Swift Generator

User’s Manual

Rev. 1.12 - August,11,1999

1 Introduction

Swift-Generator is a simple replacement for Macromedia Generator utility. It aims at generating dynamic Flash content.

Swift-Generator can replace:

- Texts (size, alignment, kerning, color, transparency and font),
- Fonts,
- Sounds,
- Images,
- Movie Clips,
- Action parameters (in frames or buttons).

It can be used as a CGI program for HTTP servers like Apache, or just as a separate utility called to produce a new Flash file whenever it needs to be updated (crontab, schedulers, ...)

2 How it works

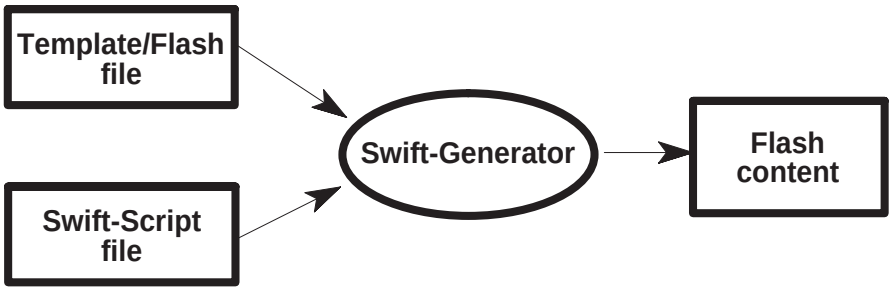
To create dynamic Flash content Swift-Generator needs either a ’Generator Template’ file (produced by Flash 3 Authoring tool from Macromedia) or standard ’Flash’ file.

It is strongly recommend to use ’Generator Template’ files that are much easier to use than standard ’Flash’ files. The section ’Template vs. Flash’ discusses about the differences.

The Template or Flash file can contain variable references that will be replaced when generation is invoked. It just consists in pattern substitutions. Swift-Generator searches for variable references in the template file and replaces them by the content defined in the Swift Script file (.sws).

A Swift-Script file is a set of statements (instructions) to tell Swift-Generator what to do. More sophisticated text replacement can be done with the swift-script text substitution command.

The following diagram shows the concept :

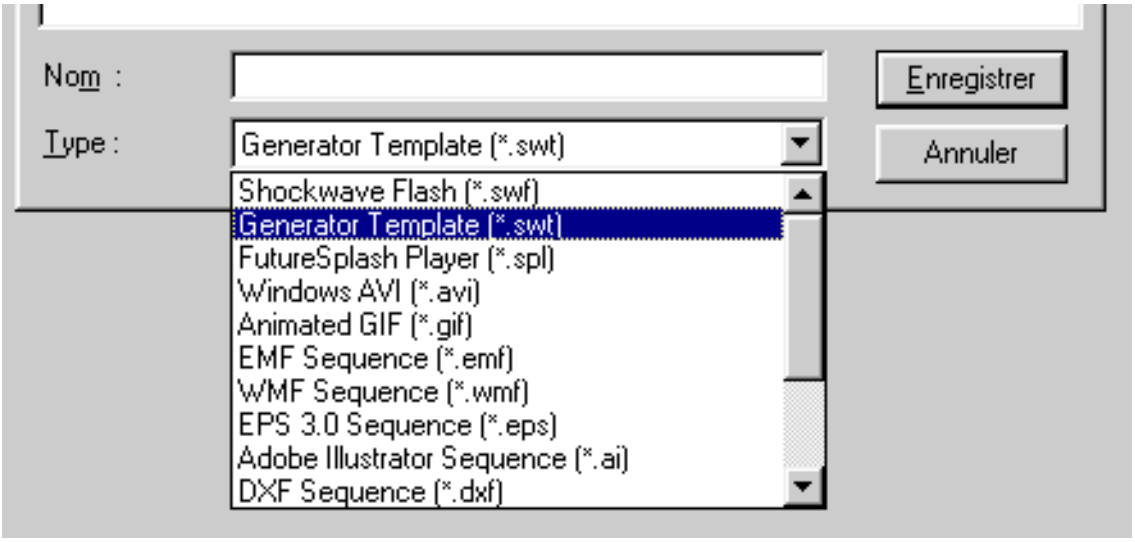


So consider that Template/Flash files and Swift-Script files are the input and the Flash content is the output. Swift-Generator will apply the transformations according to the script instructions.

3 How to create templates

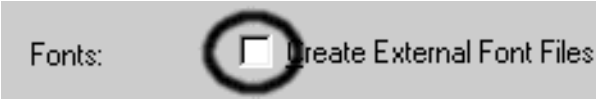
When using Flash 3 Authoring tool, you have the possibility to export your movie in various formats. The most common one is 'Shockwave Flash' format, which produces .swf files.

To save the movie as a template, select the 'Generator Template' format to produce .swt files:



Shortly, the 'Generator Template' file contains more informations than a standard Flash file. Especially the whole fonts definitions.

DO NOT check the 'Create External Font Files' box, the resulting file would not be suitable for Swift-Generator.



If you already have Flash movies, just reload the corresponding FLA file and export it as a ’Generator Template’ file.

4 Using Variables In Flash Design

This section does not explain how to design Flash movies in general. Please refer to Flash 3 user's manual for more informations about how to create Flash movies.

During Flash movie design, you can specify variables references in Text Objects, in Frame's actions and in Button Object's actions.

Variables references are specified using **{name}** notation. This is just normal text expect that the variable reference is put between curly braces. It does not affect the final file (Template or Flash). If you visualize the movie without Swift-Generator processing, you will see the notation. Variables names must consist of alphanumeric characters and/or undescores (0-9, A-Z, a-z, _). The first character cannot be a digit. The variable name is case sensitive.

4.1 Variables in Text Objects

Select the Text tool in Flash (the A in the toolset) and click somewhere in the Stage, type in the following text : Hello {who}. Change font, color or style as necessary.

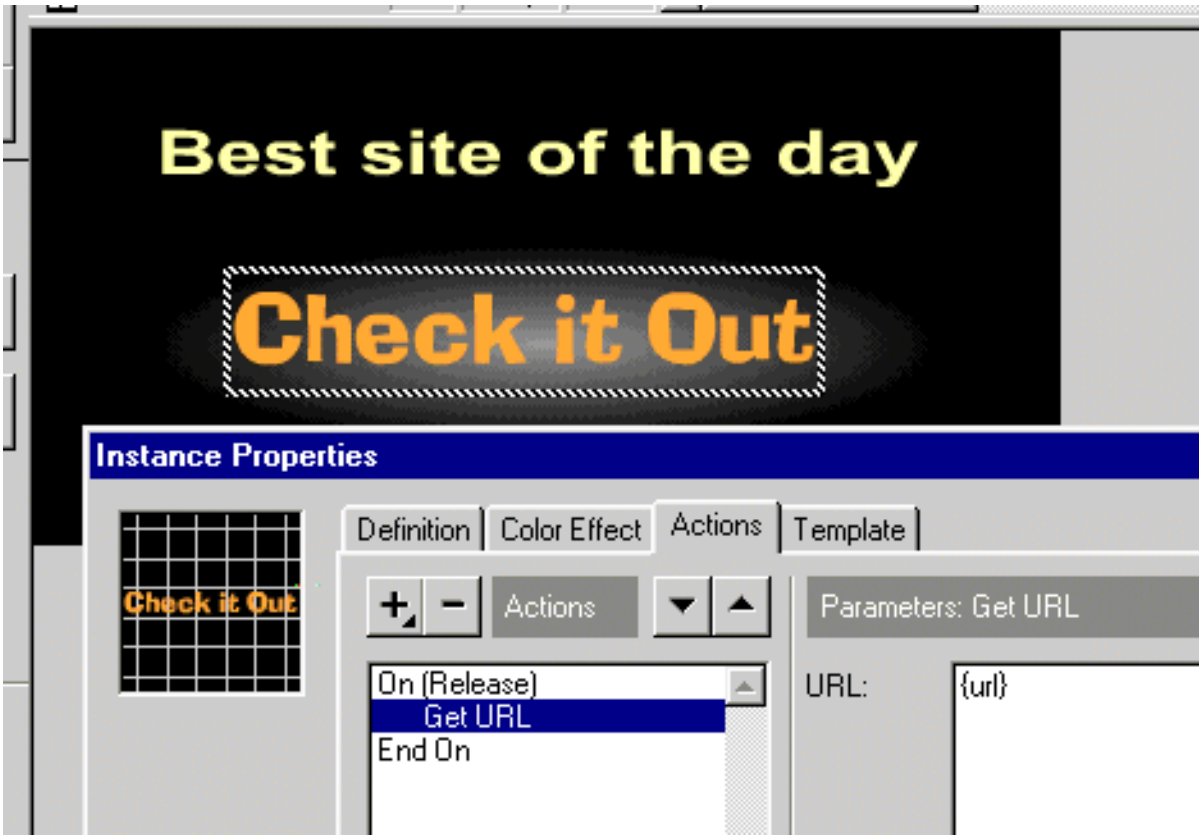
That's all. You will be now able to change the 'who' variable by whatever you want. For instance, the following swift-script statement:

```
SET who "World"
```

will produce Hello World in the final Flash movie. It's quite easy.

4.2 Variables in Action parameters

The following screen shot shows how to specify a variable reference in a button action:



See the {url} reference for the 'Get URL' parameters.

Of course, without generation process, this would cause a bad URL specification for any kind of browser. But if you specify the following statement in a swift-script:

```
SET url "http://www.mysite.net"
```

after a generation process, clicking on the 'Check it out' button will make the browser to load the page of the site 'www.mysite.net'. You can also specify a variable in the target window field.

This can also be done for the following action parameters (Frame or Button):

Action	Parameter(s)
Get URL	URL / Target Window
Goto Frame	Label[1]
FSCommand	Command / Target
Load Movie	URL / Level#
Unload Movie	Level#
Tell Target	Target

[1]This will only work if you have labeled a set of Frames in your movie. Refer to the Flash manual to see how to give a Label to a particuliar frame.

4.3 Limitations

Swift-Generator is able to assign an entire text content (multi-line text) to a variable. But the substitutions of variable references in a Template or Flash file are performed by replacing the content by **only the first line** of the multiline-text.

For instance, if you have specified the variable {foobar} somewhere in your movie, and a swift-script contains the following statement:

```
SET foobar [ cat file.txt ]
```

or, for Windows users:

```
SET foobar [ type file.txt ]
```

although the entire content of the file 'file.txt' will be assigned to foobar, only the first line of text will be produced as a replacement of foobar.

If the file.txt file contains:

```
foo
bar
```

and if you have a Text Object defined as ’Text is {foobar}’, the result would be ’Text is foo’.

If you want to perform a text substitution with multi-line text content you will have to use the SUBSTITUTE TEXT instruction (discribed later).

5 Template vs. Flash

Before starting the swift-sript instructions descriptions, it is important to tell the difference between Template files (.swt) and Flash files (.swf).

A Generator Template file (produced by Flash Authoring tool) contains full font definitions. The consequence is that it is much bigger than a Flash with the same movie content. It is actually produced for Generation tools like 'Macromedia Generator'. So it is not needed to specify True Type Font files to perform substitution text.

So, you should generally use Template files only, unless you only want to perform:

- Font substitutions,
- action parameters substitution,
- sound replacement,
- image replacament.

There is another case in which you might use Flash files: you do not have Flash Authoring tool.

When using Flash files as input for Swift-Generator, you will need True Type Font files to perform text substitution. This means that a complete generation setup implies that you install a swift-script file, a Flash file and all needed True Type Font files. This is bit more complicated than using Template Files.

However, installing True Type Font files may save disk space if you have a large number of Template Files.

If you have Flash 3 (Authoring tool), Generator Template files should always be prefered.

6 Command Line Invocation

Swift-Generator can be invoked on command line using the following ways :

`swift-generator`
will print usage of Swift-Generator (or CGI invocation, see the proper section).

`swift-generator -d file`
will produce a description of font and textual content of a Template file (.swt) or a Flash file (.swf). The output is suitable to create a Swift Script (sws file) to ease the design of a new script.

`swift-generator script.sws`
will produce a new Flash file according to script content.

`swift-generator -f true-type-fonts-directory`
will produce a True Type Fonts map of all existing fonts in the specified directory.

6.1 Template/Flash file dump

The -d option is used to dump the content of a Template/Flash file (.swt and .swf files) to determine the tagId of all fonts and texts defined in the animation. Informations regarding sounds and movie clips instances are also dumped.

The output format is exactly a Swift Script (Sws) syntax, so redirecting this output to a sws file will produce a good start to write a new script.

The resulting Sws just aims at producing the original Flash file content. The original text is human readable.

The output is different wether you dump a Template or a Flash file.

- For Template files, there will be no font definitions statements, just comments to tell the font familly associated with a FONT tagId.
- For Flash files, Swift-Generator will produce the required font definitions. If the TTFPATH variable is set to a correct font path, the script should contain font definitions that point to the correct True Type Font file (ttf), this is not useful for Windows users. If Swift-Generator cannot find the correct True

Type Font file, it produces a script comment that gives the font name, and the font file is replaced by '???.ttf'.

6.2 Generation

Generation can be performed on the command line or when invoked as a CGI program.

In both cases, Swift-Generator needs a reference to a Sws file. On the command line this is performed by specifying the Sws file name as a unique argument. For a CGI invocation the Sws file name is determined by searching the 'sws' parameter name and the associated value (see CGI setup for details).

6.3 Font Map

This only useful if you want to perform Font Substitution or if you process a Flash file as an input.

The -f option is used to scan the specified font directory and to procude a font map that is a sort of alias table. This table is an association table between fonts names (and style) and corresponding ttf files.

If you are happy with the result, it is possible (and also useful) to set the environment variable named TTFPATH to the given directory. This will ease font definitions in the File Dump operation (see below).

Unfortunately, Swift-Generator does not handle all True Type Fonts files, so, this is also a way to determine what font files can be used for successful generations.

7 Swift Script

You must be familiar with some Flash format concepts, before reading the swift script commands descriptions. Don't worry, this is simple.

Regarding the file type, Template or Flash, the internal reference to objects is done by specifying a tagId. This is just a unique identifier number. So Fonts, Texts and Sounds are specified by such numbers. Swift-Generator uses this tagId to specify a Font Definition or Text Substitution. For Sounds you can specify either tagId or the symbol name (the latter is more convenient).

If you plan to use 'variables references' only (the {} notation), you don't have to care about tagIds. But you will miss a powerful part of Swift-Generator.

Thanks to the dump capability of the tool (use of the -d option), you will easily get all needed tagIds.

A Swift Script (Sws) is a list of commands so as to produce a Flash format output (swf). It is composed of a first global settings part, and then, definitions, text-substitutions, sound replacements or movie clip definitions commands.

Here are the available Sws commands. Note that they are case sensitive (they are generally upper case written).

Commands are executed during script parsing. So operations are sequential. Any syntax error causes the execution to stop.

7.1 Comments

`% A lot of comments`

Specifies a comment. After the % sign, any trailing characters up to the next end of line are discarded.

7.2 Global settings

The first settings that should be present at the begining of the script are :

INPUT "file.sw?"
Sets the input Template/Flash file name. If "-" is specified, than standard input is used, allowing piped commands. Note: Swift-Generator determines the file type (Template or Flash) with the file extension (swt or swf respectively).

OUTPUT *[* **-cgi** *]* **"file.swf"**
Sets the output Flash file name. If "-" is specified, than standard output is used. If the '-cgi' option is present, the output is prepended by a HTTP header (as required for CGI programs).

TTFPATH "font directory"
Specifies the True Type Font directory, so subsequent ttf file specifications are shorter. Only useful for Flash Files input or font substitutions.

7.3 Setting and Using variables

Swift-Generator allows the use of variables. Variables can be set in the script, inherited from the environment, or, inherited from CGI parameters. Value of a variable is always considered as a character string. So variables can only be used where strings apply in the script. And don't forget the variable references within the input file (Template of Flash), the script is where you can define the content of variables to perform text substitution.

SET varname value
Sets the variable named 'varname' with the value 'value'. Value can be either a string, a variable reference, a subshell result or a concatenation of values.

A string is specified by characters between double-quotes : ". e.g:

```
SET myvar "the Quick brown fox..."
```

A variable reference is a variable name prefixed by a $ sign. e.g :

```
SET anothervar $myvar
```

This notation can optionally followed by a line range specification. This is convenient when the variable contains multiple lines and only a certain number of consequent lines are desired. The notation is:

```
SET section $fulltext<first_line..last_line>
```

for line range, or

```
SET section $fulltext<line>
```

to extract one specific line.

For example, if the variable *fulltext* contains:

```
Line 1
Line 2
Line 3
Line 4
```

the following statement:

```
SET section $fulltext<2..3>
```

will assign to the variable *section* this content:

```
Line 2
Line 3
```

A subshell result is a shell command line specified between brackets [ ]. e.g:

```
SET exe [ date ]
```

A shell command is: every valid commands that can be typed into a terminal window (Unix) or in a Console Window (or DOS Shell for Windows). Some Windows commands may not work (I don't know why).

All the output of a subshell is assigned to the variable, so you can have an entire multiline text assigned to it. Beware that only the first line will substitute a variable reference in a Template or Flash file. To perform the complete text substitution, use SUBSTITUTE TEXT command (see after).

It is possible to use defined variables in a subshell command. Example (Unix):

```
SET format "+%H:%M"
SET time [ date $format ]
```

Complex subshell commands can use pipes (Unix only) to obtain complex shell commands combinations.

Note: the entire subshell output is assigned to the variable. So it can result in a multi-line text content for commands more complex than 'date'.

For example:

```
SET dircontent [ ls -als ]
```

The variable dircontent will contain the whole listing of the current directory.

Values concatenation is obtained by separating values with a + sign. e.g:

```
SET format "+%H:%M"
SET time "Time is "+[ date $format ]
```

It is not allowed to define a variable which name is the same as an already existing environment variable. Trying to set a HOME variable is forbidden. It does not generate an error, but the variable's value is unchanged.

Using non-defined variables results in an empty string. This also means that if you have 'variable references' in the input file (Template or Flash) the resulting text will be empty after the generation process.

7.4 Font definition

For Flash Files input, fonts definitions must be done before referencing any font in text-substitutions.

For Template Files input, this is only to perform Font substitutions. So you can override a primary font selection done at design level.

DEFINE FONT id "file.ttf"

This defines that the font with tagId id is described in the specified True Type Font file.

The tagId must refer a font in the input Flash file (or Template), otherwise an error occurs. The True Type Font file must also be valid (Swift-Generator can handle it, the file exists, ...)

7.5 Text substitution

This is supposed to be the most interesting command.

```
SUBSTITUTE TEXT id {
<sub-commands>
}
```

The tagId id must be valid, it must be a reference to a Text object in the input file.

The sub-commands allow to redefine new content within a text definition. In a Template/Flash file, a text is composed of text records. A text-record is a list of strings with optional color, font, font height changes. If not specified, color, font and font height default to the first definitions in the original file.

Sub-commands are :

FONT id
: Change current font to Font id. The tagId must refer a predefined font. Subsequent strings will use this font.

HEIGHT height
: Change current font height to height. The font height is specified in points (positive integer decimal value).

COLOR #RRGGBB
: Change current color to RGB definition #RRGGBB. Red, green and blue values and specified with a 2 digit hexadecimal number. Be aware that color transformations can be applied in the Flash animation, so the final effect may differ from the expected setting.

ALPHA value
: Change current transparency value. Value 0 represents a total transpency, value 255 represents total opacity.

VSPACING value
: Change the vertical spacing of multi-line text. Decimal value. Default is 1.0.

KERNING value
: Change text kerning (character spacing). The decimal value represents the scale factor applied to normal kerning. Double character spacing is done with a value of 2.0. Default is 1.0.

STRING *[* **-nocr** *]* **"text"**

Define a text. The text is rendered with the current font, font height and color. Each text definition is followed by a 'carriage return' for rendering, except when the *-nocr* flag is specified.

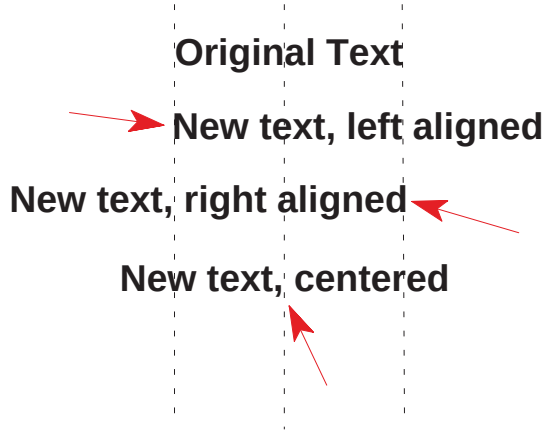
`ALIGN "position"`

Allows to specify the text alignment according to the original text boundary. Possible values for position are *left*, *right*, *center* (no case sensitivity). Default is *left*.

User can define as many new strings as necessary, even if the original Flash file described a one-line text. If the animation depends on text boundary, the resulting text rendering may overlap other graphic objects.

If a font height change is first specified and it is different from the one in the original Flash file, strange upper-left corner text positioning will occur. Swift-Generator recomputes the global text boundary but does not assure that the global text position is the same as in the original Flash file (in this case).

Text alignment works on the original text boundary. The following figure shows how positioning works:



7.6 Sound replacements

Swift-Generator is able to redefine a sound in a movie:

`DEFINE SOUND "symbol name"` *`[`* **`-bits N`** *`]`* **`"sound.wav"`**

or

```
DEFINE SOUND id [ -bits N ] "sound.wav"
```

the second form is generally used for Flash intput files.

The symbol name is the one that Flash 3 uses to specify a sound clip. The optional *-bits* parameter specifies the compression number of bits. Valid numbers are 2,3,4 or 5 (default value is 4).

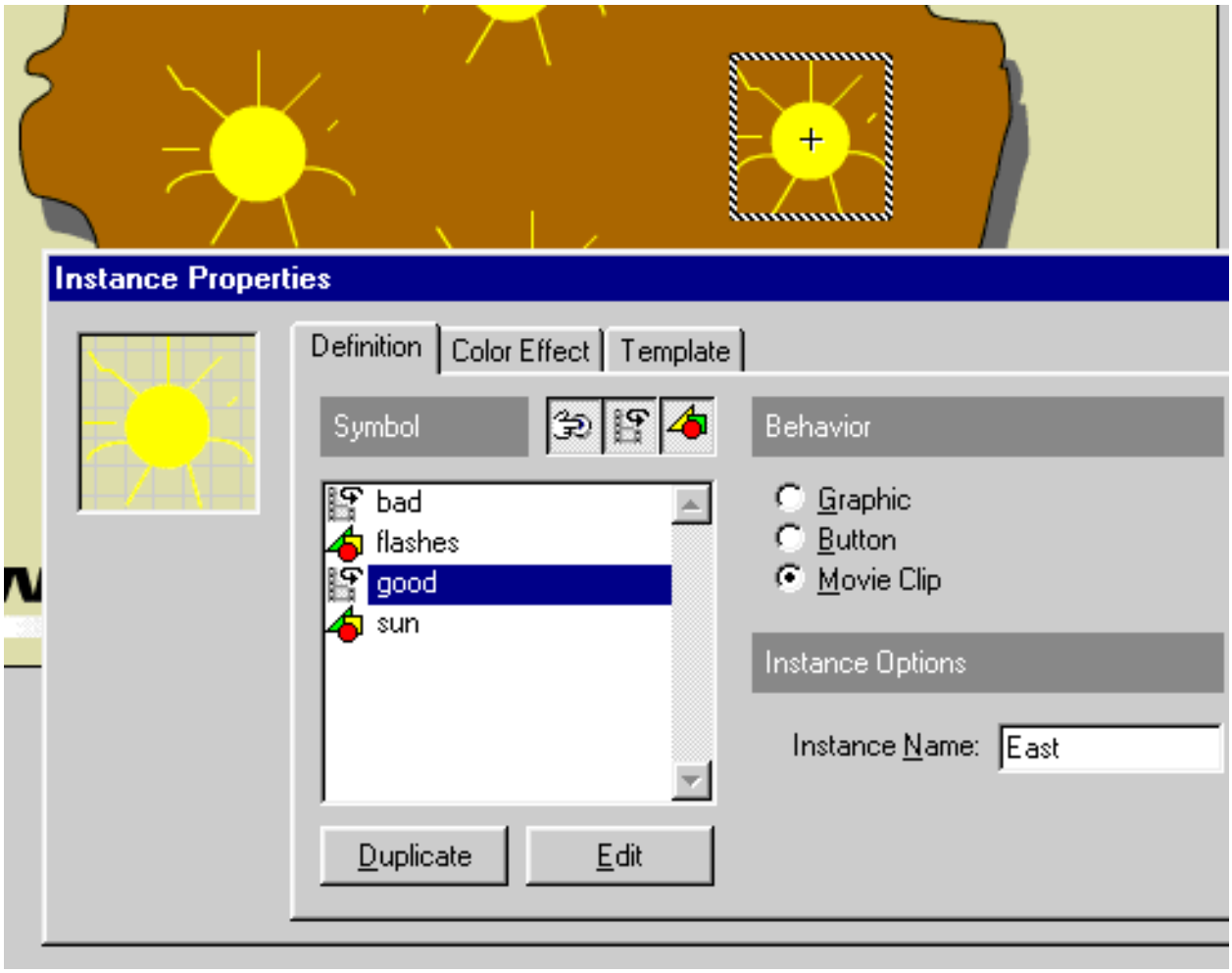
The parameter *"sound.wav"* specifies the file name that contains the new sound. Current limitations are: WAV files only (Microsoft RIFF), 8 or 16 bit PCM data and supported sound rates: 5.5, 11, 22 and 44 kHz. Many tools can perform conversion to these characteristics, if needed. Note that a sound with a rate of 11025 kHz will perfectly work. At script design level, always check the output to see if Swift-Generator has succesfully processed the sound file.

Remember that the sound filename can be replaced by a variable reference or string operations.

7.7 Movie Clip replacements

This functionality allows to redefine the Movie Clip associated to a symbol instance. When a Movie Clip is dragged from the Library window onto the stage, this creates an instance of the Movie Clip.

To perform Movie Clip replacement with Swift-Generator, you must name the instance. This can be done by modifying the Instance Properties of the selected object, then, select the Definition tab, set the instance behavior as a Movie Clip and fill the Instance Name field:



The following statement:

```
DEFINE MOVIE "instance name" "symbol name"
```

will assign the movie clip 'symbol name' to the instance 'instance name'. If the symbol name does not exist, the instance is just not redefined.

7.8 Image replacements

To replace an existing image in a movie, use:

```
DEFINE IMAGE "symbol name" [ -quality N ] "image.jpg"
```

or

DEFINE IMAGE id *[* **-quality N** *]* **"image.jpg"**

the second form is generally used for Flash intput files.

The symbol name is the one that Flash 3 uses to specify a sound clip. The optional *-quality* parameter specifies the compressed image quality. Valid values are between 10 and 100 (in percent). The higher the value is, the higher the image quality will be, but bigger will also be the resulting Flash data. The default value is 50 (%).

The parameter *"image.jpg"* specifies the JPEG file name that contains the new image. The image size may defer from the original one. Flash rendering repeats the last line and column pixels to fill the remaining space if the image is smaller than the original one. So smaller images should have a uniform colored border. This has nothing to do with Swift-Generator, it is a Flash rendering strategy.

Remember that the image file name can be specified with a variable or a string operation.

7.9 Referers list

This allows to specify a list of Internet domains that are allowed to point to your Swift-Generator application. This feature often exists in CGI programs (or scripts). It is provided to add some use-protection of your work.

REFERERS { "first.domaine.com" "second.domain.com" "localhost" }
It consists of a simple list of domains (character strings) within curly braces. The "localhost" is necessary in many cases, check all your host aliases.

If the referer is in the list Swift-Generator performs the job. Otherwise, it generates an HTML message: Access denied. This message will never be viewed if the CGI call is done from an EMBED or OBJECT tag. In these cases no movie will be displayed.

If the list is omitted, all sites can point to your CGI. This is a per-script list, it does not constrain all existing scripts on your site.

If you don’t know what a referer is, read your web server documentation or the HTTP specification (RFC 1945).

Note: this feature only affects on-line generation. Off-line generations are always performed, regardless of this list definition.

8 Tutorial

Here is the tutorial section, it will detail some basic examples.

- The first one will describe the use of Template Files, this should be the most common case.
- The second one is a Flash File input example, this can be a very useful example to know how Font description works if you want to use this functionality in Template Files.
- The third example will show a complete example with URL substitution, multiline text substitution, and use of CGI parameters.

The command swift-generator is always used to create a first script file and for testing purpose. The development cycle shoud look like:

0) create a movie with Flash 3, put variable references where they are needed,
1) dump the input file to get a basic swift script file,
2) modify the script to obtain the desired effect,
3) generate a Flash file (usually export.swf),
4) view the result with a standalone player (or with the browser).
5) Iterate over step 2). *ad lib*.

At this level you should be able to install and actually test the result on a web-server.

If you make changes into the movie, you may have to dump the file again and check if the tagIds have not changed. This may happen if you make too many changes. Of course this is only required when your script deals with tagIds.

For Windows users: the script design stage requires to type commands. So you will need to open a Console Window (or DOS Shell) to perform the very first steps. Always use NotePad for script editing (avoid the 'edit' utility). Check the default generated file (export.swf) with the standalone Flash Player (unless you want to check URL substitutions or similar actions). The following examples use a Unix prompt (#), just mentally replace it by **C:\Swift>** (depending on your working directory).

8.1 Template File Example

We assume that we have a simple movie that is supposed to present the "Best Site of The Day". Here is a snapshot:

You can see that there is a text a the bottom of the movie which is **{site_name}**. This is a first variable reference. There is actually another one, named **{url}**, that has been assigned to the URL parameter of a GetURL action of the "Check it out" button.

Remember to save the movie with: 'Export movie' and choose 'Generator Template' format.

Now we dump the content of the swt file to get a swift script:

```
# swift-generator -d best_site.swt > best_site.sws
```

We get the following script:

```
% Script template from Template file best_site.swt

INPUT "best_site.swt"

% Output for testing
OUTPUT "export.swf"
% Output for CGI
%OUTPUT -cgi "-"

% Font definitions
% FONT 2  is  Arial Bold
```

```
% FONT 4  is  TimeScrDMed
% FONT 6  is  Ad Lib

SUBSTITUTE TEXT 3 {
        FONT 2 HEIGHT 24 COLOR #ffff99
        STRING "Best site of the day"
}
SUBSTITUTE TEXT 5 {
        FONT 4 HEIGHT 18 COLOR #cccccc
        STRING "{site_name}"
}
SUBSTITUTE TEXT 7 {
        FONT 6 HEIGHT 36 COLOR #ff9933
        STRING "Check it Out"
}
SUBSTITUTE TEXT 8 {
        FONT 6 HEIGHT 36 COLOR #ff9933
        STRING "Go"
}
```

We do not want to use the SUBSTITUTE TEXT command, since we just want the variable **site_name** to contain a single text line and we do not want to change either font or text style (color and height).

So the script can be drastically reduced to:

```
INPUT "best_site.swt"

OUTPUT "export.swf"

SET url "http://www.a_good_site.net"
SET site_name "A Good Site"
```

These actions just assign *"http://www.a_good_site.net"* to the variable **url**. And the string "A Good Site" to the variable **site_name**.

The ouput file is "export.swf", so we can easily test the result of a generation:

```
# swift-generator best_site.sws
```

We can now test the export.swf movie:

If we click on the button we'll get to the home page of "A Good Site".

Well, this is not a very demonstrative dynamic content example. In fact, we could just have modified the movie to get the same result.

But imagine now, that the file "best_site.txt" contains the name of the best site of the day and the url of this site (in two separate lines).

We can modify the script so as to assign **site_name** and **url** variables to their respective value:

```
INPUT "best_site.swt"

OUTPUT "export.swf"

SET data [ cat best_site.txt ]
SET url $data<1>
SET site_name $data<2>
```

So, each time you want to update the movie with a new Best Site of the Day, you just have to change the content of the file. The content may be update by the result of a poll, a site survey, or whatever. So your movie will be automatically updated.

The file best_site.txt can look like:

```
A Good Site
http://www.a_good_site.net
```

Note that the output is always the file export.swf and that swift-generator invocation is done manually here. See "CGI Setup" section to know how to get an actual dynamic web page.

With CGI parameters we can have a shorter script file:

```
INPUT "best_site.swt"
OUTPUT -cgi "-"
```

There are no more variable assignments since they are done through CGI parameters. So the following URL will produce almost the same result as the previous scripts:

```
http://host/swift-generator.cgi?sws=best_site.sws&site_name=foo&url=bar
```

The site name will be "foo" and the url will be "bar". (Ok, the url is not valid, but this makes the example just more readable.)

A swift script cannot be shorter !

8.2 Flash File Example

Prior to writing a new script, let's assume that we have, an idea of what we want to do, the corresponding flash file and a path that leads to True Type Font files.

The idea : we want to display the current local time.

The True Type Font path is /win/winnt/Fonts.

The Flash file is linux.swf.

Let's check what Swift-Generator is able to handle in the font directory :

```
# swift-generator -f /win/winnt/Fonts
wingding.ttf     ->  Wingdings
timesi.ttf       ->  Times New Roman Italic
timesbi.ttf      ->  Times New Roman Bold Italic
timesbd.ttf      ->  Times New Roman Bold
times.ttf        ->  Times New Roman
symbol.ttf       ->  Symbol
marlett.ttf      rejected.
lucon.ttf        ->  Lucida Console
l_10646.ttf      ->  Lucida Sans Unicode
couri.ttf        ->  Courier New Italic
courbi.ttf       ->  Courier New Bold Italic
courbd.ttf       ->  Courier New Bold
cour.ttf         ->  Courier New
ariali.ttf       ->  Arial Italic
arialbi.ttf      ->  Arial Bold Italic
arialbd.ttf      ->  Arial Bold
arial.ttf        ->  Arial
```

We can see that the marlett.ttf file is not suitable for Swift-Generator. Other fonts are ok. It is now correct to set TTFPATH variable (Unix):

```
# export TTFPATH=/win/winnt/Fonts
```

Let's get a script skeleton by dumping the linux.swf content :

```
# swift-generator linux.swf > linux.sws
# cat linux.sws
% Template for file linux.swf

INPUT "linux.swf"
OUTPUT "export.swf"
TTFPATH "/win/winnt/Fonts"

% Font definitions
% Arial Bold
DEFINE FONT 2 "arialbd.ttf"
% Arial Italic
```

```
DEFINE FONT 8 "ariali.ttf"

SUBSTITUTE TEXT 3 {
        FONT 2 HEIGHT 20 COLOR #000000
        STRING "for"
}
SUBSTITUTE TEXT 4 {
        FONT 2 HEIGHT 48 COLOR #000000
        STRING "Flash"
}
SUBSTITUTE TEXT 5 {
        FONT 2 HEIGHT 48 COLOR #000000
        STRING "Linux"
}
SUBSTITUTE TEXT 9 {
        FONT 8 HEIGHT 14 COLOR #ffffff
        STRING "by Olivier Debon"
}
```

We want to replace the text "Flash" by "Time", the text "for" by "is" and finally the text "Linux" by the current time.

So we do not need to substitute Text 9, therefore Font 8 definition is useless.

So the resulting script should look like :

```
% Template for file linux.swf

INPUT "linux.swf"
OUTPUT "export.swf"

% Font definitions
TTFPATH "/win/winnt/Fonts"

% Arial Bold
DEFINE FONT 2 "arialbd.ttf"

SUBSTITUTE TEXT 3 {
        STRING "is"
}
SUBSTITUTE TEXT 4 {
        STRING "Time"
}
SUBSTITUTE TEXT 5 {
        STRING $time
}
```

The variable "time" must be defined. We choose to get the current time by calling the ’date’ command with appropriate format so as to get hours and minutes.

Let’s insert the following line (somewhere before text 5 substitution) :

```
SET time [ date ’+%H:%M’ ]
```

or for Windows NT (not 95) users:

```
SET time [ time /t ]
```

Now, we run the generator :

```
# swift-generator linux.sws
Parsing linux.sws
Exe = ’15:00’
Processing file linux.swf
        Define Font 2 -> arialbd.ttf
        Substitute Text 3 :
         Font 2 Height 20 Color 000000
         String ’is’
        Substitute Text 4 :
         Font 2 Height 48 Color 000000
         String ’Time’
        Substitute Text 5 :
         Font 2 Height 48 Color 000000
         String ’15:00’
```

At this moment the subshell command returned the string ’15:00’, then was assigned to variable ’time’.

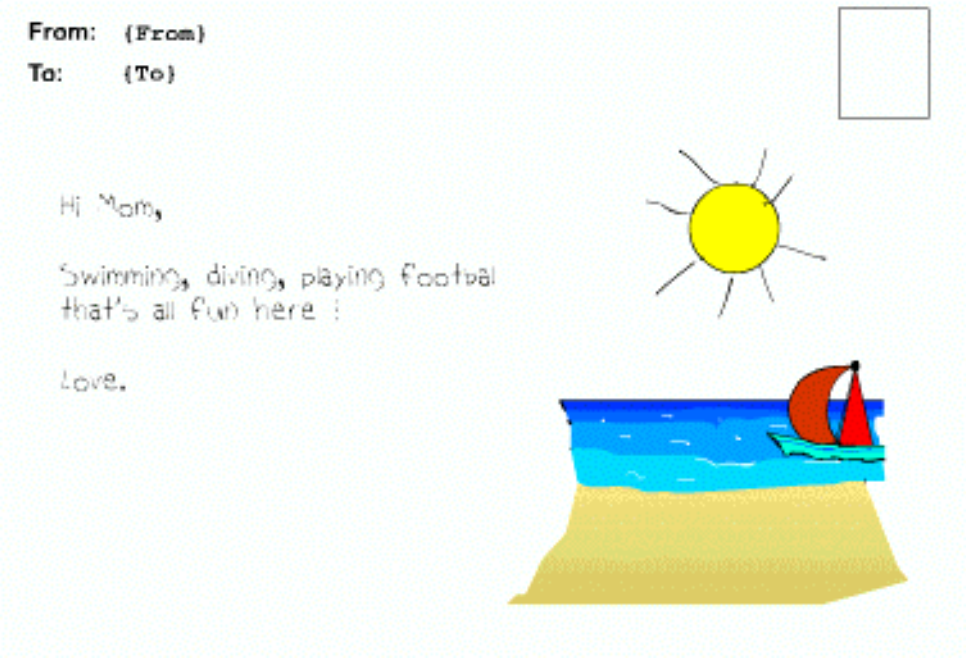
We can visualize the resulting flash file named "export.swf":

```
# netscape export.swf
```

8.3 A Flash Post Card

This example aims at generating a Flash Post Card like some sites propose to send an Electronic Post Card to a friend, for birthday, Christmas and all kind of such opportunities.

The following movie is the original PostCard:



The top fields should contain the name of the sender and the name of the receiver. The sender's name is also a button that the receiver can click to reply to the sender. The URL parameter of the GetURL button action is: *mailto:{email}*. This URL contains a reference to the variable named **email**.

The left part of the PostCard will contain a free message that the sender wants to write.

If we dump the Template file:

```
# swift-generator -d card.swt > card.sws
```

We should get:

```
% Script template from Template file card.swt

INPUT "card.swt"

% Output for testing
OUTPUT "export.swf"
% Output for CGI
%OUTPUT -cgi "-"

% Font definitions
% FONT 2  is  Arial Bold
% FONT 5  is  Courier New Bold
% FONT 8  is  Courier New
% FONT 14  is  Handwriting Bold

SUBSTITUTE TEXT 3 {
        FONT 2 HEIGHT 14 COLOR #000000
        STRING "To:"
}
SUBSTITUTE TEXT 4 {
        FONT 2 HEIGHT 14 COLOR #000000
        STRING "From:"
}
SUBSTITUTE TEXT 6 {
        FONT 5 HEIGHT 14 COLOR #000000
        STRING "{From}"
}
SUBSTITUTE TEXT 11 {
        FONT 5 HEIGHT 14 COLOR #000000
        STRING "{To}"
}
SUBSTITUTE TEXT 15 {
        FONT 14 HEIGHT 18 COLOR #000000
        STRING "Hi Mom,"
        FONT 14 HEIGHT 18 COLOR #000000
        STRING "Swimming, diving, playing footbal"
        FONT 14 HEIGHT 18 COLOR #000000
        STRING "that's all fun here !"
        FONT 14 HEIGHT 18 COLOR #000000
        STRING "Love."
}
```

Our goal is to set the **From**, **To** and **email** variables, as well as to substitute TEXT 15.

Let's modify this script:

```
INPUT "card.swt"
OUTPUT -cgi "-"

SUBSTITUTE TEXT 15 {
        STRING $Message
}
```

Once again it has been shortened because we assume that **From**, **To**, **email** and **Message** variables will be set through CGI parameters.

So we need a standard HTML page to allow the user to enter the data, here it is:

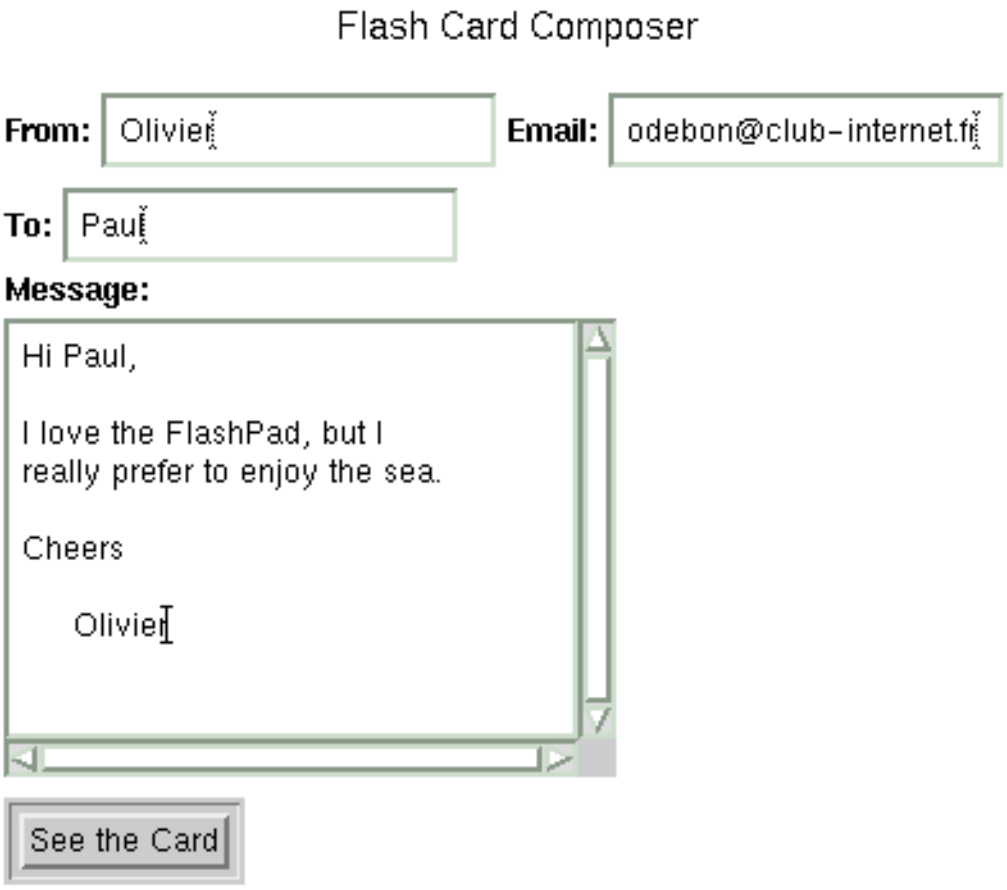
```
<HTML>
<BODY>
<CENTER>Flash Card Composer</CENTER>

<FORM ACTION="swift-generator.cgi" METHOD="POST">
<INPUT TYPE="hidden" NAME="sws" VALUE="card.sws">
From:
<INPUT TYPE="text" NAME="From">
Email:
<INPUT TYPE="text" NAME="email">
<BR>
To:
<INPUT TYPE="text" NAME="To">
<BR>
Message:
<BR>
<TEXTAREA NAME="Message" ROWS=10 COLS=30>
Hi,
</TEXTAREA>
<BR>
<INPUT TYPE="submit" VALUE="See the Card">
</FORM>

</BODY>
</HTML>
```

When the user clicks on the submition button, it results in a call to swift-generator as a CGI program receiving the expected CGI parameters. Please note, that the hidden parameter is required to tell swift-generator what script it must use.

Let’s try:



And if I click on the "See the Card" button I get:

Then, it is possible to send the resulting Flash file to Paul. When Paul receives it, he can reply by clicking on "Olivier" this will pop up a mailer window with the already set To: field.

It can be really improved. But it shows how simple it is to use Swift-Generator.

9 Server-side CGI setup

As normally expected, one of the key feature of Swift-Generator is that it can behave like a CGI program to actually generate dynamic content.

Note: If you do not administrate a Web Server, your are unlikely to be able to use Swift-Generator on the server side (where you pages are actually hosted).

If you depend on an ISP, you are generally allowed to upload CGI Perl scripts only. Some ISPs allow to upload executables on the server. In this case, you must know the OS type in order to upload the proper binary. A nice survey site provides a way to know what a site is running: www.netcraft.com. It maintains a whole Internet Web Servers survey. Available Swift-Generator binaries should cover over 80% of the servers OS types. Solaris should be the prefered OS for ISPs.

Windows users can easily test their applications by installing a local Web Server. Apache is one of the available servers for Windows, but you can find numerous servers to run on Windows 95 or Windows Workstation.

This section will take Apache as a server example to explain how to setup Swift-Generate as a CGI program. For other Web Servers refer to their respective documentations.

Like all CGI programs the server must be configured to allow Swift-Generator to be run. The simple way to do that should be to copy the executable into the cgi-bin directory. As it can be a good way to do it, it should be recalled that a CGI program is always run in the directory where it stands. But the cgi-bin directory is definitely **not** the right place to put swift scripts, HTML pages and any other data files or subshells.

The very best way to properly install Flash dynamic content stuff is to prepare a whole separate directory. Let's call this directory '*generator*'. To make swift-generator a 'clear' CGI program, copy the executable into the *generator* directory and rename it *'swift-generator.cgi'*. This means that the server has been

configured to understand that every .cgi files are supposed to be CGI programs (when they are outside of the cgi-bin directory).

Using Apache, this can be done by inserting the following directive in the *srm.conf* file:

```
AddHandler cgi-script .cgi
```

The generator directory must also be declared to the server so that is contains CGI programs. Put the appropriate directive in the *access.conf* file:

```
<Directory /home/olivier/public_html/generator>
AllowOverride None
Options ExecCGI
</Directory>
```

(Only the ExecCGI option is relevant here.)

When you change the server configuration it must usually be restarted.

In this example the *generator* directory is located in the *public_html* directory of olivier's account. The *public_html* directory is where the server looks up when the URL http://localhost/~user is requested.

Now the generator directory content should look like:

```
drwxr-xr-x   6 olivier  users      1024 Mar  6 13:01 .
drwxr-xr-x   6 olivier  users      1024 Mar  3 23:31 ..
-rwxr-xr-x   1 olivier  users    242448 Mar  7 15:52 swift-generator.cgi
```

Swift-Generator can be invoked as a CGI program with no arguments to allow a quick test. Try the following URL in your browser:

```
http://localhost/~olivier/generator/swift-generator.cgi
```

If everything is correctly configured the following page comes up:

Swift Generator
Version ?.? Copyright 1999
Olivier Debon

If it does not, the server configuration may be wrong. Also check file permissions, this might be a cause of errors.

After manual testing of the export.swf file, the swift script must be modified so the OUTPUT command should look like:

```
OUTPUT -cgi "-"
```

Most of the time, this modification is forgotten and therefore produces an error.

The CGI parameter **sws** defines the Swift-Generator script file name. So when specifying the URL do not forget this parameter (see previous examples).

Swift-Generator will process if the request method is either GET or POST and if it can find the **sws** parameter properly set. It then opens the file for processing and generating the new Flash data that are written to the standard output.

As already mentioned, variables can be set through the URL. Any other parameters will be initialized as variables and set to the corresponding value. This value is not interpreted, so it is unuseful to set a parameter with subshell commands (for security reason).

Like other variables, only variable names that are not defined in the environment variables are correctly assigned.

Complex pages can be created with JavaScript and CGI combinations.

10 Problems

There are some known issues regarding specific products use. They are described here, and workarounds are proposed.

10.1 Mac Issues

Some Templates or Flash files generated on MacOS contains bad informations about the used fonts in the animations. Resulting animations after generation may show very widely spaced characters or bad characters. Is it recommended to generate Flash files on Windows.

10.2 Internet Explorer issues

Microsoft Internet Explorer (IE) has a bug when a full-window or full-frame CGI produced multimedia object visualisation is attempted. Sometimes, a direct call to Swift-Generator (in the location field for example) may cause IE to hang. Analysis has been done on server side, it shows that IE made two or three requests in a row with partial CGI parameters transmissions in the last requests. This behavior makes IE to wait for an unexpected transmission completion, although all data have been actually sent.

IE works fine with Flash files when they are directly loaded or when they are properly encapsulated by OBJECT tags. Netscape has absolutely no problem with this. And it is NOT a Swift-Generator problem.

If you want to display a movie in full-window or full-frame, here is a simple perl script that generates the correct HTML output. It works for all browsers. Just replace the usual 'swift-generator.cgi' call by 'embed.cgi'. It's a simple wrapper.

```
#!/usr/bin/perl
print "Content-Type: text/html\n\n";

if ("$ENV{REQUEST_METHOD}" eq 'POST') {
        $query = <>;
        $query =~ s/(\r|\n)//g;
} else {
        $query = $ENV{QUERY_STRING};
}

print <<END_OF_HTML;
<HTML>
<BODY BGCOLOR=#ffffff>
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" WIDTH=100%
height=100%>
<PARAM NAME="MOVIE" VALUE="swift-generator.cgi?$query">
<PARAM NAME="QUALITY" VALUE="HIGH">
<EMBED SRC="swift-generator.cgi?$query" WIDTH=100% HEIGHT=100%
TYPE="application/x-shockwave-flash">
</EMBED></OBJECT>
</BODY>
</HTML>
END_OF_HTML
```

You may have to modify some parameters according to your system and also what you want to generate exactly. This script can be derived for presentation improvement.

10.2 Web servers issues

All possible web server products have not been tested. But some products are really not CGI1.1 compliant. Adaptations have been made to make Swift-Generator work with some of them (although the original Swift-Generator output was CGI1.1 compliant).

The test page may not appear when swift-generator is invoked without parameters (like Personal Web Server or Xitami). Just add the '?test' parameter to Swift-Generator call and the test page should show up.

Good results are obtained with Apache (Unix or Windows) and Sambar Server (Windows).

Please refer to your server documentation before sending any bug report. Most of the time, server configuration is not correct or file permissions are not properly set (for Unix).

11 Informations

Swift-Generator is a program by Olivier Debon <odebon@club-internet.fr>

It can be used for commercial or non-commercial activities. The free versions are freely redistributable providing the fact that they must come with the SwiftLogo and the README file. Any use of Swift-Generator on a Web site must be reported to the author (see above) and must also contain the provided logo without any modification of color or size. The logo should point to the Official Swift Site (see below).

All trademarks belong to their respective owners.

Official Swift site : http://www.swift-tools.com

Bug reports, inquiries to be sent to : <generator@swift-tools.com>