15

Communication Externe

CONTENEUR ET CONTENU **1**

PASSER DES VARIABLES **2**

INTÉGRATION PAR JAVASCRIPT 3
LA PROPRIETE PARAMETERS 5
LES FLASHVARS 10
PASSER DES VARIABLES DYNAMIQUES 11
ACCÉDER FACILEMENT AUX FLASHVARS 14

APPELER UNE FONCTION **17**

L'API EXTERNALINTERFACE **18**

APPELER UNE FONCTION EXTERNE DEPUIS ACTIONSCRIPT 22
APPELER UNE FONCTION ACTIONSCRIPT DEPUIS LE CONTENEUR 27
COMMUNICATION ET SECURITE 30

Conteneur et contenu

Lors du déploiement d'une application Flash sur Internet, nous intégrons généralement celle-ci au sein d'une page conteneur HTML interprétée par différents navigateurs tels Internet Explorer, Firefox, Opera ou autres. Dans d'autres situations, celle-ci peut être intégrée au sein d'une application bureau développée en C#, C++ ou Java.

Bien qu'autonome, l'animation lue par le lecteur Flash peut nécessiter des informations provenant de l'application conteneur dans le cas de développement d'applications dynamiques.

Avant de s'intéresser à la notion d'échanges, il convient de définir dans un premier temps les deux acteurs concernés par une éventuelle communication :

- Le conteneur : Une application contenant le SWF. Il peut s'agir d'une page HTML ou d'une application C#, C++ ou autres.
- L'application : Il s'agit du SWF lu par le lecteur Flash.

Au cours de ce chapitre, nous allons nous intéresser aux différentes techniques de communication possible entre ces deux acteurs tout en se préoccupant des éventuelles restrictions de sécurité pouvant intervenir.

Passer des variables

Afin d'introduire la notion d'échanges entre ces deux acteurs, nous allons nous intéresser dans un premier temps au passage de simples variables encodées au format URL.

Afin de bien comprendre l'intérêt de tels échanges, mettons nous en situation avec le cas suivant :

Vous devez développer un lecteur vidéo qui puisse charger différentes vidéos dynamiques pour un portail destiné aux cinéphiles. Afin de rendre le lecteur le plus dynamique possible, il serait judicieux de pouvoir spécifier en dehors de l'animation quelle bande-annonce jouer.

Vous pensez dans un premier temps à la création d'un fichier XML contenant le nom de la vidéo à jouer. Certes, cette technique fonctionnerait mais n'est pas optimisée. Le fichier XML devrait être dupliqué pour chaque lecteur vidéo, rendant le développement rigide et redondant.

La solution la plus efficace consiste à passer dynamiquement le nom de la vidéo à jouer depuis la page navigateur contenant le lecteur vidéo. Pour cela, deux techniques existent :

La première consiste à passer des variables encodées URL après le nom du fichier au sein du paramètre `movie` de la balise `object` et `src` de la balise `embed` :

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
#version=9,0,0,0" width="550" height="400" id="chap-15-variables"
align="middle">
  <param name="allowScriptAccess" value="sameDomain" />
  <param name="allowFullScreen" value="false" />
  <param name="movie" value="chap-15-variables.swf?maVar1=15&maVar2=50"
/><param name="quality" value="high" /><param name="bgcolor" value="#ffffff"
/>
  <embed src="chap-15-variables.swf?maVar1=15&maVar2=50" quality="high"
bgcolor="#ffffff" width="550" height="400" name="chap-15-variables"
align="middle" allowScriptAccess="sameDomain" allowFullScreen="false"
```

```
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Automatiquement, les variables `maVar1` et `maVar2` sont accessibles au sein du SWF intitulé `chap-15-variables.swf`.

Cette technique est utilisée lorsque l'animation est intégrée manuellement au sein d'une page conteneur.

Dans le cas de Flash CS3 un script JavaScript est généré automatiquement lors de la publication afin d'intégrer dynamiquement l'animation. Flash CS3 intègre un mécanisme d'intégration de l'animation Flash par script JavaScript permettant de ne pas avoir à demander l'autorisation de l'utilisateur afin de lire du contenu Flash.

Cette astuce évite d'avoir à activer l'animation en cliquant dessus au sein d'Internet Explorer.

> En avril 2007, la société Eolas avait obtenu de Microsoft la modification d'Internet Explorer visant à demander obligatoirement l'activation de l'utilisateur lorsqu'un contenu ActiveX était intégré dans une page.

Si la page contenant l'animation est directement générée depuis Flash CS3, la technique visant à intégrer les variables directement au sein des balises `object` et `embed` ne fonctionnera pas.

Il nous faut passer les variables depuis la fonction JavaScript `AC_FL_RunContent`.

A retenir

- L'utilisation de variables encodées au format URL est le moyen le plus simple de passer des données au SWF.
- Lorsque la page conteneur est générée depuis Flash CS3, l'intégration du SWF dans la page est gérée par la fonction JavaScript `AC_FL_RunContent`.

Intégration par JavaScript

Lorsque la page conteneur est directement générée depuis Flash CS3, le script suivant est intégré à la page HTML afin d'intégrer le SWF :

```
<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-15-variables',
            'quality', 'high',
            'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
            'align', 'middle',
            'play', 'true',
            'loop', 'true',
            'scale', 'showall',
            'wmode', 'window',
            'devicefont', 'false',
            'id', 'chap-15-variables',
            'bgcolor', '#ffffff',
            'name', 'chap-15-variables',
            'menu', 'true',
            'allowFullScreen', 'false',
            'allowScriptAccess','sameDomain',
            'movie', 'chap-15-variables',
            'salign', ''
            );
  }
</script>
```

La fonction `AC_FL_RunContent` intègre l'animation en ajoutant chaque attribut passé en paramètre.

Ainsi, pour passer les variables équivalentes à l'exemple précédent, nous passons les variables au sein du paramètre `movie` de la fonction `AC_FL_RunContent` :

```
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-15-external-interface',
       'quality', 'high',
       'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
       'align', 'middle',
       'play', 'true',
       'loop', 'true',
       'scale', 'showall',
       'wmode', 'window',
       'devicefont', 'false',
       'id', 'monApplication',
       'bgcolor', '#ffffff',
       'name', 'monApplication',
       'menu', 'true',
       'allowFullScreen', 'false',
       'allowScriptAccess','sameDomain',
       'movie', 'chap-15-variables?maVar1=15&maVar2=50',
       'salign', ''
);
```

Dans les précédentes versions d'ActionScript, les variables étaient directement placées sur le scénario principal `_root`.

En ActionScript 3, afin d'éviter les conflits de variables, celles-ci ne sont plus disponibles à partir du scénario principal mais depuis la propriété `parameters` de l'objet `LoaderInfo` du scénario du SWF.

A retenir

- Dans le cas de l'utilisation de la fonction `AC_FL_RunContent`, les variables doivent être passées au sein du paramètre `movie`.

La propriété parameters

Comme nous l'avons vu au cours du chapitre 13 intitulé *Chargement de contenu*. Chaque SWF contient un objet `LoaderInfo` associé, contenant différentes informations.

Dans le cas de variables passées par le navigateur, les variables passées dynamiquement deviennent des propriétés de l'objet référencé par la propriété `parameters` de l'objet `LoaderInfo`.

Dans le code suivant nous définissons la classe de document suivante, afin d'accéder aux deux variables passées :

```
package org.bytearray.document

{

  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public function Document ()

        {

              var infos:Object = root.loaderInfo.parameters;

              infosVariables.appendText ( infos.maVar1 + "\n" );
              infosVariables.appendText ( infos.maVar2 + "\n" );

        }

  }

}
```

Un champ texte `infosVariables` est posé sur le scénario principal afin d'afficher les variables réceptionnées.

Lorsque des variables sont passées au lecteur Flash, les variables sont copiées au sein de la propriété `parameters` de l'objet `LoaderInfo`. Vous remarquez que nous accédons aux variables comme des propriétés de l'objet `parameters`.

> Il est important de noter que les variables sont copiées au sein de l'objet `LoaderInfo` avant l'exécution du code. Celles-ci sont donc accessibles instantanément.

Dans le code précédent nous ciblons de manière explicite la propriété `loaderInfo` du scénario principal, à l'aide de la propriété `root`.

Dans un contexte de classe du document, l'instance en cours fait référence au scénario principal, nous pouvons donc directement accéder à l'objet `LoaderInfo` de la manière suivante :

```
package org.bytearray.document

{

  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public function Document ()

        {

              var infos:Object = loaderInfo.parameters;

              infosVariables.appendText ( infos.maVar1 + "\n" );
              infosVariables.appendText ( infos.maVar2 + "\n" );

        }

  }

}
```

Au cas où nous ne connaissons pas le nom des variables passées, nous pouvons les énumérer à l'aide d'une boucle `for in` :

```
package org.bytearray.document

{

  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public function Document ()

        {

              var infos:Object = loaderInfo.parameters;

              for ( var p:String in infos )
```

```
            {
                infosVariables.appendText ( p + "\n" );
            }
        }
    }
}
```

Si aucune variable n'est passée, la boucle n'est pas exécutée, aucun contenu n'est affiché dans le champ texte `infosVariables`.

Lors du test de l'animation au sein de Flash CS3, l'animation est lue au sein du lecteur autonome. Aucune variable ne peut donc être donc passée. De la même manière, si les variables sont passées d'une manière non appropriée, nous devons le gérer.

Il est donc important de s'assurer que les variables sont bien définies, pour cela nous écrivons le code suivant :

```
package org.bytearray.document
{
  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {
        public function Document ()
        {
            var infos:Object = loaderInfo.parameters;
            if ( infos.maVar1 != undefined && infos.maVar2 != undefined )
            {
                infosVariables.appendText ( infos.maVar1 + "\n" );
                infosVariables.appendText ( infos.maVar2 + "\n" );
            } else infosVariables.appendText ( "Les variables ne sont pas
disponibles" );
        }
  }
}
```

La figure 15-1 illustre le résultat :

Figure 15-1. Variables non accessibles.

Si nous testons l'animation au sein d'une page passant correctement les variables, nous obtenons le résultat illustré en figure 15-2 :

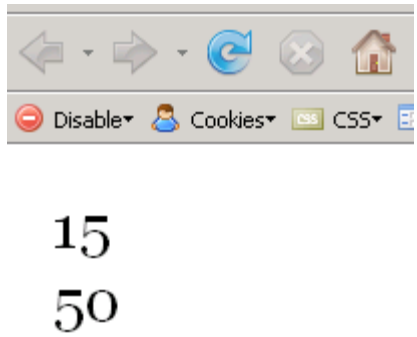

Figure 15-2. Variables correctement passées.

Il est important de noter que les variables sont copiées en tant que chaîne de caractères. Ainsi, si nous souhaitons manipuler les variables afin de faire des opérations mathématiques nous devons au préalable le convertir en nombres.

Dans le code suivant, nous tentons d'additionner les deux variables, nous obtenons alors une simple concaténation :

```
infosVariables.appendText ( infos.maVar1 + infos.maVar2 );
```

La figure 15-3 illustre le résultat :

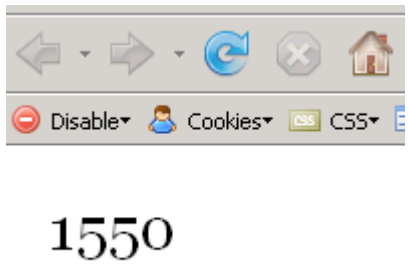

Figure 15-3. Variables concaténées.

Afin d'additionner correctement les variables nous devons au préalable les convertir en tant que nombre :

```
infosVariables.appendText ( String ( Number ( infos.maVar1 ) + Number (
infos.maVar2 ) ) );
```

La figure 15-4 illustre le résultat :

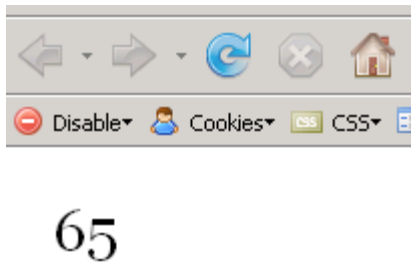

Figure 15-4. Variables additionnées.

Cette technique offre en revanche une limitation de volume de données passées propre à chaque navigateur. Depuis le lecteur 6, nous avons la possibilité de passer ces mêmes variables à l'aide d'un nouvel attribut des balises `object` et `embed` appelé `flashvars`.

De plus, le passage de variables sans l'utilisation de l'attribut `flashvars` force le rechargement du SWF empêchant donc sa mise en cache. Notons que si les variables passées sont identiques à plusieurs reprises, il n'y a pas de rechargement forcé.

A retenir

- Les variables passées sont copiées au sein de l'objet retourné par la propriété `parameters` de l'objet `LoaderInfo` du scénario principal.
- Les variables sont copiées en tant que chaînes de caractères.
- Il convient de convertir les variables en nombre avant d'effectuer des opérations mathématiques sur chacune d'elles.

Les FlashVars

L'utilisation des *FlashVars* est recommandée depuis Flash MX (Flash 6). Pour passer des variables grâce à l'attribut `flashvars`, il nous suffit d'ajouter l'attribut `flashvars` en paramètre de la fonction `AC_FL_RunContent` :

```
AC_FL_RunContent(
	'codebase',
'http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=
9,0,0,0',
	'width', '550',
	'height', '400',
	'flashvars', 'maVar1=15&maVar2=50',
	'src', 'chap-15-external-interface',
	'quality', 'high',
	'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
	'align', 'middle',
	'play', 'true',
	'loop', 'true',
	'scale', 'showall',
	'wmode', 'window',
	'devicefont', 'false',
	'id', 'monApplication',
	'bgcolor', '#ffffff',
	'name', 'monApplication',
	'menu', 'true',
	'allowFullScreen', 'false',
	'allowScriptAccess','sameDomain',
	'movie', 'chap-15-external-interface',
	'salign', ''
);
```

Si nous testons à nouveau l'animation les variables sont accessibles de la même manière.

A retenir

- L'utilisation des *FlashVars* est recommandée depuis le lecteur Flash 6.
- Les balises `object` et `embed` possèdent un attribut `flashvars` permettant un passage de variables optimisé.

Passer des variables dynamiques

Afin d'aller plus loin, nous allons récupérer dynamiquement les variables passées au sein de l'url de l'animation et les récupérer dans notre application.

Dans le cas d'un site Flash traditionnel, nous souhaitons pouvoir récupérer les variables passées en GET depuis l'url suivante :

```
http://www.monsite.org/index.php?rubrique=4&langue=fr
```

Le lecteur Flash n'a pas la capacité de les récupérer de manière autonome, nous pouvons utiliser pour cela un script JavaScript ou autre qui se chargera de les passer au SWF grâce à l'attribut `flashvars`.

Pour cela, nous ajoutons au sein de la page conteneur une fonction JavaScript nommée `recupVariables` :

```
<script language="javascript">

var position = window.location.href.indexOf ("?")+1;
var chaine = window.location.href.substr ( position );

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',
       'flashvars', chaine,
       'src', 'chap-15-flashvars-dynamique',
       'quality', 'high',
       'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
       'align', 'middle',
       'play', 'true',
       'loop', 'true',
       'scale', 'showall',
       'wmode', 'window',
       'devicefont', 'false',
       'id', 'chap-15-flashvars-dynamique',
       'bgcolor', '#ffffff',
       'name', 'chap-15-flashvars-dynamique',
       'menu', 'true',
       'allowFullScreen', 'false',
       'allowScriptAccess','sameDomain',
       'movie', 'chap-15-flashvars-dynamique',
       'salign', ''
       ); //end AC code
```

```
}
</script>
```

Nous passons dynamiquement les `flashvars` en reconstituant une chaîne encodée URL à l'aide des variables récupérées depuis l'URL.

Afin de tester si les variables sont bien passées, nous accédons à notre animation en ajoutant les variables encodées URL en fin d'adresse :

```
http://www.monsite.org/index.php?rubrique=4&langue=fr
```

Afin d'éviter les erreurs nous testons si les variables sont définies :

```
package org.bytearray.document

{

  import flash.text.TextField;
  import flash.display.MovieClip;

  public class Document extends MovieClip

  {

        public function Document ()

        {

              var infos:Object = loaderInfo.parameters;

              if ( (infos.rubrique != undefined && infos.langue != undefined) )

              {

                    infosVariables.appendText ( "langue = " + infos.langue +
"\n" );
                    infosVariables.appendText ( "rubrique = " + infos.rubrique +
"\n" );
              } else infosVariables.appendText ( "Les variables ne sont pas
disponibles" );

        }

  }

}
```

En testant notre animation, nous voyons que les variables sont correctement passées.

La figure 15-4 illustre le résultat :

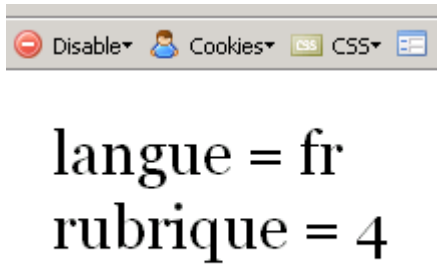

Figure 15-5. FlashVars dynamiques.

Souvenez-vous, les variables sont passées en tant que chaîne de caractères.

Ainsi, si aucune variable n'est passée dans l'url, la valeur des variables `rubrique` et `langue` seront bien différentes de `undefined`, car elles auront pour valeur `"undefined"` en tant que chaîne.

Pour gérer cela, nous ajoutons la condition suivante :

```
package org.bytearray.document

{

  import flash.text.TextField;
  import oflash.display.MovieClip;

  public class Document extends MovieClip

  {

        public function Document ()

        {

              var infos:Object = loaderInfo.parameters;

              if ( (infos.rubrique != undefined && infos.langue != undefined)
&&
              (infos.rubrique != "undefined" && infos.langue != "undefined") )

              {

                    infosVariables.appendText ( "langue = " + infos.langue +
"\n" );
                    infosVariables.appendText ( "rubrique = " + infos.rubrique +
"\n" );

              } else infosVariables.appendText ( "Les variables ne sont pas
disponibles" );

        }

  }
```

```
}
```

Ainsi, si nous accédons à l'animation en omettant les variables au sein de l'URL, nous obtenons le message illustré en figure 15-6 :

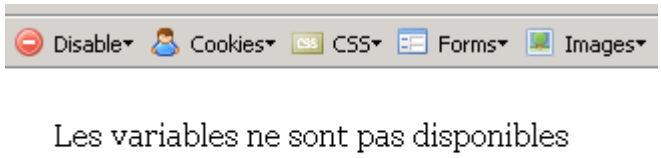

Figure 15-6. Message d'erreur.

Dans un contexte réel, il est important de s'assurer que chaque SWF composant le site puisse accéder sans problèmes aux variables passées au SWF principal.

Dans la partie suivante, nous allons découvrir comment faciliter le passage de celles-ci au reste de l'application.

A retenir

- A l'aide d'un script intégré à la page conteneur, nous pouvons passer des variables dynamiques à l'animation.

Accéder facilement aux FlashVars

Nous savons que les variables passées à l'animation sont accessibles depuis l'objet `LoaderInfo` du scénario principal.

Afin de garantir un accès simplifié de ces variables aux différents SWF composant notre application, nous allons diffuser un événement personnalisé par l'intermédiaire de la propriété `sharedEvents` de l'objet `LoaderInfo`.

Souvenez, vous au cours du chapitre 13 intitulé *Chargement de contenu*, nous avons vu que la propriété `sharedEvents` était une excellent passerelle de communication entre le SWF chargeant et chargé.

Nous définissons dans un premier temps une classe `InfosEvenement` chargée de transporter les variables :

```
package org.bytearray.evenements

{

  import flash.display.LoaderInfo;
  import flash.events.Event;
```

```
public class InfosEvenement extends Event

{

      public static const INFOS:String = "infos";

      public var infos:LoaderInfo;

      public function InfosEvenement ( pType:String, pLoaderInfo:LoaderInfo
)

      {

            super ( pType, false, false );

            infos = pLoaderInfo;

      }

      public override function clone ( ):Event

      {

            return new InfosEvenement ( type, infos );

      }

}

}
```

Puis nous modifions la classe de document du SWF principal afin de charger dynamiquement le SWF :

```
package org.bytearray.document

{

  import flash.events.Event;
  import flash.net.URLRequest;
  import flash.display.Loader;
  import flash.display.MovieClip;
  import org.bytearray.evenements.InfosEvenement;

  public class Document extends MovieClip

  {

      private var loader:Loader;

      public function Document ()

      {

            loader = new Loader();

            loader.contentLoaderInfo.addEventListener ( Event.COMPLETE,
onComplete );

            loader.load ( new URLRequest ("chap-15-animation.swf") );
```

```
            addChild ( loader );

      }

      private function onComplete ( pEvt:Event ):void

      {

            pEvt.target.sharedEvents.dispatchEvent ( new InfosEvenement (
InfosEvenement.INFOS, loaderInfo ) );

      }

  }

}
```

Afin de récupérer les variables envoyées, nous associons la classe de document suivante au SWF chargé :

```
package org.bytearray.rubriques

{

  import flash.events.Event;
  import flash.text.TextField;
  import flash.display.MovieClip;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Rubrique extends ApplicationDefaut

  {

      public var infosVariables:TextField;

      public function Rubrique ()

      {

            loaderInfo.sharedEvents.addEventListener ( InfosEvenement.INFOS,
infos );

      }

      function infos ( pEvt:InfosEvenement ):void

      {

            var infos:Object = pEvt.infos.parameters;

            if ( (infos.rubrique != undefined && infos.langue != undefined)
&&
            (infos.rubrique != "undefined" && infos.langue != "undefined") )

            {

                  infosVariables.appendText ( infos.rubrique + "\n" );
                  infosVariables.appendText ( infos.langue + "\n" );

            } else infosVariables.appendText ( "Les variables ne sont pas
disponibles" );
```

```
        }
    }
}
```

A la réception des variables la méthode écouteur `infos` est déclenchée est procède à un affichage des variables si celles-ci sont correctement définies.

A retenir

- Afin d'assurer un accès simplifié aux variables, il est intéressant de diffuser un événement depuis le SWF principal aux SWF chargés.

Appeler une fonction

La communication entre l'application conteneur et le lecteur ne se limite pas à un simple passage de variables encodées URL.

Dans le cas d'une page HTML conteneur, il est possible d'appeler une fonction JavaScript depuis ActionScript à l'aide de la fonction `navigateToURL`.

Dans le code suivant, nous ouvrons une fenêtre `alert` au sein de la page conteneur lorsque nous cliquons sur la scène :

```
package org.bytearray.document

{

  import flash.external.ExternalInterface;
  import flash.text.TextField;
  import flash.events.MouseEvent;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public function Document ()

        {

              stage.addEventListener ( MouseEvent.CLICK, click );

        }

        private function click ( pEvt:MouseEvent ):void

        {

              navigateToURL ( new URLRequest ("javascript: alert('Un message du
lecteur Flash !')"), "_self");
```

```
        }

    }

}
```

En préfixant le nom de la fonction de l'instruction `javascript:` il est aussi possible d'appeler n'importe quelle fonction JavaScript depuis ActionScript.

Dans le code suivant, nous appelons une fonction `fonctionJavaScript` :

```
navigateToURL ( new URLRequest ("javascript: fonctionJavaScript()" ),
"_self");
```

La fonction `navigateToURL` est généralement utilisée au sein d'une page conteneur afin de lancer le gestionnaire de mail configuré sur la machine :

```
private function click ( pEvt:MouseEvent ):void

{

        navigateToURL ( new URLRequest ("mailto:bob@groove.com") );

}
```

Bien qu'efficace, cette technique ne permet pas de réceptionner le retour d'une fonction JavaScript ou de passer des types précis en paramètres tels `Number` ou `Boolean`.

L'API ExternalInterface

La communication entre l'application conteneur et le lecteur ne se limite pas à un simple passage de variables encodées URL.

Il est aussi possible d'appeler différentes méthodes définies au sein du conteneur depuis ActionScript et inversement. Cette fonctionnalité était assurée auparavant par la méthode `fscommand` qui se trouve désormais dans le paquetage `flash.system`.

L'utilisation de la fonction `fscommand` est aujourd'hui dépréciée au profit de l'API `ExternalInterface`.

Afin de pouvoir utiliser celle-ci dans un contexte de navigateur, celui-ci doit prendre en charge les contrôles ActiveX ou l'API NPRuntime.

Voici un tableau récapitulatif des différents navigateurs compatible fonctionnant avec l'API `ExternalInterface` :

Navigateur	Système d'exploitation	Système d'exploitation

Internet Explorer 5.0 et versions ultérieures	Windows	
Netscape 8.0 et versions ultérieures	Windows	Macintosh
Mozilla 1.7.5 et versions ultérieures	Windows	Macintosh
Firefox 1.0 et versions ultérieures	Windows	Macintosh
Safari 1.3 et versions ultérieures		Macintosh

Tableau 1. Navigateurs compatibles.

Voici en détail les trois propriétés de la classe `ExternalInterface` :

- `available` : Renvoie l'id de la balise `object` sous Internet Explorer, ou l'attribut `name` de la balise `embed` sous Nestcape, Firefox, Opera ou autres.
- `marshallExceptions` : Cette propriété définit, si les deux acteurs peuvent recevoir les exceptions de chacun.
- `objectID` : Permet de savoir si le conteneur est compatible avec l'API `ExternalInterface`.

Dans le code suivant, nous testons si l'application évolue dans un contexte compatible avec l'API `ExternalInterface` :

```
package org.bytearray.document

{

  import flash.external.ExternalInterface;
  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public var compatible:TextField;

        public function Document ()

        {

              compatible.text = String ( ExternalInterface.available );

        }

  }

}
```

Il convient de toujours tester si le contexte actuel permet l'utilisation de l'API `ExternalInterface`. Pour cela, nous testons la propriété `available` avant toute tentative de communication.

La propriété `objectID` peut être aussi utilisée afin de déterminer si l'application évolue au sein du navigateur ou non.

Ainsi nous pourrions ajouter une propriété `navigateur` à la classe `ApplicationDefaut` permettant de savoir si l'animation évolue au sein du navigateur ou non :

```
package org.bytearray.abstrait

{

  import flash.display.MovieClip;
  import flash.events.Event;
  import flash.display.Stage;
  import flash.external.ExternalInterface;

  public class ApplicationDefaut extends MovieClip

  {

        public static var globalStage:Stage;
        public static var enLigne:Boolean;
        public static var navigateur:Boolean;
        public static var stage:Stage;
        public static var root:ApplicationDefaut;

        public function ApplicationDefaut ()

        {

              ApplicationDefaut.root = this;

              addEventListener ( Event.ADDED_TO_STAGE, activation );

              loaderInfo.addEventListener ( Event.INIT, init );

              ApplicationDefaut.navigateur = ExternalInterface.objectID !=
null;

        }

        private function activation ( pEvt:Event ):void

        {

              ApplicationDefaut.globalStage = stage;

        }

        private function init ( pEvt:Event ):void

        {

              ApplicationDefaut.enLigne = pEvt.target.url.match ( new RegExp
("^http://") ) != null;
```

```
        }

    }

}
```

Puis, dans n'importe quelle classe de document héritant de la classe `ApplicationDefaut`, nous pouvons savoir facilement si l'animation évolue ou non au sein du navigateur :

```
package org.bytearray.document

{

    import flash.text.TextField;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Document extends ApplicationDefaut

    {

        public var compatible:TextField;

        public function Document ()

        {

            if ( ApplicationDefaut.navigateur ) compatible.text =
"L'animation est lue au sein d'une page web";

            else compatible.text = "L'animation est lue hors du navigateur";

        }

    }

}
```

Si nous testons l'application hors du navigateur :

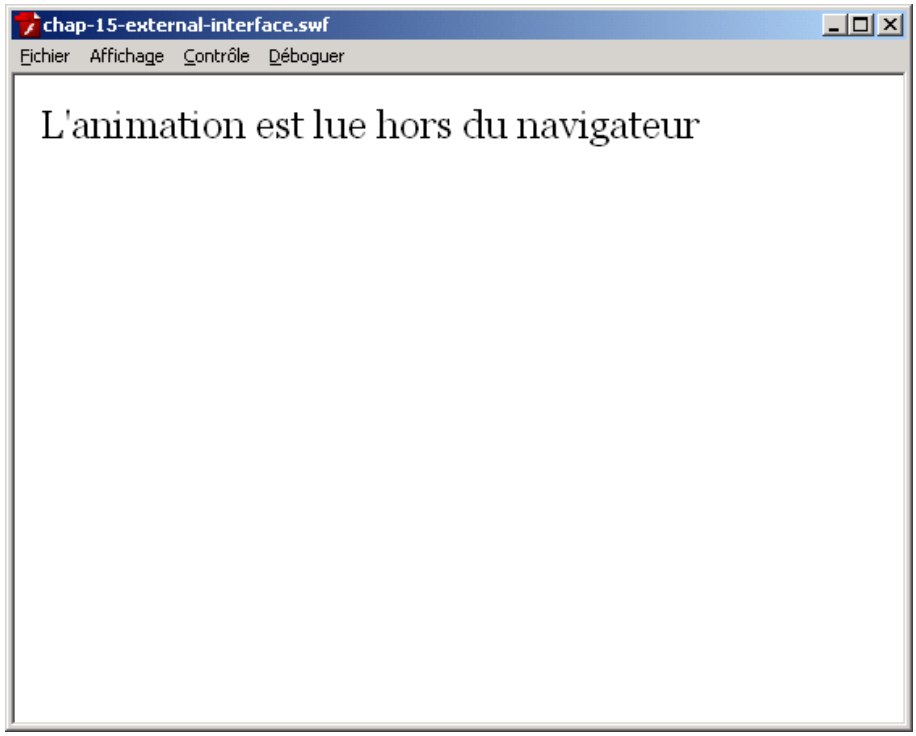

Figure 15-7. Détection du contexte.

A l'inverse, si l'animation est lue au sein du navigateur, nous le détectons comme l'illustre la figure 15-8 :

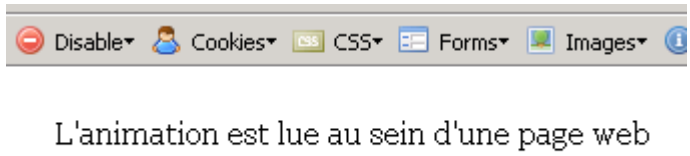

Figure 15-8. Détection du contexte.

Nous allons à présent nous attarder sur l'appel de fonctions externe depuis ActionScript.

A retenir

- L'API `ExternalInterface` est recommandée pour la communication entre ActionScript et JavaScript.
- `ExternalInterface` remplace les précédentes fonctions `fscommand`, `callFrame` et `callLabel`.

Appeler une fonction externe depuis ActionScript

Deux méthodes sont disponibles sur la classe `ExternalInterface`, voici le détail de chacune d'entre elles :

- `addCallBack` : Enregistre un alias pour une fonction ActionScript. L'alias est ensuite utilisé depuis la fonction externe pour exécuter la fonction ActionScript.
- `call` : Exécute la fonction passée en paramètre au sein du conteneur.

Dans la partie suivante, nous allons nous intéresser à l'appel d'une méthode JavaScript depuis ActionScript.

La figure 15-9 illustre le concept :

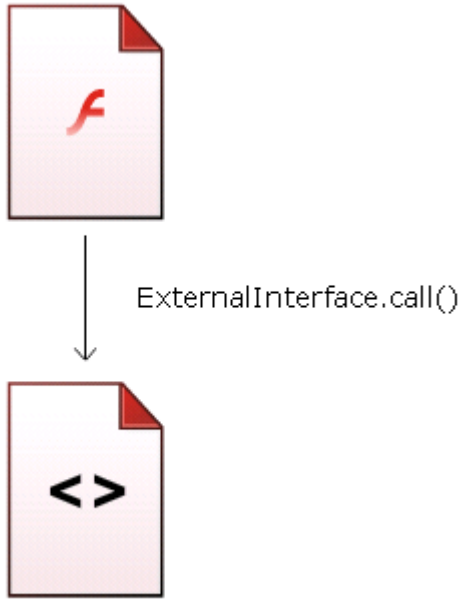

Figure 15-09. Méthode statique `call`.

Afin d'exécuter une méthode au sein de l'application conteneur, nous utilisons la méthode statique `call` de la classe `ExternalInterface` dont voici la signature :

```
public static function call(functionName:String, ... arguments):*
```

Le premier paramètre concerne le nom de la fonction à exécuter. Les paramètres suivants sont passés en paramètre à la fonction exécutée.

Nous allons commencer par un exemple simple, en appelant la méthode `direBonjour` lorsque nous cliquons sur le bouton `executeFonction` :

```
package org.bytearray.document

{

  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.external.ExternalInterface;
```

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

public class Document extends ApplicationDefaut

{

      public var executeFonction:SimpleButton;

      public function Document ()

      {

            executeFonction.addEventListener ( MouseEvent.CLICK,
declencheAppel );

      }

      private function declencheAppel ( pEvt:MouseEvent ):void

      {

            if ( ExternalInterface.available )

            {

                  ExternalInterface.call ("direBonjour");

            }

      }
}

}
```

La fonction JavaScript `direBonjour` est définie au sein de notre page conteneur :

```
function direBonjour ( )

{

  alert ("Bonjour !");

}
```

Lorsque nous cliquons sur le bouton, la fonction est déclenchée et ouvre une fenêtre d'alerte comme l'illustre la figure 15-9 :

Figure 15-10. Détection du contexte.

Nous pouvons passer dynamiquement des paramètres depuis Flash en spécifiant les paramètres à la méthode `call` :

```
ExternalInterface.call ("direBonjour", "Message de Flash !");
```

Nous modifions la fonction `direBonjour` afin d'accepter un message en paramètre :

```
function direBonjour ( pMessage )

{

  alert (pMessage);

}
```

La figure 15-11 illustre le résultat :

Figure 15-11. Détection du contexte.

La fonction JavaScript peut retourner une valeur, la valeur sera récupérée à l'appel de la méthode `call`.

Si nous modifions la fonction JavaScript afin que celle-ci renvoie le total des deux paramètres passés :

```
function calculTotal ( p1, p2 )

{

  return p1 + p2;

}
```

Nous affichons dans le champ texte `total`, le retour de l'appel de la méthode `call` :

```
package org.bytearray.document

{

  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.external.ExternalInterface;
  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {
```

```
        public var executeFonction:SimpleButton;
        public var total:TextField;

        public function Document ()

        {

             executeFonction.addEventListener ( MouseEvent.CLICK,
declencheAppel );

        }

        private function declencheAppel ( pEvt:MouseEvent ):void

        {

             if ( ExternalInterface.available )

             {

                  total.text = ExternalInterface.call ("calculTotal", 10, 15);

             }

        }

  }

}
```

La figure 15-12 illustre le résultat :

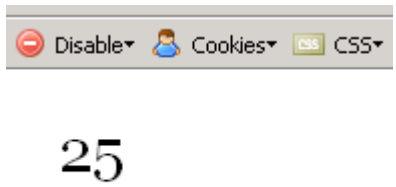

Figure 15-12. Détection du contexte.

Le code précédent illustre un des avantages de l'API `ExternalInterface` concernant le passage de données typées.

Contrairement à la fonction `fscommand` ou `navigateToURL`, nous ne sommes pas à limités au passage de chaînes de caractères avec l'API `ExternalInterface`. Nous pouvons passer entre JavaScript et ActionScript des données de type `Number`, `String`, et `Boolean`.

Nous allons maintenant communiquer dans l'autre sens en appelant depuis l'application conteneur une fonction ActionScript.

A retenir

- La méthode `call` permet d'exécuter une fonction définie au sein du conteneur depuis ActionScript.
- Dans le cas de l'utilisation d'une fonction JavaScript, des types comme `Number`, `Boolean` et `String` peuvent être échangés.

Appeler une fonction ActionScript depuis le conteneur

De la même manière, nous pouvons déclencher une fonction ActionScript depuis une fonction définie au sein du conteneur.

Afin d'enregistrer une fonction ActionScript auprès d'une fonction du conteneur, nous utilisons la méthode `addCallback`, dont voici la signature :

```
public static function addCallback(functionName:String,
closure:Function):void
```

La figure 15-13 illustre l'idée :

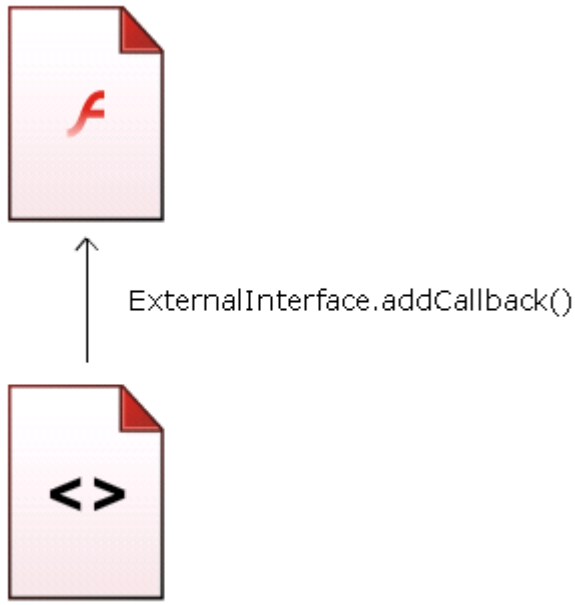

Figure 15-13. Méthode statique `addCallback`.

Voici le détail des deux paramètres de la méthode `addCallback` :

- `functionName` : Il s'agit de l'identifiant de la fonction depuis la page conteneur. Nous devrons utiliser cette chaîne pour exécuter la fonction passée au sein du paramètre `closure`.
- `closure` : La fonction à déclencher depuis la page conteneur.

Dans le code suivant, nous enregistrons une fonction `maFunction` à l'aide de l'alias `aliasFonction` :

```
package org.bytearray.document

{

  import flash.external.ExternalInterface;
  import flash.text.TextField;
  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public var messageJavascript:TextField;

        public function Document ()

        {

              // enregistre l'alias "aliasFonction" vers la méthode maFunction
              ExternalInterface.addCallback ( "aliasFonction", maFunction );

        }

        private function maFunction ( pMessage:String ):void

        {

              // nous affichons le message reçu depuis JavaScript
              messageJavascript.text = pMessage;

        }

  }

}
```

Afin de pouvoir appeler une fonction ActionScript nous devons ensuite donner un nom à notre application grâce aux attributs `id` et `name` :

```
  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-15-external-interface',
        'quality', 'high',
        'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
        'align', 'middle',
        'play', 'true',
        'loop', 'true',
        'scale', 'showall',
        'wmode', 'window',
        'devicefont', 'false',
        'id', 'monApplication',
        'bgcolor', '#ffffff',
        'name', 'monApplication',
        'menu', 'true',
        'allowFullScreen', 'false',
        'allowScriptAccess','sameDomain',
        'movie', 'chap-15-external-interface',
```

```
            'salign', ''
            );
```

Puis au sein du conteneur, nous appelons la fonction `maFunction` grâce à l'alias `aliasFonction` :

```
function recupAnimation ( pAnim )
{

  if (navigator.appName.indexOf("Microsoft") != -1) return window[pAnim];

  else return document[pAnim];

}

function declencheFonctionActionScript ( )
{

  recupAnimation("monApplication").aliasFonction("Message de JavaScript !");

}
```

La fonction `recupAnimation` permet de cibler l'animation selon le type de navigateur utilisé. Nous passons le nom de l'animation à celle-ci, qui nous renvoie l'animation intégrée dans la page, sur laquelle nous appelons la fonction grâce à l'alias passé à la méthode `addCallback`.

Il ne nous reste plus qu'à déclencher la fonction `declencheFonctionActionScript` lors du clic bouton :

```
<input type="button" name="monBouton" value="Exécute fonction ActionScript"
onClick=" declencheFonctionActionScript()">
```

La figure 15-14 illustre le résultat :

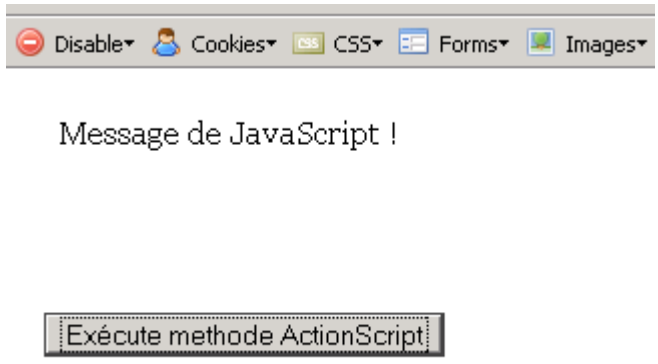

Figure 15-14. Méthode ActionScript exécutée.

Comme vous pouvez l'imaginer, la notion de communication entre le lecteur Flash et l'application conteneur est soumise à des restrictions de sécurité.

Nous allons nous intéresser dans la partie suivante, aux différentes restrictions possibles.

A retenir

- La méthode `addCallback` permet d'exécuter une fonction ActionScript depuis l'application conteneur.
- Dans le cas de l'utilisation d'une fonction JavaScript, des types comme `Number`, `Boolean` et `String` peuvent être échangés.

Communication et sécurité

Afin que les deux acteurs puissent communiquer, nous devons nous intéresser à la valeur passée à l'attribut `allowScriptAccess`.

La fonction `AC_FL_RunContent` intègre un paramètre `allowScriptAccess` régissant la communication entre l'application conteneur et le lecteur Flash :

```
  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-15-external-interface',
        'quality', 'high',
        'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
        'align', 'middle',
        'play', 'true',
        'loop', 'true',
        'scale', 'showall',
        'wmode', 'window',
        'devicefont', 'false',
        'id', 'monApplication',
        'bgcolor', '#ffffff',
        'name', 'monApplication',
        'menu', 'true',
        'allowFullScreen', 'false',
        'allowScriptAccess','sameDomain',
        'movie', 'chap-15-external-interface',
        'salign', ''
        );
```

Voici les trois valeurs possibles de l'attribut `allowScriptAccess` :

- `always` : La communication entre le l'application conteneur et ActionScript est toujours possible.
- `sameDomain` : La communication entre l'application conteneur et ActionScript est possible, uniquement si la page conteneur et le SWF évoluent dans le même domaine.
- `never` : La communication entre le l'application conteneur et ActionScript est impossible.

Nous pourrions nous demander dans quels cas l'utilisation de l'attribut `allowScriptAccess` serait nécessaire.

Imaginons le scénario suivant :

Au sein d'un forum, les utilisateurs ont la possibilité d'intégrer leurs avatars ou signature à partir d'animations Flash. Certaines d'entre elles pourraient scripter la page du forum provoquant alors un dysfonctionnement.

Afin de réguler cela, nous pourrions passer la valeur `never` à l'attribut `allowScriptAccess` empêchant toute communication entre les pages du forum et les avatars ou signatures.

Dans le code suivant, nous passons l'attribut `allowScriptAccess` à `never` :

```
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-15-external-interface',
	'quality', 'high',
	'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
	'align', 'middle',
	'play', 'true',
	'loop', 'true',
	'scale', 'showall',
	'wmode', 'window',
	'devicefont', 'false',
	'id', 'monApplication',
	'bgcolor', '#ffffff',
	'name', 'monApplication',
	'menu', 'true',
	'allowFullScreen', 'false',
	'allowScriptAccess','never',
	'movie', 'chap-15-external-interface',
	'salign', ''
);
```

Si nous tentons d'appeler à l'aide de la méthode `call` une fonction définie au sein de la page conteneur, une exception de type `SecurityError` est levée.

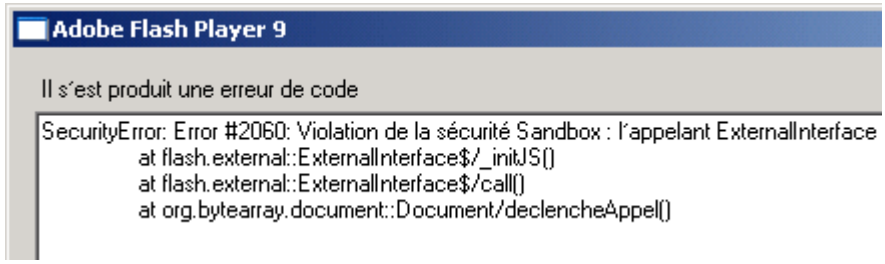

Figure 15-15. Restriction de sécurité.

A l'inverse, pour que la page conteneur puisse scripter un SWF provenant d'un domaine différent, nous pouvons utiliser la méthode `allowDomain` de la classe `Security` au sein du SWF à scripter.

Pour plus d'informations concernant le modèle de sécurité du lecteur Flash, reportez vous à la partie *Modèle de sécurité du lecteur Flash* du chapitre 13 intitulé *Chargement de contenu*.

A retenir

- La communication entre le lecteur Flash et l'application conteneur est soumise au modèle de sécurité du lecteur Flash.
- L'attribut `allowScriptAccess` permet d'autoriser ou non le lecteur Flash à scripter la page conteneur.