21

Application finale

Décomposer l'application

Au cours de ce chapitre, nous allons nous intéresser aux différents points essentiels à chaque application ActionScript 3. Afin de permettre à chacun d'apprécier ce chapitre, nous ne traiterons pas de cas particulier mais nous nous attarderons sur différents concepts réutilisables pour chaque type d'application.

Le premier point que nous allons aborder concerne la décomposition de notre application.

La première approche consiste à ce que l'application se précharge elle-même, cette technique s'avère limitée car nous devons nous assurer qu'un minimum d'éléments soient exporté sur la première image de notre application, au risque de voir notre barre de préchargement s'afficher à partir de 50%.

Nous allons donc opter pour une deuxième technique en utilisant un premier SWF qui se chargera de précharger notre application

En utilisant cette approche, nous devons nous assurer que l'application préchargeant notre application soit la plus légère possible afin de ne pas nécessiter un préchargement elle aussi.

La figure 21-1 illustre le principe :

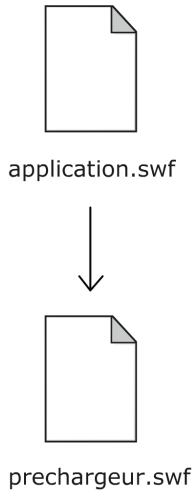

Figure 21-1. Chargement de l'application.

Chaque SWF possède une classe de document associée la figure 21-2 illustre l'idée :

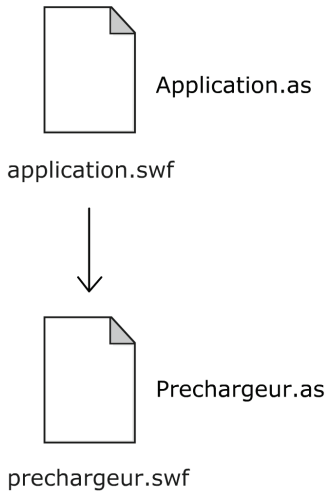

Figure 21-2. Classe de document.

Nous allons définir une première classe de document nommée `Prechargeur` pour le SWF de préchargement. Celle-ci va se charger de charger tous les éléments nécessaires à notre application.

A retenir

- Il est préférable d'utiliser un SWF dédié au préchargement de notre application.
- Ce SWF se charge de charger tous les éléments nécessaires à notre application.

Gérer les chemins d'accès

Dans la plupart des applications ActionScript, un problème se pose éternellement lié aux chemins d'accès aux fichiers. Afin de simplifier cela nous allons définir au sein d'un fichier XML de configuration les chemins d'accès.

La première chose que fera le SWF de préchargement sera de charger ce fichier XML de configuration afin de récupérer les chemins d'accès.

Création d'un fichier XML de configuration

Au sein d'un répertoire `init` nous créons un fichier XML nommé `initialisation.xml` contenant l'arbre XML suivant :

```
<CONFIG>
    <CHEMIN_XML chemin="xml/"/>
    <CHEMIN_IMAGES chemin="images/"/>
    <CHEMIN_SWF chemin="swf/"/>
    <CHEMIN_POLICES chemin="polices/"/>
</CONFIG>
```

Chaque nœud définit un chemin associé à un type de médias. Nous spécifions sont emplacement par *FlashVars* au sein de la page HTML :

```
<script language="javascript">
    if (AC_FL_RunContent == 0) {
         alert("Cette page nécessite le fichier AC_RunActiveContent.js.");
    } else {
         AC_FL_RunContent(
              'codebase',
'http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,
0,0,0',
              'width', '550',
              'height', '400',
              'src', 'chap-21-prechargeur',
              'flashvars', 'initChemin=init/',
              'quality', 'high',
              'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
              'align', 'middle',
              'play', 'true',
              'loop', 'true',
              'scale', 'showall',
              'wmode', 'window',
              'devicefont', 'false',
              'id', 'chap-21-prechargeur',
              'bgcolor', '#ffffff',
              'name', 'chap-21-prechargeur',
```

```
                'menu', 'true',
                'allowFullScreen', 'false',
                'allowScriptAccess','sameDomain',
                'movie', 'chap-21-prechargeur',
                'salign', ''
                ); //end AC code
    }
</script>
```

Nous récupérons ce chemin afin d'indiquer au SWF de préchargement où se situe le XML de configuration.

Nous associons la classe de document suivante à notre SWF de préchargement, nous réutilisons la classe `ChargeurXML` créée au cours du chapitre 14 intitulé *Charger et envoyer des données* :

```
package org.bytearray.document

{

    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.URLRequest;
    import org.bytearray.abstrait.ApplicationDefaut;
    import org.bytearray.xml.ChargeurXML;

    public class Prechargeur extends ApplicationDefaut

    {

          // adresse du fichier de configuration
          private const INITIALISATION:String = "initialisation.xml";

          // chemins
          private var cheminXML:String;
          private var cheminImages:String;
          private var cheminSWF:String;
          private var cheminPolices:String;

          // chargement
          private var chargeurInitialisation:ChargeurXML;

          public function Prechargeur ()

          {

                chargeurInitialisation = new ChargeurXML ();

                chargeurInitialisation.addEventListener ( Event.COMPLETE,
chargementInitTermine );
                chargeurInitialisation.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
                chargeurInitialisation.addEventListener (
SecurityErrorEvent.SECURITY_ERROR, erreurChargement );

                var chemin:String = loaderInfo.parameters.initChemin;

                if ( chemin == null ) chemin = "init/";
```

```
                chargeurInitialisation.charge ( new URLRequest ( chemin +
INITIALISATION ) );

            }

            private function chargementInitTermine ( pEvt:Event ):void

            {

                var initXML:XML = pEvt.target.donnees;

                cheminXML = initXML.CHEMIN_XML.@chemin;
                cheminImages = initXML.CHEMIN_IMAGES.@chemin;
                cheminSWF = initXML.CHEMIN_SWF.@chemin;
                cheminPolices = initXML.CHEMIN_POLICES.@chemin;

                // affiche : xml/ images/ swf/ polices/
                trace( cheminXML, cheminImages, cheminSWF, cheminPolices );

            }

            private function erreurChargement ( pEvt:Event ):void

            {

                trace ( pEvt );

            }

        }

}
```

Le fichier de configuration est automatiquement chargé depuis son dossier. Grâce à la variable `initChemin`, nous pouvons passer dynamiquement le chemin d'accès au fichier de configuration.

Afin de pouvoir tester notre application au sein de l'environnement auteur, nous avons ajouté le test suivant :

```
if ( chemin == null ) chemin = "init/";
```

Ce test permet simplement de définir la variable `chemin` lorsque nous testons l'application lors du développement au sein de l'environnement auteur et que l'utilisation des *FlashVars* est impossible.

Ainsi, il nous est facile de spécifier l'emplacement de ce fichier de configuration. Si lors du déploiement de notre application, le chargement d'éléments échoue, nous pouvons modifier les chemins grâce au fichier de configuration XML.

A retenir

- L'utilisation d'un fichier XML de configuration est recommandée afin de pouvoir facilement spécifier les chemins d'accès aux fichiers.
- Le chemin d'accès au fichier de configuration XML est spécifié par *FlashVars*.
- Une fois l'application mise en production, la modification des chemins d'accès aux fichiers est simplifiée.

Utiliser des librairies partagées

Lors du développement d'une application ActionScript 3, celle-ci est généralement divisée en plusieurs modules. Nous devons donc prendre en considération plusieurs points.

Le premier concerne l'optimisation du poids de l'application globale. Afin d'éviter de dupliquer les éléments graphiques et classes au sein des différents SWF nous allons utiliser le concept de librairie partagée abordée au cours du chapitre 13 intitulé *Chargement de contenu*.

Souvenez-vous, en plaçant les définitions de classes au sein d'un SWF nous avions extrait dynamiquement la définition associée et utilisée la classe au sein de notre application. Nous allons utiliser ce mécanisme de librairie partagée pour les éléments graphiques ainsi que les polices utilisées.

Les éléments graphiques seront chargés par le SWF de préchargement et rendues disponibles à tous les SWF constituant l'application. Nous économiserons ainsi du poids et gagnerons en facilité de mise à jour.

> En mettant à jour la librairie partagée, toute l'application sera mise à jour sans la nécessite d'être recompilée car tout sera extrait dynamiquement.

Imaginons que nous souhaitions utiliser le logo d'une société au sein de l'application Si nous n'optimisons pas celle-ci, voici comment celle-ci serait organisée :

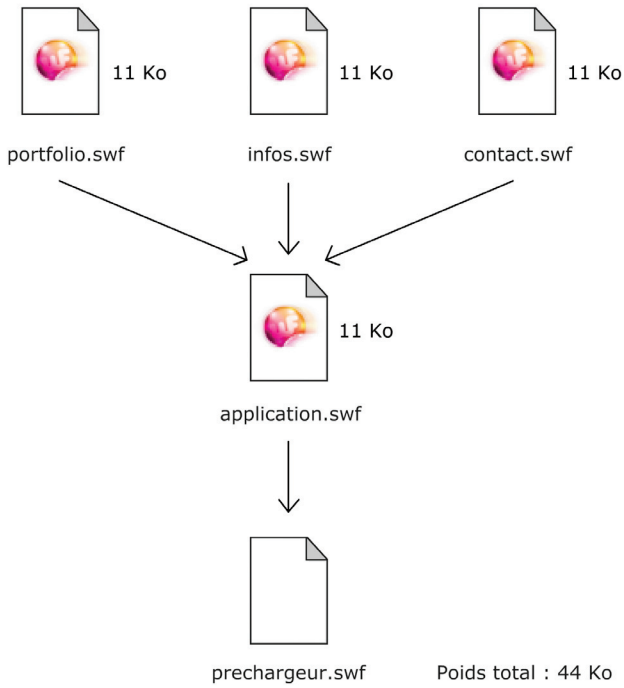

Figure 21-3. Duplication du poids de l'image.

Notre application est composée de nombreux SWF contenant chacun l'image, ce qui augmente sensiblement le poids total de l'application.

En utilisant une librairie partagée, le SWF de préchargement va extraire les classes nécessaires et permettre aux différents SWF constituant l'application leur utilisation.

Comme la figure 21-3 l'illustre, une image bitmap est utilisée par chacun des SWF. En dupliquant la classe `BitmapData` dans chaque SWF nous dupliquons et augmentons le poids de l'application globale ainsi que la bande passant utilisée.

Nous allons générer un SWF contenant l'image à réutiliser puis la réutiliser au sein des différents SWF, la figure 21-4 illustre le résultat :

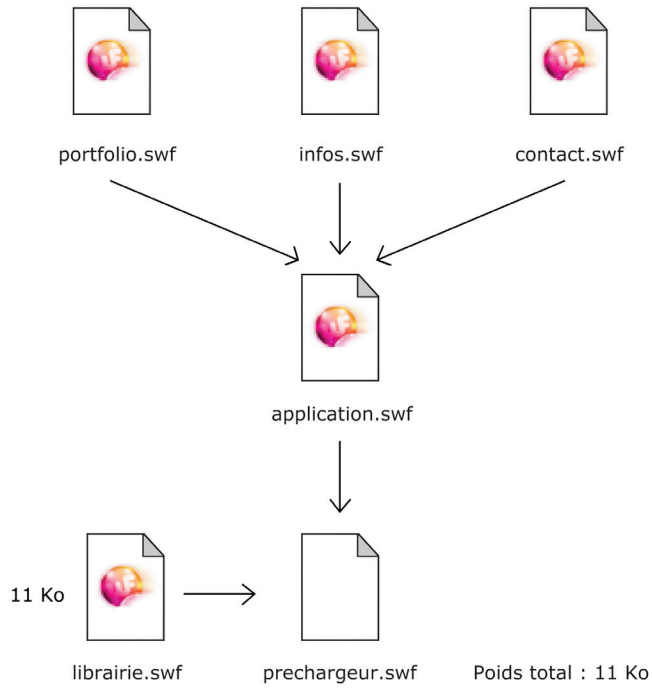

Figure 21-4. Chargement de l'image à l'aide d'une librairie partagée.

Grâce à ce procédé nous optimisons le poids de l'application mais bénéficions d'un autre avantage lié à la mise à jour de notre application. En centralisant les éléments chargés et en les réutilisant, nous facilitons la mise à jour de l'application.

Nous allons nous intéresser à cette partie dans cette nouvelle partie.

A retenir

- Afin d'optimiser la bande passante utilisée il est recommandé d'utiliser les librairies partagées.
- Leur utilisation permet de faciliter la mise à jour de l'application.
- Toutes les définitions de classes sont extraites dynamiquement, la mise à jour de l'application ne nécessite donc pas de recompilation.

Générer et charger les librairies

Afin de générer une librairie utilisable, nous créons un nouveau document Flash CS3 et importons une image dans la bibliothèque que nous associons à une classe nommée `Logo`.

Une fois définie, nous exportons un SWF sous le nom de `libraire.swf`.

Afin d'utiliser les définitions de classes issues de la librairie, le SWF gérant le chargement doit d'abord charger la librairie partagée afin de rendre les classes disponibles par tous les SWF de l'application.

Pour cela, nous devons charger la librairie dans le domaine d'application en cours. Souvenez-vous nous avions utilisé un objet `LoaderContext` lors du chapitre 13 afin de spécifier le domaine d'application dans lequel placer les classes chargées.

Nous modifions la classe de document `Prechargeur` afin de charger en premier temps la librairie partagée, puis le SWF principal `application.swf`. Nous devons donc charger notre librairie dans un premier temps, puis charger le module `application.swf`.

Afin de charger ce contenu de manière séquentielle, nous utilisons la classe de chargement séquentielle suivante :

```
package org.bytearray.chargement

{

    import flash.display.Loader;
    import flash.display.LoaderInfo;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ChargeurSequentiel extends Sprite

    {

         private var liste:Array;
         private var chargeur:Loader;
         private var contexte:LoaderContext;
         private var domaine:ApplicationDomain;
         public var contentLoaderInfo:LoaderInfo;

         public function ChargeurSequentiel ( pDomaine:ApplicationDomain )

         {

              liste = new Array();

              domaine = pDomaine;

              chargeur = new Loader();

              contentLoaderInfo = chargeur.contentLoaderInfo;

              addChild ( chargeur );

              contexte = new LoaderContext ( false, domaine );
```

```
                chargeur.contentLoaderInfo.addEventListener ( Event.INIT,
redirigeEvenement );
                chargeur.contentLoaderInfo.addEventListener (
ProgressEvent.PROGRESS, redirigeEvenement );
                chargeur.contentLoaderInfo.addEventListener ( Event.COMPLETE,
chargementTermine );
                chargeur.contentLoaderInfo.addEventListener (
IOErrorEvent.IO_ERROR, redirigeEvenement );
                chargeur.contentLoaderInfo.addEventListener (
SecurityErrorEvent.SECURITY_ERROR, redirigeEvenement );

            }

            private function chargementTermine ( pEvt:Event ):void

            {

                suivant();

            }

            private function redirigeEvenement ( pEvt:Event ):void

            {

                dispatchEvent ( pEvt );

            }

            public function ajoute ( pFichier:String ):void

            {

                liste.push ( new URLRequest ( pFichier ) );

            }

            public function demarre ( ):void

            {

                suivant();

            }

            private function suivant ( ):void

            {

                if ( liste.length ) chargeur.load ( liste.shift(), contexte );

                else dispatchEvent ( new Event ( Event.COMPLETE ) );

            }

        }
}
```


Thibault Imbert
pratiqueactionscript3.bytearray.org

Cette classe possède une méthode `ajoute` qui place chaque média en file d'attente, puis nous lançons le chargement des éléments à l'aide de la méthode `demarre`.

Nous mettons à jour le code de la classe `Prechargeur` afin de charger la librairie partagée :

```
package org.bytearray.document

{

    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import org.bytearray.abstrait.ApplicationDefaut;
    import org.bytearray.xml.ChargeurXML;
    import org.bytearray.chargement.ChargeurSequentiel;

    public class Prechargeur extends ApplicationDefaut

    {

         private const INITIALISATION:String = "initialisation.xml";

         // chemins
         private var cheminXML:String;
         private var cheminImages:String;
         private var cheminSWF:String;
         private var cheminPolices:String;

         // chargement
         private var chargeurInitialisation:ChargeurXML;
         private var chargeur:ChargeurSequentiel;

         public function Prechargeur ()

         {

              chargeur = new ChargeurSequentiel (
ApplicationDomain.currentDomain );

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( ProgressEvent.PROGRESS,
chargementEnCours );
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
              chargeur.addEventListener ( SecurityErrorEvent.SECURITY_ERROR,
erreurChargement );

              addChild ( chargeur );

              chargeurInitialisation = new ChargeurXML ();

              chargeurInitialisation.addEventListener ( Event.COMPLETE,
chargementInitTermine );
              chargeurInitialisation.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
```

```
            chargeurInitialisation.addEventListener (
SecurityErrorEvent.SECURITY_ERROR, erreurChargement );

            var chemin:String = loaderInfo.parameters.initChemin;

            if ( chemin == null ) chemin = "init/";

            chargeurInitialisation.charge ( new URLRequest ( chemin +
INITIALISATION ) );

        }

        private function chargementTermine ( pEvt:Event ):void

        {

            trace("chargement des libs et swf terminé !");

        }

        private function chargementEnCours ( pEvt:ProgressEvent ):void

        {

            trace("chargement en cours");

        }

        private function chargementInitTermine ( pEvt:Event ):void

        {

            var initXML:XML = pEvt.target.donnees;

            cheminXML = initXML.CHEMIN_XML.@chemin;
            cheminImages = initXML.CHEMIN_IMAGES.@chemin;
            cheminSWF = initXML.CHEMIN_SWF.@chemin;
            cheminPolices = initXML.CHEMIN_POLICES.@chemin;

            chargeur.ajoute ( cheminSWF + "librairie.swf" );

            chargeur.demarre();

        }

        private function erreurChargement ( pEvt:Event ):void

        {

            trace ( pEvt );

        }

    }

}
```

Nous passons au constructeur de la classe `ChargeurSequentiel` le domaine d'application en cours. Ainsi les classes issues des SWF chargés dynamiquement sont placées au sein du domaine d'application en cours.

Afin d'indiquer l'état de chargement, nous ajoutons une barre de préchargement :

```
package org.bytearray.document

{

    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import org.bytearray.abstrait.ApplicationDefaut;
    import org.bytearray.evenements.EvenementInfos;
    import org.bytearray.xml.ChargeurXML;
    import org.bytearray.chargement.ChargeurSequentiel;

    public class Prechargeur extends ApplicationDefaut

    {

         private const INITIALISATION:String = "initialisation.xml";

         // chemins
         private var cheminXML:String;
         private var cheminImages:String;
         private var cheminSWF:String;
         private var cheminPolices:String;

         // chargement
         private var chargeurInitialisation:ChargeurXML;
         private var chargeur:ChargeurSequentiel;

         // barre de préchargement
         private var jauge:Jauge;

         public function Prechargeur ()

         {

              jauge = new Jauge();

              jauge.x = int((stage.stageWidth - jauge.width) / 2);
              jauge.y = int((stage.stageHeight - jauge.height) / 2);

              chargeur = new ChargeurSequentiel (
ApplicationDomain.currentDomain );

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( ProgressEvent.PROGRESS,
chargementEnCours );
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
              chargeur.addEventListener ( SecurityErrorEvent.SECURITY_ERROR,
erreurChargement );

              addChild ( chargeur );

              chargeurInitialisation = new ChargeurXML ();
```

```
                chargeurInitialisation.addEventListener ( Event.COMPLETE,
chargementInitTermine );
                chargeurInitialisation.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
                chargeurInitialisation.addEventListener (
SecurityErrorEvent.SECURITY_ERROR, erreurChargement );

                var chemin:String = loaderInfo.parameters.initChemin;

                if ( chemin == null ) chemin = "init/";

                chargeurInitialisation.charge ( new URLRequest ( chemin +
INITIALISATION ) );

          }

          private function chargementTermine ( pEvt:Event ):void

          {

                stage.removeChild ( jauge );

          }

          private function chargementEnCours ( pEvt:ProgressEvent ):void

          {

                jauge.scaleX = pEvt.bytesLoaded / pEvt.bytesTotal;

          }

          private function chargementInitTermine ( pEvt:Event ):void

          {

                var initXML:XML = pEvt.target.donnees;

                cheminXML = initXML.CHEMIN_XML.@chemin;
                cheminImages = initXML.CHEMIN_IMAGES.@chemin;
                cheminSWF = initXML.CHEMIN_SWF.@chemin;
                cheminPolices = initXML.CHEMIN_POLICES.@chemin;

                chargeur.ajoute ( cheminSWF + "librairie.swf" );

                chargeur.demarre();

                stage.addChild ( jauge );

          }

          private function erreurChargement ( pEvt:Event ):void

          {

                trace ( pEvt );

          }

     }

}
```


Thibault Imbert
pratiqueactionscript3.bytearray.org

Le SWF de préchargement charge dans un premier temps la libraire contenant les définitions de classes :

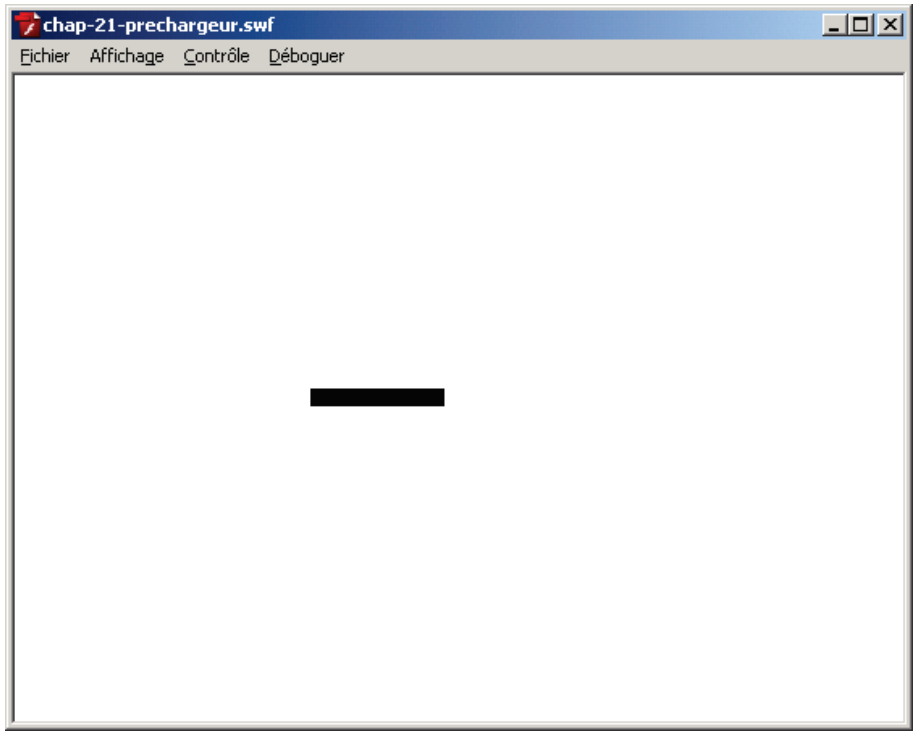

Figure 21-5. Préchargement des définitions de classes.

Une fois les définitions chargées, celles-ci pourront être utilisées par les SWF constituant l'application.

A retenir

- En chargeant la librairie partagée dans le domaine d'application en cours. Chaque SWF constituant l'application peut extraire les définitions de classe nécessaires.

Utiliser les librairies partagées

Au sein du SWF `application.swf`, nous associons la classe de document suivante :

```
package org.bytearray.document

{

    import flash.display.BitmapData;
    import flash.display.Bitmap;
    import flash.system.ApplicationDomain;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Application extends ApplicationDefaut

    {
```

```
        private var logo:BitmapData;
        private var domaineEnCours:ApplicationDomain;
        private const NOM_LOGO:String = "Logo";

        public function Application ()

        {

            domaineEnCours = ApplicationDomain.currentDomain;

            if ( domaineEnCours.hasDefinition ( NOM_LOGO ) )

            {

                var defLogo:Class = Class ( domaineEnCours.getDefinition (
NOM_LOGO ) );

                logo = new defLogo (0,0);

                addChild ( new Bitmap ( logo ) );

            }

        }

    }

}
```

Nous extrayons dynamiquement la définition de classe `Logo` à l'aide de la méthode `getDefinition` puis nous l'instancions.

Puis le SWF `application.swf` est chargé en modifiant la classe de document du SWF de préchargement :

```
private function chargementInitTermine ( pEvt:Event ):void

{

    var initXML:XML = pEvt.target.donnees;

    cheminXML = initXML.CHEMIN_XML.@chemin;
    cheminImages = initXML.CHEMIN_IMAGES.@chemin;
    cheminSWF = initXML.CHEMIN_SWF.@chemin;
    cheminPolices = initXML.CHEMIN_POLICES.@chemin;

    chargeur.ajoute ( cheminSWF + "librairie.swf" );
    chargeur.ajoute ( cheminSWF + "application.swf" );

    chargeur.demarre();

    stage.addChild ( jauge );

}
```

En testant notre application, le SWF de préchargement charge la librairie partagée puis charge le SWF `application.swf` comme l'illustre la figure 21-6 :

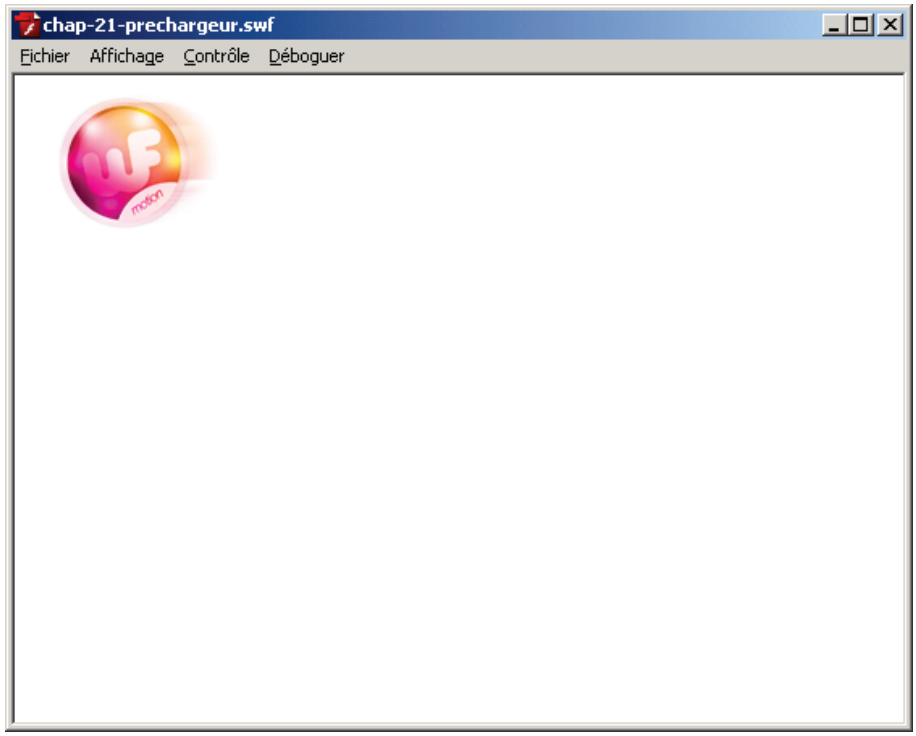

Figure 21-6. Utilisation de la définition de classe `Logo`.

Afin de nous rendre compte de la facilité de mise à jour de notre application nous ouvrons le fichier `librairie.fla` et modifions l'image `Logo` dans la bibliothèque.

Nous exportons à nouveau la librairie partagée puis nous rechargeons notre site sans aucune recompilation, la figure 21-7 illustre le résultat :

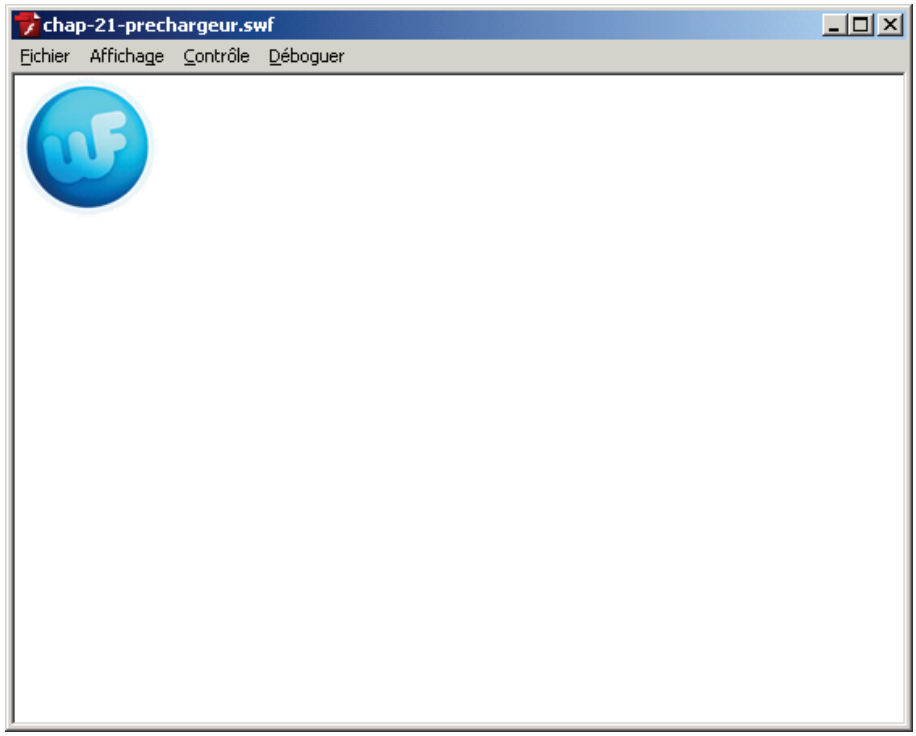

Figure 21-7. Remplacement de l'image à travers une librairie partagée.

Sans recompiler notre application nous avons mis à jour le logo utilisé au sein du module `application.swf`.

Bien que cette pratique ait un avantage certain en matière de poids et de mise à jour, celle-ci possède néanmoins un désavantage. Nous sommes obligés de passer par le SWF de préchargement afin de tester notre module SWF car les définitions de classes utilisées par ce dernier sont chargées par le module de préchargement principal.

En testant notre application nous remarquons que la définition de classe `Logo` est correctement extraire et utilisée au sein du SWF `application.swf`.

Nous allons à présent développer un menu au sein de notre module application.swf. Ce menu aura pour but de charger les autres modules `portfolio.swf`, `infos.swf` et `contact.swf`.

Nous allons utiliser la même technique afin de charger dynamiquement les polices de notre menu. Pour cela, nous intégrons au sein de la librairie une police au sein de la bibliothèque.

Nous obtenons donc deux éléments dans notre librairie :

- Une image bitmap représentant notre logo

- Une police utilisée par notre menu

La figure 21-8 illustre la bibliothèque :

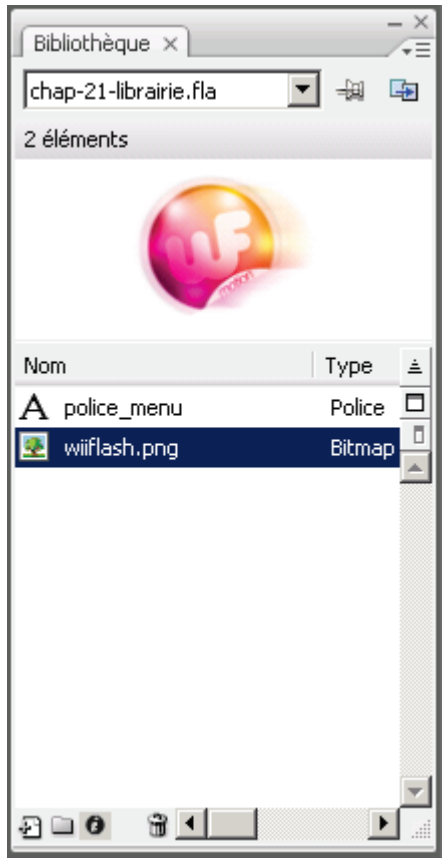

Figure 21-8. Bibliothèque de la librairie partagée.

Comme nous l'avons vu lors du chapitre 16 intitulé *Le texte*, les polices peuvent être embarquées à l'exécution. De la même manière que notre image bitmap, nous allons extraire la définition de classe de police et ajouter celle-ci dans la liste des polices disponibles.

La figure 21-9 illustre l'idée :

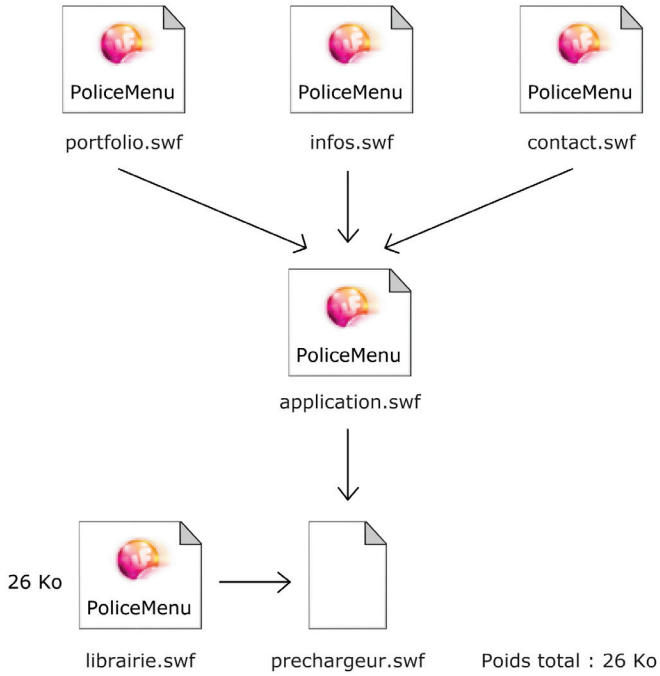

Figure 21-9. Chargement d'une police à l'aide d'une librairie partagée.

Nous modifions la classe `Application` afin d'extraire la police et l'ajouter :

```
package org.bytearray.document

{

    import flash.display.BitmapData;
    import flash.display.Bitmap;
    import flash.system.ApplicationDomain;
    import flash.text.Font;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Application extends ApplicationDefaut

    {

        private var logo:BitmapData;
        private var police:Font;
        private const NOM_LOGO:String = "Logo";
        private const NOM_POLICE:String = "PoliceMenu";

        public function Application ()

        {
```

```
                if ( ApplicationDomain.currentDomain.hasDefinition ( NOM_LOGO ) )
                {
                    var defLogo:Class = Class (
ApplicationDomain.currentDomain.getDefinition ( NOM_LOGO ) );
                    logo = new defLogo (0,0);
                    addChild ( new Bitmap ( logo ) );
                }
                if ( ApplicationDomain.currentDomain.hasDefinition ( NOM_POLICE
))
                {
                    var defPolice:Class = Class (
ApplicationDomain.currentDomain.getDefinition ( NOM_POLICE ) );
                    police = new defPolice();
                    Font.registerFont ( defPolice );
                }
            }
        }
    }
```

Afin d'utiliser la police extraite, nous créons un menu dynamique. Nous avons donc besoin de charger un fichier XML contenant les chemins d'accès au XML. Pour cela, nous allons diffuser un événement depuis le SWF de préchargement, cela va nous permettre de passer au module `application.swf` les chemins d'accès aux fichiers.

Nous définissons un fichier XML nommé `donnees.xml` :

```
<MENU>
<BOUTON legende="Accueil" url="accueil.swf"/>
<BOUTON legende="Photos" url="photos.swf"/>
<BOUTON legende="Blog" url="blog.swf"/>
</MENU>
```

Puis nous créons une classe événementielle `EvenementInfos` contenant les différentes propriétés indiquant les chemins :

```
package org.bytearray.evenements

{
    import flash.events.Event;

    public class EvenementInfos extends Event

    {
```

```
        public static const CHEMINS:String = "chemins";

        public var images:String;
        public var polices:String;
        public var swf:String;
        public var xml:String;

        public function EvenementInfos ( pType:String, pCheminImages:String,
pCheminPolices:String, pCheminSWF:String, pCheminXML:String  )

        {

            super ( pType );

            images = pCheminImages;
            polices = pCheminPolices;
            swf = pCheminSWF;
            xml = pCheminXML;

        }

    }

}
```

Le SWF de préchargement doit donc indiquer au SWF principal les chemins d'accès aux fichiers, pour cela nous diffusons un événement au module `application.swf` une fois les fichiers chargés :

```
private function chargementTermine ( pEvt:Event ):void

{

    stage.removeChild ( jauge );

    pEvt.target.contentLoaderInfo.sharedEvents.dispatchEvent ( new
EvenementInfos ( EvenementInfos.CHEMINS , cheminImages, cheminPolices,
cheminSWF, cheminXML ) );

}
```

Notre SWF `application.swf` n'a plus qu'à écouter cet événement et enregistrer les chemins puis charger le XML afin de créer le menu :

```
package org.bytearray.document

{

    import flash.display.BitmapData;
    import flash.display.Bitmap;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.MouseEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.text.Font;
    import flash.text.TextFormat;
    import org.bytearray.evenements.EvenementInfos;
```

```
    import org.bytearray.xml.ChargeurXML;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Application extends ApplicationDefaut

    {

         private var logo:BitmapData;
         private var police:Font;
         private var donneesXML:ChargeurXML;
         private var domaineEnCours:ApplicationDomain;
         private var conteneurMenu:Sprite;
         private var chargeur:Loader;

         private const NOM_LOGO:String = "Logo";
         private const NOM_POLICE:String = "PoliceMenu";
         private const MENU_XML:String = "donnees.xml";

         private var cheminXML:String;
         private var cheminImages:String;
         private var cheminSWF:String;
         private var cheminPolices:String;

         public function Application ()

         {

              loaderInfo.sharedEvents.addEventListener( EvenementInfos.CHEMINS,
initialisation );

              chargeur = new Loader();

              addChild ( chargeur );

              conteneurMenu = new Sprite();

              conteneurMenu.addEventListener ( MouseEvent.CLICK, clicBouton,
true );

              conteneurMenu.x = 110;
              conteneurMenu.y = 25;

              addChildAt ( conteneurMenu, 0 );

              donneesXML = new ChargeurXML ( true );

              donneesXML.addEventListener ( Event.COMPLETE, chargementTermine
);
              donneesXML.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
              donneesXML.addEventListener ( SecurityErrorEvent.SECURITY_ERROR,
erreurChargement );

              domaineEnCours = ApplicationDomain.currentDomain;

              if ( domaineEnCours.hasDefinition ( NOM_LOGO ) )

              {

                   var defLogo:Class = Class ( domaineEnCours.getDefinition (
NOM_LOGO ) );
```

```
                    logo = new defLogo (0,0);

                    addChild ( new Bitmap ( logo ) );

               }

          }

          private function initialisation ( pEvt:EvenementInfos ):void

          {

               cheminXML = pEvt.xml;
               cheminImages = pEvt.images;
               cheminSWF = pEvt.swf;
               cheminPolices = pEvt.polices;

               if ( domaineEnCours.hasDefinition ( NOM_POLICE ))

               {

                    var defPolice:Class = Class ( domaineEnCours.getDefinition (
NOM_POLICE ) );

                    police = new defPolice();

                    Font.registerFont ( defPolice );

                    donneesXML.charge ( new URLRequest ( cheminXML + MENU_XML )
);

               }

          }

          private function chargementTermine ( pEvt:Event ):void

          {

               var donnees:XML = pEvt.target.donnees;

               var boutonMenu:Bouton;
               var i:int = 0;

               var formatage:TextFormat = new TextFormat ( police.fontName, 8 );

               for each ( var noeud:XML in donnees.* )

               {

                    boutonMenu = new Bouton();

                    boutonMenu.buttonMode = true;
                    boutonMenu.mouseChildren = false;
                    boutonMenu.lien = noeud.@url;

                    boutonMenu.legende.embedFonts = true;

                    boutonMenu.legende.defaultTextFormat = formatage;

                    boutonMenu.legende.text = noeud.@legende;
```

```
                    boutonMenu.x = (boutonMenu.width + 5) * i;
                    conteneurMenu.addChild ( boutonMenu );
                    i++;
               }
          }
          private function clicBouton ( pEvt:MouseEvent ):void
          {
               chargeur.load ( new URLRequest ( cheminSWF + pEvt.target.lien )
);
          }
          private function erreurChargement ( pEvt:Event ):void
          {
               trace(pEvt);
          }
     }
}
```

En utilisant un nom de police générique, nous pouvons ainsi remplacer plus tard la police utilisée pour le menu par une autre sans créer des erreurs de logique Ainsi une police Verdana, Times ou autre pourra être utilisée de manière transparente pour désigner la police du menu.

Lorsque nous testons le module `application.swf`, nous ne voyons pas le menu affiché. A l'inverse, en testant le SWF par le module principal nous obtenons le résultat suivant :

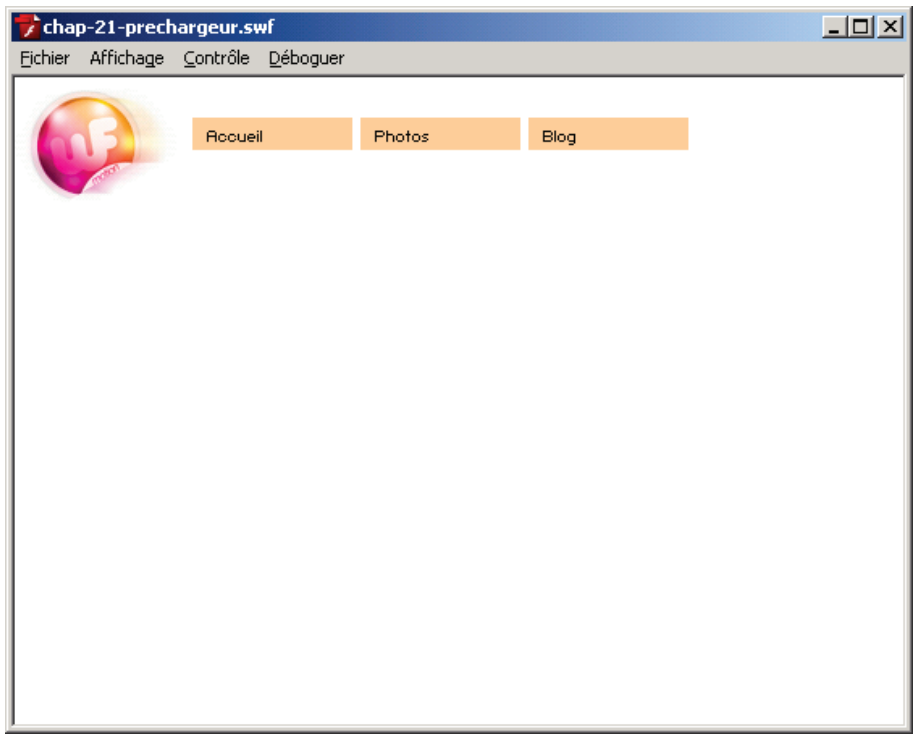

Figure 21-10. Utilisation de la police au sein du menu dynamique.

La police est ainsi intégrée au sein du site et chargée une seule et unique fois. Nous économisons bande passante et gagnons en facilité de mise à jour.

Afin de mettre à jour la police, nous modifions la police présente au sein de la bibliothèque et régénérons le fichier `librairie.swf`.

La figure 21-11 illustre l'application relancée en ayant modifié la police présente au sein de la librairie partagée :

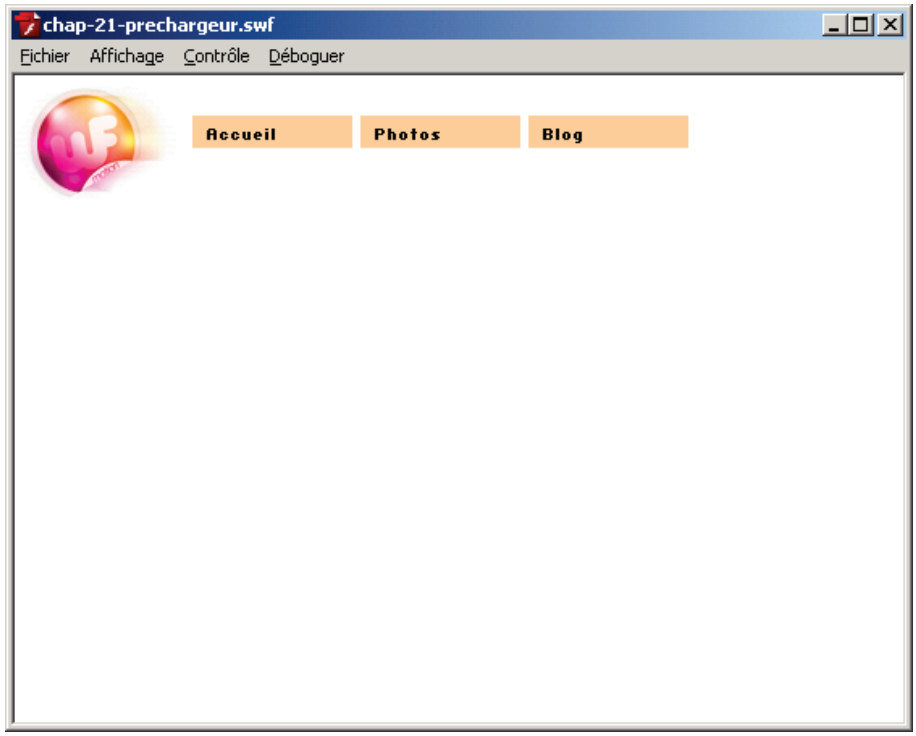

Figure 21-11. Remplacement de la police à travers la librairie partagée.

La police extraite est désormais disponible au sein de tous les SWF de l'application. En cliquant sur chacun des boutons nous chargeons de nouveaux SWF ayant recours à cette police.

Nous ajoutons au sein de la librairie partagée une nouvelle police destinée à être utilisée par le contenu. Celle-ci est associée à une classe `PoliceContenu`.

La figure 21-12 illustre la bibliothèque de la librairie partagée :

Figure 21-12. Bibliothèque de la librairie partagée.

Chaque bouton du menu charge un SWF représentant une partie du site, ici encore chaque SWF va utiliser les définitions de polices chargées le SWF de préchargement. Nous évitons ainsi d'intégrer des polices au sein de chaque SWF et optimisons le poids du site.

En cliquant sur le bouton *Accueil* nous chargeons le SWF `accueil.swf`.

Nous ajoutons au sein de ce dernier le code suivant afin de créer un champ texte et d'utiliser la définition de police `PoliceContenu` :

```
var domaineEnCours:ApplicationDomain = ApplicationDomain.currentDomain;
var policeContenu:String = "PoliceContenu";

if ( domaineEnCours.hasDefinition ( policeContenu ) )

{

        var defPolice:Class = Class ( domaineEnCours.getDefinition ( policeContenu ) );

        var police:Font = new defPolice();

        var formatage:TextFormat = new TextFormat ( police.fontName, 8 );
```

```
            Font.registerFont ( defPolice );

            var monContenu:TextField = new TextField()

            addChild ( monContenu );

            monContenu.x = 40;
            monContenu.y = 110;

            monContenu.embedFonts = true;
            monContenu.defaultTextFormat = formatage;
            monContenu.autoSize = TextFieldAutoSize.LEFT;
            monContenu.wordWrap = true;
            monContenu.width = 450;
            monContenu.text = "Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Duis consectetuer enim fringilla ligula. Donec facilisis scelerisque sem.
Nunc lobortis ante eget turpis. Donec auctor ullamcorper diam. Phasellus sed
enim non urna aliquam consectetuer. Morbi sit amet purus. Suspendisse eros leo,
volutpat eu, eleifend sit amet, bibendum ac, lacus. Suspendisse elit. In
egestas lorem in nisi. Integer imperdiet felis et ligula. Quisque semper
dapibus pede. Mauris ac neque vel erat porttitor hendrerit. Duis justo augue,
scelerisque a, rutrum nec, rutrum ut, justo. Maecenas luctus. Aenean a leo et
eros blandit sollicitudin. Sed bibendum placerat ligula. Integer varius.
Aliquam in felis in felis convallis laoreet. Vivamus nulla. Quisque rutrum
massa id nunc."

        }
```

Nous extrayons dynamiquement la police `PoliceContenu`, la figure 21-13 illustre le résultat :

Figure 21-13. Utilisation de la police au sein du texte de présentation.

Afin de tester notre mécanisme, nous modifions le fichier de police puis nous relançons notre site sans recompiler les autres SWF constituant l'application.

Nous obtenons le résultat illustré par la figure suivante :

Figure 21-14. Remplacement de la police à travers la librairie partagée.

De cette manière nous centralisons nos images ainsi que les polices par le biais de la librairie partagée. La mise à jour de l'application est ainsi simplifiée et les temps de chargement optimisés.

A retenir

- Afin d'extraire dynamiquement une classe nous utilisons la méthode `getDefinition` de l'objet `ApplicationDomain`.

Ainsi s'achève notre aventure au cœur de l'ActionScript 3. Rendez-vous sur le forum dédié à l'ouvrage pour en discuter et ainsi répondre à d'éventuelles questions.

A très vite !