./assets/NEWTOW__.TTF</fontPath>
                <fontSize>0.02</fontSize>
                <size>0.14 0.04</size>
            </text>

            <text name="md_description">
                <color>393a3b</color>
                <fontPath>./assets/NEWTOW__.TTF</fontPath>
                <fontSize>0.025</fontSize>
                <pos>0.046 0.607</pos>
                <size>0.42 0.26</size>
            </text>

            <datetime name="md_releasedate, md_lastplayed">
                <color>393a3b</color>
                <fontPath>./assets/NEWTOW__.TTF</fontPath>
                <fontSize>0.02</fontSize>
                <size>0.14 0.04</size>
            </datetime>

            <textlist name="gamelist">
                <selectorColor>ef8100</selectorColor>
                <selectedColor>ffffff</selectedColor>
                <primaryColor>065a00</primaryColor>
                <secondaryColor>ffffff</secondaryColor>
                <fontPath>./assets/NEWTOW__.TTF</fontPath>
                <fontSize>0.04</fontSize>
            </textlist>

        </view>

</theme>
```

Wow, look at all that! What does it all mean?? If you look at it a section at time, you can pick through it pretty easily! Give it a good looking over before reading the explanation and see if you can identify with the structure here.

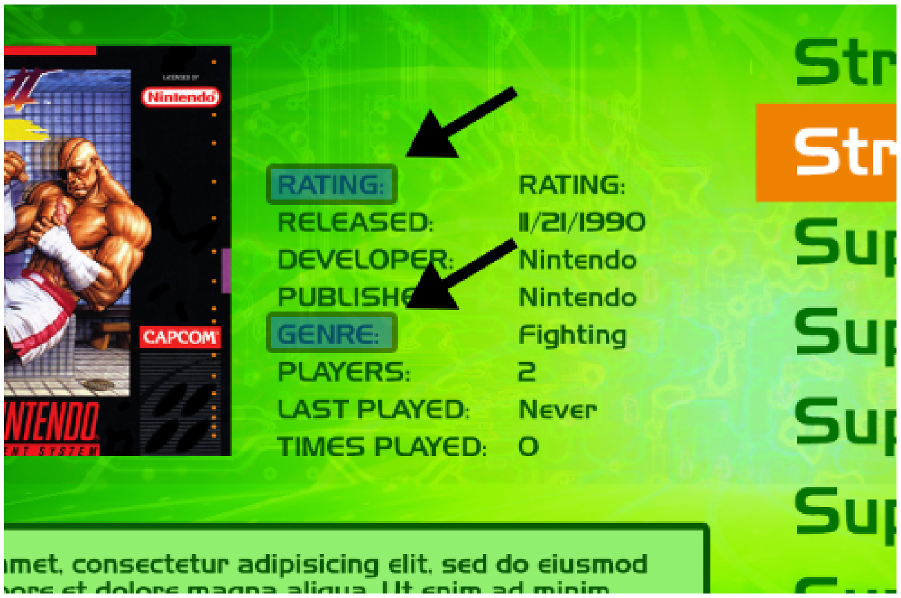
First off, we defined the purpose of the XML document and the format with <theme> and <formatVersion>. Then, we told ES that we were talking about the Detailed View with <view

name=”detailed”>. Then, we started telling ES how to handle a chunk of different elements on ***ALL*** Detailed View pages (remember, we are editing the Global XML file here…)

So what are the elements, and what did we tell ES to do with them? Let’s start with the first section:

```
        <text name="md_lbl_rating, md_lbl_releasedate, md_lbl_developer, md_lbl_publisher,
md_lbl_genre, md_lbl_players, md_lbl_lastplayed, md_lbl_playcount">
                <color>393a3b</color>
                <forceUppercase>1</forceUppercase>
                <fontPath>./assets/NEWTOW__.TTF</fontPath>
                <fontSize>0.02</fontSize>
                <size>0.12 0.04</size>
        </text>
```

In this text element…wait…what text element are we referencing here? Oh my, there are like…100 elements in there. Actually, only 8, but it looks like 100. md_lbl_* defines the label of a metadata value. So, md_lbl_rating refers to the actual text “Rating” that is displayed on the Detailed View page. md_lbl_genre refers to the actual text “Genre” that is displayed on the Detailed View page. It is important to note that elements ***must be of the same type*** to be able to be grouped into one set like this. I.e. you can’t define properties for an image and a text element in the same section.



RATING:	RATING:
RELEASED:	11/21/1990
DEVELOPER:	Nintendo
PUBLISHER:	Nintendo
GENRE:	Fighting
PLAYERS:	2
LAST PLAYED:	Never
TIMES PLAYED:	0

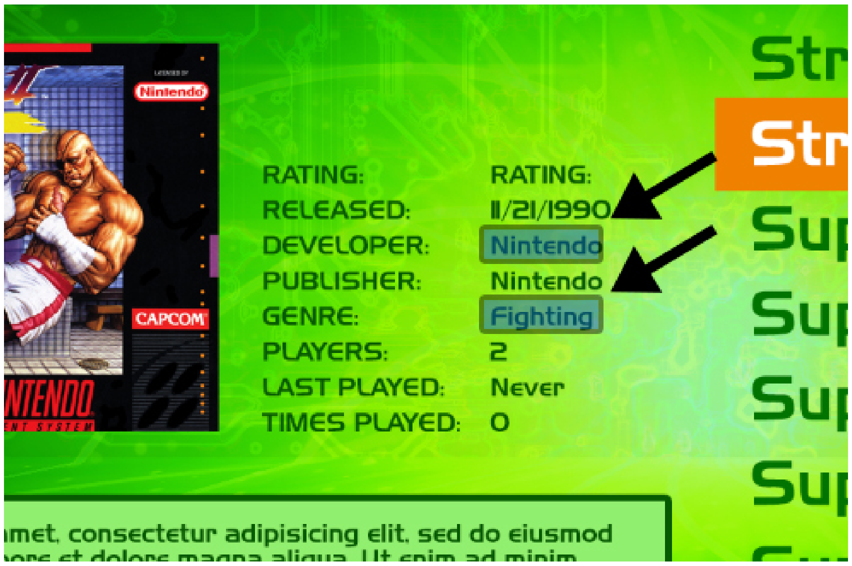
met, consectetur adipisicing elit, sed do eiusmod

So we are taking 8 different metadata labels and applying different properties to them. In the above example, we set the color to #393a3b with <color>, we forced uppercase lettering with <forceuppercase>1</forceuppercase> (this uses a Boolean value: 1=true, 0=false, but it is false by default), then we told ES to use our sweet font by pointing to it with <fontPath>, then we defined the font size with <fontSize>, and finally we told ES the total amount of space on the screen these elements are allowed to take up with <size>. Awesome!

Let’s look at the next piece of code:

```
<text name="md_developer, md_publisher, md_genre, md_players, md_playcount">
        <color>393a3b</color>
        <fontPath>./assets/NEWTOW__.TTF</fontPath>
        <fontSize>0.02</fontSize>
        <size>0.14 0.04</size>
</text>
```

What is going on here? Same thing as before, but we are working with a different set of text elements. You might notice that these elements have the same names as the previous elements, sans “lbl.” That’s because these are the *values* of the metadata. By default, these sit directly to the right of the metadata labels unless a <pos> is defined. What’s that? You want to define a specific position for these guys? Not yet!! Remember, we are still working on global properties. The actual position of these elements may vary depending on the arrangement of your individual system’s page. But we know that the color, font, font size and size will remain constant from system to system, so we define them here.

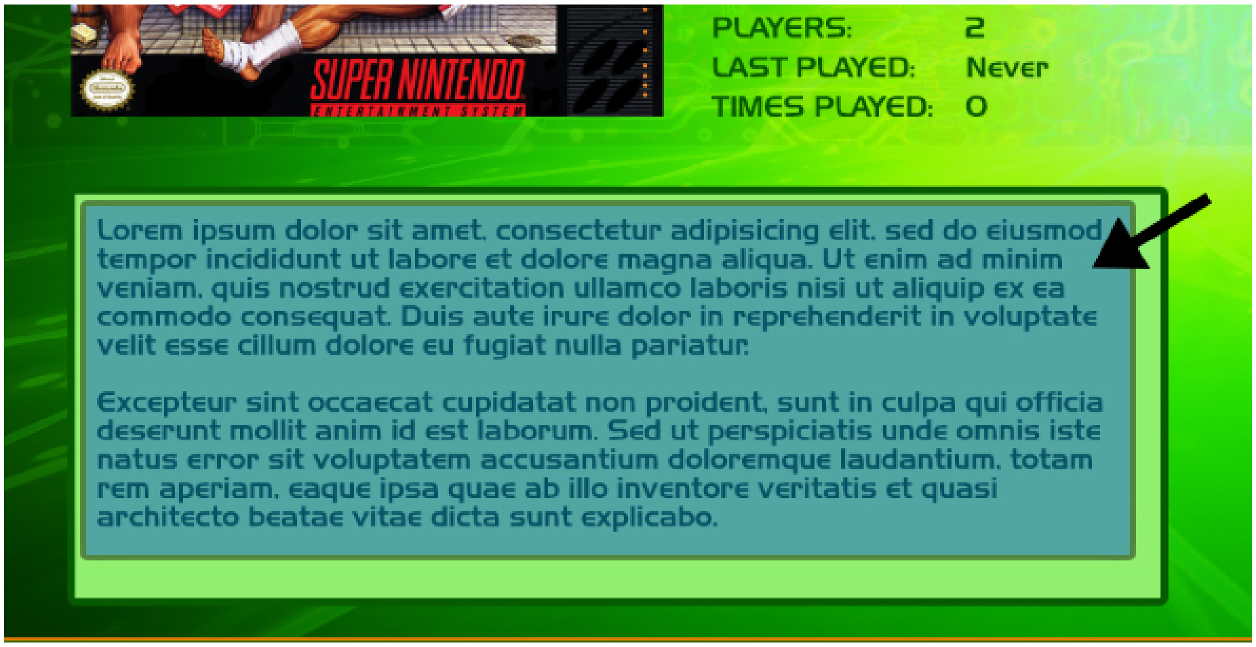


RATING:	RATING:
RELEASED:	11/21/1990
DEVELOPER:	Nintendo
PUBLISHER:	Nintendo
GENRE:	Fighting
PLAYERS:	2
LAST PLAYED:	Never
TIMES PLAYED:	0

Done and done. Ok, what do we have next?

```
<text name="md_description">
	<color>393a3b</color>
	<fontPath>./assets/NEWTOW__.TTF</fontPath>
	<fontSize>0.025</fontSize>
	<pos>0.046 0.607</pos>
	<size>0.42 0.26</size>
</text>
```

This is another text element. md_description defines the game description. Looks like the rest, yes? Wait, why is the <pos> defined here? Because in this theme, let's say we know that we wanted to keep the description in the same place for every system's page. That box isn't moving anywhere, so defining the position globally is perfect! We can easily define the position of this element by measuring the pixel dimensions in our graphic editing program and converting it to decimals as previously discussed in the definitions section. Awwww schniz.



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Rock & Roll, what's next?!

```
<datetime name="md_releasedate, md_lastplayed">
	<color>393a3b</color>
	<fontPath>./assets/NEWTOW__.TTF</fontPath>
	<fontSize>0.02</fontSize>
	<size>0.14 0.04</size>
</datetime>
```

Same as the others, but Release Date and Last Played are both dates (01/23/97 for example) so they need to be defined as <datetime> elements instead of <text>.

Good, good. Last one in this example!

```
	<textlist name="gamelist">
		<selectorColor>ef8100</selectorColor>
		<selectedColor>ffffff</selectedColor>
		<primaryColor>065a00</primaryColor>
		<secondaryColor>ffffff</secondaryColor>
		<fontPath>./assets/NEWTOW__.TTF</fontPath>
		<fontSize>0.04</fontSize>
	</textlist>
```

This is defining properties in the <textlist> element that is the ever-important game list! The color definitions are outlined on Aloshi's GitHub page, as well as in the beginning of this guide.

Fantastic! We just defined global properties for a bunch of elements in the Detailed View!

So what if there are properties that you want on 2...3…or even 4 pages? Do you want to type all that out over and over like a chump? Of course not, that's crazy. Not only is it a PITA, but it makes for some repetitious code that is no fun to look at.

Let's take the gamelist example from above and see what that would look like if you were applying it to both the Detailed and Basic view:

```
<theme>
	<formatVersion>3</formatVersion>

		<view name=”basic, detailed”>
			<textlist name="gamelist">
				<selectorColor>ef8100</selectorColor>
				<selectedColor>ffffff</selectedColor>
				<primaryColor>065a00</primaryColor>
				<secondaryColor>ffffff</secondaryColor>
				<fontPath>./assets/NEWTOW__.TTF</fontPath>
				<fontSize>0.04</fontSize>
			</textlist>
		</view>
</theme>
```

See what we did there? By defining multiple views in the view name, we are able to apply these properties to these elements in several places. Boo-ya.

Individual XMLs

Ok, we defined our global properties so that all of the common properties are taken care of. There are a few things that need to happen on an individual basis depending on how our individual system's pages are laid out. There are also a few things that need to be taken care of at this level to properly affect the System View layout.

Let's say we are working on a Super Nintendo set. We would start by creating our XML file and saving it at ./snes. Emulation Station looks for files named "theme," so we will save this as **theme.xml**. (not an option – ***you have to name it theme.xml***)

What do we do at the beginning of EVERY XML page we are creating? We define the purpose of the page and the format.

```
<theme>
	<formatVersion>3</formatVersion>
</theme>
```

Now we put it together!

First thing's first. We need to **include** our Global XML file so this page knows about all the hard work we just did. This will need to happen on EVERY INDIVIDUAL XML page that you make.

So now our page looks like this:

```
<theme>
	<formatVersion>3</formatVersion>
	<include> ./../Bitchin.xml</include>
</theme>
```

Now everything that we defined in the Global XML is already set here! What's that? You have a slight tweak to this page but don't want to adjust the Global XML for whatever reason? No big deal. Anything you define in the Individual XML will override the Global XML file.

So now we go back to laying out our 3(4) sections – System, Basic, Defined, (Grid)

2 major things that need defined in an individual system's page are the logo and background image that are displayed on the System View. So as you are scrolling through your emulators, these are the logos you see and the images that are displayed in the background. It would look something like this:

```
<view name="system">

	<image name="logo">
		<path>./assets/snes.svg</path>
	</image>
	<image name="background" extra="true">
```

```
                <size>0 1</size>
                <pos>0.5 0.5</pos>
                <origin>0.5 0.5</origin>
                <path>./assets/snesBground.jpg</path>
        </image>

</view>
```

So here we told ES that we were talking about the SNES System view with <view name=”system”> (it knows this is for SNES because we are editing the individual SNES theme.xml), that we were talking about the main logo images with <image name=”logo”>, where that logo was with <path>, then that we were talking about the background image with <image name=”background” extra=”true”> (extra is used to define an element that isn’t in the base program), then defined the <size>, <pos> , <origin> and <path>. Easy peasy!

What’s next? In my theme, I wanted a custom header for each emulator, so I defined this element for both the Basic and Detailed views. It looks like this:

```
<view name="basic, detailed">

                <image name="logo">
                        <path>./assets/snesHeader.jpg</path>
                        <pos>0 0</pos>
                        <size>1 0</size>
                        <maxSize>1 0.166</maxSize>
                        <origin>0 0</origin>
                </image>

</view>
```

Nothing that we haven’t done yet! ES defines the header area as “logo,” so it made sense to use that predefined asset. You could, however, create a new image and just not define the “logo” element. That would look the same, but be defined with <image name=”header” extra=”true”>. Awesome.

The only other things that I need to define for my theme on an individual basis are the positions of the metadata labels and the box art for the Detailed view, as I made all system layouts consistent with each other. The only thing that changes is the size of the box art, which in turn changes the position of the metadata labels and values. Check out the following for how mine looks. In the interest of space, I only put the example of the box art, the rating and the last played tags. Note that the last played value (not the label) needs to be set as a <datetime> element:

```
<view name="detailed">

                <image name="md_image">
                        <pos>0.03 0.215</pos>
                        <maxSize>0.245 0.301</maxSize>
                        <origin>0 0</origin>
                </image>

                <text name="md_lbl_rating">
                        <pos>0.292 0.315</pos>
                </text>

                <text name="md_lbl_lastplayed">
                        <pos>0.292 0.465</pos>
                </text>

                <datetime name="md_lastplayed">
                        <pos>0.4 0.461</pos>
                </datetime>

                <rating name="md_rating">
                        <pos>0.4 0.322</pos>
                </rating>

            </view>
```

So, to break this down, we told ES that we were talking about the Detailed view, that we were talking about several elements in this view and defining only the positions of these elements. All of the formatting for these elements was defined Globally and referenced with the <include> tag at the beginning of our file.

All of these sections we just laid out would need to be inside of the <theme> tag. That's it! We're done with at least one emulator! Create copies of the individual XML file for each of the other systems and adjust accordingly.

Wrap it up, B!

Test it out! Copy the root folder and all subfolders over to /etc/emulationstation/themes and you should be good to go! Change your theme from the menu inside of ES and scope all that sweet work you just did!

Oh crap, the labels don't line up right! Adjust the position values in your individual XML file. WTH?! The font isn't my sweet font that I added! Make sure your path is correct in the Global or Local files and make sure the font is actually in your assets directory! OMF G all I get is a white screen with blue text and black highlights!! You broke it. It's done. You completely failed and should give up now. Sike. There is an open tag somewhere, or you have conflicting paths in your Global file, or something else that when you see it will smack your head hard enough to leave a mark. Don't freak out. If you were all OCD about writing your code by keeping it clean and globablizing as much as possible, it should be a relatively easy process to scan your code and find that (usually) one missing or mistyped character, or see the path that your incorrectly typed, or notice that you didn't copy over your assets folder. There is always an answer! Check out the Simple Theme and see how it is put together. Check out other posted themes online at the PetRockBlog Forum and see what others did. Comparing code can be a great way to identify problems and mishaps with your work.

I hope this guide helped in some capacity and that you enjoy your new setup!!