14

Chargement et envoi de données

LA CLASSE URLLOADER 1

CHARGER DU TEXTE 3
L'ENCODAGE URL 14
CHARGER DES VARIABLES 14
CHARGER DES DONNÉES XML 20

CHARGEMENT DE DONNÉES ET SECURITÉ 30

CHARGER DES DONNÉES BINAIRES 32

CONCEPT D'ENVOI DE DONNÉES 37

ENVOYER DES VARIABLES 38
LA MÉTHODE GET OU POST 45
ENVOYER DES VARIABLES DISCRÈTEMENT 49
RENVOYER DES DONNÉES DEPUIS LE SERVEUR 50
ALLER PLUS LOIN 56
ENVOYER UN FLUX BINAIRE 58

TÉLÉCHARGER UN FICHIER 58

PUBLIER UN FICHIER 68
PUBLIER PLUSIEURS FICHIERS 73
CRÉATION DE LA CLASSE ENVOIMULTIPLE 80
RETOURNER DES DONNÉES UNE FOIS L'ENVOI TERMINÉ 89

ALLER PLUS LOIN 93

La classe URLLoader

Afin de concevoir une application dynamique nous avons besoin de pouvoir envoyer ou charger des données externes au sein du lecteur Flash. Le chargement dynamique de données offre une grande souplesse de développement en permettant la mise à jour complète du contenu d'une application sans recompiler celle-ci.

Parmi les formats les plus couramment utilisés nous pouvons citer le texte ainsi que le XML mais aussi un format brut comme le binaire. Nous reviendrons en détail sur la manipulation de données binaire au cours du chapitre 20 intitulé *ByteArray*.

ActionScript 3 offre la possibilité de charger et d'envoyer ces différents types de données grâce à la classe `flash.net.URLLoader` qui remplace l'objet `LoadVars` ainsi que les différentes fonctions `loadVariables`, `loadVariablesNum` utilisées dans les précédentes versions d'ActionScript.

Nous retrouvons ici l'intérêt d'ActionScript 3 consistant à centraliser les fonctionnalités du lecteur.

Il est important de noter que contrairement à la classe `Loader`, la classe `URLLoader` diffuse directement les événements et ne dispose pas d'objet `LoaderInfo` interne. Ainsi l'écoute des différents événements se fait directement auprès de l'objet `URLLoader`.

En revanche, les événements diffusés sont quasiment les mêmes que la classe `LoaderInfo`. Voici le détail de chacun d'entre eux :

- `Event.OPEN` : diffusé lorsque le lecteur commence à charger les données.
- `ProgressEvent.PROGRESS` : diffusé lorsque le chargement est en cours. Celui-ci renseigne sur le nombre d'octets chargés et totaux.
- `Event.COMPLETE` : diffusé lorsque le chargement est terminé.
- `HTTPStatusEvent.HTTP_STATUS` : indique le code d'état de la requête HTTP.
- `IOErrorEvent.IO_ERROR` : diffusé lorsque le chargement échoue.
- `SecurityErrorEvent.SECURITY_ERROR` : diffusé lorsque le lecteur tente de charger des données depuis un domaine non autorisé.

Tout en gérant les différentes erreurs pouvant survenir lors du chargement de données, nous allons commencer notre apprentissage en chargeant de simples données au format texte, puis XML.

Puis nous traiterons en détail l'envoi et réception de variables mais aussi de fichiers grace aux classes `flash.net.FileReference` et `flash.net.FileReferenceList`.

A retenir

- Les fonctions et méthodes `loadVariables`, `loadVariablesNum` et la classe `LoadVars` sont remplacées par la classe `URLLoader`.
- La classe `URLLoader` permet de charger des données au format texte, XML et binaire.

Charger du texte

Afin de charger des données nous créons une instance de la classe `URLLoader`, puis nous utilisons la méthode `load` dont voici la signature :

```
public function load(request:URLRequest):void
```

Comme nous l'avons vu lors du précédent chapitre intitulé chargement de contenu, toute url doit être spécifiée au sein d'un objet `flash.net.URLRequest`. A l'aide d'un éditeur tel le bloc notes, nous créons un fichier texte nommé `donnees.txt` ayant le contenu suivant :

```
Voici le contenu du fichier texte !
```

Dans un nouveau document Flash CS3, nous créons une instance de la classe `URLLoader` puis nous chargeons le fichier texte :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  trace("données chargées");

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
  trace("code HTTP : " + pEvt.status);

}
```

```
function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");

}
```

A l'instar de la classe `flash.display.LoaderInfo` la classe `URLLoader` diffuse deux événements, une fois les données chargées. L'événement `HTTPStatsEvent.HTTP_STATUS` puis l'événement `Event.COMPLETE`.

> Souvenez-vous qu'en local la propriété `status` de l'objet événementiel `HTTPStatusEvent` vaut toujours 0, même si le chargement échoue.

Dans le code suivant, nous chargeons le même fichier texte depuis un serveur, la propriété `status` de l'objet événementiel renvoie 200 :

```
// chargement des données
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// chargement des données
chargeurDonnees.load ( new URLRequest
("http://www.monserveur.org/donnees.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  trace("données chargées");

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 200
  trace("code HTTP : " + pEvt.status);

}

function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");
```

```
}
```

Si le lecteur ne parvient pas à charger le fichier distant, l'événement `IoErrorEvent.IO_ERROR` est diffusé ainsi que l'événement `HTTPStatusEvent.HTTP_STATUS`. Dans ce cas la propriété `status` contient le code d'erreur HTTP permettant de connaître la raison de l'echec.

> Pour un tableau récapitulatif des différents codes d'erreurs possibles reportez-vous au chapitre 13 intitulé *Chargement de contenu*.

Afin d'accéder aux données chargées nous utilisons la propriété `data` de l'objet `URLLoader` :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // affiche : Voici le contenu du fichier texte !
  trace( contenu );

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
  trace("code HTTP : " + pEvt.status);

}

function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");
```

```
}
```

La propriété `dataFormat` de l'objet `URLLoader` permet de définir quel format de données nous chargeons. Celle-ci a la valeur `text` par défaut :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// affiche : text
trace( chargeurDonnees.dataFormat );
```

Il est recommandé pour des questions de lisibilité de code et de travail en équipe de toujours spécifier le type de données que nous chargeons même si il s'agit de données texte.

Pour cela nous utilisons trois propriétés constantes de la classe `flash.net.URLLoaderDataFormat`.

Voici le détail de chacune d'entre elles :

- `URLLoaderDataFormat.BINARY` : permet de charger des données au format binaire.
- `URLLoaderDataFormat.TEXT` : permet de charger du texte brut.
- `URLLoaderDataFormat.VARIABLES` : permet de charger des données url encodées.

Ainsi, même si nous souhaitons charger du texte, nous préférons l'indiquer pour des questions de lisibilité :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données texte
chargeurDonnees.dataFormat = URLLoaderDataFormat.TEXT;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // affiche : Voici le contenu du fichier texte !
  trace( contenu );
```

```
        // affiche : text
        trace( pEvt.target.dataFormat );

    }

    function codeHTTP ( pEvt:HTTPStatusEvent ):void

    {

        // affiche : 0
        trace("code HTTP : " + pEvt.status);

    }

    function erreurChargement ( pEvt:IOErrorEvent ):void

    {

        trace("erreur de chargement");

    }
```

Dans les précédentes versions d'ActionScript la classe `LoadVars` était utilisée afin de charger des données externes. Celle-ci possédait un événement `onData` nous permettant de récupérer les données chargées brutes, sans passer par une interprétation des données chargées.

L'équivalent n'existe plus en ActionScript 3. Si nous souhaitons charger des données brutes, nous utilisons le format `URLLoaderDataFormat.BINARY`.

Dans certains cas, le chargement de fichier texte peut être utile, en particulier lors de chargement de fichiers comme le CSV.

> Le CSV est un format simple de représentation de données sous forme de valeurs séparées par des virgules. Il est couramment utilisé en format d'export de logiciels comme Microsoft Excel ou Microsoft Outlook.

Dans l'exemple suivant nous chargeons un fichier CSV exporté depuis Microsoft Excel contenant des statistiques.

Voici un aperçu du contenu du fichier :

```
100
133.46
144.02
148
94.04
87.17
92.27
96.83
98.81
```

```
113.8
113.2
```

Dans le code suivant nous chargeons le fichier CSV en tant que données texte, puis nous transformons la chaîne en un tableau à l'aide de la méthode `split` de la classe `String` :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données texte
chargeurDonnees.dataFormat = URLLoaderDataFormat.TEXT;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.csv") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // transformation de la chaîne en un tableau en séparant les données à
chaque saut de ligne
  var tableauDonnees:Array = contenu.split("\n");

  // affiche : 100
  trace( tableauDonnees[0] );

  // affiche : 94
  trace( tableauDonnees.length );

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
  trace("code HTTP : " + pEvt.status);

}

function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");

}
```

Nous calculons l'amplitude du graphique en extrayant les valeurs minimum et maximum :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données texte
chargeurDonnees.dataFormat = URLLoaderDataFormat.TEXT;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.csv") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

var hauteurMaximum:Number = 180;

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // affiche : Voici le contenu du fichier texte !
  var tableauDonnees:Array = contenu.split("\n");

  // extraction des valeurs minimum et maximum
  var valeurMinimum:Number = calculeMinimum ( tableauDonnees );
  var valeurMaximum:Number = calculeMaximum ( tableauDonnees );

  // calcul de l'amplitude du graphique
  var ratio:Number = hauteurMaximum / ( valeurMaximum - valeurMinimum );

  // affiche : 1.2699308593198815
  trace( ratio );

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
  trace("code HTTP : " + pEvt.status);

}

function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");

}
```

```
function calculeMaximum ( pTableau:Array ):Number

{

  var lng:int = pTableau.length;
  var valeurMaximum:Number = Number.MIN_VALUE;

  while ( lng-- ) valeurMaximum = Math.max ( valeurMaximum, pTableau[lng] );

  return valeurMaximum;

}

function calculeMinimum ( pTableau:Array ):Number

{

  var lng:int = pTableau.length;
  var valeurMinimum:Number = Number.MAX_VALUE;

  while ( lng-- ) valeurMinimum = Math.min ( valeurMinimum, pTableau[lng] );

  return valeurMinimum;

}
```

Puis nous dessinons le graphique à l'aide de la fonction `dessineGaphique` :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données texte
chargeurDonnees.dataFormat = URLLoaderDataFormat.TEXT;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees.csv") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

var hauteurMaximum:Number = 180;

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // affiche : Voici le contenu du fichier texte !
  var tableauDonnees:Array = contenu.split("\n");

  // extraction des valeurs minimum et maximum
  var valeurMinimum:Number = calculeMinimum ( tableauDonnees );
  var valeurMaximum:Number = calculeMaximum ( tableauDonnees );
```

```
        // calcul de l'amplitude du graphique
        var ratio:Number = hauteurMaximum / ( valeurMaximum - valeurMinimum );

        // dessine le graphique
        dessineGraphique ( ratio, valeurMaximum, tableauDonnees );

      }

      function codeHTTP ( pEvt:HTTPStatusEvent ):void

      {

        // affiche : 0
        trace("code HTTP : " + pEvt.status);

      }

      function erreurChargement ( pEvt:IOErrorEvent ):void

      {

        trace("erreur de chargement");

      }

      function calculeMaximum ( pTableau:Array ):Number

      {

        var lng:int = pTableau.length;
        var valeurMaximum:Number = Number.MIN_VALUE;

        while ( lng-- ) valeurMaximum = Math.max ( valeurMaximum, pTableau[lng] );

        return valeurMaximum;

      }

      function calculeMinimum ( pTableau:Array ):Number

      {

        var lng:int = pTableau.length;
        var valeurMinimum:Number = Number.MAX_VALUE;

        while ( lng-- ) valeurMinimum = Math.min ( valeurMinimum, pTableau[lng] );

        return valeurMinimum;

      }

      var courbe:Shape = new Shape();

      courbe.graphics.lineStyle ( 1, 0x990000, 1 );

      var conteneurGraphique:Sprite = new Sprite();

      conteneurGraphique.addChild( courbe );

      conteneurGraphique.opaqueBackground = 0xFCEEBC;
```

```
addChild ( conteneurGraphique );

function dessineGraphique  ( pRatio:Number, pMaximum:Number, pDonnees:Array
):void

{

  for ( var i:int = 0; i< pDonnees.length; i++ )

  {

        if ( i == 0 ) courbe.graphics.moveTo ( i, (pMaximum-pDonnees[0]) *
pRatio );

        else courbe.graphics.lineTo ( i * 4, (pMaximum-pDonnees[i]) * pRatio
);

  }

  // centre le graphique
  conteneurGraphique.x = ( stage.stageWidth - conteneurGraphique.width ) / 2;
  conteneurGraphique.y = ( stage.stageHeight - conteneurGraphique.height ) /
2;

}
```

La figure 14-1 illustre le graphique généré depuis les données extraites du fichier CSV :

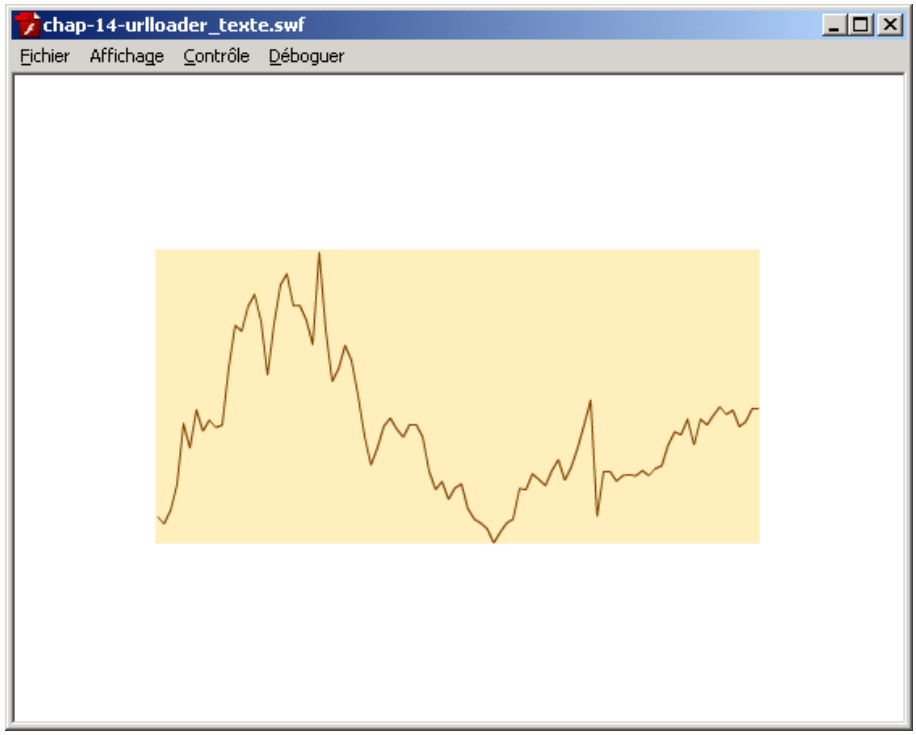

Figure 14-1. Courbe de statistique.

Si nous modifions la valeur de la variable `hauteurMaximum` :

```
var hauteurMaximum:Number = 80;
```

Nous voyons que le graphique est dessiné en conservant les proportions, comme l'illustre la figure 14-2 :

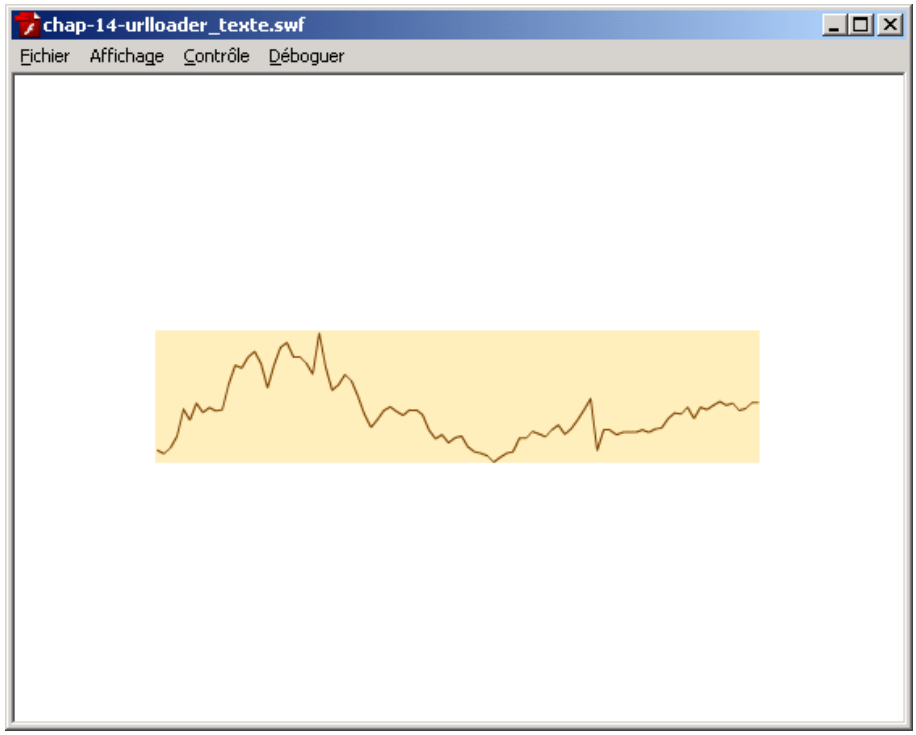

Figure 14-2. Courbe de statistique réduite.

Bien entendu, d'autres formats de données peuvent être utilisés. Nous allons nous intéresser au chargement de variables encodées au format URL.

A retenir

- Afin de charger des données au format texte, nous passons la valeur `URLLoaderDataFormat.TEXT` à la propriété `dataFormat`.
- L'objet `URLLoader` ne possède pas d'objet `LoaderInfo` interne.
- Afin d'accéder aux données chargées, nous utilisons la propriété `data` de l'objet `URLLoader`.

L'encodage URL

L'encodage URL est le standard de transmition d'informations par formulaire. Il permet de rendre compatible différents paramètres avec le protocole HTTP.

L'encodage respecte les règles suivantes :

- Les champs sont séparés par des eperluettes (caractère &)
- Un champ comporte un nom de champ, suivi de = puis de la valeur.

- Les espaces sont remplacés par des +.
- Les caractères non alphanumériques sont remplacés par %XX où XX représente le code ASCII en hexadécimal du caractère.

L'encodage URL va être utilisé au cours des prochaines parties afin de charger ou d'envoyer des variables.

Charger des variables

Le chargement de données texte brut convient lorsque nous chargeons des données non structurées. Pour des données plus détaillées nous pouvons utiliser le format d'encodage URL détaillé précédemment.

La chaîne suivante est une chaîne encodée URL :

```
titre=Voici un titre&contenu=Voici le contenu !
```

Nous allons placer la chaîne précédente au sein d'un fichier texte nommé `donnees_url.txt`.

Au sein d'un nouveau document Flash CS3, nous chargeons le fichier texte à l'aide de la méthode `load` :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données texte
chargeurDonnees.dataFormat = URLLoaderDataFormat.TEXT;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees_url.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );

// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );

// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  // accès aux données chargées
  var contenu:String = pEvt.target.data;

  // affiche : titre=Voici un titre&contenu=Voici le contenu !
  trace( contenu );

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
```

```
    trace("code HTTP : " + pEvt.status);

  }

  function erreurChargement ( pEvt:IOErrorEvent ):void

  {

    trace("erreur de chargement");

  }
```

Les données sont ainsi chargées en tant que données texte brut. Il nous est difficile d'accéder pour le moment à la valeur de chaque variable `titre` ou `contenu`.

Afin d'interpréter et d'extraire des données de la chaîne encodée, deux possibilités s'offrent à nous :

La première consiste à décoder les données texte à l'aide de la classe `flash.net.URLVariables`. En passant la chaîne encodée URL au constructeur de celle-ci nous la décodons afin d'extraire la valeur de chaque variable :

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

  {

    // accès aux données chargées
    var contenu:String = pEvt.target.data;

    // décodage de la chaîne encodée url sous forme d'objet
    var variables:URLVariables = new URLVariables ( contenu );

    // itération sur chaque variable décodée
    /*
    affiche :
    titre --> Voici un titre
    contenu --> Voici le contenu !
    */
    for ( var p in variables ) trace( p, "--> " + variables[p] );

  }
```

Une fois l'objet `URLVariables` crée, nous accédons à chaque variable par la syntaxe pointée.

> Nous retrouvons ici le même comportement que la classe `LoadVars` des précédentes versions d'ActionScript. Les variables chargées deviennent des propriétés de l'objet `URLVariables`.

Dans le code suivant, nous accédons manuellement aux deux variables `titre` et `contenu` devenues propriétés :

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

```
{

  // accès aux données chargées
  var donnees:String = pEvt.target.data;

  // décodage de la chaîne url encodée sous forme d'objet
  var variables:URLVariables = new URLVariables ( donnees );

  var titre:String = variables.titre;

  // affiche : Voici un titre
  trace( titre );

  var contenu:String = variables.contenu;

  // affiche : Voici le contenu !
  trace( contenu );

}
```

Au sein de notre document, nous plaçons deux champs texte dynamique auxquels nous associons deux noms d'occurrence. Puis nous remplissons dynamiquement ces derniers avec le contenu chargé correspondant :

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

{

  // accès aux données chargées
  var donnees:String = pEvt.target.data;

  // décodage de la chaîne url encodée sous forme d'objet
  var variables:URLVariables = new URLVariables ( donnees );

  var titre:String = variables.titre;
  var contenu:String = variables.contenu;

  // affectation des données chargées au champs texte
  champTitre.text = titre;
  champContenu.text = contenu;

}
```

La figure 14-3 illustre le résultat :

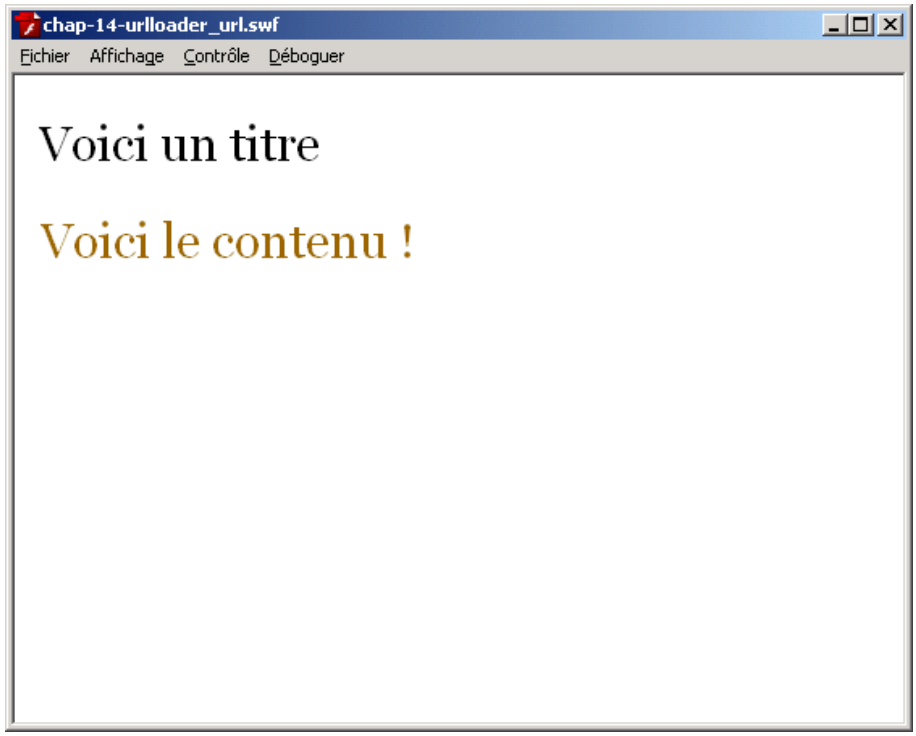

Figure 14-3. Contenu texte chargé dynamiquement.

Une seconde approche plus rapide consiste à spécifier au préalable à la propriété `dataFormat` la valeur `URLLoaderDataFormat.VARIABLES`.

Ainsi, les données chargées sont automatiquement décodées par un objet `URLVariables` :

```
// création de l'objet URLLoader
var chargeurDonnees:URLLoader = new URLLoader();

// nous souhaitons charger des données url encodées
chargeurDonnees.dataFormat = URLLoaderDataFormat.VARIABLES;

// chargement des données
chargeurDonnees.load ( new URLRequest ("donnees_url.txt") );

// écoute de l'événement Event.COMPLETE
chargeurDonnees.addEventListener( Event.COMPLETE, chargementTermine );
// écoute de l'événement HTTPStatusEvent.HTTP_STATUS
chargeurDonnees.addEventListener( HTTPStatusEvent.HTTP_STATUS, codeHTTP );
// écoute de l'événement IOErrorEvent.IO_ERROR
chargeurDonnees.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );

function chargementTermine ( pEvt:Event ):void

{

  // accès à l'objet URLVariables
  var variables:URLVariables = pEvt.target.data;

  var titre:String = variables.titre;
```

```
  var contenu:String = variables.contenu;

  // affectation des données chargées au champs texte
  champTitre.text = titre;
  champContenu.text = contenu;

}

function codeHTTP ( pEvt:HTTPStatusEvent ):void

{

  // affiche : 0
  trace("code HTTP : " + pEvt.status);

}

function erreurChargement ( pEvt:IOErrorEvent ):void

{

  trace("erreur de chargement");

}
```

Lorsque nous demandons explicitement de charger les données sous la forme de variables encodées URL, la propriété `data` de l'objet `URLLoader` retourne un objet `URLVariables` crée automatiquement par le lecteur Flash.

La classe `URLVariables` définie une méthode `decode` appelée en interne lors de l'interprétation de la chaîne encodée. Celle-ci peut aussi être appelée manuellement lorsque l'objet `URLVariables` est déjà crée et que nous souhaitons décoder une nouvelle chaîne.

Au cas où les variables chargées ne seraient pas formatées correctement, l'appel de celle-ci lève une erreur à l'exécution.

Dans le cas de la chaîne suivante :

```
Voici un titre&contenu=Voici le contenu !
```

Nous avons omis volontairement la première variable `titre` associée au contenu `Voici un titre`. Si nous testons à nouveau le code précédent, l'erreur suivante est levée lors de l'événement `Event.COMPLETE` :

```
Error: Error #2101: La chaîne transmise à URLVariables.decode() doit être une
requête au format de code URL contenant des paires nom/valeur.
```

Si le texte chargé contient le caractère & et que nous ne souhaitons pas l'interpréter comme séparateur mais comme caractère composant une chaîne, nous pouvons indiquer son code caractère ASCII au format héxadécimal à l'aide du symbole %.

Ainsi, pour ajouter comme titre la chaîne de caractère suivante :

```
Bobby & The Groove Orchestra
```

Nous remplaçons le caractère & par son code caractère ASCII de la manière suivante :

```
titre=Bob %26 The Groove Orchestra&contenu=Sortie de l'album en Août 2008 !
```

La figure 14-4 illustre le résultat :

Figure 14-4. Caractère spécial encodé en hexadécimal.

Malheureusement, pour des questions de lisibilité et d'organisation, le format encodé URL s'avère rapidement limité. Nous préférerons l'utilisation du format XML qui s'avère être un format standard et universel adapté à la représentation de données complexes.

A retenir

- Afin de décoder une chaîne encodée URL nous utilisons un objet `URLVariables`.
- Afin de charger des variables encodées URL, nous passons la valeur `URLLoaderDataFormat.VARIABLES` à la propriété `dataFormat`.
- Afin de décoder une chaine de caractères encodée URL nous pouvons la passer au constructeur de la classe `URLVariables` où à la méthode `decode`.

Charger des données XML

Si nous devions représenter une petite partie de la discographie de Stevie Wonder au format encodé URL nous obtiendrons la chaîne suivante :

```
&album_1=Talking Book&artiste_1=Stevie
Wonder&label_1=Motown&annee_1=1972&album_2=Songs in the key of
life&artiste_2=Stevie Wonder&label_2=Motown&annee_2=1976
```

Les mêmes données au format XML seraient formatées de la manière suivante :

```
<DISCOGRAPHIE>
    <ALBUM>
        <TITRE>
            Talking Book
        </TITRE>
        <ANNEE DATE="1972"/>
        <ARTISTE NOM="Wonder" PRENOM="Stevie"/>
        <LABEL NOM="Motown"/>
    </ALBUM >
    <ALBUM>
        <TITRE>
            Songs in the key of life
        </TITRE>
        <ANNEE DATE="1976"/>
        <ARTISTE NOM="Wonder" PRENOM="Stevie"/>
        <LABEL NOM="Motown"/>
    </ALBUM >
</DISCOGRAPHIE>
```

La représentation XML est plus naturelle et plus adaptée dans le cas de données structurées.

Nous avons manipulé le format XML au cours du chapitre 2 intitulé *Langage et API du lecteur Flash*. Nous allons découvrir comment charger dynamiquement un fichier XML afin de construire une interface graphique.

En ActionScript 3, la classe XML ne fait plus partie de l'API du lecteur Flash, mais appartient au langage ActionScript 3 reposant sur ECMAScript, c'est donc une classe *haut niveau*.

La classe XML ne gère donc plus le chargement de données comme c'était le cas dans les précédentes versions d'ActionScript. Afin de charger des données XML, nous chargeons simplement une chaîne de caractères sous la forme de texte brut, puis nous la transformons en objet XML.

Dans un nouveau document Flash CS3, nous associons la classe de document suivante :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

       public function Document ()

       {

       }


  }

}
```

A côté du document Flash en cours nous sauvons un fichier XML sous le nom `donnees.xml` contenant les données suivantes :

```
<MENU>
<BOUTON legende="Accueil" couleur="0x887400" vitesse="1" swf="accueil.swf"/>
<BOUTON legende="Photos" couleur="0x005587" vitesse="1" url="photos.swf"/>
<BOUTON legende="Blog" couleur="0x125874" vitesse="1" url="blog.swf"/>
<BOUTON legende="Liens" couleur="0x59CCAA" vitesse="1" url="liens.swf"/>
<BOUTON legende="Forum" couleur="0xEE44AA" vitesse="1" url="forum.swf"/>
</MENU>
```

Puis nous le chargeons :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {
```

```
        private var chargeur:URLLoader;

        public function Document ()

        {

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.TEXT;

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              chargeur.load ( new URLRequest ("donnees.xml") );

        }

        private function chargementTermine ( pEvt:Event ) :void

        {

              var donnneesXML:XML = new XML ( pEvt.target.data );

              /*
              affiche :
              <MENU>
                    <BOUTON legende="Accueil" couleur="0x887400" vitesse="1"
url="accueil.swf"/>
                    <BOUTON legende="Photos" couleur="0x005587" vitesse="1"
url="photos.swf"/>
                    <BOUTON legende="Blog" couleur="0x125874" vitesse="1"
url="blog.swf"/>
                    <BOUTON legende="Liens" couleur="0x59CCAA" vitesse="1"
url="liens.swf"/>
                    <BOUTON legende="Forum" couleur="0xEE44AA" vitesse="1"
url="forum.swf"/>
              </MENU>
              */
              trace( donnneesXML );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

              // affiche : 0
              trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

              trace("erreur de chargement");

        }
```

```
  }

}
```

Lorsque la méthode écouteur `chargementTermine` se déclenche nous accédons aux données chargées puis nous transformons la chaîne de caractères en objet XML.

Nous allons reprendre le menu construit lors du chapitre 10 intitulé *Diffusion d'événements personnalisés* afin de charger dynamiquement les données du menu depuis un fichier XML.

Afin de créer notre menu, nous reprenons la classe `Bouton` utilisée lors du chapitre 10 puis nous l'importons ainsi que la classe `Sprite` :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.ui.Bouton;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;
  import flash.display.Sprite;

  public class Document extends ApplicationDefaut

  {

        private var chargeur:URLLoader;
        private var conteneurMenu:Sprite;

        public function Document ()

        {

              conteneurMenu = new Sprite();

              addChild ( conteneurMenu );

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.TEXT;

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              chargeur.load ( new URLRequest ("donnees.xml") );

        }

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

```
        {

            var donnneesXML:XML = new XML ( pEvt.target.data );

            creerMenu ( donnneesXML );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

            // affiche : 0
            trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

            trace("erreur de chargement");

        }

        private function creerMenu ( pXML:XML ):void

        {

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

            for each ( var enfant:XML in pXML.* )

            {

                // récupération des infos
                var legende:String = enfant.@legende;
                var couleur:Number = enfant.@couleur;
                var vitesse:Number = enfant.@vitesse;
                var swf:String = enfant.@url;

                // création des boutons
                monBouton = new Bouton( 60, 40, swf, couleur, vitesse,
legende );

                // positionnement
                monBouton.y = (monBouton.height + 5) * i;

                // ajout à la liste d'affichage
                conteneurMenu.addChild ( monBouton );

                i++;

            }

        }

    }

}
```

La figure 14-4 illustre le résultat :

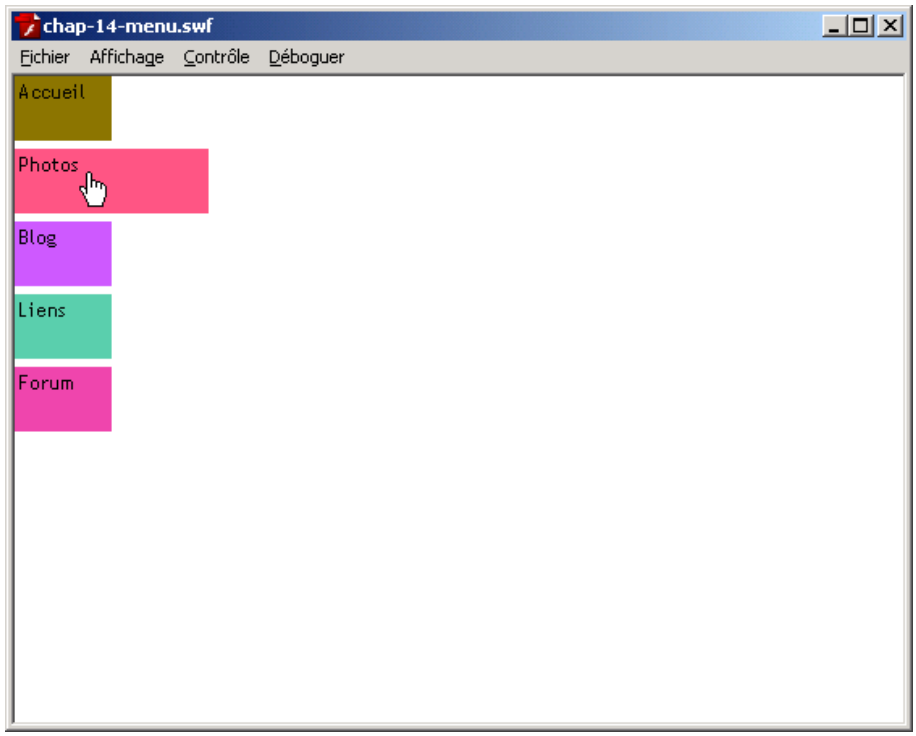

Figure 14-4. Menu dynamique XML.

Afin d'écouter chaque clic bouton, nous importons la classe `EvenementBouton` créée lors du chapitre 10 puis nous ciblons l'événement `EvenementBouton.CLICK` en utilisant la phase de capture :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.ui.Bouton;
  import org.bytearray.evenements.EvenementBouton;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;
  import flash.display.Sprite;

  public class Document extends ApplicationDefaut

  {

        private var chargeur:URLLoader;
        private var conteneurMenu:Sprite;

        public function Document ()
```

```
        {

            conteneurMenu = new Sprite();

            addChild ( conteneurMenu );

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

            chargeur = new URLLoader();

            chargeur.dataFormat = URLLoaderDataFormat.TEXT;

            chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
            chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
            chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

            chargeur.load ( new URLRequest ("donnees.xml") );

        }

        private function chargementTermine ( pEvt:Event ) :void

        {

            var donnneesXML:XML = new XML ( pEvt.target.data );

            creerMenu ( donnneesXML );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

            // affiche : 0
            trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

            trace("erreur de chargement");

        }

        private function creerMenu ( pXML:XML ):void

        {

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

            for each ( var enfant:XML in pXML.* )

            {

                // récupération des infos
```

```
                var legende:String = enfant.@legende;
                var couleur:Number = enfant.@couleur;
                var vitesse:Number = enfant.@vitesse;
                var swf:String = enfant.@url;

                // création des boutons
                monBouton = new Bouton( 60, 40, swf, couleur, vitesse,
legende );

                // positionnement
                monBouton.y = (monBouton.height + 5) * i;

                // ajout à la liste d'affichage
                conteneurMenu.addChild ( monBouton );

                i++;

            }

        }

        private function clicBouton( pEvt:EvenementBouton ):void

        {

            // affiche : photos.swf
            trace( pEvt.lien );

        }

    }

}
```

Nous avons ici réutilisé la classe `Bouton` créée lors du chapitre 10, en prévoyant un modèle simple d'utilisation nous avons pu réutiliser cette classe sans aucun problème.

Afin de désactiver totalement notre menu, nous devons supprimer l'objet `conteneurMenu` de la liste d'affichage, puis passer sa référence à `null` :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.ui.Button;
  import org.bytearray.events.ButtonEvent;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;
  import flash.events.MouseEvent;
  import flash.display.Sprite;

  public class Document extends ApplicationDefaut
```

```
  {

        private var chargeur:URLLoader;
        private var conteneurMenu:Sprite;

        public function Document ()

        {

              conteneurMenu = new Sprite();

              addChild ( conteneurMenu );

              stage.doubleClickEnabled = true;

              stage.addEventListener ( MouseEvent.DOUBLE_CLICK, desactive );

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

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.TEXT;

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              chargeur.load ( new URLRequest ("donnees.xml") );

        }

        private function chargementTermine ( pEvt:Event ) :void

        {

              var donnneesXML:XML = new XML ( pEvt.target.data );

              creerMenu ( donnneesXML );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

              // affiche : 0
              trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

              trace("erreur de chargement");

        }

        private function creerMenu ( pXML:XML ):void
```

```
            {

                var i:int = 0;
                var monBouton:Button;

                for each ( var enfant:XML in pXML.* )

                {

                    // récupération des infos
                    var legende:String = enfant.@legende;
                    var couleur:Number = enfant.@couleur;
                    var vitesse:Number = enfant.@vitesse;
                    var swf:String = enfant.@url;

                    // création des boutons
                    monBouton = new Button( 60, 40, swf, couleur, vitesse,
    legende );

                    // positionnement
                    monBouton.y = (monBouton.height + 5) * i;

                    // ajout à la liste d'affichage
                    conteneurMenu.addChild ( monBouton );

                    i++;

                }

            }

            private function clicBouton( pEvt:ButtonEvent ):void

            {

                // affiche : photos.swf
                trace( pEvt.lien );

            }

            private function desactive ( pEvt:MouseEvent ):void

            {

                removeChild ( conteneurMenu );
                conteneurMenu = null;

            }

        }

    }
```

Souvenez-vous que pour correctement désactiver un élément interactif, nous devons supprimer toutes ses références.

Dans cet exemple, les boutons sont seulement référencés de par leur présence au sein de l'objet graphique `conteneurMenu`. En désactivant ce dernier nous rendons alors inaccessible ses enfants. Ces

derniers deviennent ainsi éligible à la suppression par le ramasse miettes.

Nous avons vu lors du chapitre précédent que le modèle de securité du lecteur intégrait des restrictions conçernant le chargement de contenu externe. Nous allons au cours de la partie suivante nous intéresser aux restrictions de securité dans un contexte de chargement de données.

A retenir

- Afin de charger un flux XML, nous passons la valeur `URLLoaderDataFormat.TEXT` à la propriété `dataFormat` de l'objet `URLLoader`.
- La classe XML ne s'occupe plus du chargement du flux XML.
- Une fois la chaîne de caractères chargée par l'objet `URLLoader`, nous créons un objet XML à partir de celle-ci.

Chargement de données et securité

Le chargement de données est soumis aux mêmes restrictions de sécurité que le chargement de contenu.

Imaginons le scénario suivant :

Vous devez développer une application Flash permettant de lire des flux XML provenant de différents blogs. Cette application sera hebergée sur votre serveur et ne peut pour des raisons de securité accéder aux flux distants.

Au cours du chapitre 13, nous avons vu qu'il était possible d'autoriser de trois manières un SWF tentant de charger du contenu depuis un serveur distant :

- Par un fichier de régulation (XML).
- Par l'appel de la méthode `allowDomain` de la classe `flash.system.Security` dans le SWF à charger.
- Par l'utilisation d'un fichier de proxy.

Attention, dans notre cas, nous ne chargeons plus de contenu SWF. Il est donc impossible d'appeler la méthode `allowDomain` de l'objet `Security`.

Ainsi, dans le cas de chargement de flux XML, texte, ou autres seules deux méthodes d'autorisations s'offrent à vous :

Par un fichier de régulation XML comme l'illustre la figure 14-5 :

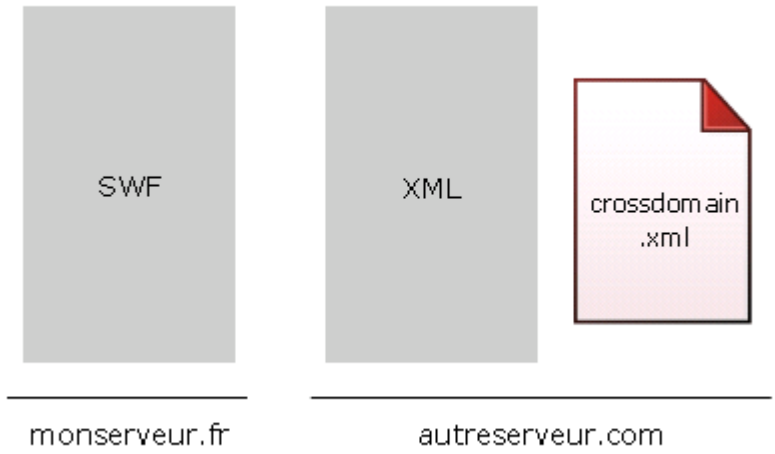

Figure 14-5. Autorisation par fichier de régulation.

Ou bien par l'utilisation d'un fichier proxy :

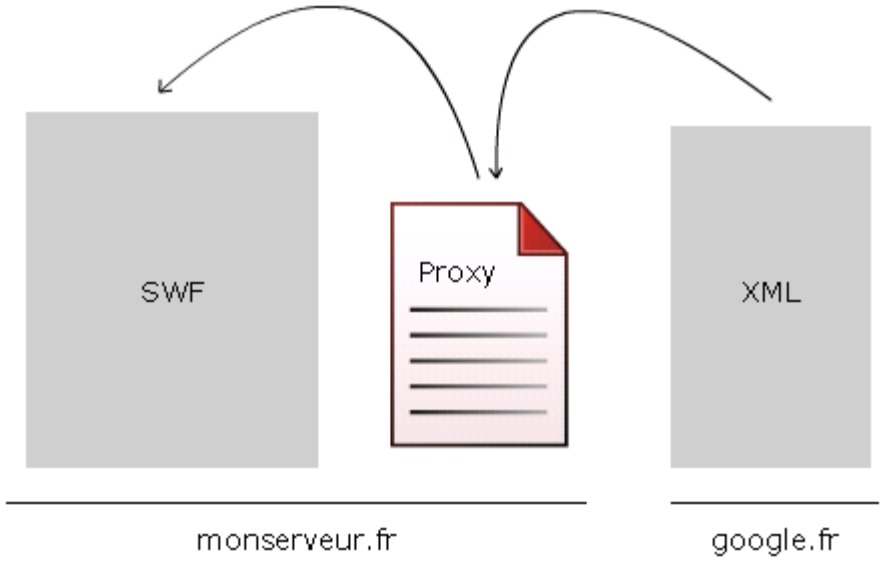

Figure 14-6. Utilisation d'un proxy.

Reportez vous au chapitre 13 intitulé chargement de contenu pour plus d'informations relatives au modèle de sécurité du lecteur Flash 9 et l'utilisation de fichier de régulation ou proxy.

A retenir

- Les mêmes restrictions de securité liées au chargement de contenu, s'appliquent lors du chargement de données.
- Il est impossible de placer au sein des données chargées, un appel à la méthode `allowDomain` de l'objet `Security`.

Charger des données binaires

La grande puissance du lecteur Flash 9 en ActionScript 3 réside dans la lecture de données au format binaire grâce à la classe bas niveau `flash.utils.ByteArray`. Afin de charger des données binaires brutes, nous devons passer à la propriété `dataFormat` la valeur `URLLoaderDataFormat.BINARY`.

Dans un nouveau document Flash CS3, nous chargeons un fichier PSD en associant une classe du document contenant le code suivant :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.URLRequest;
  import flash.utils.ByteArray;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var chargeur:URLLoader;

        public function Document ()

        {

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.BINARY;

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              chargeur.load ( new URLRequest ("maquette.psd") );

        }

        private function chargementTermine ( pEvt:Event ) :void

        {
```

```
            var donneesBinaire:ByteArray = pEvt.target.data;

            // affiche : 182509
            trace( donneesBinaire.length );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

            // affiche : 0
            trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

            trace("erreur de chargement");

        }

    }

}
```

La variable `donneesBinaire` contient le flux binaire du fichier PSD chargé. En créant une classe `EntetePSD`, nous allons lire l'entête du fichier afin d'extraire différentes informations comme la taille du document, la version, ainsi que le modèle de couleur utilisé.

Pour cela, nous créons une classe `EntetePSD` contenant le code suivant :

```
package org.bytearray.psd

{

  import flash.utils.ByteArray;

  public class EntetePSD

  {

        private var flux:ByteArray;
        private var _signature:String;
        private var _version:int;
        private var _canal:int;
        private var _hauteur:int;
        private var _largeur:int;
        private var _profondeur:int;
        private var _mode:int;

        private static const MODES_COULEURS:Array = new Array ("Bitmap", "Mode
niveaux de gris", "Indexé", "RVB", "CMJN", "Multi Canal", "Deux tons",
"Lab");

        public function EntetePSD ( pFlux:ByteArray )
```

```
        {

            flux = pFlux;

            // extrait la signature de l'entête PSD (doit être 8BPS)
            _signature = flux.readUTFBytes(4);

            // extrait la version (doit êter égal à 1)
            _version = flux.readUnsignedShort();

            // nous sautons 6 octets
            flux.position += 6;

            // extrait le canal utilisé
            _canal = flux.readUnsignedShort();

            // extrait la largeur du document
            _largeur = flux.readInt();

            // extrait la hauteur du document
            _hauteur = flux.readInt();

            // bpp
            _profondeur = flux.readUnsignedShort();

            // mode colorimétrique (Bitmap=0, Mode niveaux de gris=1,
Indexé=2, RVB=3, CMJN=4, Multi Canal=7, Deux tons=8, Lab=9)
            _mode = flux.readUnsignedShort();

        }

        public function toString ( ):String

        {

            return "[EntetePSD signature : " + signature +",  version : " +
version + ", canal : " + canal + ", largeur : " +
                    largeur + ", hauteur : " + hauteur + ", profondeur : " +
profondeur + ", mode colorimétrique : " + MODES_COULEURS [ mode ] +"]";

        }

        public function get signature ():String

        {

            return _signature;

        }

        public function get version ():int

        {

            return _version;

        }

        public function get canal ():int

        {
```

```
                return _canal;
            }
            public function get largeur ():int
            {
                return _largeur;
            }
            public function get hauteur ():int
            {
                return _hauteur;
            }
            public function get profondeur ():int
            {
                return _profondeur;
            }
            public function get mode ():int
            {
                return _mode;
            }
        }
    }
```

Le flux binaire généré par le lecteur Flash est passé au constructeur, puis nous lisons le flux à l'aide des méthodes de la classe `ByteArray`. Nous reviendrons sur celles-ci au cours du chapitre 20 intitulé *ByteArray*.

Afin d'extraire les informations du PSD nous instancions la classe `EntetePSD` en passant le flux binaire au constructeur :

```
package org.bytearray.document
{
  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.psd.EntetePSD;
  import flash.net.URLRequest;
  import flash.utils.ByteArray;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.events.Event;
```

```
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var chargeur:URLLoader;

        public function Document ()

        {

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.BINARY;

              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS, codeHTTP
);
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              chargeur.load ( new URLRequest ("maquette.psd") );

        }

        private function chargementTermine ( pEvt:Event ) :void

        {

              var donneesBinaire:ByteArray = pEvt.target.data;

              var infosPSD:EntetePSD = new EntetePSD( donneesBinaire );

              // affiche : [EntetePSD signature : 8BPS,  version : 1, canal :
3, largeur : 450, hauteur : 562, profondeur : 8, mode colorimétrique : RVB]
              trace( infosPSD );

        }

        private function codeHTTP ( pEvt:HTTPStatusEvent ):void

        {

              // affiche : 0
              trace("code HTTP : " + pEvt.status);

        }

        private function erreurChargement ( pEvent:IOErrorEvent ):void

        {

              trace("erreur de chargement");

        }

  }

}
```

Grâce à la classe `ByteArray` nous pouvons charger n'importe quel type de fichiers puis en extraire des informations. Nous pourrions imaginer une application RIA permettant d'héberger tout type de fichier. Celle-ci pourrait extraire les informations provenant de fichiers PSD, AI, FLA ou autres.

Nous pourrions optimiser la classe `EntetePSD` en diffusant un événement personnalisé `EvenementEntetePSD.INFOS`. L'objet événementiel diffusé contiendrait toutes les propriétés nécessaires à la description du fichier PSD.

La classe `ByteArray` ouvre des portes à toutes sortes de possibilités. Nous reviendrons en détail sur la puissance de cette classe au cours du chapitre 20 intitulé *ByteArray*.

A retenir

- Afin de charger un flux binaire, nous passons la valeur `URLLoaderDataFormat.BINARY` à la propriété `dataFormat` de l'objet `URLLoader`.
- Le flux binaire généré est representé par la classe `flash.utils.ByteArray`.

Concept d'envoi de données

Comme nous l'avons vu lors du chapitre 13 intitulé *Chargement de contenu*, toute URL doit être spécifiée au sein d'un objet `URLRequest`.

Celui-ci offre pourtant bien d'autres fonctionnalités que nous n'avons pas encore exploitées. Nous allons nous intéresser au cours des prochaines parties au concept d'envoi de données.

Pour cela, voyons en détail les propriétés de la classe `URLRequest` :

- `contentType` : type de contenu MIME des données envoyées en POST.
- `data` : contient les données à envoyer. Peut être de type `String`, `URLVariables` ou `ByteArray`.
- `method` : permet d'indiquer si les données doivent être envoyées par la méthode GET ou POST.
- `requestHeaders` : tableau contenant les entêtes HTTP définies par des objets `flash.net.URLRequestHeader`.
- `url` : contient l'url à atteindre.

Nous allons au cours des exercices suivants utiliser ces différentes propriétés afin de découvrir leurs intérêts.

Au sein du lecteur Flash nous pouvons distinguer trois types d'envoi de données :

- Envoi simple : les variables sont envoyées à l'aide d'une nouvelle fenêtre navigateur.
- Envoi discret : l'envoi des données est transparent pour l'utilisateur. Aucune fenêtre navigateur n'est ouverte durant l'envoi.
- Envoi discret avec validation : l'envoi des données est transparent, le lecteur Flash reçoit un retour du serveur permettant une validation d'envoi des données au sein de l'application Flash.

Nous allons traiter chacun des cas et comprendre les différences entre chaque approche.

Envoyer des variables

Nous allons commencer par un envoi simple de données en développant un formulaire permettant d'envoyer un email depuis Flash. Ce type de développement peut être intégré dans une rubrique « Contactez-nous » au sein d'un site.

Nous verrons qu'il est possible sous certaines conditions d'envoyer un email depuis le lecteur Flash sans passer par un script serveur grâce à la classe `flash.net.Socket` que nous traiterons au cours du chapitre 19 intitulé *Les sockets*.

Par défaut, le lecteur Flash n'a pas la capacité d'envoyer un email de manière autonome. Afin d'y parvenir, nous devons passer les informations nécessaires à un script serveur afin que celui-ci puisse envoyer le message.

Dans un nouveau document Flash CS3, nous associons la classe du document suivante :

```
package org.bytearray.document
{
  import org.bytearray.abstrait.ApplicationDefaut;
  public class Document extends ApplicationDefaut
  {
        public function Document ()
        {
        }
```

```
    }
}
```

Puis nous plaçons trois champs texte de saisie et un bouton afin de créer une interface d'envoi de mail.

La figure 14-7 illustre l'interface :

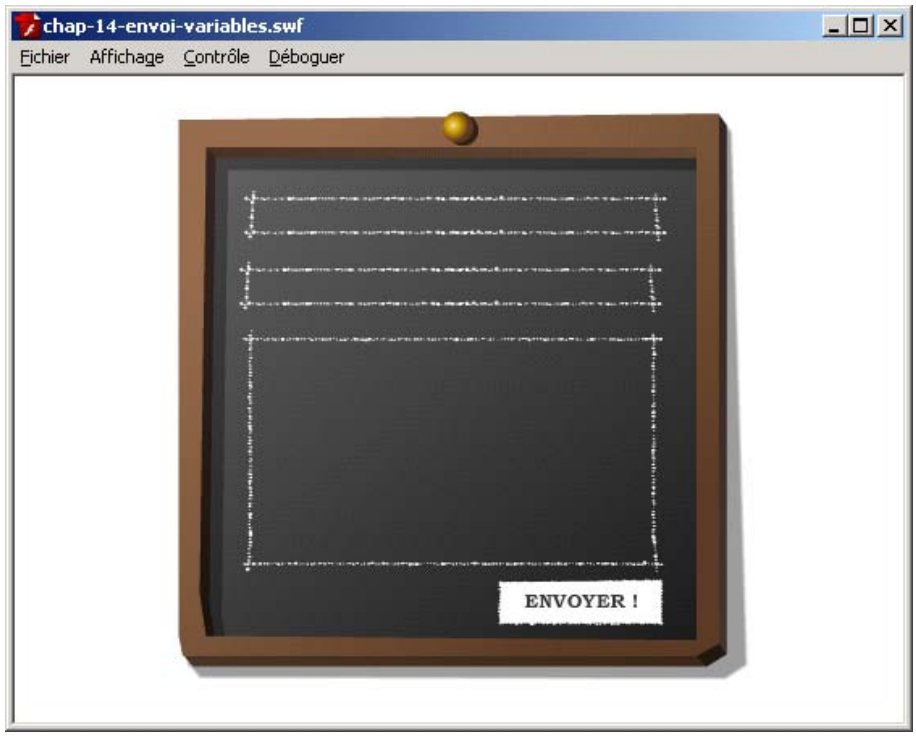

Figure 14-7. Formulaire d'envoi d'email.

Chaque instance est définie au sein de la classe du document :

```
package org.bytearray.document

{

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

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;

        public function Document ()
```

```
        {


        }
    }
}
```

Nous allons créer à présent le code PHP nous permettant de réceptionner les variables transmises depuis Flash. Nous allons utiliser dans un premier temps la méthode GET.

Les variables sont ainsi accessible par l'intermédiaire du tableau associatif `$_GET` :

```
<?php

$destinataire = $_GET ["destinataire"];
$sujet = $_GET ["sujet"];
$message = $_GET ["message"];

if ( isset ( $destinataire ) && isset ( $sujet ) && isset ( $message ) )

{

  echo $destinataire. "<br>".  $sujet. "<br>" . $message;

} else echo "Variables non transmises";

?>
```

Il convient de placer ce script sur un serveur local ou distant afin de pouvoir tester l'application. Dans notre cas, le script serveur est placé sur un serveur local. Le script est donc accessible en `localhost` :

```
http://localhost/mail/envoiMail.php
```

Nous ajoutons à présent le code nécessaire afin d'envoyer les variables à notre script distant :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.display.SimpleButton;
  import flash.text.TextField;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import flash.events.MouseEvent;

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;
```

```
            public function Document ()

            {

                  boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

            }

            private function envoiMail ( pEvt:MouseEvent ):void

            {

                  // affectation des variables à envoyer coté serveur
                  var destinaireEmail:String = destinataire.text;
                  var sujetEmail:String = sujet.text;
                  var messageEmail:String = message.text;

                  // création de l'objet URLRequest
                  var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

                  // ouvre une nouvelle fenêtre navigateur et envoi les variables
                  navigateToURL ( requete );

            }

      }

}
```

Nous stockons le destinataire, le sujet ainsi que le message au sein de trois variables. Puis nous les ajoutons en fin d'url du script distant de la manière suivante :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.display.SimpleButton;
  import flash.text.TextField;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import flash.events.MouseEvent;

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;

        public function Document ()

        {

              boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

        }
```

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

        {

            // affectation des variables à envoyer coté serveur
            var destinaireEmail:String = destinataire.text;
            var sujetEmail:String = sujet.text;
            var messageEmail:String = message.text;

            var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php?destinataire="+destinaireEmail+"&sujet=
"+sujetEmail+"&message="+messageEmail);

            // ouvre une nouvelle fenêtre navigateur et envoi les variables
            navigateToURL ( requete );

        }

    }

}
```

Le point d'interrogation au sein de l'URL indique au navigateur que le texte suivant contient les variables représentées par des paires noms/valeurs séparées par des esperluettes (caractère &).

Une fois les informations saisies comme l'illustre la figure 14-8 :

Figure 14-8. Formulaire d'envoi d'email.

Nous cliquons sur le bouton `boutonEnvoi`, une nouvelle fenêtre navigateur s'ouvre, les variables sont automatiquement placées en fin d'url :

```
http://localhost/mail/envoiMail.php?destinataire=thibault@bytearray.org&sujet
=Infos%20Album%202008%20!&message=Quand%20pr%C3%A9voyez%20vous%20le%20retour%
20de%20Peter%20mc%20Bryan%20%C3%A0%20la%20basse%20?
```

Le script serveur récupère chaque variable et affiche son contenu comme l'illustre la figure 14-9 :

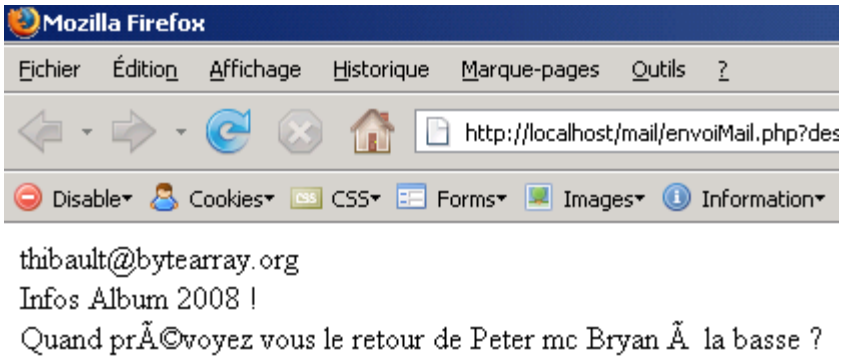

Figure 14-9. Variables récupérées.

Nous remarquons que les caractères spéciaux ne s'affichent pas correctement. Cela est du au fait que le lecteur Flash fonctionne en UTF-8, tandis que PHP fonctionne par défaut en ISO-8859-1.

Nous devons donc décoder la chaîne UTF-8 à l'aide de la méthode PHP `utf8_decode` :

```
<?php

$destinataire = $_GET["destinataire"];
$sujet = $_GET["sujet"];
$message = $_GET["message"];

if ( isset ( $destinataire ) && isset ( $sujet ) && isset ( $message ) )

{

  echo
utf8_decode($destinataire)."<br>".utf8_decode($sujet)."<br>".utf8_decode($mes
sage);

} else echo "Variables non transmises";

?>
```

Si nous testons à nouveau le code précédent, les chaînes sont correctement décodées :

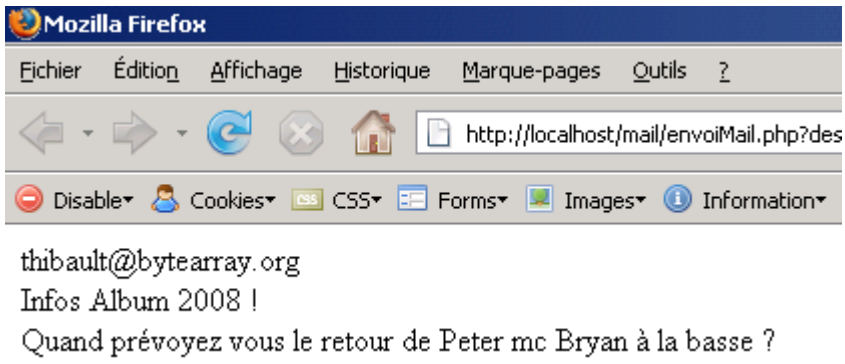

Figure 14-10. Variables correctement décodées.

Une fois assuré que les variables passées sont bien réceptionnées, nous pouvons ajouter le code nécessaire pour envoyer l'email.

Pour cela, nous ajoutons l'appel à la fonction PHP `mail` en passant le destinataire, le sujet ainsi que le contenu du message :

```
<?php

$destinataire = $_GET["destinataire"];
$sujet = $_GET["sujet"];
$message = $_GET["message"];

if ( isset ( $destinataire ) && isset ( $sujet ) && isset ( $message ) )

{

  if ( @mail ( utf8_decode($destinataire), utf8_decode($sujet),
utf8_decode($message) ) ) echo "Mail bien envoyé !";

  else echo "Erreur d'envoi !";

} else echo "Variables non transmises";

?>
```

Nous ajoutons le caractère @ devant la fonction `mail` afin d'éviter que celle-ci lève une exception PHP en cas de mauvaise configuration du serveur SMTP.

En testant notre application formulaire Flash, nous voyons que le mail est bien envoyé si les variables sont correctement transmises :

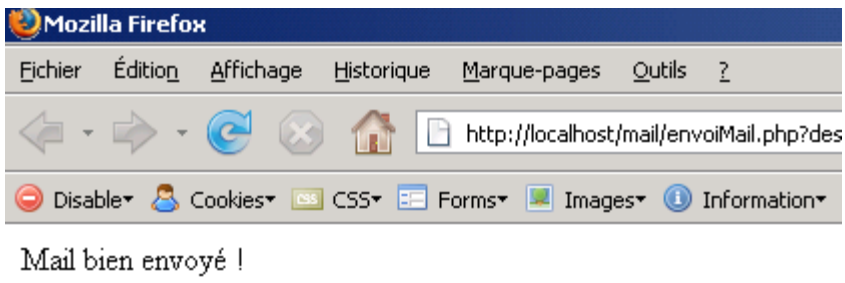

Figure 14-11. Email envoyé.

Même si le code ActionScript précédent fonctionne, il ne fait pas usage de la classe `URLVariables` recommandée dans un contexte d'envoi de variables. Grâce à celle-ci nous allons pouvoir choisir quelle méthode utiliser afin d'envoyer les variables.

Mais avant d'aller plus loin, qu'entendons nous par méthode d'envoi ?

A retenir

- Nous pouvons envoyer des variables à un script serveur en les ajoutant en fin d'URL.
- Le format d'encodage URL doit alors être respecté.

La méthode GET ou POST

Lorsque des données sont passées à un script serveur. Celles-ci peuvent être transmises de deux manières différentes. Par l'intermédiaire du tableau GET ou POST.

En utilisant la méthode GET, les variables sont obligatoirement ajoutées en fin d'url. Dans le cas d'un site de vente en ligne, l'adresse suivante permet d'accéder à un article spécifique :

```
http://www.funkrecords.com/index.php?rubrique=soul&langue=fr&article=Breakwater
```

Un script serveur récupère les variables passées en fin d'URL afin d'afficher le disque correspondant. Lorsque la méthode GET est utilisée, les variables en fin d'URL sont automatiquement accessible en PHP au sein du tableau associatif `$_GET`.

Il est important de noter que la méthode GET possède une limitation de 1024 caractères, il n'est donc possible de passer un grand volume de données par cette méthode.

Au contraire, lorsque les données sont transmises par la méthode POST, les variables n'apparaissent pas dans l'URL du navigateur. Ainsi, cette méthode ne souffre pas de la limitation de caractères.

Le W3C définit à l'adresse suivante quelques règles concernant l'utilisation des deux méthodes :

http://www.w3.org/2001/tag/doc/whenToUseGet.html

Il est conseillé d'utiliser la méthode GET lorsque les données transmises ne sont pas sensibles ou ne sont pas liées à un processus d'écriture.

A l'inverse, la méthode POST est préférée lorsque les données transmises n'ont pas à être exposées et qu'une écriture en base de données intervient par la suite.

L'utilisation d'un objet `URLVariables` va nous permettre de spécifier la méthode à utiliser dans nos échanges entre le lecteur Flash et le script serveur. Au lieu de passer les variables directement après l'url du script distant :

```
var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php?destinataire="+destinaireEmail+"&sujet=
"+sujetEmail+"&message="+messageEmail);
```

Nous allons préférer l'utilisation d'un objet `URLVariables` qui permet de stocker les variables à transmettre. Cet objet est ensuite associé à la propriété `data` de l'objet `URLRequest` utilisé.

Nous pouvons spécifier la méthode à utiliser grâce à la propriété `method` de la classe `URLRequest` et aux constantes de la classe `flash.net.URLRequestMethod`.

Nous modifions le code précédent en utilisant un objet `URLVariables` tout en conservant l'envoi des données avec la méthode GET :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.display.SimpleButton;
  import flash.text.TextField;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import flash.net.URLVariables;
  import flash.net.URLRequestMethod;
  import flash.events.MouseEvent;

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;

        public function Document ()

        {

              boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

        }

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

```
            {

                // création d'un objet URLVariables
                var variables:URLVariables = new URLVariables();

                // affectation des variables à envoyer coté serveur
                variables.sujet = sujet.text;
                variables.destinataire = destinataire.text;
                variables.message = message.text;

                // création de l'objet URLRequest
                var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

                // nous passons les variables dans l'url (tableau GET)
                requete.method = URLRequestMethod.GET;

                // nous associons les variables à l'objet URLRequest
                requete.data = variables;

                // ouvre une nouvelle fenêtre navigateur et envoi les variables
                navigateToURL ( requete );

            }

        }

    }
```

Si nous testons à nouveau notre application Flash. Nous remarquons que le script serveur reçoit de la même manière les informations, le mail est bien envoyé.

Afin d'utiliser la méthode POST, nous passons la valeur `URLRequestMethod.POST` à la propriété `method` de l'objet `URLRequest` :

```
// nous passons les variables dans l'url (tableau POST)
requete.method = URLRequestMethod.POST;
```

Désormais les données seront envoyées au sein du tableau associatif `$_POST`. Nous devons donc impérativement modifier le script serveur afin de réceptionner les variables au sein du tableau correspondant :

```
<?php

$destinataire = $_POST ["destinataire"];
$sujet = $_POST ["sujet"];
$message = $_POST ["message"];

if ( isset ( $destinataire ) && isset ( $sujet ) && isset ( $message ) )

{

  if ( @mail ( utf8_decode($destinataire), utf8_decode($sujet),
utf8_decode($message) ) ) echo "Mail bien envoyé !";

  else echo "Erreur d'envoi !";

} else echo "Variables non transmises";
```

```
?>
```

Lorsque la méthode POST est utilisée, les variables sont automatiquement accessible en PHP au sein du tableau associatif `$_POST`.

> Attention, si nous tentons d'envoyer des variables à l'aide de la fonction `navigateToURL` à l'aide de la méthode POST depuis l'environnement de test de Flash CS3, celles-ci seront tout de même envoyées par la méthode GET.
>
> Il convient donc de toujours tester au sein du navigateur, une application Flash utilisant la fonction `navigateToURL` et la méthode POST.

Nous venons de traiter le moyen le plus simple d'envoyer des variables à un script serveur. Dans la plupart des applications, nous ne souhaitons pas ouvrir de nouvelle fenêtre navigateur lors de la transmission des données. Nous préférerons généralement un envoi plus « discret ».

Afin d'envoyer discrètement des données à un script distant, nous préférerons l'utilisation de la classe `URLLoader` à la fonction `navigateToURL`.

A retenir

- Lors de l'envoi de variables par la méthode GET, les variables sont automatiquement ajoutés en fin d'url.
- Lors de l'envoi de variables par la méthode POST, les variables ne sont pas affichées au sein de l'url.
- Il convient d'utiliser la méthode GET pour la lecture de données.
- Il convient d'utiliser la méthode POST pour l'écriture de données.

Envoyer des variables discrètement

Nous allons modifier notre formulaire développé précédemment afin d'envoyer les informations au script distant, sans ouvrir de nouvelle fenêtre navigateur.

Ainsi l'expérience utilisateur sera plus élégante. Notre application donnera l'impression de fonctionner de manière autonome :

```
package org.bytearray.document

{

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

```
import flash.display.SimpleButton;
import flash.text.TextField;
import flash.net.navigateToURL;
import flash.net.URLRequest;
import flash.net.URLVariables;
import flash.net.URLRequestMethod;
import flash.net.URLLoader;
import flash.events.MouseEvent;

public class Document extends ApplicationDefaut

{

      public var destinataire:TextField;
      public var sujet:TextField;
      public var message:TextField;
      public var boutonEnvoi:SimpleButton;
      public var echanges:URLLoader;

      public function Document ()

      {

            echanges = new URLLoader();

            boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

      }

      private function envoiMail ( pEvt:MouseEvent ):void

      {

            var variables:URLVariables = new URLVariables();

            // affectation des variables à envoyer coté serveur
            variables.sujet = sujet.text;
            variables.destinataire = destinataire.text;
            variables.message = message.text;

            // création de l'objet URLRequest
            var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

            // nous passons les variables dans l'url (tableau POST)
            requete.method = URLRequestMethod.POST;

            // associe les variables à l'objet URLRequest
            requete.data = variables;

            // envoi les données de manière transparente, sans ouvrir de
nouvelle fenêtre navigateur
            echanges.load ( requete );

      }

 }

}
```

Aussitôt la méthode `load` exécutée, les variables sont transmises au script serveur, l'email est envoyé.

En plus de permettre le chargement de données, la méthode `load` permet aussi l'envoi. Cela diffère des précédentes versions d'ActionScript où la classe `LoadVars` possédait une méthode `load` pour le chargement et `send` pour l'envoi.

Nous allons continuer l'amélioration de notre formulaire en ajoutant un mécanisme de validation. Ne serait-il pas intéressant de pouvoir indiquer à Flash que le mail a bien été envoyé ?

Pour le moment, notre code n'intègre aucune gestion du retour serveur. C'est ce que nous allons ajouter dans la partie suivante.

A retenir

- Afin d'envoyer des données de manière transparente, nous utilisons la méthode `load` de l'objet `URLLoader`.
- La méthode `load` permet le chargement de données mais aussi l'envoi.

Renvoyer des données depuis le serveur

Nous allons à présent nous intéresser à l'envoi de données avec validation. Afin de savoir au sein de Flash si l'envoi du message a bien été effectué, nous devons simplement écouter l'événement `Event.COMPLETE` de l'objet `URLLoader`.

Celui-ci est diffusé automatiquement lorsque le lecteur reçoit des informations de la part du serveur :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.display.SimpleButton;
  import flash.text.TextField;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import flash.net.URLVariables;
  import flash.net.URLRequestMethod;
  import flash.net.URLLoader;
  import flash.events.Event;
  import flash.events.MouseEvent;

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;
```

```
        public var echanges:URLLoader;

        public function Document ()

        {

             echanges = new URLLoader();

             echanges.addEventListener ( Event.COMPLETE, retourServeur );

             boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

        }

        private function envoiMail ( pEvt:MouseEvent ):void

        {

             var variables:URLVariables = new URLVariables();

             // affectation des variables à envoyer coté serveur
             variables.sujet = sujet.text;
             variables.destinataire = destinataire.text;
             variables.message = message.text;

             // création de l'objet URLRequest
             var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

             // nous passons les variables dans l'url (tableau POST)
             requete.method = URLRequestMethod.POST;

             // associe les variables à l'objet URLRequest
             requete.data = variables;

             // envoi les données de manière transparente, sans ouvrir de
nouvelle fenêtre navigateur
             echanges.load ( requete );

        }

        private function retourServeur ( pEvt:Event ):void

        {

             sujet.text = destinataire.text = message.text = "";

             message.text = pEvt.target.data;

        }

    }

}
```

La méthode `retourServeur` est exécutée lorsque le serveur retourne des informations à Flash par l'intermédiaire du mot clé PHP `echo`.

En testant à nouveau l'application, nous obtenons un retour dans le champ message nous indiquant le mail a bien été envoyé, comme l'illustre la figure 14-12 :

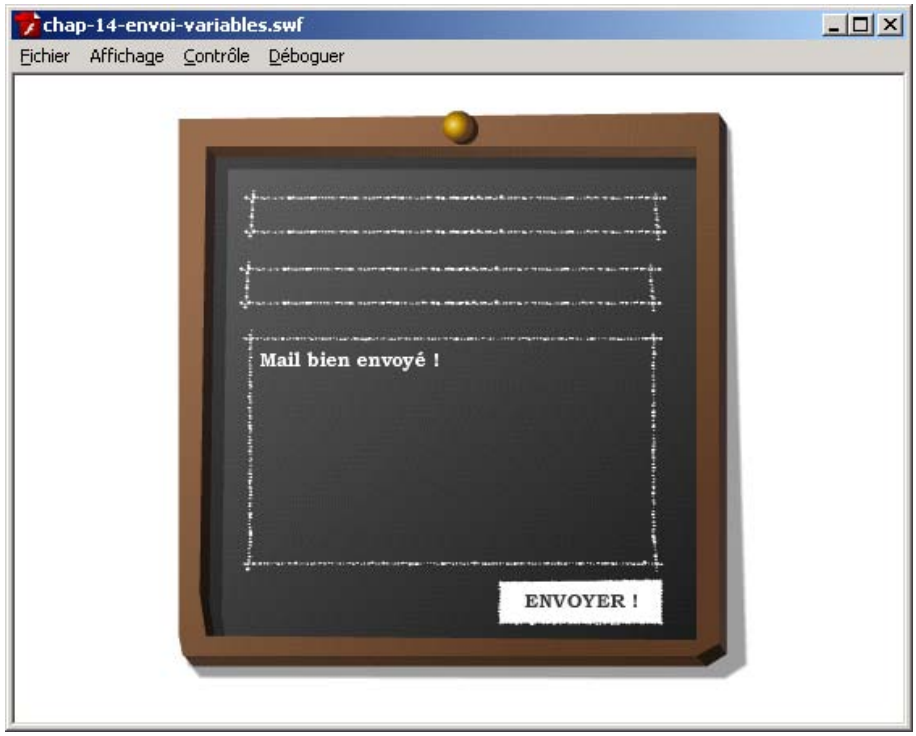

Figure 14-12. Accusé de réception de l'envoi du message.

Pour des questions de localisation, il n'est pas recommandé de conserver ce script serveur. Car si l'application devait être traduite nous serions obligé de modifier le script serveur.

Afin d'éviter cela, nous allons renvoyer la valeur 1 lorsque le mail est bien envoyé, et 0 le cas échéant. Nous renvoyons la valeur 2 lorsque les variables ne sont pas bien réceptionnées :

```
<?php

$destinataire = $_POST ["destinataire"];
$sujet = $_POST ["sujet"];
$message = $_POST ["message"];

if ( isset ( $destinataire ) && isset ( $sujet ) && isset ( $message ) )

{

  if ( @mail ( utf8_decode($destinataire), utf8_decode($sujet),
utf8_decode($message) ) ) echo "resultat=1";

  else echo "resultat=0";

} else echo "resultat=2";

?>
```

Nous décidons ainsi côté client quel message afficher. Nous renvoyons donc une chaîne encodée URL qui devra être décodée coté client.

Pour cela nous demandons à l'objet `URLLoader` de décoder toutes les chaînes reçues afin de pouvoir facilement extraire le contenu de la variable `resultat`. Nous modifions la méthode `retourServeur` afin d'afficher le message approprié :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.display.SimpleButton;
  import flash.text.TextField;
  import flash.net.navigateToURL;
  import flash.net.URLRequest;
  import flash.net.URLVariables;
  import flash.net.URLRequestMethod;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.events.Event;
  import flash.events.MouseEvent;
  import flash.events.HTTPStatusEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;
        public var echanges:URLLoader;

        public function Document ()

        {

              echanges = new URLLoader();

              echanges.dataFormat = URLLoaderDataFormat.VARIABLES;

              echanges.addEventListener ( Event.COMPLETE, retourServeur );
              echanges.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );
              echanges.addEventListener ( HTTPStatusEvent.HTTP_STATUS,
statutHTTP );

              boutonEnvoi.addEventListener (  MouseEvent.CLICK, envoiMail );

        }

        private function envoiMail ( pEvt:MouseEvent ):void

        {

              var variables:URLVariables = new URLVariables();
```

```
            // affectation des variables à envoyer coté serveur
            variables.sujet = sujet.text;
            variables.destinataire = destinataire.text;
            variables.message = message.text;

            // création de l'objet URLRequest
            var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

            // nous passons les variables dans l'url (tableau POST )
            requete.method = URLRequestMethod.POST;

            // associe les variables à l'objet URLRequest
            requete.data = variables;

            // envoi les données de manière transparente, sans ouvrir de
nouvelle fenêtre navigateur
            echanges.load ( requete );

      }

      private function retourServeur ( pEvt:Event ):void

      {

            sujet.text = destinataire.text = message.text = "";

            var messageRetour:String;

            if ( Number ( pEvt.target.data.resultat ) == 1 ) messageRetour =
"Mail bien envoyé !";

            else if ( Number ( pEvt.target.data.resultat ) == 2 )
messageRetour = "Erreur de transmission des données !";

            else messageRetour = "Erreur d'envoi du message !";

            message.text = messageRetour;

      }

      private function statutHTTP ( pEvt:HTTPStatusEvent ):void

      {

            trace( "Code HTTP : " + pEvt.status );

      }

      private function erreurChargement ( pEvt:IOErrorEvent ):void

      {

            trace("Erreur de chargement !");

      }

  }

}
```

Si nous n'envoyons pas les variables au script serveur ou que la fonction `mail` échoue nous obtenons un message approprié dans notre application Flash. Dans le code suivant, nous n'affectons aucune variable à l'objet `URLVariables` :

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

{

	var variables:URLVariables = new URLVariables();

	// création de l'objet URLRequest
	var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

	// nous passons les variables dans l'url (tableau POST )
	requete.method = URLRequestMethod.POST;

	// associe les variables à l'objet URLRequest
	requete.data = variables;

	// envoi les données de manière transparente, sans ouvrir de nouvelle
fenêtre navigateur
	echanges.load ( requete );

}
```

L'email ne peut être envoyé, la figure 14-13 illustre le message affiché par l'application :

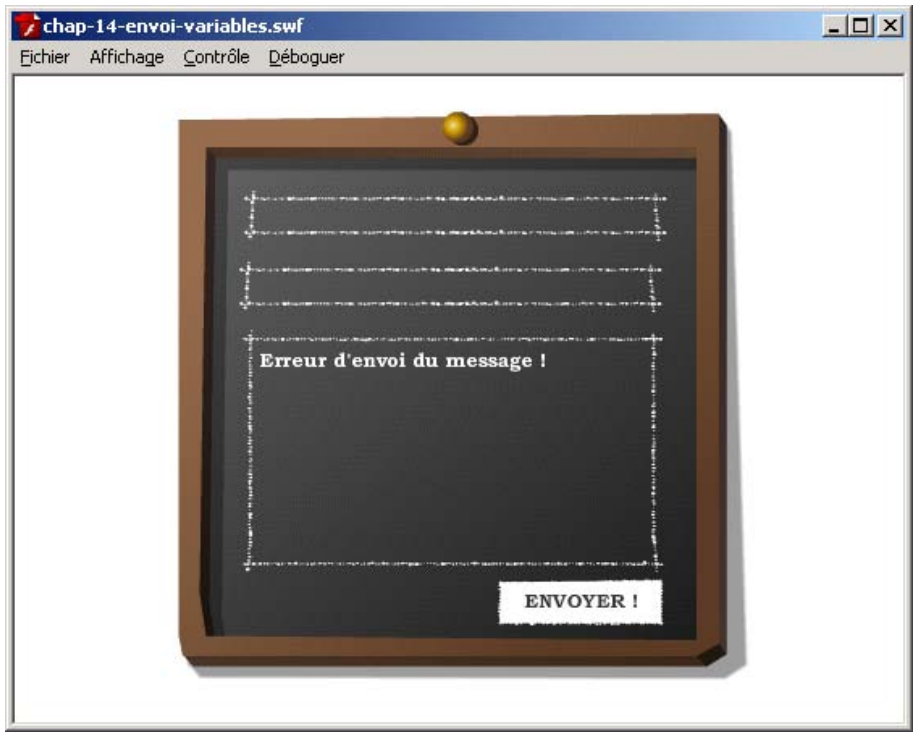

Figure 14-13. Echec d'envoi de l'email.

Nous pourrions aller plus loin dans cet exemple et enregistrer l'utilisateur dans une base de données et renvoyer d'autres types d'informations.

Nous reviendrons en détail sur l'envoi et la réception de données issues d'une base de données au cours du chapitre 19 intitulé *Remoting*.

A retenir

- Afin de récupérer les données issues du serveur nous écoutons l'événement `Event.COMPLETE` de l'objet `URLLoader`.
- Afin de retourner des données à Flash, nous devons utiliser le mot clé PHP `echo`.

Aller plus loin

Souvenez-vous, au cours du chapitre 2, nous avons découvert la notion d'expressions régulières permettant d'effectuer des manipulations complexes sur des chaînes de caractères.

Afin d'optimiser notre application nous allons intégrer une vérification de l'email par expression régulière.

Nous allons donc créer une classe `OutilsFormulaire` globale à tous nos projets, intégrant une méthode statique `verifieEmail`. Celle-ci renverra `true` si l'email est correcte ou `false` le cas échéant.

Rappelez-vous, au cours du chapitre 12 intitulé *Programmation Bitmap*, nous avions crée un répertoire global de classes nommé `classes_as3`. Au sein du répertoire `outils` nous créeons la classe `OutilsFormulaire` suivante :

```
package org.bytearray.outils

{

  public class FormulaireOutils

  {

        private static var emailModele:RegExp = /^[a-z0-9][-._a-z0-9]*@([a-z0-
9][-_a-z0-9]*\.)+[a-z]{2,6}$/

        public static function verifieEmail ( pEmail:String ):Boolean

        {

              var resultat:Array = pEmail.match( FormulaireOutils.emailModele
);

              return resultat != null;
```

```
        }
    }
}
```

Une fois définie, nous pouvons l'utiliser dans notre application en l'important :

```
import org.bytearray.outils.FormulaireOutils;
```

Puis nous modifions la méthode `envoiMail` afin d'intégrer une vérification de l'email :

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

{

  if ( FormulaireOutils.verifieEmail ( destinataire.text ) )

  {

        var variables:URLVariables = new URLVariables();

        // affectation des variables à envoyer coté serveur
        variables.sujet = sujet.text;
        variables.destinataire = destinataire.text;
        variables.message = message.text;

        // création de l'objet URLRequest
        var requete:URLRequest = new URLRequest
("http://localhost/mail/envoiMail.php");

        // nous passons les variables dans l'url (tableau POST )
        requete.method = URLRequestMethod.POST;

        // associe les variables à l'objet URLRequest
        requete.data = variables;

        // envoi les données de manière transparente, sans ouvrir de nouvelle
fenêtre navigateur
        echanges.load ( requete );

  } else destinataire.text = "Email non valide !";

}
```

Ainsi, toutes nos applications pourront utiliser la classe `FormulaireOutils` afin de vérifier la validité d'un email. Bien entendu, nous pouvons ajouter d'autres méthodes pratiques à celle-ci.

Nous retrouvons ici l'intérêt d'une méthode statique, pouvant être appelée directement sur le constructeur de la classe, sans avoir à instancier un objet `FormulaireOutils`.

Envoyer un flux binaire

Rendez vous au chapitre 20 intitulé `ByteArray` pour découvrir comment transmettre un flux binaire grâce à la classe `URLLoader`.

Télécharger un fichier

Nous avons vu jusqu'à present comment charger ou envoyer des variables à partir du lecteur Flash. Certaines applications peuvent néanmoins nécessiter un téléchargement ou un envoi de fichiers entre le serveur et l'ordinateur de l'utilisateur.

La classe `flash.net.FileReference` permet de télécharger n'importe quel fichier grace à la méthode `download`.

Imaginons que nous devions télécharger un fichier zip depuis une URL spécifique telle :

http://www.monserveur.org/archive.zip

Au sein d'un nouveau document Flash CS3 nous associons la classe de document suivante :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;

  public class Document extends ApplicationDefaut

  {

        public function Document ()

        {


        }

  }

}
```

Puis nous créons un objet `FileReference` :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReference;

  public class Document extends ApplicationDefaut

  {

        private var telechargeur:FileReference;

        public function Document ()

        {
```

```
                telechargeur = new FileReference();
            }
        }
    }
```

Afin de gérer les différentes étapes liées au téléchargement de données la classe `FileReference` diffuse différents événements dont voici le détail :

- `Event.CANCEL` : diffusé lorsqu'un envoi ou téléchargement est annulé par la fenêtre de parcours.
- `Event.COMPLETE` : diffusé lorsqu'un envoi ou un téléchargement a réussi.
- `HTTPStatusEvent.HTTP_STATUS` : diffusé lorsqu'un envoi ou chargement échoue.
- `IOErrorEvent.IO_ERROR` : diffusé lorsque l'envoi ou la réception des données échoue.
- `Event.OPEN` : diffusé lorsque l'envoi ou la réception des données commence.
- `ProgressEvent.PROGRESS` : diffusé lorsque l'envoi ou le chargement est en cours.
- `SecurityErrorEvent.SECURITY_ERROR` : diffusé lorsque le lecteur tente d'envoyer ou de charger des données depuis un domaine non autorisé.
- `Event.SELECT` : diffusé lorsque le bouton OK de la boite de dialogue de recherches de fichiers est relâche
- `DataEvent.UPLOAD_COMPLETE_DATA` : diffusé lorsqu'une opération d'envoi ou de chargement est terminé et que le serveur renvoie des données.

Afin de télécharger un fichier nous utilisons la méthode `download` dont voici la signature :

```
public function download(request:URLRequest, defaultFileName:String =
null):void
```

Celle-ci requiert deux paramètres dont voici le détail :

- `request` : un objet `URLRequest` contenant l'adresse de l'élément à télécharger.
- `defaultFileName` : ce paramètre permet de spécifier le nom de l'élément téléchargé au sein de la boite de dialogue.

Dans le code suivant, nous téléchargeons un fichier distant :

```
package org.bytearray.document
```

```
{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;

  public class Document extends ApplicationDefaut

  {

        private var telechargeur:FileReference;
        private var lienFichier:String = "http://alivepdf.bytearray.org/wp-
content/images/logo_small.jpg";
        private var requete:URLRequest = new URLRequest( lienFichier );
        public var boutonTelecharger:SimpleButton;

        public function Document ()

        {

              // création d'un objet FileReference
              telechargeur = new FileReference();

              boutonTelecharger.addEventListener ( MouseEvent.CLICK,
telechargeFichier );

        }

        private function telechargeFichier ( pEvt:MouseEvent ):void

        {

              // téléchargement du fichier distant
              telechargeur.download ( requete );

        }

  }

}
```

Lorsque nous cliquons sur le bouton `boutonTelecharger` une fenêtre de téléchargement s'ouvre comme l'illustre la figure 14-14 :

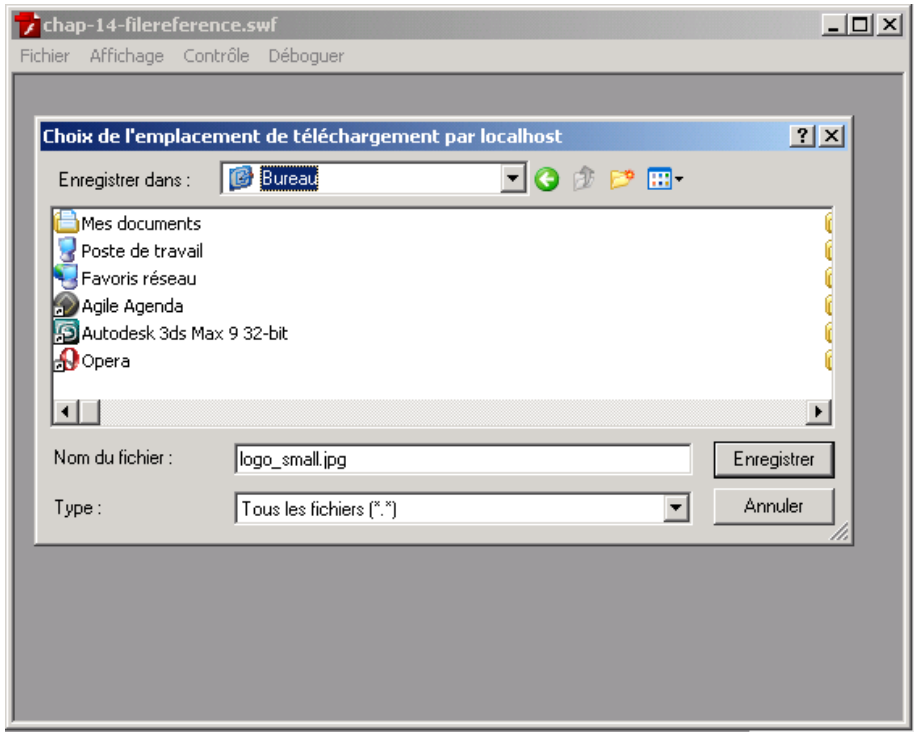

Figure 14-14. Boite de dialogue.

Une fois l'emplacement sélectionné, nous cliquons sur le bouton `Enregistrer`. Le fichier est alors sauvé en local sur l'ordinateur exécutant l'application. Bien que nous n'ayons pas spécifié le nom du fichier, le lecteur Flash extrait automatiquement celui-ci à partir de son URL.

Attention, lorsque cette boite de dialogue apparaît, la tentative d'accès au fichier distant n'a pas encore démarré. Si nous modifions l'adresse du fichier distant par l'adresse suivante :

```
private var lienFichier:String =
"http://www.monserveurInexistant.org/logo_small.jpg";
```

Le lecteur affiche tout de même la boite de dialogue ainsi que le nom du fichier.

Afin de gérer l'état du transfert nous écoutons les principaux événements :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
```

```
  import flash.events.HTTPStatusEvent;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var telechargeur:FileReference;
        private var lienFichier:String = "http://alivepdf.bytearray.org/wp-
content/images/logo_small.jpg";
        private var requete:URLRequest = new URLRequest( lienFichier );
        public var boutonTelecharger:SimpleButton;

        public function Document ()

        {

              // création d'un objet FileReference
              telechargeur = new FileReference();

              telechargeur.addEventListener ( ProgressEvent.PROGRESS,
chargementEnCours );
              telechargeur.addEventListener ( Event.COMPLETE, chargementTermine
);
              telechargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

              boutonTelecharger.addEventListener ( MouseEvent.CLICK,
telechargeFichier );

        }

        private function chargementEnCours ( pEvt:ProgressEvent ):void

        {

              trace( "chargement en cours  : " + pEvt.bytesLoaded /
pEvt.bytesTotal );

        }

        private function chargementTermine ( pEvt:Event ):void

        {

              trace( "chargement terminé" );

        }

        private function erreurChargement ( pEvt:IOErrorEvent ):void

        {

              trace( "erreur de chargement du fichier !" );

        }

        private function telechargeFichier ( pEvt:MouseEvent ):void

        {
```

```
            // téléchargement du fichier distant
            telechargeur.download ( requete );

        }

    }

}
```

La méthode `download` n'accèpte qu'un seul fichier à télécharger. Il est donc impossible de télécharger plusieurs fichiers en même temps à l'aide de la classe `FileReference`.

Il est important de noter que dans un contexte inter domaines, le téléchargement de fichiers est interdit si le domaine distant n'est pas autorisé.

Ainsi, si nous publions l'application actuelle sur un serveur distant, l'événement `SecurityErrorEvent.SECURITY_ERROR` est diffusé affichant le message suivant :

```
Error #2048: Violation de la sécurité Sandbox :
http://monserveur.fr/as3/chap-14-filereference.swf ne peut pas charger de
données à partir de http://alivepdf.bytearray.org/wp-
content/images/logo_small.jpg.
```

Afin de pouvoir charger le fichier distant nous devons utiliser un *fichier de régulation* ou un *proxy*.

Nous allons utiliser dans le code suivant un *proxy* afin de feindre le lecteur Flash :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.URLRequestMethod;
  import flash.net.URLVariables;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
  import flash.events.HTTPStatusEvent;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;
  import flash.events.SecurityErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var telechargeur:FileReference;
        private var lienFichier:String = "proxy.php";
        private var requete:URLRequest = new URLRequest( lienFichier );
        public var boutonTelecharger:SimpleButton;
```

```
		public function Document ()

		{

			// création d'un objet FileReference
			telechargeur = new FileReference();

			// création d'un objet URLVariables pour passer le chemin du
fichier à charger au proxy
			var variables:URLVariables = new URLVariables();

			// spécification du chemin
			variables.chemin = "http://alivepdf.bytearray.org/wp-
content/images/logo_small.jpg";

			// affectation des variables et de la méthode utilisée
			requete.data = variables;
			requete.method = URLRequestMethod.POST;

			telechargeur.addEventListener ( ProgressEvent.PROGRESS,
chargementEnCours );
			telechargeur.addEventListener ( Event.COMPLETE, chargementTermine
);
			telechargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

			boutonTelecharger.addEventListener ( MouseEvent.CLICK,
telechargeFichier );

		}

		private function chargementEnCours ( pEvt:ProgressEvent ):void

		{

			trace( "chargement en cours  : " + pEvt.bytesLoaded /
pEvt.bytesTotal );

		}

		private function chargementTermine ( pEvt:Event ):void

		{

			trace( "chargement terminé" );

		}

		private function erreurChargement ( pEvt:IOErrorEvent ):void

		{

			trace( "erreur de chargement du fichier !" );

		}

		private function telechargeFichier ( pEvt:MouseEvent ):void

		{

			// téléchargement du fichier distant
			telechargeur.download ( requete );
```

```
            }
        }
    }
```

Une fois l'application publiée, l'appel de la méthode download ne lève plus d'exeception. La boite de dialogue s'ouvre et propose comme nom de fichier le nom du fichier proxy utilisé comme l'illustre la figure 14-15 :

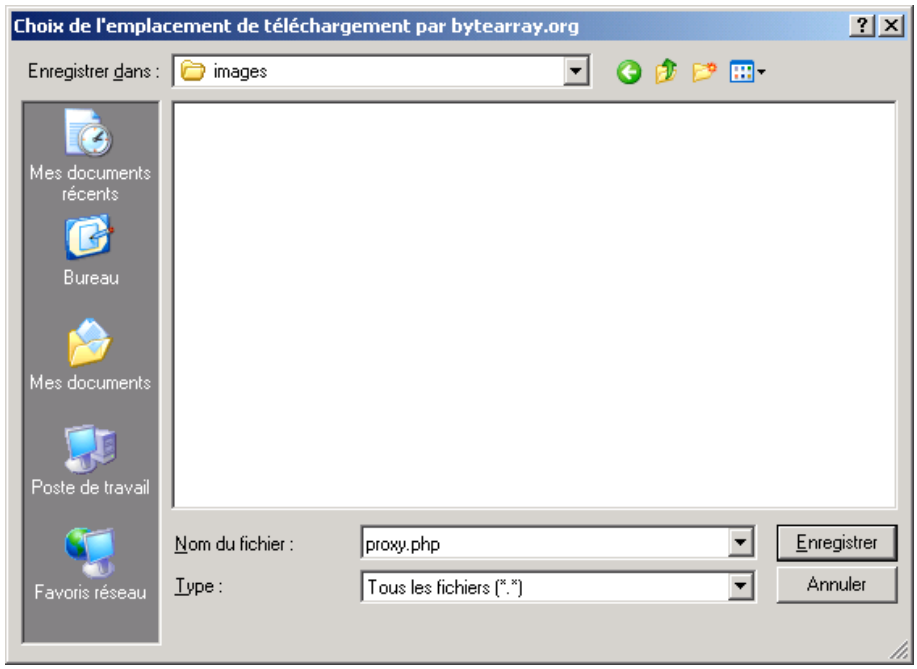

Figure 14-15. Nom d'enregistrement du fichier.

Comme nous l'avons vu précédemment, le lecteur devine automatiquement le nom du fichier téléchargé depuis l'URL du fichier. Dans notre cas, ce dernier considère que le fichier à charger est le proxy et propose donc comme nom de fichier `proxy.php`.

Afin de proposer le nom réel du fichier chargé, nous allons utiliser le deuxième paramètre `defaultFileName` de la méthode `download` :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.URLRequestMethod;
  import flash.net.URLVariables;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
```

```
import flash.events.HTTPStatusEvent;
import flash.events.ProgressEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;

public class Document extends ApplicationDefaut

{

      private var telechargeur:FileReference;
      private var lienFichier:String = "proxy.php";
      private var requete:URLRequest = new URLRequest( lienFichier );
      private var urlFichier:String;
      public var boutonTelecharger:SimpleButton;

      public function Document ()

      {

            // création d'un objet FileReference
            telechargeur = new FileReference();

            // création d'un objet URLVariables pour passer le chemin du
fichier à charger au proxy
            var variables:URLVariables = new URLVariables();

            // url du fichier distant à charger par le proxy
            urlFichier = "http://alivepdf.bytearray.org/wp-
content/images/logo_small.jpg";

            // spécification du chemin
            variables.chemin = urlFichier;

            // affectation des variables et de la méthode utilisée
            requete.data = variables;
            requete.method = URLRequestMethod.POST;

            telechargeur.addEventListener ( ProgressEvent.PROGRESS,
chargementEnCours );
            telechargeur.addEventListener ( Event.COMPLETE, chargementTermine
);
            telechargeur.addEventListener ( IOErrorEvent.IO_ERROR,
erreurChargement );

            boutonTelecharger.addEventListener ( MouseEvent.CLICK,
telechargeFichier );

      }

      private function chargementEnCours ( pEvt:ProgressEvent ):void

      {

            trace( "chargement en cours  : " + pEvt.bytesLoaded /
pEvt.bytesTotal );

      }

      private function chargementTermine ( pEvt:Event ):void

      {
```

```
            trace( "chargement terminé" );

        }

        private function erreurChargement ( pEvt:IOErrorEvent ):void

        {

            trace( "erreur de chargement du fichier !" );

        }

        private function telechargeFichier ( pEvt:MouseEvent ):void

        {

            // expression régulière
            var modele:RegExp = /[A-Za-z0-9_]*\.\D{3}$/

            // extrait le nom du fichier de l'url
            var resultat:Array = urlFichier.match ( modele );

            // téléchargement du fichier distant
            if ( resultat != null ) telechargeur.download ( requete,
resultat[0] );

        }
    }

}
```

Le nom du fichier est extrait manuellement puis passé en deuxième paramètre de la méthode `download`. Lors du clic sur le bouton `boutonTelecharger` la boite de dialogue s'ouvre en proposant le bon nom de fichier :

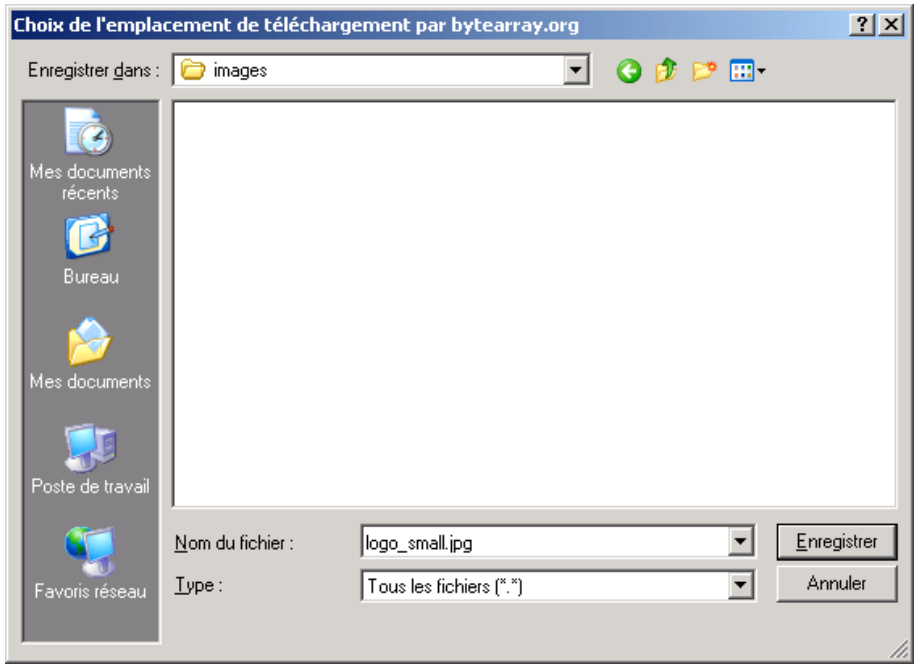

Figure 14-16. Nom d'enregistrement du fichier.

Il est donc possible de rendre totalement transparent l'intervention d'un *proxy* dans le téléchargement de notre fichier distant. Lorsque nous cliquons sur le bouton `Enregistrer`, le fichier est téléchargé et enregistré.

Nous allons nous attarder à présent sur l'envoi de fichiers par la classe `flash.net.FileReference`.

A retenir

- Afin de télécharger un fichier depuis le lecteur Flash nous utilisons la méthode `download` de la classe `FileReference`.
- L'appel de la méthode `download` entraîne l'ouverture d'une boite de dialogue permettant de sauver le fichier sur l'ordinateur de l'utilisateur.

Publier un fichier

A l'inverse si nous souhaitons mettre en ligne un fichier selectionné depuis l'ordinateur de l'utilisateur, nous pouvons utiliser la méthode `upload` de la classe `FileReference`.

Dans un nouveau document Flash CS3, nous associons la classe de document suivante :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var envoyeur:FileReference;
        private var lienScript:String =
"http://localhost/envoi/envoiFichier.php";
        private var requete:URLRequest = new URLRequest( lienScript );
        public var boutonEnvoyer:SimpleButton;

        public function Document ()

        {

              // création d'un objet FileReference
```

```
            envoyeur = new FileReference();

            envoyeur.addEventListener ( ProgressEvent.PROGRESS, envoiEnCours
);
            envoyeur.addEventListener ( Event.COMPLETE, envoiTermine );
            envoyeur.addEventListener ( IOErrorEvent.IO_ERROR, erreurEnvoi );

            boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

      }

      private function envoiEnCours ( pEvt:ProgressEvent ):void

      {

            trace( "envoi en cours  : " + pEvt.bytesLoaded / pEvt.bytesTotal
);

      }

      private function envoiTermine ( pEvt:Event ):void

      {

            trace( "envoi terminé" );

      }

      private function erreurEnvoi ( pEvt:IOErrorEvent ):void

      {

            trace( "erreur d'envoi du fichier !" );

      }

      private function parcoursFichiers ( pEvt:MouseEvent ):void

      {

            // ouvre la boite de dialogue permettant de parcourir les
fichiers
            envoyeur.browse ();

      }

  }

}
```

Au sein de l'application, un bouton `boutonEnvoyer` déclenche l'ouverture de la boite de dialogue permettant de parcourir les fichiers à transférer à l'aide de la méthode `browse` de l'objet `FileReference`.

La propriété `lienScript` pointe vers un script serveur permettant l'enregistrement du fichier transféré sur le serveur. Voici le code PHP permettant de sauver le fichier transféré :

```
<?php

$fichier = $_FILES["Filedata"];

if ( isset ( $fichier ) )

{

  $nomFichier = $fichier['name'];

  move_uploaded_file ( $fichier['tmp_name'], "monRepertoire/".$nomFichier);

}

?>
```

Le lecteur Flash place le fichier transféré ua sein du tableau `$_FILES` et de la propriété `Filedata`.

Lors de la gestion d'envoi de fichiers par `FileReference` il convient d'écouter l'événement `Event.SELECT` diffusé lorsque l'utilisateur a selectionné un fichier. Ainsi nous pouvons récupérer différentes informations liées au fichier selectionné :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var envoyeur:FileReference;
        private var lienScript:String =
"http://localhost/envoi/envoiFichier.php";
        private var requete:URLRequest = new URLRequest( lienScript );
        public var boutonEnvoyer:SimpleButton;

        public function Document ()

        {

              // création d'un objet FileReference
              envoyeur = new FileReference();

              envoyeur.addEventListener ( Event.SELECT, selectionFichier );
              envoyeur.addEventListener ( ProgressEvent.PROGRESS, envoiEnCours
);
              envoyeur.addEventListener ( Event.COMPLETE, envoiTermine );
              envoyeur.addEventListener ( IOErrorEvent.IO_ERROR, erreurEnvoi );
```

```
            boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

      }

      private function selectionFichier ( pEvt:Event ):void

      {

            var fichier:FileReference = FileReference ( pEvt.target );

            // affiche : Tue Oct 23 19:59:27 GMT+0200 2007
            trace( fichier.creationDate );

            // affiche : Interview.doc
            trace( fichier.name );

            // affiche : 37888
            trace( fichier.size );

            // affiche : .doc
            trace( fichier.type );

      }

      private function envoiEnCours ( pEvt:ProgressEvent ):void

      {

            trace( "envoi en cours  : " + pEvt.bytesLoaded / pEvt.bytesTotal
);

      }

      private function envoiTermine ( pEvt:Event ):void

      {

            trace( "envoi terminé" );

      }

      private function erreurEnvoi ( pEvt:IOErrorEvent ):void

      {

            trace( "erreur d'envoi du fichier !" );

      }

      private function parcoursFichiers ( pEvt:MouseEvent ):void

      {

            // ouvre la boite de dialogue permettant de parcourir les
fichiers
            envoyeur.browse ();

      }

 }
```

```
}
```

Voici le détail de chacune des propriétés définies par la classe `FileReference` :

- `creationDate` : date de création du fichier.
- `creator` : type de créateur Macintosh du fichier.
- `modificationDate` : date de dernière modification du fichier.
- `name` : nom du fichier.
- `size` : taille du fichier en octets.
- `type` : extension du fichier.

Une fois l'objet séléctionné, nous appelons la méthode `upload` sur l'objet `FileReference` :

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

{

	var fichier:FileReference = FileReference ( pEvt.target );

	// affiche : Tue Oct 23 19:59:27 GMT+0200 2007
	trace( fichier.creationDate );

	// affiche : Interview.doc
	trace( fichier.name );

	// affiche : 37888
	trace( fichier.size );

	// affiche : .doc
	trace( fichier.type );

	// envoi du fichier
	fichier.upload ( requete );

}
```

Comme l'illustre la figure 14-17, vous remarquerez qu'il est impossible de sélectionner plusieurs fichiers au sein de la boite de dialogue ouverte par la méthode `browse` de l'objet `FileReference` :

Figure 14-17. Selection multiple impossible.

Afin de pouvoir sélectionner plusieurs fichiers nous devons obligatoirement utiliser la classe `flash.net.FileReferenceList`.

A retenir

- Afin de selectionner un fichier à transférer sur un serveur depuis le lecteur Flash nous utilisons d'abord la méthode `browse` de l'objet `FileReference`.
- Une fois le fichier sélectionné, nous appelons la méthode `upload` en spécifiant l'URL du script serveur permettant l'enregistrement du fichier.

Publier plusieurs fichiers

Afin de pouvoir envoyer plusieurs fichiers en même temps, nous modifions la classe du document afin de créer un objet `FileReferenceList` :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReferenceList;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

       private var envoyeurMultiple:FileReferenceList;
```

```
		private var lienScript:String =
"http://localhost/envoi/envoiFichier.php";
		private var requete:URLRequest = new URLRequest( lienScript );
		public var boutonEnvoyer:SimpleButton;

		public function Document ()

		{

			// création d'un objet FileReference
			envoyeurMultiple = new FileReferenceList();

			envoyeurMultiple.addEventListener ( Event.SELECT,
selectionFichier );
			envoyeurMultiple.addEventListener ( ProgressEvent.PROGRESS,
envoiEnCours );
			envoyeurMultiple.addEventListener ( Event.COMPLETE, envoiTermine
);
			envoyeurMultiple.addEventListener ( IOErrorEvent.IO_ERROR,
erreurEnvoi );

			boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

		}

		private function selectionFichier ( pEvt:Event ):void

		{

			// référence le tableau contenant chaque objet FileReference
			var listeFichiers:Array = pEvt.target.fileList;

			// affiche : n fichiers séléctionnés
			trace( listeFichiers.length + " fichier(s) selectionnés");

		}

		private function envoiEnCours ( pEvt:ProgressEvent ):void

		{

			trace( "envoi en cours  : " + pEvt.bytesLoaded / pEvt.bytesTotal
);

		}

		private function envoiTermine ( pEvt:Event ):void

		{

			trace( "envoi terminé" );

		}

		private function erreurEnvoi ( pEvt:IOErrorEvent ):void

		{

			trace( "erreur d'envoi du fichier !" );

		}
```

```
            private function parcoursFichiers ( pEvt:MouseEvent ):void
            {
                // ouvre la boite de dialogue permettant de parcourir les
fichiers
                envoyeurMultiple.browse ();
            }
        }
    }
```

La boite de dialogue ouverte par la méthode `browse` de l'objet `FileReferenceList` permet la sélection multiple, comme l'indique la figure 14-18 :

Figure 14-18. Selection multiple possible.

En réalité lorsque nous sélectionnons plusieurs fichiers à travers la boite de dialogue et que nous validons la sélection. La classe `FileReferenceList` crée un interne un objet `FileReference` associé à chaque fichier référence au sein de la propriété `fileList` de l'objet `FileReferenceList`.

Nous devons donc manuellement parcourir ce tableau interne contenant les objets `FileReference` et appeler la méthode `upload` sur chacun d'entre eux.

Il est important de signaler que l'objet `FileReference` ne diffuse que les événements `Event.SELECT` et `Event.CANCEL` liés à la sélection des fichiers. La gestion du chargement se fait par l'intermédiaire des objets `FileReference` crées en interne par l'objet `FileReferenceList` :

```
package org.bytearray.document
```

```
{

  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReferenceList;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var envoyeurMultiple:FileReferenceList;
        private var lienScript:String =
"http://localhost/envoi/envoiFichier.php";
        private var requete:URLRequest = new URLRequest( lienScript );
        public var boutonEnvoyer:SimpleButton;

        public function Document ()

        {

              // création d'un objet FileReference
              envoyeurMultiple = new FileReferenceList();

              envoyeurMultiple.addEventListener ( Event.SELECT,
selectionFichier );

              boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

        }

        private function selectionFichier ( pEvt:Event ):void

        {

              // référence le tableau contenant chaque objet FileReference
              var listeFichiers:Array = pEvt.target.fileList;

              // longueur du tableau
              var lng:int = listeFichiers.length;

              for ( var i:int = 0; i< lng; i++ )

              {

                    var fichier:FileReference = listeFichiers[i];

                    fichier.addEventListener ( ProgressEvent.PROGRESS,
envoiEnCours );
                    fichier.addEventListener ( Event.COMPLETE, envoiTermine );
                    fichier.addEventListener ( IOErrorEvent.IO_ERROR,
erreurEnvoi );

                    fichier.upload ( requete );
```

```
            }
        }
        private function envoiEnCours ( pEvt:ProgressEvent ):void
        {
            trace( "envoi en cours  : " + pEvt.bytesLoaded / pEvt.bytesTotal
);
        }
        private function envoiTermine ( pEvt:Event ):void
        {
            trace( "envoi terminé" );
        }
        private function erreurEnvoi ( pEvt:IOErrorEvent ):void
        {
            trace( "erreur d'envoi du fichier !" );
        }
        private function parcoursFichiers ( pEvt:MouseEvent ):void
        {
            // ouvre la boite de dialogue permettant de parcourir les
fichiers
            envoyeurMultiple.browse ();
        }
    }
}
```

En testant notre application, les fichiers sont bien transférés sur le serveur.

Nous allons ajouter à présent une jauge de chargement indiquant l'état du transfert. Pour cela nous créons un clip de forme rectangulaire en bibliothèque auquel nous associons une classe `Prechargeur`.

```
package org.bytearray.document
{
  import org.bytearray.abstrait.ApplicationDefaut;
  import flash.net.FileReferenceList;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.events.Event;
```

```
  import flash.events.ProgressEvent;
  import flash.events.IOErrorEvent;

  public class Document extends ApplicationDefaut

  {

        private var envoyeurMultiple:FileReferenceList;
        private var lienScript:String =
"http://localhost/envoi/envoiFichier.php";
        private var requete:URLRequest = new URLRequest( lienScript );
        public var boutonEnvoyer:SimpleButton;
        private var prechargeur:Prechargeur;

        public function Document ()

        {

              // création d'un objet FileReference
              envoyeurMultiple = new FileReferenceList();

              prechargeur = new Prechargeur();

              envoyeurMultiple.addEventListener ( Event.SELECT,
selectionFichier );

              boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

        }

        private function selectionFichier ( pEvt:Event ):void

        {

              // référence le tableau contenant chaque objet FileReference
              var listeFichiers:Array = pEvt.target.fileList;

              // longueur du tableau
              var lng:int = listeFichiers.length;

              for ( var i:int = 0; i< lng; i++ )

              {

                    var fichier:FileReference = listeFichiers[i];

                    fichier.addEventListener ( ProgressEvent.PROGRESS,
envoiEnCours );
                    fichier.addEventListener ( Event.COMPLETE, envoiTermine );
                    fichier.addEventListener ( IOErrorEvent.IO_ERROR,
erreurEnvoi );

                    fichier.upload ( requete );

              }

              // ajout du préchargeur à l'affichage
              if ( !contains ( prechargeur ) ) addChild ( prechargeur );

        }
```

```
        private function envoiEnCours ( pEvt:ProgressEvent ):void
        {
            // ajuste la taille de la jauge par rapport  à l'état du
transfert
            prechargeur.scaleX = pEvt.bytesLoaded / pEvt.bytesTotal;
        }
        private function envoiTermine ( pEvt:Event ):void
        {
            trace( "envoi terminé" );
        }
        private function erreurEnvoi ( pEvt:IOErrorEvent ):void
        {
            trace( "erreur d'envoi du fichier !" );
        }
        private function parcoursFichiers ( pEvt:MouseEvent ):void
        {
            // ouvre la boite de dialogue permettant de parcourir les
fichiers
            envoyeurMultiple.browse ();
        }
    }
}
```

Lors du transfert des fichiers, la jauge de transfert indique l'état du chargement. En revanche, lors du transfert de multiples fichiers la jauge reçoit les événements de chaque fichier en cours de transfert et indique le chargement de tous les fichiers en même temps.

De la même manière, nous ne pouvons pas savoir simplement, si le transfert de tous les fichiers est terminé. Nous ne bénéficions pas d'événement approprié. Pour remédier à tout cela nous allons développer notre propre version de la classe `FileReferenceList`.

A retenir

- Seule la classe `FileReferenceList` permet l'envoi de plusieurs fichiers en même temps.

Création de la classe EnvoiMultiple

Nous allons améliorer la gestion d'envoi des fichiers en créant une classe `EnvoiMultiple`.

Celle si étend la classe `FileReferenceList` et améliore le transfert de chaque fichier en s'assurant que chaque fichier est transféré l'un après l'autre. De plus, nous allons ajouter un événement indiquant la fin du transfert de tous les fichiers.

Nous commençons par définir notre classe `EnvoiMultiple` en étendant la classe `FileReferenceList` :

```
package org.bytearray.envoi

{

  import flash.net.FileReferenceList;
  import flash.events.Event;
  import flash.net.URLRequest;

  public class EnvoiMultiple extends FileReferenceList

  {

        private var requete:URLRequest;

        public function EnvoiMultiple ( pRequete:URLRequest )

        {

              requete = pRequete;

              addEventListener ( Event.SELECT, selectionFichiers );

        }

        private function selectionFichiers ( pEvt:Event ):void

        {

              trace("fichiers selectionnés");

        }

  }

}
```

La classe `EnvoiMultiple` accèpte en paramètre un objet `URLRequest` pointant vers un script serveur permettant l'enregistrement du fichier.

Nous souhaitons que la classe `EnvoiMultiple` puisse gérer de manière synchronisée chaque envoi de fichiers. Pour cela nous allons lancer le premier transfert puis attendre que celui soit terminé pour déclencher les suivants :

```
package org.bytearray.envoi

{

  import flash.events.Event;
  import flash.events.ProgressEvent;
  import org.bytearray.events.ProgressQueueEvent;
  import org.bytearray.events.QueueEvent;
  import flash.net.FileReference;
  import flash.net.URLRequest;

  public class EnvoiMultiple extends FileReferenceList

  {

        private var historiqueFichiers:Array;
        private var fichierEnCours:FileReference;
        private var fichiers:Array;
        private var requete:URLRequest;

        public function EnvoiMultiple ( pRequete:URLRequest )

        {

              requete = pRequete;

              addEventListener ( Event.SELECT, selectionFichiers );

        }

        private function selectionFichiers ( pEvt:Event ):void

        {

              historiqueFichiers = new Array();

              fichiers = pEvt.target.fileList;

              suivant();

        }

        private function suivant ( ):void

        {

              var fichierEnCours:FileReference = fichiers.shift();

              historiqueFichiers.push ( fichierEnCours );

              fichierEnCours.addEventListener ( Event.COMPLETE,
transfertTermine );
              fichierEnCours.addEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );

              fichierEnCours.upload ( requete );
```

```
        }

        private function transfertTermine ( pEvt:Event ):void

        {


        }

        private function transfertEnCours ( pEvt:ProgressEvent ):void

        {


        }

    }
}
```

Lorsque le premier fichier a été transféré la méthode écouteur `transfertTermine` est déclenchée. Nous devons ajouter au sein de celle-ci le code nécessaire afin de lancer le transfert suivant, si cela est nécessaire. Nous en profitions pour supprimer l'écoute des événements `Event.COMPLETE` et `ProgressEvent.PROGRESS` auprès de l'objet `FileReference` en cours.

Nous consommons le tableau `fichierEnCours` en déclenchant la méthode `suivant`, si d'autres fichiers sont en attendre de transfert au sein du tableau `fichiers` :

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

{

        pEvt.target.removeEventListener ( Event.COMPLETE, transfertTermine );
        pEvt.target.removeEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );

        if ( fichiers.length ) suivant();

        trace("transfert terminé !");

}

private function transfertEnCours ( pEvt:ProgressEvent ):void

{

        trace("transfert en cours...");

}
```

Il ne nous reste plus qu'à diffuser deux événements pour chaque phase :

- `EvenementEnvoiMultiple.TERMINE` : cet événement est diffusé lorsque la totalité des fichiers séléctionnés sont transférés.

- `ProgressionEvenementEnvoiMultiple.PROGRESSION` : cet événement est diffusé lorsqu'un fichier est en cours de transfert.

Au sein du paquetage `evenements` nous créons la classe `EvenementEnvoiMultiple` :

```
package org.bytearray.evenements

{

  import flash.events.Event;

  public class EvenementEnvoiMultiple extends Event

  {

        public static const TERMINE:String = "termine";
        public var historique:Array;

        public function EvenementEnvoiMultiple ( pType:String,
pHistorique:Array=null )

        {

              // initialisation du constructeur de la classe Event
              super( pType, false, false );

              historique = pHistorique;

        }

        // la méthode clone doit être surchargée
        public override function clone ():Event

        {

              return new EvenementEnvoiMultiple ( type, historique );

        }

        // la méthode toString doit être surchargée
        public override function toString ():String

        {

              return '[EvenementEnvoiMultiple type="'+ type +'" bubbles=' +
bubbles + ' eventPhase='+ eventPhase + ' cancelable=' + cancelable +']';

        }

  }

}
```

Puis la classe `ProgressionEvenementEnvoiMultiple` :

```
package org.bytearray.evenements

{

  import flash.events.ProgressEvent;
```

```
  import flash.events.Event;
  import flash.net.FileReference;

  public class ProgressionEvenementEnvoiMultiple extends ProgressEvent

  {

        public static const PROGRESSION:String = "progression";
        public var fichier:FileReference;

        public function ProgressionEvenementEnvoiMultiple ( pType:String,
pOctetsCharges:Number, pOctetsTotaux:Number, pFichier:FileReference )

        {

              // initialisation du constructeur de la classe Event
              super( pType, false, false, pOctetsCharges, pOctetsTotaux );

              fichier = pFichier;

        }

        // la méthode clone doit être surchargée
        public override function clone ():Event

        {

              return new ProgressionEvenementEnvoiMultiple ( type, bytesLoaded,
bytesTotal, fichier );

        }
        // la méthode toString doit être surchargée
        public override function toString ():String

        {

              return '[ProgressionEvenementEnvoiMultiple type="'+ type +'"
bubbles=' + bubbles + ' eventPhase='+ eventPhase + ' cancelable=' +
cancelable + ' bytesLoaded=' + bytesLoaded +' bytesTotal=' + bytesTotal +']';

        }

  }

}
```

Enfin nous diffusons les événements appropriés au sein des méthodes `transfertEnCours` et `transfertTermine` :

```
package org.bytearray.envoi

{

  import flash.events.Event;
  import flash.events.ProgressEvent;
  import org.bytearray.evenements.EvenementEnvoiMultiple;
  import org.bytearray.evenements.ProgressionEvenementEnvoiMultiple;
  import flash.net.FileReference;
  import flash.net.FileReferenceList;
  import flash.net.URLRequest;

  public class EnvoiMultiple extends FileReferenceList
```

```
{
	private var historiqueFichiers:Array;
	private var fichierEnCours:FileReference;
	private var fichiers:Array;
	private var requete:URLRequest;

	public function EnvoiMultiple ( pRequete:URLRequest )

	{

		requete = pRequete;

		addEventListener ( Event.SELECT, selectionFichiers );

	}

	private function selectionFichiers ( pEvt:Event ):void

	{

		historiqueFichiers = new Array();

		fichiers = pEvt.target.fileList;

		suivant();

	}

	private function suivant ( ):void

	{

		var fichierEnCours:FileReference = fichiers.shift();

		historiqueFichiers.push ( fichierEnCours );

		fichierEnCours.addEventListener ( Event.COMPLETE,
transfertTermine );
		fichierEnCours.addEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );

		fichierEnCours.upload ( requete );

	}

	private function transfertTermine ( pEvt:Event ):void

	{

		pEvt.target.removeEventListener ( Event.COMPLETE,
transfertTermine );
		pEvt.target.removeEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );

		if ( fichiers.length ) suivant();

		else dispatchEvent ( new EvenementEnvoiMultiple (
EvenementEnvoiMultiple.TERMINE, historiqueFichiers ) );

	}
```

```
        private function transfertEnCours ( pEvt:ProgressEvent ):void

        {

            dispatchEvent ( new ProgressionEvenementEnvoiMultiple (
ProgressionEvenementEnvoiMultiple.PROGRESSION, pEvt.bytesLoaded,
pEvt.bytesTotal, FileReference ( pEvt.target ) ) );

        }

    }
}
```

Nous pouvons à présent utiliser la classe `EnvoiMultiple`. Dans un nouveau document Flash CS3, nous associons la classe de document suivante :

```
package org.bytearray.document

{

  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.envoi.EnvoiMultiple;
  import org.bytearray.evenements.EvenementEnvoiMultiple;
  import org.bytearray.evenements.ProgressionEvenementEnvoiMultiple;
  import flash.display.MovieClip;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.text.TextField;

  public class Document extends ApplicationDefaut

  {

        private var envoyeur:EnvoiMultiple;
        public var boutonEnvoyer:SimpleButton;
        public var prechargeur:MovieClip;
        public var legende:TextField;

        public function Document ()

        {

            // création de l'objet EnvoiMultiple
            envoyeur = new EnvoiMultiple( new URLRequest
("http://www.bytearray.org/as3/upload.php" ));

            // écoute des événements personnalisés liés au chargement
            envoyeur.addEventListener (
ProgressionEvenementEnvoiMultiple.PROGRESSION, transfertEnCours );
            envoyeur.addEventListener ( EvenementEnvoiMultiple.TERMINE,
transfertTermine );

            boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

        }

        private function transfertEnCours (
pEvt:ProgressionEvenementEnvoiMultiple ):void
```

```
            {

                // mise à jour de la barre de progression
                prechargeur.scaleX = pEvt.bytesLoaded / pEvt.bytesTotal;

                // informations liées au fichier en cours de transfert
                legende.text = "Transfert en cours : " + pEvt.fichier.name;

            }

            private function transfertTermine ( pEvt:EvenementEnvoiMultiple ):void

            {

                // indique le nombre de fichiers transférés
                legende.text = pEvt.historique.length + " fichiers(s)
    transféré(s)";

            }

            private function parcoursFichiers ( pEvt:MouseEvent ):void

            {

                envoyeur.browse();

            }

        }

    }
```

La document en cours contient une barre de préchargement `prechargeur` ainsi qu'un champ texte `legende` posés sur la scène afin d'afficher les informations relatives au transfert des fichiers.

La figure 14-19 illustre l'application :

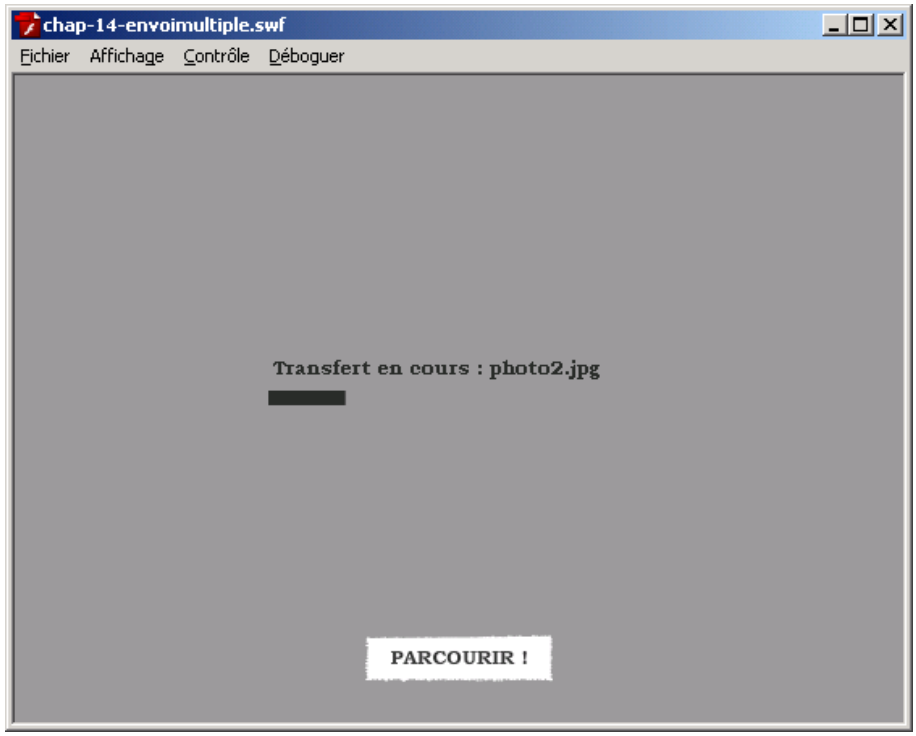

Figure 14-19. Transfert de fichiers.

Lorsque les images ont été transférées, l'événement `EvenementEnvoiMultiple.TERMINE` est diffusé, cela nous permet d'afficher un message approprié et de réagir en conséquence :

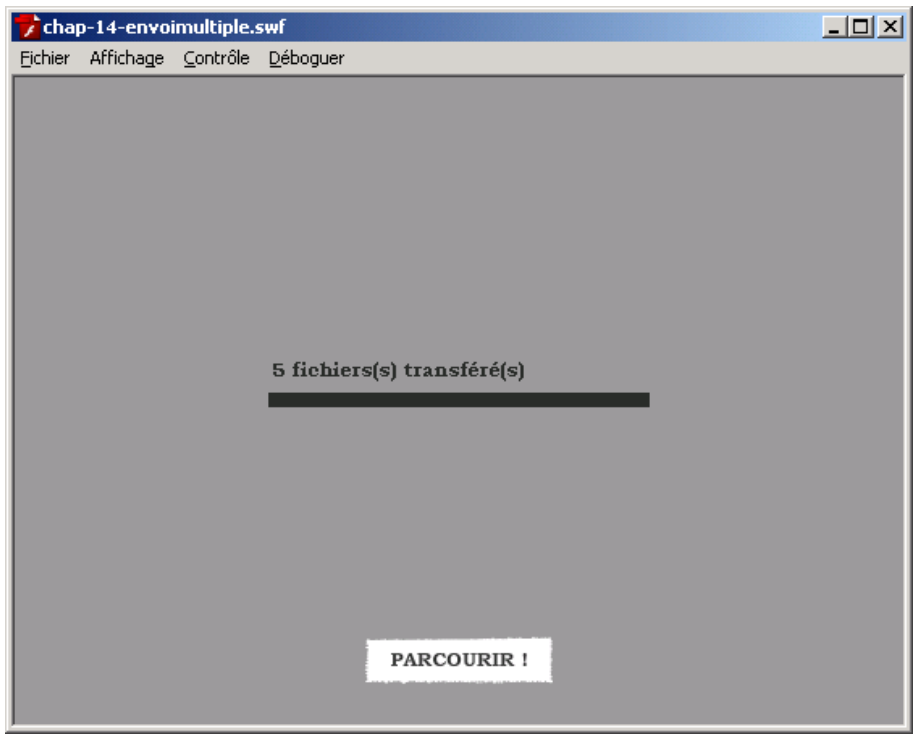

Figure 14-20. Transfert de fichiers terminé.

La propriété `historique` de l'objet événementiel `EvenementEnvoiMultiple` est un tableau contenant des objets `FileReference` associé à chaque fichier transféré. En récupérant sa longueur nous pouvons indiquer le nombre de fichiers transférés.

A vous d'ajouter à présent, une instanciation dynamique de la jauge de transfert et de la légende puis de les supprimer une fois l'événement `EvenementEnvoiMultiple.TERMINE` est diffusé.

Retourner des données une fois l'envoi terminé

ActionScript 3 ajoute un événement très intéressant à la classe `FileReference` que nous allons découvrir à présent.

Dans les précédentes versions d'ActionScript, il était impossible d'obtenir des informations du serveur lorsque le transfert était terminé. Très souvent, les développeurs souhaitaient retourner des informations indiquant si le transfert des données avait échoué ou non.

L'événement `FileFerence.COMPLETE` n'offre pas la possibilité de renvoyer les informations provenant du serveur. En revanche, ActionScript 3 introduit un événement `DataEvent.UPLOAD_COMPLETE_DATA` offrant cette possibilité.

Nous allons ajouter cette fonctionnalité à notre classe `EnvoiMultiple` et renvoyer des informations afin d'indiquer à l'application Flash que l'écriture sur le serveur a réussi ou non.

Nous allons modifier le script serveur gérant l'enregistrement des fichiers de manière à indiquer si l' :

```
<?php

$fichier = $_FILES["Filedata"];

if ( isset ( $fichier ) )

{

  $nomFichier = $fichier['name'];

  if ( move_uploaded_file ( $fichier['tmp_name'], $nomFichier) ) echo
"resultat=1";

  else echo "resultat=0";

} else echo "resultat=0";

?>
```

Voici le code complet de la classe `EnvoiMultiple` :

```
package org.bytearray.envoi

{

  import flash.events.DataEvent;
  import flash.events.Event;
  import flash.events.ProgressEvent;
  import org.bytearray.evenements.EvenementEnvoiMultiple;
  import org.bytearray.evenements.ProgressionEvenementEnvoiMultiple;
  import flash.net.FileReference;
  import flash.net.FileReferenceList;
  import flash.net.URLRequest;

  public class EnvoiMultiple extends FileReferenceList

  {

        private var historiqueFichiers:Array;
        private var fichierEnCours:FileReference;
        private var fichiers:Array;
        private var requete:URLRequest;

        public function EnvoiMultiple ( pRequete:URLRequest )

        {

              requete = pRequete;

              addEventListener ( Event.SELECT, selectionFichiers );

        }

        private function selectionFichiers ( pEvt:Event ):void

        {

              historiqueFichiers = new Array();

              fichiers = pEvt.target.fileList;

              suivant();

        }

        private function suivant ( ):void

        {

              var fichierEnCours:FileReference = fichiers.shift();

              historiqueFichiers.push ( fichierEnCours );

              fichierEnCours.addEventListener ( Event.COMPLETE,
transfertTermine );
              fichierEnCours.addEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );
              fichierEnCours.addEventListener ( DataEvent.UPLOAD_COMPLETE_DATA,
retourServeur );

              fichierEnCours.upload ( requete );

        }
```

```
        private function retourServeur ( pEvt:DataEvent ):void

        {

             dispatchEvent ( pEvt );

             pEvt.target.removeEventListener ( DataEvent.UPLOAD_COMPLETE_DATA,
retourServeur );

        }

        private function transfertTermine ( pEvt:Event ):void

        {

             pEvt.target.removeEventListener ( Event.COMPLETE,
transfertTermine );
             pEvt.target.removeEventListener ( ProgressEvent.PROGRESS,
transfertEnCours );

             if ( fichiers.length ) suivant();

             else dispatchEvent ( new EvenementEnvoiMultiple (
EvenementEnvoiMultiple.TERMINE, historiqueFichiers ) );

        }

        private function transfertEnCours ( pEvt:ProgressEvent ):void

        {

             dispatchEvent ( new ProgressionEvenementEnvoiMultiple (
ProgressionEvenementEnvoiMultiple.PROGRESSION, pEvt.bytesLoaded,
pEvt.bytesTotal, FileReference ( pEvt.target ) ) );

        }

    }
}
```

Il ne reste plus qu'à écouter cet événement auprès de l'objet `EnvoiMultiple` crée :

```
package org.bytearray.document

{

  import flash.events.DataEvent;
  import flash.net.URLRequest;
  import flash.net.URLVariables;
  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.envoi.EnvoiMultiple;
  import org.bytearray.evenements.EvenementEnvoiMultiple;
  import org.bytearray.evenements.ProgressionEvenementEnvoiMultiple;
  import flash.display.MovieClip;
  import flash.display.SimpleButton;
  import flash.events.MouseEvent;
  import flash.text.TextField;

  public class Document extends ApplicationDefaut
```

```
  {

        private var envoyeur:EnvoiMultiple;
        public var boutonEnvoyer:SimpleButton;
        public var prechargeur:MovieClip;
        public var legende:TextField;

        public function Document ()

        {

              // création de l'objet EnvoiMultiple
              envoyeur = new EnvoiMultiple( new URLRequest
("http://www.bytearray.org/as3/upload.php" ));

              // écoute des événements personnalisés liés au chargement
              envoyeur.addEventListener (
ProgressionEvenementEnvoiMultiple.PROGRESSION, transfertEnCours );
              envoyeur.addEventListener ( EvenementEnvoiMultiple.TERMINE,
transfertTermine );
              envoyeur.addEventListener ( DataEvent.UPLOAD_COMPLETE_DATA,
retourServeur );

              boutonEnvoyer.addEventListener ( MouseEvent.CLICK,
parcoursFichiers );

        }

        private function retourServeur ( pEvt:DataEvent ):void

        {

              var variables:URLVariables = new URLVariables ( pEvt.data );

              var retour:Number = Number ( variables.resultat );

              legende.text = retour ? "Transfert réussi !" : "Echec de
transfert";

        }

        private function transfertEnCours (
pEvt:ProgressionEvenementEnvoiMultiple ):void

        {

              // mise à jour de la barre de progression
              prechargeur.scaleX = pEvt.bytesLoaded / pEvt.bytesTotal;

              // informations liées au fichier en cours de transfert
              legende.text = "Transfert en cours : " + pEvt.fichier.name;

        }

        private function transfertTermine ( pEvt:EvenementEnvoiMultiple ):void

        {

              // indique le nombre de fichiers transférés
              legende.text = pEvt.historique.length + " fichiers(s)
transféré(s)";
```

```
            }

            private function parcoursFichiers ( pEvt:MouseEvent ):void

            {

                  envoyeur.browse();

            }

      }

}
```

Vous remarquerez que nous utilisons la classe `URLVariables` afin de décoder la chaîne encodée URL retournée par le serveur.

Nous avons achevé notre classe `EnvoiMultiple` qui pourra être réutilisée dans tous les projets nécessitant un transfert de fichiers synchronisés.

A retenir

- La classe `FileReferenceList` envoie tous les fichiers en même temps et ne diffuse pas d'événement approprié lorsque tous les fichiers ont été transférés.
- Afin de corriger cela, nous avons crée une classe `EnvoiMultiple`.

Aller plus loin

Souvenez-vous, lors du chapitre 10 intitulé *Diffusion d'événements personnalisés* nous avions entamé la conception d'une classe `ChargeurXML` pouvant charger un flux XML et diffuser un événement `Event.COMPLETE` :

```
package

{

  import flash.events.Event;
  import flash.events.EventDispatcher;
  import flash.net.URLRequest;

  public class ChargeurXML extends EventDispatcher

  {

        public function ChargeurXML ()

        {

        }

        public function charge ( pRequete:URLRequest ):void
```

```
        {

        }

        public function chargementTermine ( pEvt:Event ):void

        {

        }

  }


}
```

Nous allons la compléter en ajoutant créant un objet `URLoader` interne :

```
package org.bytearray.xml

{

  import flash.events.Event;
  import flash.events.IOErrorEvent;
  import flash.events.HTTPStatusEvent;
  import flash.events.SecurityErrorEvent;
  import flash.events.EventDispatcher;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;

  public class ChargeurXML extends EventDispatcher

  {

        private var chargeur:URLLoader;
        private var fluxXML:XML;

        public function ChargeurXML ()

        {

              chargeur = new URLLoader();

              chargeur.dataFormat = URLLoaderDataFormat.TEXT;

              chargeur.addEventListener ( Event.OPEN, redirigeEvenement );
              chargeur.addEventListener ( ProgressEvent.PROGRESS,
redirigeEvenement );
              chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
              chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS,
redirigeEvenement );
              chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
redirigeEvenement );
              chargeur.addEventListener ( SecurityErrorEvent.SECURITY_ERROR,
redirigeEvenement );

        }

        private function redirigeEvenement ( pEvt:Event ):void

        {
```

```
                dispatchEvent ( pEvt );
            }
            public function charge ( pRequete:URLRequest ):void
            {
                chargeur.load ( pRequete );
            }
            private function chargementTermine ( pEvt:Event ):void
            {
                try
                {
                    fluxXML = new XML ( pEvt.target.data );
                } catch ( pErreur:Error )
                {
                    trace (pErreur);
                }
                dispatchEvent ( pEvt );
            }
            public function get donnees ( ):XML
            {
                return fluxXML;
            }
    }
}
```

Une fois le fichier XML chargé nous diffusons un événement `Event.COMPLETE`. Ainsi, le code permettant de charger un fichier XML est extrêmement réduit :

```
package org.bytearray.document
{
  import org.bytearray.abstrait.ApplicationDefaut;
  import org.bytearray.xml.ChargeurXML;
  import flash.events.Event;
  import flash.net.URLRequest;

  public class Document extends ApplicationDefaut
  {
```

```
        public function Document ()
        {
            var chargeur:ChargeurXML = new ChargeurXML ();
            chargeur.charge ( new URLRequest ("donnees.xml") );
            chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
        }
        private function chargementTermine ( pEvt:Event ):void
        {
            /* affiche :
            <MENU>
            <BOUTON legende="Accueil" couleur="0x887400" vitesse="1"
url="accueil.swf"/>
            <BOUTON legende="Photos" couleur="0x005587" vitesse="1"
url="photos.swf"/>
            <BOUTON legende="Blog" couleur="0x125874" vitesse="1"
url="blog.swf"/>
            <BOUTON legende="Liens" couleur="0x59CCAA" vitesse="1"
url="liens.swf"/>
            <BOUTON legende="Forum" couleur="0xEE44AA" vitesse="1"
url="forum.swf"/>
            </MENU>
            */
            trace( pEvt.target.donnees );
        }
  }
}
```

La classe `ChargeurXML` peut ainsi être utilisée dans de nombreux projets ayant recourt au XML. Si un matin, un membre de votre équipe se plaint de la mise en cache des XML par le navigateur.

Assurez-lui, qu'un système anti-cache peut être ajouté à la classe `ChargeurXML` :

```
package org.bytearray.xml

{

  import flash.events.Event;
  import flash.events.IOErrorEvent;
  import flash.events.HTTPStatusEvent;
  import flash.events.ProgressEvent;
  import flash.events.SecurityErrorEvent;
  import flash.events.EventDispatcher;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;
  import flash.net.URLRequestHeader;
```

```
public class ChargeurXML extends EventDispatcher

{

      private var chargeur:URLLoader;
      private var fluxXML:XML;
      private var antiCache:Boolean;

      public function ChargeurXML ( pAntiCache:Boolean=false )

      {

            antiCache = pAntiCache;

            chargeur = new URLLoader();

            chargeur.dataFormat = URLLoaderDataFormat.TEXT;

            chargeur.addEventListener ( Event.OPEN, redirigeEvenement );
            chargeur.addEventListener ( ProgressEvent.PROGRESS,
redirigeEvenement );
            chargeur.addEventListener ( Event.COMPLETE, chargementTermine );
            chargeur.addEventListener ( HTTPStatusEvent.HTTP_STATUS,
redirigeEvenement );
            chargeur.addEventListener ( IOErrorEvent.IO_ERROR,
redirigeEvenement );
            chargeur.addEventListener ( SecurityErrorEvent.SECURITY_ERROR,
redirigeEvenement );

      }

      public function charge ( pRequete:URLRequest ):void

      {

            if ( antiCache ) pRequete.requestHeaders.push ( new
URLRequestHeader ("pragma", "no-cache") );

            chargeur.load ( pRequete );

      }

      private function redirigeEvenement ( pEvt:Event ):void

      {

            dispatchEvent ( pEvt );

      }

      private function chargementTermine ( pEvt:Event ):void

      {

            try

            {

                  fluxXML = new XML ( pEvt.target.data );

            } catch ( pErreur:Error )
```

```
            {
                trace (pErreur);
            }
            dispatchEvent ( pEvt );
        }
        public function get donnees ( ):XML
        {
            return fluxXML;
        }
    }
}
```

Grace à la classe `URLRequestHeader`, nous avons modifié l'entête HTTP du lecteur Flash lors du chargement du fichier XML et empêché sa mise en cache par le navigateur.

Ainsi chaque développeur souhaitant tirer profit de cette fonctionnalité devra simplement le préciser lors de l'instanciation de l'objet `ChargeurXML` :

```
// crée un objet ChargeurXML en précisant de jamais mettre en cache les fichiers XML chargés
var chargeur:ChargeurXML = new ChargeurXML ( true );
```

D'autres entêtes HTTP peuvent être utilisées, nous reviendrons sur la classe `URLRequestHeader` au cours du chapitre 20 intitulé `ByteArray`.

A retenir

- La classe `URLRequestHeader` permet de modifier les entêtes HTTP du lecteur Flash.

Dans le prochain chapitre, nous explorerons les nouveautés apportées par ActionScript 3 en matière d'échanges entre le lecteur Flash et les différents navigateurs.