Flash MX/MX2004 and ActionScript Secrets

Phillip Kerman

No matter how fast an internet connection, a small file always downloads quicker than a large one. No matter how fast your computer, a better performing application is always snappier than a slow one. And no matter how much you might charge per hour, finishing a project quicker with fewer mistakes is always beneficial.

The "secrets" I've collected for this paper range from simple tips to significant workflow changes. First, realize they're only secrets until you know about them—many you may already be using. Second, the general rule that I live by is that you must invest time to save time. That is, even if a technique takes some time to set up it will be worth it if it saves you time down the road. That said, a technique that takes months to set up but only saves you a few minutes is not worthwhile. These tips may not be appropriate for every project or work style. They're just tips, not rules. Try to keep them in balance with practical considerations.

Filesize

Although you and your collogues might all have broadband, we still live in a relatively slow connected world. Even if everyone *did* had high speed connections there's no reason to abuse it with larger than necessary files.

1. Learn how to use the bandwidth profiler (when testing a movie select View>Bandwidth Profiler). Alternatively, learn to read the size report that you can generate (from the publish settings).
2. Lines are smaller than fills.
3. Never use Modify>Shape>Soften Fill Edges (provided you're delivering to the internet).
4. Preview the effects on both quality and filesize from using Modify>Shape>Optimize.
5. Breaking apart text usually increases filesize. However, if only one character from a particular fontface appears then it might be smaller to break apart than to use static text.
6. Embedding font outlines for dynamic or input text (via the Character button) is only necessary under the following conditions: you're using a font that may not be installed your users' machines; the text must appear anti-aliased; the text must appear under a mask; or you're rotating the text. However, embedding fonts makes the filesize grow significantly. Don't use this option unless you need to and only embed the characters that you need.
7. Nested Movie Clip instances make for smaller filesizes than nested Graphic instances. However, Movie Clips don't appear on screen quite as fast and they don't stream. All frames in a Movie Clip must download before the first one appears on stage. That's because Flash doesn't know if you're going to trigger a script that makes the clip jumps to a later frame. Bottom line: use Movie Clips whenever you can; but go ahead and use Graphics when you want to progressively download the contents of an animation.

Filesize continued.

8. Avoid shape tweens unless you really need something to "morph".
9. The only thing with a bigger impact on filesize than audio is video. Half of video is audio. The point is that audio is a biggie.
10. Mono sounds are half the size of stereo. You can still pan between left and right channels using a mono sound.
11. Don't fall for the common misconception that voice can withstand more compression than music. In fact, when artifacts appear in sounds that are familiar to our ears (like the human voice or natural sounds) it's more obvious than when quality is sacrificed in unnatural or commonly distorted sounds (say, an electric guitar). If you don't know how something is *supposed* to sound then you don't know if it sounds right. Anyway, judge the results of compression using your ears on a case by case basis.
12. For video, use Sorenson Squeeze for the best quality/filesize tradeoff.
13. Although vector graphics are generally smaller than raster graphics, don't expect Modify>Trace Bitmap to save filesize. Imagine a raster graphic converted to thousands of 1-pixel wide vector circles—it's likely bigger than the original raster graphic.

Performance

Flash is a dog. Sure, performance has gotten way better in recent versions of the player, but—from the beginning—the idea behind Flash was "make small files that download quickly and offload display work onto the user's machine". This makes sense because CPU performance continues to grow at a much faster rate than bandwidth improvements. Shaving a few milliseconds off here or there can add up to a noticeably snappier application.

1. Multiplication is faster than division.
2. Don't repeatedly evaluate an expression inside a loop.

   bad:
   ```
   for (var i=0; i<myArray.length; i++){
   }
   ```
   good:
   ```
   var total=myArray.length;
   for (var i=0; i<total; i++){
   }
   ```

3. In MX2004 consider casting your variables using the colon. Just precede the first use of a variable with `var` and follow the variable name with `:type` where "type" is the data type (like `Number`, `String`, `Array`, etc). You can even specify the data type that a function is expected to return (or `Void` if none).
   Here's an example:
   ```
   var gAge:Number=10;
   function birthday():Void{
     gAge++;
   }
   ```

Performance continued.

4. Scripting tips (like above) pale in comparison to graphic display issues. You can download a test-bed application I built (from www.phillipkerman.com/ddw04) and you'll see how anti aliased text, scaling, and alpha tweens all affect performance.

Production Efficiency

The less you work the better your life. Reducing meticulous and detailed work makes you happier and less likely to make mistakes. Most people accept this basic theorem but fail live their life by it. You really want to identify places where you can work faster and more efficiently.

1. Use a simple text editor to copy and paste code blocks. This way you can effectively maintain multiple clipboards. Similarly, you can use a third party text editor like PrimalScript or SciTEFlash. Note that the undo-hierarchy changed in Flash MX 2004.
2. Flash MX 2004 supports a scripting language called JSFL that runs during author time. You can use it to create graphics, position objects on stage, or really anything that you normally do manually.
3. Building software is an iterative process. So, expect to plan-prototype-analyze and then repeat. No project is ever finished; you just stop working on it. You can also use this fact as a client management technique—say "yes, let's add that feature to the list of things we're adding in the next version".
4. If your movie has lots of audio you can significantly reduce the publish time (when you select Control>Test Movie) by temporarily setting the audio compression to Raw. Remember to set it back before you publish for real or your files will be huge.
5. Take the time to program features that save time while testing. For example, you might add a "skip" button even if you need to remove it before you deliver. Interestingly, you'll sometimes find features that are useful while testing turn out to be worth incorporating into the real project.

Scripting Techniques

After you learn the basics of the ActionScript language you need to practice using it. There's always more than one way to achieve a particular effect. These tips involve both general approaches and conventional "best practices".

1. You can quickly loop through all the properties in a particular object using the following code:

```
for (var i in myObject){
   trace(i+":"+myObject[i]);
}
```

Scripting Techniques continued.

2. Modularize code so that addressing bugs is focused and so that you can recycle work for future projects. Flash MX supports `#include "code.as"` and Flash MX Professional 2004 supports `import classFiles` which are both ways to put code in external files. (Note that in both cases, you have to republish your .swf to incorporate any changes in the code files.)
3. Another modular idea is to try to keep your main timeline free from too many clip instances—put everything in nested clips so it's easier to later nest inside a different project.
4. Develop automatic script generating systems. For example, you can populate an array with mouse locations as the offline version of your application runs instead of typing them in by hand.
5. Understand how you can extend objects using prototype (in Flash MX) and class files (in Flash MX 2004).
6. Do what your mom would say and comment your code. Most often it helps you read your own code months down the road—though it can also help others in your workgroup.
7. Strunk and White said (in *The Elements of Style*) that "I used the toilet" is preferable to "I utilized the facilities". Similarly, when coding don't look for eloquent or overly fancy solutions—look for the easiest clearest way to script something. Here are two pieces of code that do the same thing—see if you can tell which is easier to read:

   cryptic:
   ```
   score=(score!=undefined)?((score+questionValue)*cheat):0;
   ```

   lucent, albeit longer:
   ```
   if(score==undefined){
      score=0;
      return;
   }

   score=score+questionValue;

   if(cheat==true){
      score=0;
   }
   ```

8. Develop a good naming style for your variables, functions, and classes. Here are a few general guidelines:
   --functions that return values should start "get" (as in `getScore()`).
   --functions triggered directly from components can start with "do" (as in `doSubmit()`).
   --variables should be short but descriptive. They're usually equivalent property names and, as such, should be nouns.
   --use prefixes to identify variable types (like `g` for globals as in `gScore`).
9. Put `var` before the first use of any variable in order to make it behave as a local variable. That way it won't linger in memory when you're done using it.
10. Use a monospaced font for the ActionScript panel. This way multiple lines of similar code will align perfectly. Andale Mono is a particularly good choice because Os look different than 0s.
11. Put `owner=this` in the first frame and then use `owner` in place of `_root`
12. Sudo code even if you can't spell pseudo.

Quality Assurance

Even the least perceptive user can tell the difference between an amateur and professional application. Sometimes it's the subtle—almost subconscious—enhancements that make the difference between good and great.

1. For video, use Sorenson Squeeze for the best quality/filesize tradeoff.
2. Place raster graphics in whole-number pixel locations. To help, Flash MX has the option View>Snap to Pixels. (In MX2004 it's under View>Snapping>Snap to Pixels.)
3. Do a save-as periodically and keep copies of old revisions (that is, don't just save over old files).
4. Backup revisions to a separate computer, webspace, or removable media.
5. Log changes as you make them so that you can associate the changes made with each revision. If you have recover from a loss or other problem you will know the features to re-add. Redoing work is easy when you have a list—plus it can help you validate your invoices:
   ```
   added back button
   changed blip sound
   new background graphic added
   -------above changes in 20-feb-11am.zip
   removed close up button
   fixed blinking bug
   -------above changes in 20-feb-1pm.zip
   ```
6. Develop proofing systems using scripts instead of manual labor. For example, you could write a script that steps through each image in your application (so someone can view it) instead of expecting someone to step through each one by hand (and accidentally skipping some).
7. Import uncompressed sounds and images and use Flash's compression settings. If you do import compressed media (JPGs or MP3s) don't recompress.

Tips and Tricks

While these tips can wall into any number of the other categories, they're all sneaky ways of tricking Flash into doing something for you.

1. Keep track of time spent on every project even personal or non-paying jobs. This way you'll become better at estimating the time it takes to complete a project.
2. Participate in local user groups. This is invaluable for both finding work and giving you a chance to carve a niche for yourself. Also, online listservs are very useful.
3. Although Flash imposes a limit of 8 simultaneous sounds, that's stereo sounds. This means you can actually have 16 tracks of mono sound (just separate sounds into the left and right channel).
4. Because it's impossible to really learn everything, learn techniques to assess new technologies without investing too much time. For example, don't really *read* everything—but learn accurate ways to skim read.
5. Find your own tips! The process of collating this list involved looking at my own work style and talking to many others. By just analyzing the way I do things I've become better at applying my own tips.