

Compilation de thèmes et de classes en bibliothèques SWC

par Vincent Petithory ([Mes articles](#))

Date de publication : 01/04/2008

Dernière mise à jour :

Dans cet article, nous verrons comment créer des bibliothèques SWC à l'aide du compilateur en ligne de commande `compc` et étudierons deux applications possibles : la création et l'utilisation d'un SWC pour un thème Flex et la création et l'utilisation d'une bibliothèque de classes ActionScript 3.0.

- I - Introduction
- II - Rôle et contenu généré
- III - Création d'un SWC regroupant des classes
- IV - Utilisation de ce SWC
- V - Compilation et utilisation de thèmes Flex en SWC
- VI - Pour les utilisateurs de AIR
- VII - Conclusion

I - Introduction

Toujours dans le thème de la compilation, nous allons voir cette fois-ci comment utiliser le compilateur en ligne de commande `compc`, fourni dans le Flex SDK, puis l'utilisation du résultat de nos compilations. Il y aura une partie sur l'archivage et utilisation de classes, et une sur l'archivage et l'utilisation de thèmes pour Flex.

Pour les utilisateurs de Flex Builder, vous reconnaîtrez ce que l'on peut faire avec ce compilateur si vous avez déjà expérimenté un projet de type "Flex Library Project".

Vous pourrez retrouver les sources créées pour cet article [ici](#) ([miroir](#)).

II - Rôle et contenu généré

Compac a pour but de générer des bibliothèques de composants. Ces bibliothèques compilées, sous forme d'archives SWC, fournissent un moyen efficace pour partager et utiliser un ensemble de données. En outre, pour ceux qui souhaitent ne pas partager leurs sources, et autoriser malgré tout l'utilisation de leurs classes, les compiler dans un SWC est un moyen idéal, car leur code ne sera pas visible. (même si pour moi, l'intérêt est plutôt la facilité de manipuler une archive plutôt qu'un ensemble de fichiers).

Une archive SWC pourra ainsi contenir un certain nombre d'assets, de composants, de classes, et tout autre fichier requis par ces éléments.

Il est possible de décompresser l'archive (via tout utilitaire de compression supportant *PKZip*, *7-Zip* par exemple), et d'explorer certains éléments. Deux fichiers importants sont à noter :

- Le fichier *catalog.xml*, qui liste le contenu du SWC. Cela comprend vos propres éléments, mais également les imports qui ont été nécessaires, comme des classes du framework Flex par exemple.
- Un *swf* compilé regroupant certaines ressources de l'archive (par exemple vos classes ActionScript, vos MXML, symboles et composants).

La lecture du *catalog.xml* peut vous permettre de vérifier le contenu de votre SWC, ou de prendre connaissance du contenu d'un SWC que vous avez récupéré.

III - Création d'un SWC regroupant des classes

Voilà le vif du sujet. Pour l'exemple, nous allons prendre un ensemble de 3 classes d'un même package, qui a pour rôle de créer un environnement basique 2D où peuvent rebondir des objets (youpi :D).

Nous avons donc un répertoire racine contenant nos classes dans leur package, ainsi que les éléments nécessaires pour réaliser la compilation, à savoir :

- Un **batch** afin d'appeler facilement le compilateur `compc` et lui passer des paramètres.
- Un **xml** comprenant les paramètres à passer au compilateur.
- Un **fichier manifest** par namespace à inclure. Ici, un seul.

Il existe plusieurs commandes du compilateur pour inclure des fichiers sources :

- Les inclure une par une, via `-include-classes`.
- Les inclure par répertoire, via `-include-sources`.
- Les inclure par namespace, via `-include-namespaces`.

Cette dernière est la plus appropriée lorsque le nombre de fichiers sources augmentent : les sources sont regroupées dans des espaces de nom, et on choisit d'inclure tel ou tel espace de nom dans la bibliothèque compilée.

En revanche, cela est plus long à mettre en place au début car les fichiers de configuration doivent être écrits. L'utilisation de templates résout néanmoins ce souci. C'est la solution que j'utilise pour le framework (assez modeste) que je développe en ce moment, et dont je parlerai dans un article d'ici quelques jours.

Pour accélérer la mise en place, les options du compilateur sont spécifiées dans un xml. J'irai vite là-dessus, donc si vous n'êtes pas familiers avec cette façon de faire, regardez [mon article sur le compilateur mxmhc](#), qui en parle plus en détail.

Ecrivons le batch appelant le compilateur :

```
@echo off

REM the path of the compc compiler
SET compcPath="C:\Program Files\Adobe\Flex SDK 3\bin\compc.exe"

REM the path of your xml configuration
SET xmlConfigPath=library.config.withManifest.xml

echo SWC a l aide d un fichier manifest

%compcPath% -load-config+=%xmlConfigPath%
pause
```

Ainsi que le xml de configuration :

```
<?xml version="1.0" encoding="utf-8"?>
<flex-config>
  <compiler>
    <source-path>
      <!-- current directory -->
```

```
<path-element>src</path-element>
  </source-path>
  <namespaces>
    <namespace>
      <uri>http://blog.lunar-dev.net/tutorials/env2D</uri>
      <manifest>manifest.xml</manifest>
    </namespace>
  </namespaces>
</compiler>

<include-namespaces>
  <uri>http://blog.lunar-dev.net/tutorials/env2D</uri>
</include-namespaces>

<directory>>false</directory>

<output>env2DWithManifest.swc</output>

<metadata>
  <title>Classes de rebond d objets 2D</title>
  <description>Un ensemble de classes permettant de creer un environnement 2D ou evoluent des
objets.</description>
<publisher>examples.lunar-dev.net</publisher>
  <creator>examples.lunar-dev.net</creator>
  <language>FR</language>
</metadata>
</flex-config>
```

Décortiquons-le :

Ensuite, nous définissons un ensemble d'espaces de nom. Ces espace de nom comprennent un URI ainsi qu'une liste de sources, spécifiées dans un fichier manifest. Attention, l'URI que vous spécifiez doit être unique ! Rappelons aussi que les chemins spécifiés sont relatifs à l'emplacement de ce xml.

Une fois les namespaces définis, nous devons indiquer lesquels nous allons inclure dans notre bibliothèque. Dans une balise `<include-namespaces>`, il suffit d'ajouter une balise `<uri>` pour chaque espace de nom à inclure, cette balise contenant l'URI de l'espace de nom à inclure.

L'option `directory=false` indique que nous voulons obtenir un SWC en sortie (valeur par défaut), dont le nom et l'emplacement sont spécifiés avec la commande `-o`. Si `directory=true`, nous obtenons un dossier qui contiendra ce que vous auriez obtenu en décompressant l'archive SWC. (en conséquence, la balise `<output>` devra contenir un chemin valide vers un dossier, et non le nom d'une archive SWC)

Si vous envisagez d'utiliser votre bibliothèque en tant que RSL ou de la partager avec d'autres applications en parallèle, fixez l'option à true : Flex a besoin d'accéder directement au swf compilé pour le charger dynamiquement dans vos applications.

Pour chaque espace de nom, nous devons créer un fichier manifest. C'est la seule étape nécessitant un travail d'écriture un peu long. Dans notre cas, ça sera court, il n'y a que 3 classes d'un même package à inclure.

```
<?xml version="1.0" encoding="utf-8"?>
<componentPackage>

  <!-- package net.lunar.engines.engine2D -->
  <component id="Engine2D" class="net.lunar.engines.engine2D.Engine2D"/>
  <component id="AbstractObject2D" class="net.lunar.engines.engine2D.AbstractObject2D"/>
  <component id="Sphere2D" class="net.lunar.engines.engine2D.Sphere2D"/>

</componentPackage>
```

Pour chaque classe à inclure, il faut créer une balise `<component />` avec deux attributs :

- `id` : un identifiant unique (typiquement le nom de la classe). Cet identifiant sera utilisé lorsque vous voudrez instancier ce composant dans un mxml.
- `class` : le chemin vers la class, en notation pointée, comme pour un import.

Concernant l'id, certaines classes que vous pourriez inclure dans le manifest ne pourront évidemment pas être instanciées dans un mxml si ces composants ne sont pas visuels. Pour pallier cet aspect illogique, Il faut créer une classe, dans le package racine, avec simplement les imports des composants non-visuels qu'on veut inclure. Toutes les classes seront ainsi ajoutées lors de l'analyse des dépendances par le compilateur.

Malheureusement, cela exige de devoir trier en deux catégories les classes que nous avons créées. Nous aurons donc nos classes visuelles associées à un namespace dans un fichier manifest, et nos classes non-visuelles importées dans une classe en package racine. Cela ajoute un peu de travail en plus ; en revanche, vous aurez un résultat bien propre :).

Pour bien montrer la différence, je vais compiler une autre bibliothèque SWC avec cette dernière méthode. Evidemment, ici ce n'est pas la bonne solution, puisque nos 3 classes sont des composants visuels, mais ça permettra de voir la façon de faire.

Nous créons ce fichier actionscript à la racine du source-path :

```
package
{

internal class Env2DClasses
{
import net.lunar.engines.engine2D.AbstractObject2D; AbstractObject2D;
import net.lunar.engines.engine2D.Engine2D; Engine2D;
import net.lunar.engines.engine2D.Sphere2D; Sphere2D;
// Maintain alphabetical order
}
}
```

Nous avons besoin d'un autre xml de configuration qui va inclure cette classe :

```
<?xml version="1.0" encoding="utf-8"?>
<flex-config>
<compiler>
<source-path>
<!-- current directory -->
<path-element>src</path-element>
</source-path>

</compiler>

<directory>>false</directory>

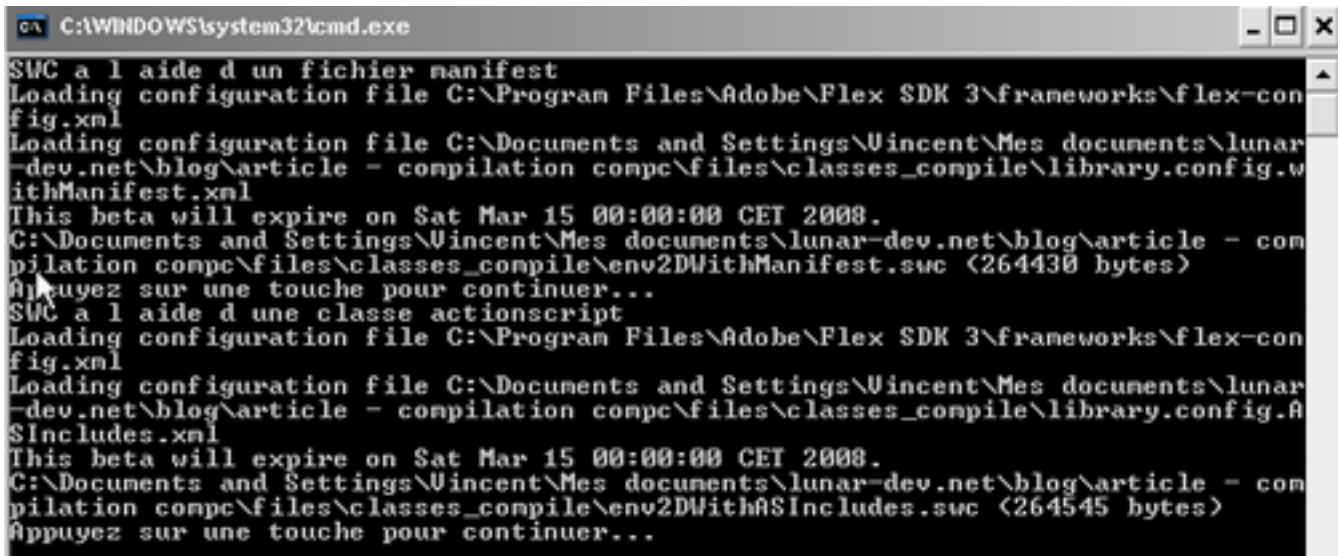
<output>env2DWithASIncludes.swc</output>

<include-classes>
<class>Env2DClasses</class>
</include-classes>

<metadata>
<title>Classes de rebond d objets 2D</title>
```

```
<description>Un ensemble de classes permettant de creer un environnement 2D ou evoluent des
objets.</description>
<publisher>examples.lunar-dev.net</publisher>
<creator>examples.lunar-dev.net</creator>
<language>FR</language>
</metadata>
</flex-config>
```

On est fin prêt pour compiler ; on lance le batch (qui compile les deux bibliothèques) :



```
C:\WINDOWS\system32\cmd.exe
SWC a l aide d un fichier manifest
Loading configuration file C:\Program Files\Adobe\Flex SDK 3\frameworks\flex-con
fig.xml
Loading configuration file C:\Documents and Settings\Vincent\Mes documents\lunar
-dev.net\blog\article - compilation compc\files\classes_compile\library.config.w
ithManifest.xml
This beta will expire on Sat Mar 15 00:00:00 CET 2008.
C:\Documents and Settings\Vincent\Mes documents\lunar-dev.net\blog\article - com
pilation compc\files\classes_compile\env2DWithManifest.swc (264430 bytes)
Appuyez sur une touche pour continuer...
SWC a l aide d une classe actionsript
Loading configuration file C:\Program Files\Adobe\Flex SDK 3\frameworks\flex-con
fig.xml
Loading configuration file C:\Documents and Settings\Vincent\Mes documents\lunar
-dev.net\blog\article - compilation compc\files\classes_compile\library.config.A
SIncludes.xml
This beta will expire on Sat Mar 15 00:00:00 CET 2008.
C:\Documents and Settings\Vincent\Mes documents\lunar-dev.net\blog\article - com
pilation compc\files\classes_compile\env2DWithASIncludes.swc (264545 bytes)
Appuyez sur une touche pour continuer...
```

Et nous obtenons nos SWC là où nous les attendions. Magnifique :).

Dans les sources, vous pourrez décompresser les deux archives SWC. Vous constaterez dans les catalog.xml que les 3 classes ont bien été incluses :

- Dans le cas du swc compilé à l'aide d'un manifest, les 3 classes figurent dans la balise 'components', avec chacune des attributs donnant leur tag MXML ainsi que leur namespace.
- Dans le cas du swc compilé à l'aide d'un .AS, les 3 classes figurent comme scripts dans la description du library.swf.

IV - Utilisation de ce SWC

C'est très simple. Il faut déjà ajouter notre SWC au projet :

Dans Flex Builder, il suffit d'ajouter le swc dans le library path (projects -> properties).

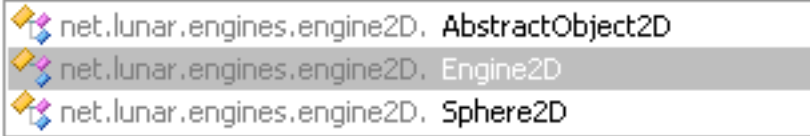
Si vous utilisez le compilateur en ligne de commande mxmhc, je vous invite à regarder [l'article sur la compilation avec mxmhc](#), qui vous dira tout là-dessus.

Une fois ceci fait, les classes incluses dans le SWC sont disponibles dans votre code, par simple import (que ce soit avec la bibliothèque que j'ai compilé à l'aide d'un .as ou celle compilée à l'aide du fichier manifest).

```

<mx:Script>
    <![CDATA[
        import engine
    ]]>
</mx:Script>

```



Pour les composants qu'on peut déclarer en mxml, il faut déclarer l'espace de nom lié aux composants qu'on veut utiliser. Dans notre exemple, les 3 classes sont incluses dans le même espace de nom :

```

<?xml version="1.0" encoding="utf-8" ?>
<mx:Application
backgroundGradientColors="[#444444, #111111]"
layout="absolute"
xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:engines="http://blog.lunar-dev.net/tutorials/env2D">

<mx:Panel>
<mx:LinkButton id="addBall" label="Ajouter boule" click="factory2D.addRandomObject2D()"/>
<mx:LinkButton id="removeBall" label="Retirer boule" click="factory2D.removeRandomObject2D()"/>
<mx:LinkButton id="clearBalls" label="Retirer tout" click="factory2D.clear()"/>
</mx:Panel>

<engines:Engine2D width="100%" height="100%" id="factory2D" creationComplete="init();" />

</mx:Application>

```

Rien de plus simple. Et l'avantage pour les composants visuels, c'est qu'on s'affranchit de devoir créer un espace de nom pour chaque package différent ; chose qu'on aurait dû faire si on avait les sources directement dans les packages.

Dans l'exemple qui suit, j'utilise le swc compilé sans manifest, et donc sans espace de nom. Vous pouvez constater que je suis obligé de créer mon objet Engine2D en ActionScript, car il n'a pas d'espace de nom affecté ni de tag MXML.

```


```

```
<?xml version="1.0" encoding="utf-8" ?>
<mx:Application
  creationComplete="init()"
  backgroundGradientColors="[#444444, #111111]"
  layout="absolute"
  xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>

    <![CDATA[

      import net.lunar.engines.engine2D.Engine2D;

      private var factory2D:Engine2D;

      private function init():void
      {
        factory2D = new Engine2D();
        factory2D.width = 1024;
        factory2D.height = 768;
        this.addChild(factory2D);
      }

    ]]>

  </mx:Script>

  <mx:Panel>
    <mx:LinkButton id="addBall" label="Ajouter boule" click="factory2D.addRandomObject2D()"/>
    <mx:LinkButton id="removeBall" label="Retirer boule" click="factory2D.removeRandomObject2D()"/>
    <mx:LinkButton id="clearBalls" label="Retirer tout" click="factory2D.clear()"/>
  </mx:Panel>

</mx:Application>
```

V - Compilation et utilisation de thèmes Flex en SWC

Une archive SWC peut contenir d'autres fichiers que des classes en .as et .mxml. Pour un thème plus évolué qu'un simple CSS, on aurait besoin de stocker des images, des fichiers css et éventuellement des classes skinnant des composants par le code.

Pour illustrer ceci, nous allons recompiler le thème haloclassic d'Adobe. Nous avons donc à inclure une feuille de style, un swf contenant les symboles de nos composants, ainsi qu'un package de classes réalisant des skins de composants par le code.

Comme d'habitude, nous allons utiliser un couple batch/xml pour faire ça (J'omets le batch, vous vous doutez de son contenu). Voici le xml qui va correctement configurer le compilateur :

```
<?xml version="1.0" encoding="utf-8"?>
<flex-config>
  <compiler>
    <source-path>
      <!-- current directory -->
<path-element>haloclassic/src/</path-element>
    </source-path>

  </compiler>

  <!-- include all files the theme requires -->
  <include-file>
    <name>Assets.swf</name>
<path>haloclassic/assets/Assets.swf</path>
  </include-file>

  <!-- name of the class enumerating all other classes -->
  <!-- Must be found in the source-path -->
  <include-classes>
    <class>HaloClassicClasses</class>
  </include-classes>

  <include-stylesheet>
    <name>defaults.css</name>
<path>haloclassic/defaults.css</path>
  </include-stylesheet>

  <output>haloclassic.swc</output>

  <metadata>
    <title>Haloclassic</title>
    <description>Theme Haloclassic d Adobe.</description>
<publisher>Adobe</publisher>
    <creator>Adobe</creator>
    <language>EN</language>
  </metadata>
</flex-config>
```

En détail :

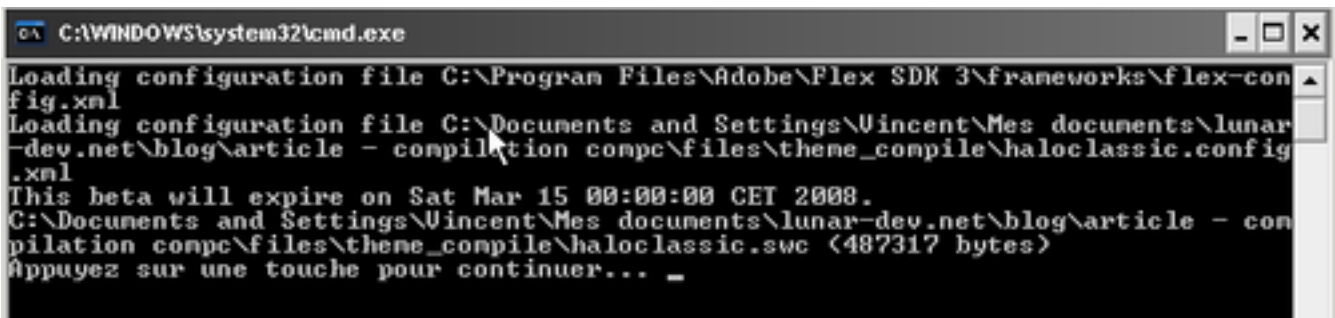
On spécifie comme d'habitude le source-path où se trouvent nos sources actionscript.

On utilise ensuite la balise *<include-file>* afin de d'intégrer le swf contenant les composants skinnés. On aurait fait de même pour chaque image (comme un background par défaut) ou autre fichier à inclure.

Pour inclure la feuille de style, on pourrait faire de même, mais pour plus de propreté, on utilisera la balise `<include-stylesheet>`. La différence entre les deux est que le compilateur va rechercher les références externes que fait la feuille de style et les compiler ; ce sont entre autres des skins par le code.

Pour finir, nous devons inclure proprement les sources de notre thème. Etant des composants non-visuels, si vous avez suivi ce qui précède, vous devinez que c'est un fichier ActionScript énumérant ces classes qui va se charger du boulot. Nous avons juste à inclure cette classe, via `<include-classes>`.

C'est tout cuit, on lance le batch :



```

C:\WINDOWS\system32\cmd.exe
Loading configuration file C:\Program Files\Adobe\Flex SDK 3\frameworks\flex-con
fig.xml
Loading configuration file C:\Documents and Settings\Vincent\Mes documents\lunar
-dev.net\blog\article - compilation compc\files\theme_compile\haloclassic.config
.xml
This beta will expire on Sat Mar 15 00:00:00 CET 2008.
C:\Documents and Settings\Vincent\Mes documents\lunar-dev.net\blog\article - con
pilation compc\files\theme_compile\haloclassic.swc (487317 bytes)
Appuyez sur une touche pour continuer... _
  
```

Et notre beau SWC est là. Pour l'utiliser, c'est simple. Si vous voulez que votre application utilise ce skin par défaut en lieu et place du skin par défaut de flex, vous pouvez :

- ajouter la commande `-theme="chemin vers votre swc"`
- utiliser la balise `<theme>` dans un xml de configuration.

En considérant que tous les fichiers sont dans un même répertoire, avec l'option en ligne de commande, on aurait :

```

@echo off

REM the path of the mxmclc compiler
SET mxmclcPath="C:\Program Files\Adobe\Flex SDK 3\bin\mxmclc.exe"

SET swcTheme=haloclassic.swc

%mxmclcPath% -theme=%swcTheme% SkinView.mxml
pause
  
```

Et avec un xml de configuration (extrait):

```

<compiler>
...
...
<theme>
  <filename>haloclassic.swc</filename>
</theme>
</compiler>
  
```

On lance l'une des compilations possibles :

```
C:\WINDOWS\system32\cmd.exe
Loading configuration file C:\Program Files\Adobe\Flex SDK 3\frameworks\flex-con
fig.xml
Loading configuration file C:\Documents and Settings\Vincent\Mes documents\lunar
-dev.net\blog\article - compilation compc\files\theme_use\build.config.xml
This beta will expire on Sat Mar 15 00:00:00 CET 2008.
C:\Documents and Settings\Vincent\Mes documents\lunar-dev.net\blog\article - con
pilation compc\files\theme_use\haloclassic.swc$defaults.css(513): Warning: ignor
ing AS2 code within 'mx.skins.cursor.BusyCursor'; Flex 2 applications only suppo
rt AS3

C:\Documents and Settings\Vincent\Mes documents\lunar-dev.net\blog\article - con
pilation compc\files\theme_use\SkinView.swf (427054 bytes)
Appuyez sur une touche pour continuer...
```

Magnifique, notre application est skinné par défaut par ce thème :).

Dans les [sources de cet article](#), vous trouverez également des templates de batch, xml et manifest afin de mettre assez rapidement en place tout ce que vous avez vu dans cet article pour vos propres besoins.

VI - Pour les utilisateurs de AIR

Afin d'inclure les classes propres à AIR dans votre bibliothèque compilée, utilisez de la même manière `acompc.bat` (dans le même dossier que `compc.exe`). Le comportement est identique à `compc` : `air-config.xml` préalablement chargé au lieu de `flex-config.xml`.

VII - Conclusion

Dans cet article, nous avons vu comment créer des bibliothèques SWC à l'aide du compilateur en ligne de commande `comp.c`. Nous avons vu deux applications possibles : la création et l'utilisation d'un SWC pour un thème Flex et la création et l'utilisation d'une bibliothèque de classes ActionScript 3.0.

