

# Comment personnaliser une fenêtre Apollo

par [Olivier Bugalotto \(Mes articles\)](#)

Date de publication : 17/04/2007

Nous sommes habitués à voir les applications Flash ou Flex cantonnées dans un navigateur ou dans le player flash. Toutes interactions avec l'extérieur devaient passer par un conteneur qui intégrait l'activeX Flash (dans le cas Windows) écrit dans des technologies comme C++, VB, Java ou encore C#. Aujourd'hui avec Apollo, ces applications se libèrent et nous verrons cela au travers de ce tutorial...

- I - Avant-propos
- II - Une première apparence
- III - Fermer l'application
- IV - Déplacer notre application
- V - Reduire, agrandir et restaurer
- VI - Redimensionner
- VII - Téléchargement

## I - Avant-propos

Voici l'apparence d'une fenêtre d'application Apollo :



Rien de bien extraordinaire vous me direz mais avec Apollo, nous allons pouvoir personnaliser cette apparence.

Toute application Apollo écrite avec Flex est basée sur un super conteneur : **ApolloApplication** mais surtout avec un fichier de configuration XML (voir [Ma première application Apollo](#)) où nous pouvons spécifier l'apparence de la fenêtre.

Voici le fichier de configuration pour personnaliser notre fenêtre :

```
<rootContent systemChrome="none" transparent="true" visible="true">[SWF reference is generated]</rootContent>
```

mais cela ne suffira pas, il faut aussi modifier notre fichier d'application MXML. Il suffit de remplacer le super conteneur **ApolloApplication** par le conteneur **Application**, comme ca :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
</mx:Application>
```

Nous allons d'ailleurs aussi supprimer la couleur de fond et l'image de fond à personnalisant le style de l'application. Comme pour les pages HTML, nous allons utiliser un style :

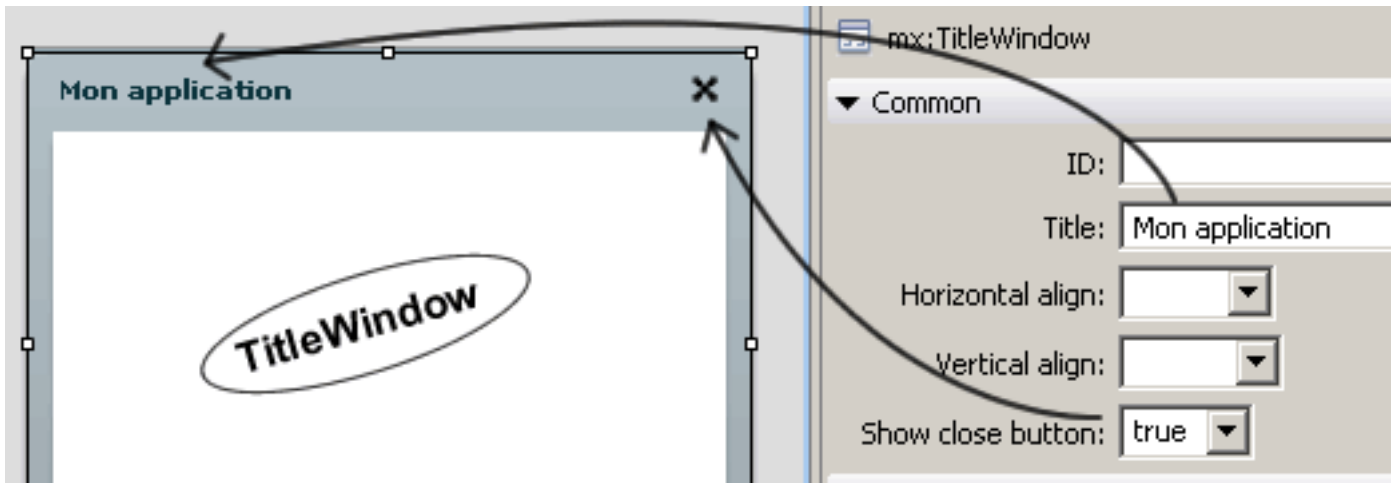
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical" width="300"
height="250">
  <mx:Style>
    Application {
      background-color:"";
      background-image:"";
      padding-bottom: 0px;
      padding-left: 0px;
      padding-right: 0px;
```

```
padding-top: 0px;  
    }  
    </mx:Style>  
</mx:Application>
```


Le problème dans tout cela, c'est que nous n'avons plus de contrôle sur la fenêtre puisqu'elle n'existe plus. Nous allons donc devoir reproduire le comportement standard d'une fenêtre à savoir : la fermer, la réduire, l'agrandir et enfin la déplacer.

## II - Une première apparence

Plaçons d'abord un composant TitleWindow pour avoir une première apparence :



Modifions aussi sa taille en mettant 100% en hauteur comme en largeur de manière à couvrir la totalité de la surface du conteneur Application.

 *Nous avons le droit de mettre des pourcentages dans les attributs d'une balise MXML mais si vous voulez le faire de manière programmé, vous devez utiliser les propriétés `percentWidth` et `percentHeight` valeur comprise entre 0 et 100, bien entendu.*

Avant de tester, mettons en place et le mécanisme de fermeture de notre application.

### III - Fermer l'application

Pour manipuler notre fenêtre d'application, nous devons récupérer une instance de cette dernière. Nous pouvons récupérer cette instance de la manière suivante :

```
stage.window
// ou encore a partir de n'importe quel DisplayObject
// si bien entendu celui-ci a ete ajoute a la display list
monSprite.stage.window
```

Voici donc le code nécessaire pour fermer notre application, lorsque nous cliquons sur la croix du composant TitleWindow :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical" width="300" height="250">
  <mx:Style>
    Application {
      background-color: "";
      background-image: "";
      padding-bottom: 0px;
      padding-left: 0px;
      padding-right: 0px;
      padding-top: 0px;
    }
  </mx:Style>
  <mx:Script>
    <![CDATA[
      private function onClose():void {
        var win:NativeWindow = stage.window;
        win.close();
      }
    ]]>
  </mx:Script>
  <!-- Remarquer la gestion de l'evenement close -->
  <mx:TitleWindow id="titleWin" width="100%" height="100%"
    layout="absolute" title="Mon application"
    close="onClose()" showCloseButton="true">
  </mx:TitleWindow>
</mx:Application>
```

## IV - Déplacer notre application

Il est évident que nous allons trouver l'ensemble des fonctionnalités à implémenter de notre exemple dans les méthodes de la classe **WindowNative** :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical" width="300" height="250"
  creationComplete="onInit()">
  <!-- Le style Application personnalise dans une feuille de style -->
  <mx:Style source="style.css" />
  <mx:Script>
  <![CDATA[
      // Lorsque l'application est cree nous ajoutons un ecouteur sur
      // l'instance du composant TitleWindow lorsque nous laissons le
      // bouton gauche de notre souris enfoncee
      private function onInit():void {
        titleWindow.addEventListener(MouseEvent.CLICK, onMouseDown);
      }

      private function onMouseDown(e:MouseEvent):void {
        stage.window.startMove();
      }

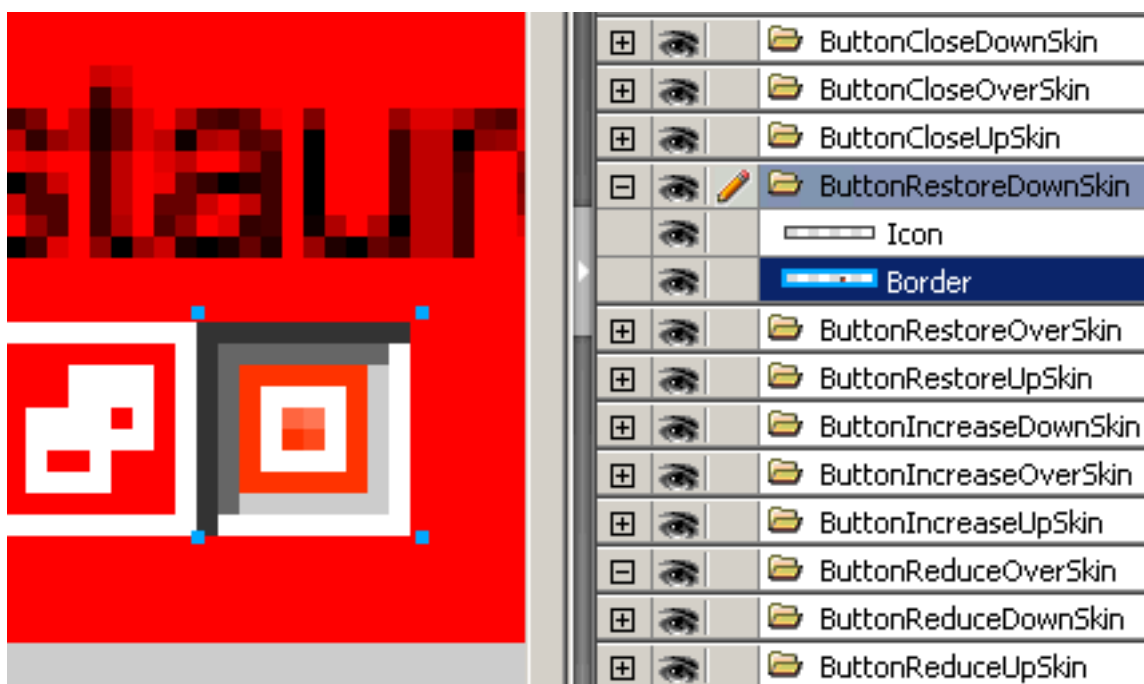
      private function onClose():void {
        var win:NativeWindow = stage.window;
        win.close();
      }
  ]]>
</mx:Script>
<mx:TitleWindow id="titleWindow" width="100%" height="100%"
  layout="absolute" title="Mon application"
  close="onClose()" showCloseButton="true">
</mx:TitleWindow>
</mx:Application>
```

## V - Reduire, agrandir et restaurer

Le composant TitleWindow ne propose pas d'autres boutons comme celui de reduire, d'agrandir ou encore de restaurer. Nous allons donc devoir les créer. Voici leurs apparences :



Nous réalisons ces habillages sous Fireworks, en organisant son travail de la manière suivante : Chaque habillage dans un calque, exporter les par calque en PNG



Il nous suffira simplement d'utiliser une feuille de style pour personnaliser nos boutons :

```
.closeButton {
  upSkin: Embed(source="assets/ButtonCloseUpSkin.png");
  overSkin: Embed(source="assets/ButtonCloseOverSkin.png");
  downSkin: Embed(source="assets/ButtonCloseDownSkin.png");
}

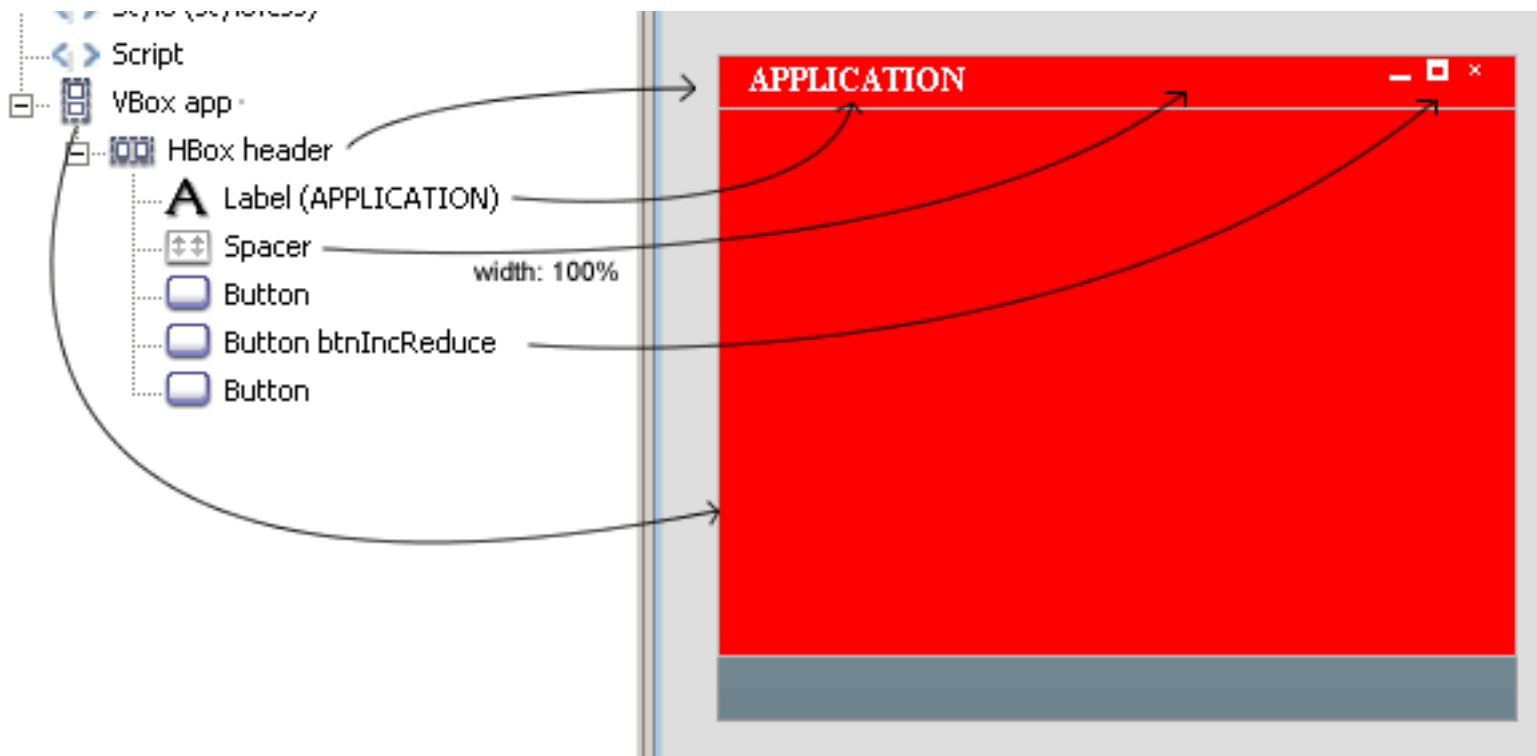
.reduceButton {
  upSkin: Embed(source="assets/ButtonReduceUpSkin.png");
  downSkin: Embed(source="assets/ButtonReduceDownSkin.png");
  overSkin: Embed(source="assets/ButtonReduceOverSkin.png");
}

.increaseRestoreButton {
  upSkin: Embed(source="assets/ButtonIncreaseUpSkin.png");
  downSkin: Embed(source="assets/ButtonIncreaseDownSkin.png");
  overSkin: Embed(source="assets/ButtonIncreaseOverSkin.png");
  /* apparence du bouton lorsqu'il est selectionne */
  selected-up-skin: Embed(source="assets/ButtonRestoreUpSkin.png");
  selected-over-skin: Embed(source="assets/ButtonRestoreOverSkin.png");
  selected-down-skin: Embed(source="assets/ButtonRestoreDownSkin.png");
}
```

```
}  
}
```

Comme nous pouvons le constater, nous utilisons deux styles différents pour un seul bouton de manière à ce qu'il représente le bouton agrandir et restaurer. Nous jouons sur le fait qu'un bouton est avant tout un bouton poussoir, il est ou pas sélectionnable. Pour rendre un bouton poussoir, il suffit de mettre la propriété `toggle` à `true`.

Voici la composition de notre application :



Le positionnement des boutons à droite est fait à l'aide d'un spacer dont sa largeur est mise à 100%.

## VI - Redimensionner

Nous allons faire de la même manière pour le redimensionnement de la fenêtre dont voici le code:

```
private function onInit():void {  
    ...  
    btnResize.addEventListener(MouseEvent.CLICK, onResize);  
}  
  
...  
  
private function onResize(e:MouseEvent):void {  
    stage.window.startResize(NativeWindowResize.BOTTOM_RIGHT);  
}
```

Il vous suffit maintenant d'effectuer le déploiement de votre application (voir [Ma première application Apollo](#)).

## VII - Téléchargement

Vous trouverez le code source et le fichier d'installation ici : **WindowApollo.air** ( [miroir](#) )

