

Flash Internals

*Session Notes for the FlashForward Austin '06 Conference
Edwin van Rijkom*

1 Introduction

(These notes are work in progress. For the latest version of this document, please refer to <http://www.vanrijkom.org/ffa/notes.pdf>)

Flash files cannot be played back without the help of the Flash player. This makes sense from the perspective of Shockwave Flash Files (SWFs) being media files: we don't expect MPEG or JPG files to be self-contained programs that are able of showing their content without the help of a play back program either. Instead we are used to having Media Player playing back movies and Preview opening images. The Flash player playing back Flash movies follows naturally.

Traditionally though, we expect a different behavior from applications. After installing a RSS viewer for example, it should be able to run stand-alone on clicking its main icon. The odd thing is though, that nowadays Flash movies really are applications, as a result of the importance and power ActionScript having evolved so dramatically over the past years. Yet, the play-back metaphor has not changed. This duplicity raises some question about what exactly happens when a SWF gets displayed: are we running a program or are we playing back a movie? This session is about the answer to that question, providing some more insight into what exactly is stored in SWFs, and how the Flash player interprets them.

2 .FLA Files

We are all familiar with the Flash authoring tool storing projects in .FLA files. At first glance, it is a bit odd though, that apparently there's two formats for storing the same information, namely Flash content. However, the way in which the Flash authoring tool represents Flash movies does not match the final representation of the movie once converted to a SWF. For example, a SWF file does not contain the layers you can define in the authoring tool to organize the canvas. In short, the FLA file contains a lot of meta-information, that is not strictly necessary for correct playback but awfully handy while still doing work on a project. This also explains why

doing an import of a SWF file into the Flash authoring tool results in a mangled representation.

Clearly, the motivation for this scheme is optimization. By omitting meta information file size gets diminished whereas performance is increased.

3 Shockwave Flash Files, Part I

SWFs are binary. Opening them with a text editor does not give any human readable information.

All SWFs start with a header, that stores important information for the movie as a whole, such as the Flash player version required for playback, the size of its canvas, the desired playback frame rate and the size of the SWF as a whole.

After the header follows a series of tags. Tags are records. Each record starts with a number identifying the tag and is followed by a number indicating the length of the record specific data.

Record specific data is stored in a highly optimized fashion. Whereas typical binary formats use the byte (8 bits) as their atomic unit, the SWF format allows numbers to be stored with precisely the number of bits strictly required to represent the value at hand. Although this adds a little administrative overhead for registering how many bits to expect for a certain number, it nets quite a bit of file size savings when for example working with series of floating point numbers. Series of floating point numbers being what is typically required for storing vector graphics, this scheme attributes greatly to Flash being the efficient file format we know it to be.

Since Flash content can be streamed, the tags in a SWF file are organized sequentially, per frame. A frame is build from various tags defining visual and auditive content, as well as tags defining actions. A series of tags constituting a frame is always followed by a *ShowFrame* tag. After that, the next frame gets defined, and so on, until the end of the movie is reached.

Visual and auditive content is written to SWF in a fairly straightforward fashion: vectors get their coordinates and fill/line styles and coordinates registered, bitmaps and audio get stored in the native format you select from the authoring tool's symbol properties. ActionScript is a different cup of tea, though. Apart from literals (strings and numbers), and all class-, function- and variable names, all code is transferred into a lower level format.

4 The Flash Player

On the receiving end of the SWF format, is the Flash player. The different versions of the player (stand-alone and the browser plug-in) behave differently in terms of getting their content loaded. The browser plug-in uses

its host to retrieve data from the web, whereas the stand-alone player contains its own routines for doing so. In the latter case, the SWF is often already packed with the with the player itself - so obviously this requires a whole different loading mechanism than is present in the plug-in version of the player. Still, a stand-alone player is capable of downloading additional content, and for that it has its own build-in routines.

Apart from these minor differences and a few extra features and limitations found in the stand-alone version, both players interpret SWFs in a similar fashion.

Since the SWF format is streaming, the player will start playing back content upfront receiving the entire Flash movie. The player will initially keep loading SWF data until it has successfully loaded all the files header, and the visual, auditive an ActionScript data for the first frame.

Once that part has been received, it will first render the visuals and start playing the auditive content. Unless the SWF that is being played back has a pre-loader installed, the player will continue to repeat this process for as long as there is data. In case the next frame's loading time exceeds the amount of time that the current frame should be shown, the movie will be delayed until the next frame is fully loaded.

In case the frames load faster than the movie is set to play, than the player will read-ahead, trying to slurp in as many frames up-front as possible.

So far, this does not sound very much different from how you view an MPEG movie using the Media Player, as mentioned earlier in the introduction text. What is interesting though, is that besides the player only being just that, it is also a virtual machine for the ActionScript code that is part of the movie that is being loaded.

5 ActionScript Virtual Machine (AVM)

In general, applications get compiled to a target platform. What happens is, that a compiler translates higher-level code into lower-level instructions that make sense to an Operating System's (OS) Advanced Programming Interface (API) and, at the lowest level, a specific micro-processor. As a result, most applications we know are either available for only one specific OS - or there are different versions of the application available per OS.

When working with Flash, we are not troubled by all that, but still we are successfully deploying applications over multiple OSs!

The Flash player is helping is out here by, besides being a playback device, also acting as an intermediate between multiple platforms (Windows, OSX, Linux, Mobile Devices, etc.) and the ActionScript code we have defined in our SWFs. This is done by Flash running a so called *Virtual*

Machine. Those familiar with Java will know what a Virtual Machine is, but for others it might require some explaining:

A virtual machine is a piece of software that emulates a computer that has a certain instruction set. ActionScript code contained in our SWFs is executed against that emulated computer. Since the front-end of the AVM is the same on all OS's (it is software, so the Adobe engineers could code it to be like that), ActionScript can be played back on any machine that has a Flash player available. What's going on at the back-end of the AVM, is that all instructions get translated into OS specific instructions.

The AVM has a fixed instruction set, that is, fixed per Flash version. Over time the instruction set got bigger and bigger, but remained backwards compatible. With Flash 9 the Adobe engineers decided it was time for a whole new, more efficient instruction set and the broke with the old one. This new virtual machine is called AVM2. In order to still stay compatible with pre v.9 SWFs, Flash player 9 will include both the Flash 8 AVM (AVM1) as well as the new AVM (AVM2). Knowing that virtual machines are really emulated computers clarifies why it is quite impossible to mix code targeted for AVM1 with code targeted for AVM2: the code is really running on two different (virtual) machines.

6 Shockwave Flash Files, Part II

As mentioned earlier, ActionScript code, as defined in your .as files, is not stored literally to SWF. Instead, on pressing CTRL-ENTER, all code gets translated into so called *byte code*. Byte code means a series of AMV instructions. The transformation of human readable (ActionScript) code into byte code is called *compilation*.

For the remainder of this presentation, we have explored how several common ActionScript code snippets translate into byte code. The sample scripts and their accompanying bytecode are available for download at: <http://www.vanrijkom.org/ffa/samples.zip>