June 22, 2006
Rebuild the Flash V2 Components?!
Eep! 20 days... no entries!

Disclaimer: This is a really flash specific entry.

I've been working with the flash components for about 3 years now (and I've been working on this blog for 3 weeks now ;) both with the v1 components, then later the v2 components.

After beating my head against walls night after night trying to get these things to work for me, I think I can tentatively say I've got a grasp on them. But, it's been a long uphill battle. After all the confusion and hair tearing, I gotta say it, I love these friggin' things. We've been building applications in flash that interface with .NET or Cold Fusion web services and in this scenario, for this they simply kick ass. Despite my acquired affection, however, it started to become more apparent that for me to use these with complete confidence, I've got to fix the things that trip me up time and again.

I set off in a pursuit to recompile them with my own custom "fixes" that would help me love them better. I read articles on their structure and looked through the classes, but it seemed like anything I did to them didn't fix some of the basic problems I was having. They are very well integrated into each other, and I simply couldn't tell what to cut out. I've grown to use the "throw out everything and start brand new" method of programming sparingly these days. You can burn many more hours redoing something and rethinking decisions that were previously made and still end up with only moderate improvements if you aren't completely sure what you're building or understand why the existing codebase is in the state it's in. So I figured this might be the perfect opportunity to get my hands dirty developing components. So I decided to rewrite the whole codebase.

The problems I hope to fix are:

"Adding a list box makes my movie 65k?! What the @&&$$?!"?? ?
BIG AS HELL. My basic belief is that if people are going to be using flash in applications, it should be as light and small as possible. Flash is built to be lightweight! Unfortunately in it's early days it developed this reputation for "Loading 30KB/10MB for the Intro" and I think it's a reputation we need to forcibly overcome. It's always bothered me that simple vectors, code, and text take so long just to load. It is really disheartening to us filesize misers to add a combo box to a movie and have it jump up to 65k or add a simple button and have it jump up to 25k without even adding one sweet alpha graphic.

"LOOOOOADDDD DAMN YOU!"
Looooooooooongg load times. I can watch a component draw itself. I can understand this happening with components that have several thousand pieces of information in their dataProviders, but not on the load of the actual component itself.

"Hmmm, I wonder if it's done compiling ... nope, not yet."
When I use that import statement on the component class, the compile times become so long that my easily distractable mind is already working on something else by the time my movie compiles.

"I like the listbox, but the look needs to change ..."
Skinning. It would seem to me that skinning should be as simple as saying "here's a theme in my library or an swf in an accessible folder that I want to use for my button but retain all of the methods and properties of my button. Currently to skin, you have to open up an fla file copy the correct skin, change it, and put it on the first frame of your project so that you import it, then you... it's just not easy. It should be easier than it is.

For the LOVE OF MARY... can I just remove that component?!
Ignoring the MovieClip Object. The movie clip object is a great object! It does everything! It seems like the components are trying to reinvent the movieclip object in favor of component object, which can be a bit confusing for everyone who's familiar with the beautiful movieClip. Why can't I remove the component with removeMovieClip?

I'm sure that the developers who made these components faced every single one of these problems and have continued to face them on a regular basis based on the comments that I've read in newsgroups about them. I really love these components. I love them because of all of the data goodness that they give me access to. However, I think the process can be refined and tightened up a bit to alleviate my stress and give me a few more years back to the end of my life.

It seems like every developer I've talked to who is using these components has their own fix for them (they came out 3 years ago fer chrissake), but nobody has posted a real, live, actual, solid, realistic replacement for them. Those that are public usually have a price tag associated with them, and have their own set of incompetencies to learn and deal with. So, feeling totally frustrated, I set out to create a set of UI components that would be:

- **Easy to Skin ... I mean REALLY easy.** Every part of the component should be easily changed and not just with code, but with your mouse too. I'm really into tactile touch. I want to put a label where I want to put it and see that I'm putting it there.
- **Lightweight. small small small.** These codechunks should be so small that html navigation graphics load slower than they do.
- Usable as a MovieClip. I love that movieclip!
- **V2 Architecture.** I really dig with the listener broadcaster event model. Plus I'm already using v2 components in my current clips, so if I build new components, I want to very easily be able to switch them in without having to rewrite code for existing method calls. So all the methods and properties have to mirror the V2 components.
- **FREE.** I really like the *concept* of open source. I don't like the idea of having to pay for something that already comes free with flash. If you purchase a compiled component that has a bug in it, or should work in a slightly different way than it's meant to, and you don't have the source code then you have to rely on the company for the fix (which I've had to do). Release your components! Make 'em FREE!

Definitely pretty lofty goals, I must admit. The problem with components is the same that you have with most software, they work great until they don't. The smallest thing can stop you from using your nifty little V2 Replacements. Scenario:

A: I have my button that I've just created as a component that I'm extremely proud of! I want to use it in an application that I'm building that is using existing V2 components, however, when my component is on the stage with the other components, it breaks both of them, and they both just sit there as empty boxes when compiled.

B: I have my button that I've just created as a component that I'm extremely proud of! I want to use

it in an application that I'm building that is using existing V2 components, however, it won't focus like the other components focus.

The moment you run into either of these scenarios, you can either a) not use your component, or b)rebuild all of the V2 components that you're using before you can use a single one of them. If you find yourself in that scenario, you've just dropped into the world of building crapware. Custom components should work with everything in flash, including V2 components. I've run into all kinds of stupid problems like these with previous components that I've built or bought.

I've been working on it all for the past couple months between projects, and with the help of some very good component resources (person13.com, actionscript.org, and www.ultrashock.com), I've built the keystones of our new component set that will be available for public download, source code and all.

They aren't posted yet because they haven't been fully tested so I'll post a code example for you to check out and see where we're headed. Click the button in this flash movie:

Ok, so why is this even remotely cool? It doesn't actually DO anything! Well, actually it does. It does quite a bit with only a small amount of code:

```
var displayedTheme = "ThemeMM";

myButton.click = function(){

  if(displayedTheme == "ThemeMM"){

    myButton.template = "ThemeXPButton";
    myCheckBox.template = "ThemeXPCheckBox";
    myRadioButton1.template = "ThemeXPRadioButton";
    myRadioButton2.template = "ThemeXPRadioButton";
    myRadioButton3.template = "ThemeXPRadioButton";
    MC_Back.attachMovie("MC_XPBack","MC_Back", 1);

    displayedTheme = "ThemeXP";

  }else{
    myButton.template = "ThemeMMButton";
    myCheckBox.template = "ThemeMMCheckBox";
    myRadioButton1.template = "ThemeMMRadioButton";
    myRadioButton2.template = "ThemeMMRadioButton";
    myRadioButton3.template = "ThemeMMRadioButton";
    MC_Back.attachMovie("MC_MMBack","MC_Back", 1);

    displayedTheme = "ThemeMM";
  }

}

myButton.addEventListener("click", myButtonLO);
```

What are you looking at? First things first, the filesize for this movie is 12.8K. This means, that I've added three components to my flash document AND added an XP skin for each of the components and I'm still at 12.8 K. If you add the V2 button, checkbox, and radiobutton to a typical flash movie, you're looking at a movie of 30K. If you add skins, then you'll add at least 5K more. so we've basically cut our file size down to almost a third of it's size. If we keep doing this with all the components, we're looking at drastically dropping the filesize of your flash movies that use components without losing functionality or changing your code at all.

Runtime Skinning? Yep, they do that. You can skin a component at runtime with nothing more than this code:

myComponent.template = "somemovieinmylibrary"

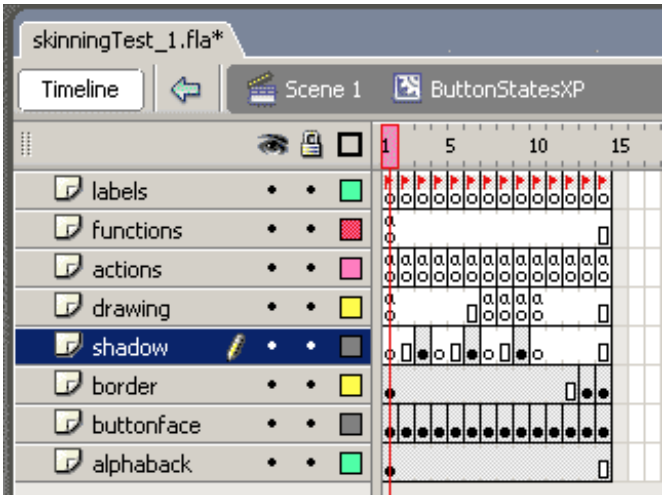
OR this code:

myComponent.template = "someswfmovieinafolder.swf"

So what's a template? It's a 14 frame Movieclip that you can slap on the component codebase. The fourteen frames represent the possible states of a button component. They are:

- up
- over
- down
- selectedUp
- selectedOver
- selectedDown
- focusUp
- focusOver
- focusDown
- focusSelectedUp
- focusSelectedOver
- focusSelectedDown
- disabledUp
- disabledSelected

Here is the XP button Skin:



That's a pretty neat trick, but why would I want runtime skinning? Well, runtime skinning allows for you to better adjust the appearence of your apps without losing any of the methods that are associated with the component. Say if you are creating a calendar application, and you would like to signify a date as being different from the other dates, but you would like it to *behave* the same way as the other dates, skinning that particular date with a custom skin in your library would be easy to do. just build the date skin with it's states, apply it to the calendar, and you're done.

What if you would like to create an application that detects whether a person is on OSX or XP and show the components in the specific style of the operating system? Wouldn't it be nice to have a folder of skins on your site that you could simply open up and change the look without having to worry about the functionality or touching your main flash file? Ahh that would be soooo nice...

Or, how about the fact that skinning is suddenly really easy to do :) ?

This style of skinning works very well for button type components, but how would you skin a list box, or a combo box or ... the DATAGRID?

Eeverything that you interact with is simply either, a) a button, or b) a collection of buttons. A list box consists of the following buttons:

- ListItem
- scrollBar component - which consists of increaseArrow, decreaseArrow, barBackground, thumbScroll (wich consists of top, bottom, back)

See where I'm going with this? The simplebutton is really the psuedo base class of everything you interact with.

This is not a novel concept and I didn't make this up. The methodology is already there in flash, I'm just reinterpretting it to make it easier to skin components without having to get bound up in the drawing API, and using flash the way I'm used to using flash: build a movieclip, give it functionality. Anyway, this is all I'm going to post for right now. I'll keep posting updates, and hopefully soon, I'll have a releasable component set for you to test out on your own.

Posted At : 10:42 PM. | Posted By : seth