Converting a Flash object to an HTML5 object

François Lionet - 20/01/2014

This document will guide you step by step through the conversion of the source code of a Flash extension object to a HTML5 extension object.
ActionScript, as a language, is very similar to JavaScript. It is almost, in fact, a strongly typed JavaScript. Most of the syntax remains the same, and the default classes contain the same functions.
While writing the HTML5 runtime, I discovered that by carefully using search / replace in the correct order, most Flash objects can be converted to HTML5 in a couple of hours. The conversion process is very safe in that if you respect what is written in this document, there is little chance of conversion errors, and your object should work at first run.
The only thing that needs to be rewritten is graphics routines, but as JavaScript is such a simple language to use, this should not take long.

Setup

You have to configure the search/replace functions of your editor or IDE. The search should be **case sensitive** and **whole word only**.
Be careful when you copy the strings to replace from this document, as many of them contain dots, brackets, commas etc. Using them exactly as I describe is **essential** for the conversion to work.
I suggest that you regularly check if the conversion is correct by scrolling through the code, so that you can undo if necessary.

The beginning

- Open your IDE and load the extension you want to convert. In our example, I'll be working on the Inventory object, so I open the file named "CRunInventory.as"
- Open the editor you want to use use for HTML5 (I use Aptana), and create a new file named "**extensionname.js**" (in my example it will be "**Inventory.js**"). Note that the name is the name of the extension, respecting the case, and **without** "CRun" at the beginning.

Initial copy of the code

Start by copying any comments at the begining of the code if they are present. Javascript uses the same syntax for comments (// or /* and */).

Global variables

You certainly have global variables in your Flash code : variables defined as "const" or as "static". Copy and paste the list of these variables at the beginning of your HTML5 code.

- Using search/replace, replace the variable definition (for example "**public static var**") by an **empty string**. This should leave you with a list of variable names like this one :
  ```
  CND_NAMEDITEMSELECTED:int=0;
  CND_NAMEDCOMPARENITEMS:int=1;
  ```
- Do another global search/replace, this time search for "**:int**" (with a column) and replace it with an **empty string**. Your list of global variables should already look nicer.
- Eventually set the indentation of the list of variable using SHIFT-TAB
- Will will come back to this list later.

The class constructor

We are going to have our Javascript object structured as neatly as possible. Let's create the class constructor.

- Enter the constructor function like this :
- ```
  function CRunnameofobject()
  {
  }
  ```

  in our case, it would give :

  ```
  function CRunInventory()
  {
  }
  ```

  This function will be called when the object is created. Although it is not mandatory, we will declare and initialize all the class variables in this function. Flash sets all the variables to 0 when the application start, Unexpected results might occur if we do not do the same in the Javascript version.
- Copy and paste all the class variable definitions from your ActionScript code to your Javascript constructor function. This should look like this :
  ```
  public var type:int;
  public var logFont:CFontInfo;
  public var pDisplayString:String;
  public var bUpdateList:Boolean;
  ```
- Now search and replace "**public var**" (or may be in your case "**private var**" by an **empty string**, it should look like this now :
  ```
  type:int;
  logFont:CFontInfo;
  pDisplayString:String;
  ```
- Do a global search/replace of "**:int**" with "**=0**"
  ```

- Do a global search/replace of "**:String**" with "**=null**" (if you use "string" in your actionscript code, replace that too)
- Do a global search/replace of "**:Boolean**" with "**=false**" (if you use "bool" in your actionscript code, replace that too)
- Do a global search/replace of "**:Number**" with "**=0**"
- If you use any other objects as class variables, set them to "**=null**".
- Your class constructor should look better now:

```
type=0;
logFont=null;
pDisplayString=null;
```

- Leave the list as it is, we will come back to it later.

Copying the main code

The main code of the extension will be defined in the prototype of the JavaScript object. Let's define the prototype:

- Type the following:

```
CRunnameofextension.prototype = CServices.extend(new
CRunExtension(),
{

});
```

In our example:

```
CRunInventory.prototype = CServices.extend(new
CRunExtension(),
{

});
```

Be very careful to respect the curly and regular bracket structure. The functions of the object will be added as a list of properties to the prototype.

- Now copy a large block of the ActionScript code (all the functions of the object), and paste it inside the curly brackets in the code we just entered
- Correct the indentation with SHIFT-TAB
- We are now ready to begin

Removing ActionScript variable types

- Do a global search/replace of "**:int**", replacing it with an **empty string**
- Do a global search/replace of "**:Boolean**" replacing it with an **empty string**
  You might also use "**bool**" for booleans in your ActionScript code. If this is the case, replace **" :bool"** with an empty string too.

- Do a global search/replace of " **:String**" with an **empty string**
  Same here if you use "**string**" instead of "String"
- Do a global search/replace of "**.CValue**" with an **empty string**
- Do a global search/replace of "**.CActExtension**" with an **emtpy string**
- Do a global search/replace of "**.CCndExtension**" with an **empty string**
- Do a global search/replace of "**.CObject**" with an **empty string**
- Do a global search/replace of "**:void**" with an **empty string**
- In a general manner, if you use class names for variables, replace them (including the "." with an empty string. For example, in my Inventory object, I use a class named "CRunInventoryItem", so I do a global search/replace of ".CRunInventoryItem" with an empty string.

Inserting the name of the class in the global variables

Lets go back to the top of the source, in the list of global variables. We need to define them as variables of the class itself, like this :

```
CRunextensionname.VARIABLE=0;
```

In my case, for example, I would get :

```
CRunInventory.ACT_LEFT=0;
```

- Start with the first line. Copy the name of the variable in the clipboard
- Paste the name from the clipboard in the search field of the search/replace dialog. For our example, I would enter "**ACT_LEFT**"
- Enter "**nameofextension.VARIABLENAME**" in the replace field. For our previous example, I would enter "**CRunInventory.ACT_LEFT"** in the replace field
- Do a global replace. All occurrences of this global variable are now modified
- Proceed the same way for all the global variables.
- At the end of the process, your list of global variables should be perfect, and most importantly, all the code in the object reflect the changes.

Inserting "this" before the class variables

Scroll to the constructor of the object in the editor. Javascript needs the "**this**" keyword to indicate if a variable is part of the current object. We will proceed in a similar way as above, for all the class variables. The code will become :

```
this.variable=0;
```

In my example :

```
this.type=0;
```

- Start with the first variable defined in the constructor. Copy the name of the variable to the clipboard.
- Paste the name in the search field of the search/replace dialog. In my example I would enter "**type**"
- Enter "**this.variablename**" in the replace field of the search/replace dialog. In my case, I would enter "**this.type**"
- Proceed with the global replace. All occurrences of the variable should be changed in the source.
- Do the same for each one of the variables defined in the constructor.

Warning: this procedure will not work if you use variables with the same name as class variables (for example, if I used a variable named "type" in one of the functions – doing this is a bad habit anyway). You may need to redo the changes manually for such a variable.

At the end of the procedure, all of your class member variables should now have **this.** before them throughout the source code.

A few more things to search/replace

You may not use them, but doing a search/replace for the next list of items will not take you a lot of time.

- Do a global search/replace for "**ho**" with "**this.ho**"
- Do a global search/replace for "**rh**" with "**this.rh**"
- Do a global search/replace for "**new CValue(**" with "**(**"
- Do a global search/replace for "**readInt**" with "**readAInt**"
- Do a global search/replace for "**readShort**" with "**readAShort**"
- Do a global search/replace for "**readColor**" with **readAColor**"
- Do a global search/replace for "**readString**" with "**readAString**"
- Do a global search/replace for "**readStringSize**" with "**readAString**"

All of these changes are to cope with the differences between **CBinaryFile** and **CFile** in the CreateRunObject function. I am really sorry for creating such a mess – I should never have used CBinaryFile in ActionScript...

- Do a global search/replace of "**.getInt()**" with an **empty string**
- Do a global search/replace of "**.getDouble()**" with an **empty string**
- Do a global search/replace of "**.getString()**" with an **empty string**

Changing the function definitions

Now comes the longest part of the job. Starting at the top of the source code, you will have to modify the declaration of each function to make it compatible with JavaScript. At the same time, you can review the code of the functions and make any manual changes that are necessary.

Let's take this function:

```
    public override function getNumberOfConditions()
```

It becomes in Javascript :

```
    getNumberOfConditions:function()
```

Or this one :

```
      public override function createRunObject(file:CBinaryFile,
cob:CCreateObjectInfo, version)
```

It becomes in Javascript :

```
      createRunObject:function(file, cob, version)
```

As you can see, the syntax of the declaration of a function is :

```
      functionname:function(parameters)
```

You should at the same time you do this job, remove any type declaration for the parameters. Do it in all the source code as soon as you encounter one. In my example above, I would do a global search/replace of "**:CBinaryFile**" with an **empty string**, and another global search/replace of "**:CCreateObjectInfo**" with an **empty string**.

By doing the search/replace as soon as you encounter a new type declaration, you will slowly but surely eliminate all the occurrences in the code and it will go faster and faster as you progress.

Ending the functions by a comma

The syntax of Javascript enforces you to end the declaration of each function with a **comma**. So you **must** add a comma at the end of each function. Our example function would become :

```
    getNumberOfConditions:function()
    {
        return Inventory.CND_LAST;
    },
```

A comma must added at the end of each function **except** for the **last** one!

Beware, if you forget a comma somewhere in the code, your extension will **not** work (and the HTML5 runtime will crash usually at the beginning of the code.

Function calls

Your code might contain internal functions, called in other places of your code.
Calling such a function must be done with the "**this**" operator.
When re-reading the code (while changing the function definitions) if you encounter such a function, I suggest you do a global search replace of "**nameoffunction**" to "**this**.**nameoffunction**", which will change all the places where the function is called.

For example:

```
private function GetFixedValue(pHo)
```

becomes

```
GetFixedValue:function(pHo)
```

Then I do a global search/replace of "**GetFixedValue**" with "**this.GetFixedValue**". All of the function calls are changed in the code.

As a side effect, my function definition becomes :

```
this.GetFixedValue:function(pHo)
```

Which is obviously wrong, so I need to remove the "this." to complete the code.
You should do this as soon as you encounter an internal function definition when exploring the code. Do not forget to remove the "this." prefix in the function definition.

Changing the typecasts

As Javascript is not a strongly typed language, you do not need to cast a variable from class to class. You do need to in ActionScript, and the syntax of a cast in ActionScript is:

```
=class(variable)
```

So it could be possible to do a global search/replace of "**newclass(**" with "**(**", leaving only the brackets in the code. But beware! Doing this will also change a line like this one :

```
var variable=new class();
```

to

```
var variable=();
```

So you may save time by doing a global search/replace of the casts as indicated above, providing you know where the occurrences of "**new class()**" are located and change them manually.

Displaying your object

One thing is very different between Flash and HTML5: how the object is displayed.

No need to create sprites and add them to the display list

In the HTML5 runtime, the display of the object is handled automatically. This means you can remove all of the following from your Flash source:

- Creation of sprite objects
- Creation of TextField objects
- Adding these objects to the display list of the object's layer

Usually, these objects are created in the CreateRunObject function.

MMF automatically calls the displayRunObject function when necessary. It takes into account whether the object is a background or not. Hiding and showing the object is also performed automatically. Same for the functions that change the display priority of the object. (it should have been like that in the Flash runtime actually 8-|).

In fact, the only thing you have to implement in the HTML5 runtime is the displayRunObject function.

The displayRunObject function now has 3 parameters: (renderer, xDraw, yDraw)

- renderer : contains the renderer object. Use this parameter for drawing your object
- xDraw : the X offset at which to draw the object. This offset is different from zero if, for example, the layer on which the object exists has been scrolled. You should add this value to the position of your object (hoX)
- yDraw : the Y offset at which to draw the object.

You can find a pointer to the layer on which the object exists in **this.ho.pLayer**. You can find out if the object is shown or hidden with **this.ho.bShown** (boolean). You should calculate the coordinate at which to draw your object with the following formula:

```
var x=this.ho.hoX-this.rh.rhWindowX+this.ho.pLayer.x+xDraw;
var y=this.ho.hoY-this.rh.rhWindowY+this.ho.pLayer.y+yDraw;
```

You can find the specifications of the Renderer object in the main documentation of this SDK.