Swiffotron overview

# Summary

This document serves as a simple high level overview of the Swiffotron projects in the SWF processing solution and how they relate to one another.

If you want more detail on the components, refer to the individual documentation for each project.



# SWFModeller

SWFModeller provides functionality that allows SWF files to be created, loaded, modified and saved. Using SWFModeller classes you can create and piece together a new SWF file entirely in memory and save it to a new SWF file. Alternatively you can load an existing SWF file, modify it and re-save it as a new SWF.

SWFModeller makes some attempt to present itself as a programmatic version of the flash IDE. Creating a new SWF object will create a new SWF with the same defaults as a new .fla document created in the IDE. It also attempts to present features available in the IDE that are not available in SWF files, e.g. the notion of layers and frame labels.

It supports only flash player 9 format files. In theory version 10 files should be ok too but no effort has been made to test this. This means it supports only Actionscript 3. Prior versions cannot be created, and will be rejected by the SWFModeller parser.

Also supported is manipulation of bytecode (ABC bytecode). Classes and methods can be inspected and disassembled. Bodies of code from different files can also be merged together into one new body of code. Constant values such as strings and numbers used in the code can be found and altered.

As well as providing IDE-like functionality, SWFModeller aims to provide features not possible in the flash IDE. Using SWFModeller, a loaded SWF file can be treated as a MovieClip and inserted into another SWF as though it was a library item. Unlike the IDE, this will result in the inserted SWF retaining its Actionscript functionality.

# Swiffotron

Swiffotron builds on top of the SWFModeller interface by providing a set of operations and manipulations expressed as XML files. SWF files can be created, loaded and manipulated. They can be tagged with IDs within the XML and reference each other. A Swiffotron job file might declare a SWF which is loaded from an existing file and declares that it should produce a new output SWF. That SWF might reference another SWF within the same XML that is imported as a functioning MovieClip.

As well as an XSD for Swiffotron jobs, the project exposes an XML interface for configuration. This configuration allows you to declare implementation assemblies for caching and persistence. Swiffotron will automatically determine what work needs to be done on a job XML by inspecting the cache via the configured cache assemblies.

In this way Swiffotron can be integrated with different persistence implementations (File system, database..) and cache mechanisms (Memcached, membase...). All that needs to be done on the front end is provide Swiffotron with a job XML file and it will work out the caching itself.

By carefully crafting an XML file that constructs a complex SWF in stages, Swiffotron will be able to cache partial steps along the way to a complete job. E.g. it may be that a weemee could be created in two steps leading to a cache of undressed weemees of different skin tones and facial expressions ready to be completed.

# SwiffotronCLI

This was initially intended to be a simple CLI onto Swiffotron for demo purposes. It accepts configuration and job XML files on the command line and invokes Swiffotron to produce its output. It has however taken on slightly more than that and can be used as a very simple stand-in for MSTest when working on the solution in Visual Studio 2010 express which lacks unit testing functionality and fails to load the test projects within the solution.

# SWFModellerTest and SwiffotronTest

These two projects are how code gets run in the solution. Each contains a set of tests and test files which exercise various aspects of the SWFModeller and Swiffotron functionality.

If a new feature is started, or a bug is discovered, the first step is always to create a test or tests that replicate the problem or test an intended outcome.

Testing is intended to expose the inner workings of the projects rather than simple input/expected output testing. To achieve this, the tested projects are littered with test output that gets logged in the MSTest output folders so that the operation of a failed test run can be seen in as much detail as possible.

This also allows tests to have predicted logs that can be compared automatically to catch subtle changes that may not necessarily be detected in simple ‘The output was produced’ style tests.