

19

Flash Remoting

LA TECHNOLOGIE.....	1
UN FORMAT OPTIMISÉ	2
PASSERELLE REMOTING	4
DÉPLOIEMENT.....	5
LE SERVICE.....	7
SE CONNECTER AU SERVICE	15
LA CLASSE RESPONDER	16
APPELER UNE MÉTHODE DISTANTE	18
ÉCHANGER DES DONNEES PRIMITIVES	22
ÉCHANGER DES DONNEES COMPOSITES	27
ÉCHANGER DES DONNEES TYPEES	32
ENVOYER UN EMAIL AVEC FLASH REMOTING	33
EXPORTER UNE IMAGE	40
SE CONNECTER A UNE BASE DE DONNEES.....	49
SECURITE	61
LA CLASSE SERVICE	62

La technologie

Développé à l'origine par Macromedia, *Flash Remoting* est une technologie visant à optimiser grandement les échanges client-serveur.

Le lecteur Flash 6 fut le premier à en bénéficier, mais la puissance de celle-ci ne fut pas perçue immédiatement par la communauté due en partie à un manque de documentation et d'informations à ce sujet. Grâce à Flash Remoting nous allons optimiser notre temps de développement d'applications dynamiques et apprendre à penser différemment nos échanges client-serveur. Des frameworks tels Flex

ou AIR s'appuient aujourd'hui fortement sur Flash Remoting trop souvent inconnu des développeurs Flash.

Nous allons découvrir au sein de ce chapitre comment tirer profit de cette technologie en ActionScript 3 à travers différentes applications.

A retenir

- La technologie Flash Remoting vu le jour en 2001 au sein du lecteur 6 (Flash MX).
- Flash Remoting permet d'optimiser les échanges client-serveur.

Un format optimisé

Toute la puissance de Flash Remoting réside dans le format d'échanges utilisé, connu sous le nom d'AMF (*Action Message Format*).

Afin de bien comprendre l'intérêt de ce format, il convient de revenir sur le fonctionnement interne du lecteur Flash dans un contexte de chargement et d'envoi de données.

Nous avons vu au cours du chapitre 14 intitulé *Chargement et envoi de données* que les données échangées entre le lecteur Flash et le script serveur étaient par défaut réalisées au format texte.

Dans le cas d'échanges de variables encodées URL, une représentation texte des variables doit être réalisée. Nous devons donc nous assurer manuellement de la sérialisation et désérialisation des données ce qui peut s'avérer long et fastidieux, surtout dans un contexte de large flux de données.

Nous pouvons alors utiliser le format XML, mais là encore bien que l'introduction de la norme ECMAScript pour XML (E4X) ait sensiblement amélioré les performances d'interprétation de flux XML, dans un contexte d'échanges, le flux XML reste transporté au format texte ce qui n'est pas optimisé en terme de bande passante.

Grâce au format AMF, le lecteur Flash se charge de sérialiser et désérialiser les données ActionScript nativement. Nous pouvons ainsi échanger des données complexes typées, sans se soucier de la manière dont cela est réalisé.

Lorsque nous souhaitons transmettre des données par Flash Remoting, le lecteur encode un paquet AMF binaire compressé contenant les données sérialisées, puis le transmet au script serveur par la méthode POST par le biais du protocole HTTP.

Voici un extrait de la transaction HTTP :

```
POST gateway.php HTTP/1.1
Host: www.bytearray.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0
.8,image/png,*/*;q=0.5
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.bytearray.org/wp-
content/projects/jpegencoder/media_snapshot.swf
Content-type: application/x-amf
Content-length: 4944
Paquet AMF
```

Une fois le paquet AMF décodé par le serveur, ce dernier répond au lecteur Flash en lui transmettant un nouveau paquet AMF de réponse :

```
HTTP/1.1 200 OK
Date: Sat, 02 Feb 2008 02:55:40 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) PHP/4.3.10-22 mod_ssl/2.0.54
OpenSSL/0.9.7e mod_perl/1.999.21 Perl/v5.8.4
X-Powered-By: PHP/4.3.10-22
Expires: Sat, 2 Feb 2008 03:55:40 GMT
Cache-Control: no-store
Pragma: no-store
Content-length: 114
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: application/x-amf
Paquet AMF
```

Il est important de comprendre que contrairement au format texte, le format AMF est un format binaire compressé natif du lecteur Flash.

L'utilisation du format AMF nous permet donc d'optimiser la bande passante utilisée et d'augmenter les performances d'une application en s'affranchissant totalement de toute sérialisation manuelle des données.

James Ward a dernièrement publié une application comparant les performances d'échanges au format JSON, XML et AMF.

L'application est disponible à l'adresse suivante :

<http://www.jamesward.org/blazebench/>

Bien que le lecteur Flash puisse nativement encoder ou décoder un paquet AMF, l'application serveur se doit elle aussi de pouvoir décoder ce même paquet.

C'est ici qu'intervient la notion de *passerelle remoting*.

A retenir

- L'acronyme AMF signifie *Action Message Format*.
- Flash Remoting est basé sur un échange des données au format AMF binaire par le biais du protocole HTTP.
- Encoder et décoder un paquet AMF est beaucoup plus rapide et moins gourmand qu'une sérialisation et désérialisation au format texte.
- Deux versions du format AMF existent aujourd'hui : AMF0 (ActionScript 1 et 2) et AMF3 (ActionScript 3).
- Les formats AMF0 et AMF3 sont ouverts et documentés.
- Aujourd'hui, le format AMF est utilisé au sein des classes `NetConnection`, `LocalConnection`, `SharedObject`, `ByteArray`, `Socket` et `URLStream`.

Passerelle remoting

Lorsque le lecteur Flash transmet un paquet AMF à un script serveur, celui-ci n'est pas par défaut en mesure de le décoder.

Des projets appelés passerelles remoting ont ainsi vu le jour, permettant à des langages serveurs tels PHP, Java ou C# de comprendre le format AMF. Chaque passerelle est ainsi liée à un langage serveur et se charge de convertir le flux AMF transmis en données compatibles.

Afin de développer ces passerelles, le format AMF a dû être piraté par la communauté afin de comprendre comment le décoder. Initialement non documenté, Adobe a finalement rendu public les spécifications du format AMF en décembre 2007. Toute personne souhaitant développer une passerelle pour un langage spécifique peut donc se baser sur ces spécifications fournies par Adobe.

Celles ci peuvent être téléchargées à l'adresse suivante :

<http://labs.adobe.com/technologies/blazeds/>

En plus d'ouvrir le format AMF, Adobe a décidé de fournir une passerelle officielle pour la plateforme Java J2EE nommée *BlazeDS* téléchargeable à la même adresse.

Il existe aujourd'hui un grand nombre de passerelles remoting pour la plupart open source, le tableau suivant regroupe les plus connues associées à chaque langage :

Nom	Langage	Open Source	Lien
-----	---------	-------------	------

AMFPHP	PHP	Oui	http://www.amfphp.org
Web ORB	PHP, .NET, Ruby, Java.	Non	http://www.themidnightcoders.com
BlazeDS	Java	Oui	http://labs.adobe.com/technologies/blazeds/
Ruby AMF	Ruby	Oui	http://www.rubyamf.org
Fluorine	.NET	Oui	http://fluorine.thesilentgroup.com

Tableau 1. Passerelles AMF.

La figure 19-1 illustre le modèle de communication entre le lecteur Flash et la passerelle remoting :

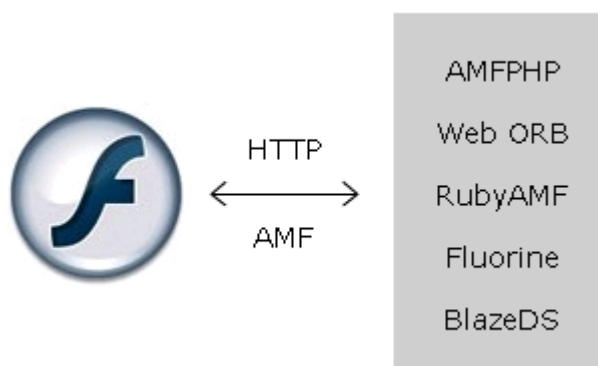


Figure 19-1. Communication entre le lecteur Flash et la passerelle remoting.

Les paquets AMF devant forcément transiter par la passerelle afin d’être décodés, le lecteur ne se connecte donc *jamais* directement auprès du script serveur, mais auprès de la passerelle.

Nous allons utiliser tout au long du chapitre, la passerelle remoting AMFPHP afin de simplifier nos échanges avec le serveur.

Déploiement

Avant de déployer AMFPHP il convient de télécharger le projet à l’adresse suivante :

<http://sourceforge.net/projects/amfphp>

Attention, le format AMF3 n'est pris en charge que depuis la version 1.9, nous veillerons donc à télécharger une version égale ou ultérieure.

Souvenez-vous, ActionScript 3 utilise le format AMF3, les anciennes versions d'ActionScript utilisent le format AMF0.

Une fois AMFPHP téléchargé, nous extrayons l'archive et obtenons les répertoires illustrés par la figure 19-2 :

Nom	Taille	Type
browser		Dossier de fichiers
core		Dossier de fichiers
services		Dossier de fichiers
.htaccess	1 Ko	Fichier HTACCESS
gateway.php	6 Ko	OpenStudio File
globals.php	1 Ko	OpenStudio File
json.php	1 Ko	OpenStudio File
phpinfo.php	1 Ko	OpenStudio File
xmlrpc.php	1 Ko	OpenStudio File

Figure 19-2. Fichiers AMFPHP.

Voici le détail des trois répertoires ainsi que du fichier `gateway.php` :

- `browser` : contient une application de débogage que nous allons découvrir très bientôt.
- `core` : il s'agit des sources d'AMFPHP. Dans la majorité des cas nous n'irons jamais au sein de ce répertoire.
- `services` : les services distants sont placés au sein de ce répertoire. C'est le répertoire le plus important pour nous.
- `gateway.php` : la passerelle à laquelle nous nous connectons depuis Flash.

Une des forces des différentes passerelles remoting réside dans leur simplicité de déploiement sur le serveur.

Les passerelles sont déployables sur des serveurs mutualisés et ne nécessitent aucun accès privilégié au niveau de l'administration du serveur. AMFPHP requiert une version PHP 4.3.0 ou supérieure, l'utilisation de PHP 5 ajoutant quelques fonctionnalités intéressantes à AMFPHP.

Nous plaçons le répertoire `amfphp` contenant les fichiers illustrés précédemment sur notre serveur au sein d'un répertoire `echanges` et accédons à l'adresse suivante pour vérifier que tout fonctionne correctement :

<http://localhost/echanges/gateway.php>

Si tout fonctionne, nous obtenons le message suivant :

```
amfphp and this gateway are installed correctly. You may now connect to this gateway from Flash.
```

```
Note: If you're reading an old tutorial, it will tell you that you should see a download window instead of this message. This confused people so this is the new behaviour starting from amfphp 1.2.
```

Si vous souhaitez utiliser une autre passerelle remoting que AMFPHP, soyez rassurés, le fonctionnement de la plupart d'entre elles est similaire.

A retenir

- La passerelle remoting joue le rôle d'intermédiaire entre le lecteur Flash et l'application serveur.
- Le déploiement d'une passerelle remoting ne prend que quelques secondes.
- Aucun droit spécifique n'est nécessaire auprès du serveur. Il est donc possible d'utiliser Flash Remoting sur un serveur mutualisé.
- AMFPHP requiert une version de PHP 4.3.0 au minimum.

Le service

Contrairement aux moyens de communication étudiés précédemment, Flash Remoting définit la notion de *service*. Le service est une classe définissant les méthodes que nous souhaitons appeler depuis l'application Flash.

Nous pourrions ainsi appeler le service "classe distante".

Un des avantages de Flash Remoting est lié à sa conception orientée objet. Au lieu d'appeler différents scripts serveurs stockés au sein de plusieurs fichiers PHP, nous appelons directement depuis Flash des méthodes définies au sein du service.

La figure 19-3 illustre l'idée :

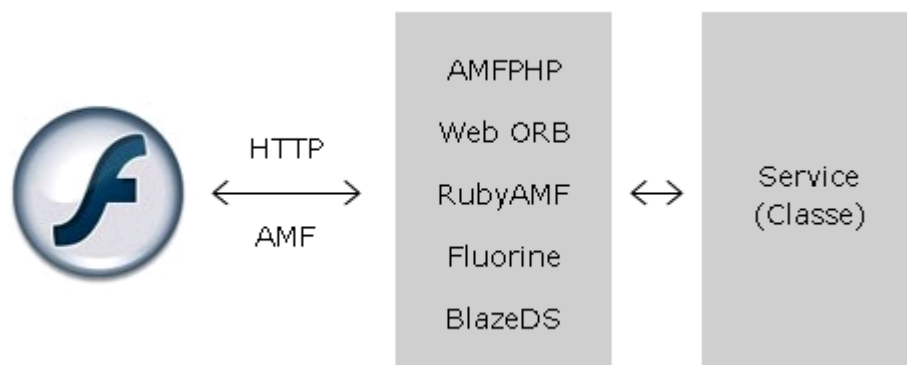


Figure 19-3. Service distant associé à la passerelle.

Nous allons créer notre premier service en plaçant au sein du répertoire `services`, une classe PHP nommée `Echanges.php`.

Celle-ci est définie au sein d'un paquetage `org.bytearray.test` :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function premierAppel ( )
    {
    }
}
?>
```

La classe `Echanges` doit donc être accessible à l'adresse suivante :

<http://localhost/echanges/services/org/bytearray/test/Echanges.php>

Notons que contrairement à ActionScript 3, PHP 4 et 5 n'intègrent pas de mot clé `package`, ainsi la classe ne contient aucune indication liée à son paquetage. La classe PHP `Echanges` définit une seule méthode `premierAppel` que nous allons pouvoir tester directement depuis un navigateur grâce à un outil extrêmement pratique appelé *Service Browser* (explorateur de services).

Dans notre exemple, l'explorateur de service est accessible à l'adresse suivante :

<http://localhost/echanges/browser/>

L'explorateur de service est une application Flex développée afin de pouvoir tester depuis notre navigateur les différentes méthodes de notre service. La partie gauche de l'application indique les services actuellement déployés. En déroulant les nœuds nous pouvons accéder à un service spécifique.

La figure 19-4 illustre l'explorateur :

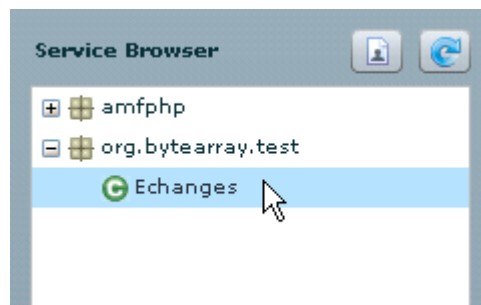


Figure 19-4. Explorateur de service.

En sélectionnant un service, la partie droite de l'application découvre les méthodes associées au service.

Chaque méthode est accessible et peut être testée directement depuis le navigateur. Cela permet de pouvoir développer la logique serveur sans avoir à tester depuis Flash.

L'onglet *Results* affiche alors le résultat de l'exécution de la méthode distante `premierAppel` :

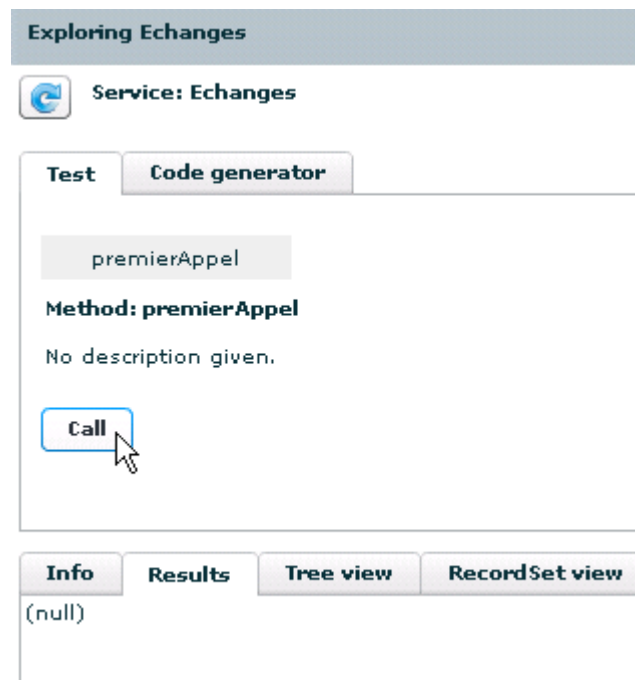


Figure 19-5. Explorateur de méthodes liées au service.

Dans notre exemple, la méthode `premierAppel` ne renvoie aucune valeur, l'onglet *Results* affiche donc `null`.

Afin de retourner des informations à Flash, la méthode doit *obligatoirement* utiliser le mot clé `return` :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function premierAppel ( )
    {
        return "cela fonctionne sans problème !";
    }
}
?>
```

Si nous testons à nouveau la méthode, nous pouvons voir directement les valeurs retournées :



Figure 19-6. Explorateur de méthodes liées au service.

La possibilité de pouvoir exécuter depuis le navigateur les méthodes distantes est un avantage majeur. L'explorateur service doit être considéré comme un véritable outil de débogage de service.

Au cas où une erreur PHP serait présente au sein du service, l'explorateur de service nous l'indique au sein de l'onglet *Results*

Dans le code suivant, nous oublions d'ajouter un point virgule en fin d'expression :

```
<?php  
  
class Echanges  
{  
  
    function Echanges ( )  
  
    {  
  
    }  
  
    function premierAppel ()  
  
    {  
  
        return "cela fonctionne sans problème !" ;  
  
    }  
  
}  
  
?>
```

En tentant d'exécuter la méthode, le message suivant est affiché au sein de l'onglet *Results* :

```
Parse error: parse error, unexpected '}' in C:\Program Files\EasyPHP  
2.0b1\www\echanges\services\org\bytearray\test\Echanges.php on line 18
```

Les onglets situés en dessous de l'onglet *Test* nous apportent d'autres informations comme les temps d'exécution ou le poids des données transférées.

La figure 19-7 illustre les informations liées à l'appel de la méthode *premierAppel* :

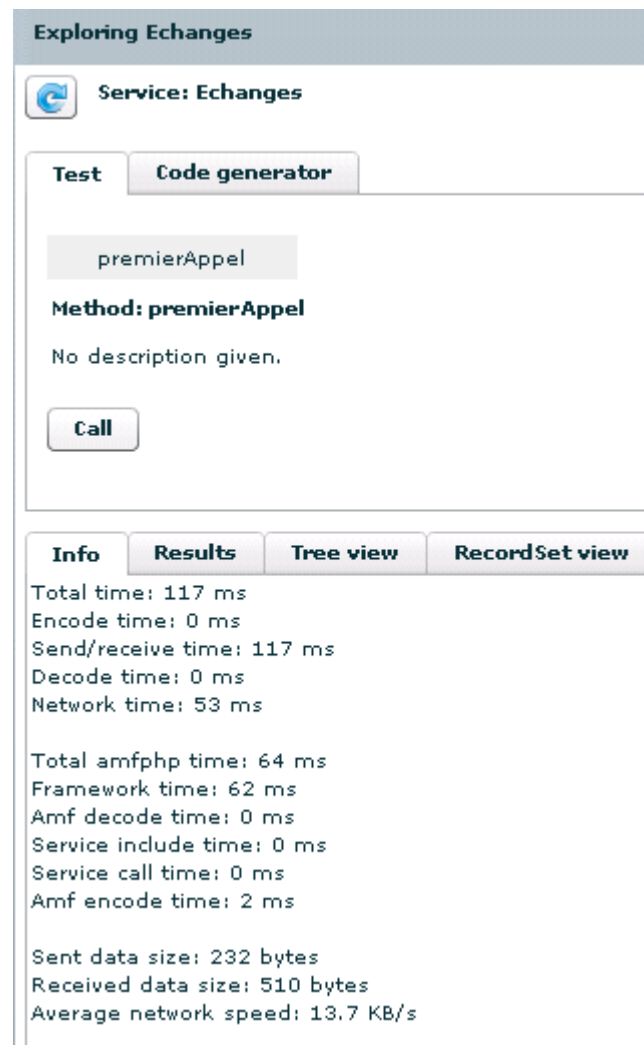


Figure 19-7. Onglet Info.

Imaginons que nous souhaitons afficher lors de l'exécution de la méthode `premierAppel` la longueur d'une chaîne.

Nous pouvons utiliser pour cela la méthode statique `trace` de la classe `NetDebug` intégrée à AMFPHP.

Dans le code suivant, nous affichons la longueur de la variable `$chaine`:

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }
}
```

```
function premierAppel ()
{
    $chaine = "cela fonctionne sans problème !";
    NetDebug::trace( "infos persos : " . strlen ( $chaine ) );
}
}
?>
```

En exécutant la méthode, l'onglet *Trace* retourne le message personnalisé telle une fenêtre de sortie classique.

La figure 19-7 illustre le message affiché :

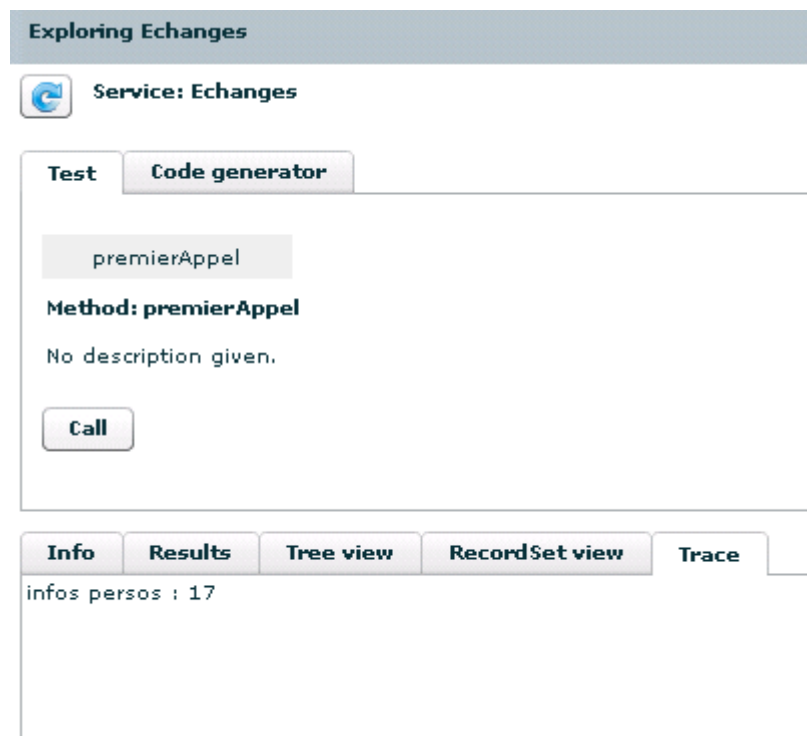


Figure 19-7. Utilisation du débogage personnalisé.

Cette fonctionnalité s'avère très précieuse lors de débogage de scripts, nous découvrirons l'intérêt des autres onglets très rapidement.

Si vous disposez de PHP 5 sur votre serveur, vous avez la possibilité d'utiliser les modificateurs de contrôle d'accès similaire à ActionScript 3.

En définissant une méthode comme privée, celle-ci ne pourra plus être appelée depuis Flash, mais seulement depuis une autre méthode de la classe :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    private function premierAppel ( )
    {
        return "cela fonctionne sans problème !";
    }
}
?>
```

L'explorateur de méthodes considère alors que cette méthode n'est plus disponible depuis l'extérieur.

La figure 19-8 illustre l'état de l'explorateur de service :

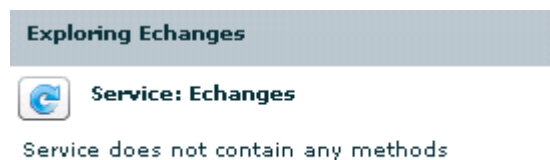


Figure 19-8. Aucune méthode disponible auprès du service.

Nous retrouvons ici encore l'intérêt de la technologie Flash Remoting où nous évoluons dans un contexte orienté objet. L'utilisation de tels mécanismes améliore fortement la souplesse de développement d'une application dynamique.

Il est temps de coder quelques lignes d'ActionScript, nous allons dès maintenant nous connecter au service **Echanges** et exécuter la méthode **premierAppel**.

A retenir

- Le service distant est une classe définissant différentes méthodes accessibles depuis Flash.
- L'explorateur de service nous permet de tester les méthodes en direct afin d'optimiser le débogage.

Se connecter au service

Afin de se connecter à un service distant en ActionScript 3, nous disposons de différentes classes dont voici le détail :

- `flash.net.NetConnection` : la classe `NetConnection` permet de se connecter à un service distant
- `flash.net.Socket` : la classe `Socket` offre une connectivité TCP/IP offrant la possibilité d'échanger des données au format AMF à l'aide du protocole HTTP.

Afin d'appeler la méthode `premierAppel`, nous créons une instance de la classe `NetConnection` :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();
```

La classe `NetConnection` définit une propriété `objectEncoding` permettant de définir la version d'AMF utilisé pour les échanges.

En ActionScript 3, le format AMF3 est utilisé par défaut :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// affiche : 3
trace( connexion.objectEncoding );

// affiche : true
trace( connexion.objectEncoding == ObjectEncoding.AMF3 );
```

Au cas où nous souhaiterions nous connecter à une passerelle n'étant pas compatible avec AMF3, nous devons spécifier explicitement d'utiliser le format AMF0 :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// utilisation du format AMF0 pour les échanges
connexion.objectEncoding = ObjectEncoding.AMF0;
```

La version 1.9 d'AMFPHP étant compatible avec le format AMF3, nous ne modifions pas la propriété `objectEncoding` et nous connectons à la passerelle `gateway.php` à l'aide de la méthode `connect` :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// connexion à la passerelle AMFPHP
```

```
| connexion.connect ("http://localhost/echanges/gateway.php");
```

Même si l'adresse de la passerelle est erronée ou inaccessible, l'objet `NetConnection` ne renverra aucune erreur lors de l'appel de la méthode `connect`.

L'objet `NetConnection` diffuse différents événements dont voici le détail :

- `AsyncErrorEvent.ASYNC_ERROR` : diffusé lorsqu'une erreur asynchrone intervient.
- `IOErrorEvent.IO_ERROR` : diffusé lorsque la transmission des données échoue.
- `NetStatusEvent.NET_STATUS` : diffusé lorsqu'une erreur fatale intervient.
- `SecurityErrorEvent.SECURITY_ERROR` : diffusé lorsque nous tentons d'appeler la méthode d'un service évoluant sur un autre domaine sans en avoir l'autorisation.

Nous écoutons les différents événements en passant une seule fonction écouteur afin de centraliser la gestion des erreurs :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// connexion à la passerelle amfphp
connexion.connect ("http://localhost/echanges/gateway.php");

// écoute des différents événements
connexion.addEventListener( NetStatusEvent.NET_STATUS, erreurConnexion);
connexion.addEventListener( IOErrorEvent.IO_ERROR, erreurConnexion);
connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion);
connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR, erreurConnexion);

function erreurConnexion ( pEvt:Event ):void
{
    trace( pEvt );
}
```

Nous devons maintenant gérer le retour du serveur, Flash CS3 intègre pour cela une classe spécifique.

La classe `Responder`

Avant d'appeler une méthode distante, il convient de créer au préalable un objet de gestion de retour des appels grâce à la classe `flash.net.Responder`.

Le constructeur de la classe `Responder` possède la signature suivante :


```
public function Responder(result:Function, status:Function = null)
```

Seul le premier paramètre est obligatoire :

- **result** : référence à la fonction gérant le retour du serveur en cas de réussite de l'appel.
- **status** : référence à la fonction gérant le retour du serveur en cas d'échec de l'appel.

Dans le code suivant nous créons un objet **Responder** en passant deux fonctions **succes** et **echec** :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// connexion à la passerelle amfphp
connexion.connect ("http://localhost/echanges/gateway.php");

// écoute des différents événements
connexion.addEventListener( NetStatusEvent.NET_STATUS, erreurConnexion );
connexion.addEventListener( IOErrorEvent.IO_ERROR, erreurConnexion );
connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR, erreurConnexion );
connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR, erreurConnexion );

function erreurConnexion ( pEvt:Event ):void
{
    trace( pEvt );
}

// création des fonctions de gestion de retour serveur
function succes ( pRetour:* ):void
{
    trace("retour serveur");
}

function echec ( pErreur:* ):void
{
    trace("echec de l'appel");
}

// création d'un gestionnaire de retour des appels
var retourServeur:Responder = new Responder (succes, echec);
```

Une fois l'objet **Responder** défini, nous pouvons appeler la méthode distante.

A retenir

- Les classes `NetConnection` et `Socket` permettent de se connecter à une passerelle remoting à l'aide de la méthode `connect`.
- En ActionScript 3, le format AMF3 est utilisé par défaut.
- Afin de gérer le retour du serveur, nous devons créer un objet `Responder`.
- Ce dernier nécessite deux fonctions qui seront déclenchées en cas de succès ou échec de l'appel.

Appeler une méthode distante

Afin d'appeler la méthode distante, nous utilisons la méthode `call` de l'objet `NetConnection` dont voici la signature :

```
public function call(command:String, responder:Responder, ... arguments):void
```

Les deux premiers paramètres sont obligatoires :

- `command` : la méthode du service à appeler.
- `responder` : l'objet `Responder` traitant les retours serveurs.
- `arguments` : les paramètres à transmettre à la méthode distante.

Lorsque la méthode `call` est appelée, un paquet AMF est créé contenant le nom de la méthode à exécuter au sein du service ainsi que les données à transmettre.

Dans le code suivant, nous passons le chemin complet de la méthode à appeler ainsi qu'une instance de la classe `Responder` pour gérer le retour du serveur :

```
// création de la connexion
var connexion:NetConnection = new NetConnection ();

// connexion à la passerelle amfphp
connexion.connect ("http://localhost/echanges/gateway.php");

// création des fonctions de gestion de retour serveur
function succes ( pRetour:* ):void
{
    trace("retour serveur");
}

function echec ( pErreur:* ):void
{
    trace("echec de l'appel");
}

// création d'un objet de gestion de l'appel
var retourServeur:Responder = new Responder (succes, echec);
```

```
// appel de la méthode distante
connexion.call ("org.bytearray.test.Echanges.premierAppel", retourServeur);
```

En testant le code précédent, nous voyons que la fonction `succes` est exécutée.

Attention, n'oubliez pas que Flash fonctionne de manière asynchrone. La méthode `call` de l'objet `NetConnection` ne renvoie donc aucune valeur. Seul l'objet `Responder` passé en paramètre permet de gérer le résultat de l'appel.

Pour récupérer les données renvoyées par le serveur nous utilisons le paramètre `pRetour` de la fonction `succes` :

```
function succes ( pRetour:* ):void
{
    // affiche : cela fonctionne sans problème !
    trace ( pRetour );
}
```

Nous venons d'appeler notre première méthode distante par Flash Remoting en quelques lignes seulement.

Souvenez-vous, lors du chapitre 14 intitulé *Chargement et envoi de données* nous devions nous assurer manuellement du décodage et de l'encodage UTF-8 afin de préserver les caractères spéciaux.

Le format AMF encode les chaînes de caractères en UTF-8, et AMFPHP se charge automatiquement du décodage. Il n'est donc plus nécessaire de se soucier de l'encodage des caractères spéciaux avec Flash Remoting.

Nous allons à présent modifier la méthode `premierAppel` afin que celle-ci accepte un paramètre `$pMessage` :

```
<?php

class Echanges
{
    function Echanges ( )
    {
    }

    function premierAppel ( $pMessage )
    {
        return "vous avez dit : $pMessage ?";
    }
}
```

```
}
?>
```

Lorsqu'un paramètre est ajouté à une méthode, l'explorateur de service nous donne la possibilité de passer dynamiquement les valeurs au sein d'un champ texte de saisie :



Figure 19-9. Paramètre.

Si nous testons à nouveau le code ActionScript précédent, nous remarquons que la fonction `echec` est déclenchée.

La méthode `premierAppel` attend désormais un paramètre. En l'omettant, une erreur est levée que nous pouvons intercepter grâce à la fonction `echec` passée en paramètre à l'objet `Responder`.

En ciblant la propriété `description` de l'objet retourné nous obtenons des informations liées à l'erreur en cours :

```
function echec ( pErreur:* ):void
{
    // affiche : Missing argument 1 for Echanges::premierAppel()
    trace( pErreur.description );
}
```

Les informations renvoyées par AMFPHP nous permettent ainsi de déboguer l'application plus facilement. Ce type de mécanisme illustre la puissance de Flash Remoting en matière de débogage.

Dans le cas présent, nous apprenons de manière explicite que nous avons omis le paramètre attendu de la méthode distante `premierAppel`.

Si nous itérons sur l'objet retourné nous découvrons de nouvelles propriétés :

```
function echec ( pErreur:* ):void
{
    /* affiche :
    details : C:\wamp\www\echanges\services\org\bytearray\test\Echanges.php
    level : Unknown error type
    description : Missing argument 1 for Echanges::premierAppel()
    line : 12
    code : AMFPHP_RUNTIME_ERROR
    */
    for ( var p:String in pErreur )
    {
        trace( p + " : " + pErreur[p] );
    }
}
```

Dans le code suivant, nous tentons d'appeler une méthode inexistante dû à une faute de frappe :

```
connexion.call ( "org.bytearray.test.Echanges.premierApel", retourServeur);
```

La propriété `description` de l'objet retourné par le serveur nous indique qu'une telle méthode n'existe pas au sein du service :

```
function echec ( pErreur:* ):void
{
    // affiche : The method {premierApel} does not exist in class {Echanges}.
    trace( pErreur.description );
}
```

De la même manière, si nous nous trompons de service :

```
connexion.call ( "org.bytearray.test.Echange.premierAppel", retourServeur);
```

AMFPHP nous oriente à nouveau vers la cause de l'erreur :

```
function echec ( pErreur:* ):void
{
    // affiche : The class {Echange} could not be found under the class path
    {C:\wamp\www\echanges\services\org\bytearray\test\Echange.php}
    trace( pErreur.description );
}
```

```
| }  
|
```

Nous allons à présent nous intéresser aux différents types de données échangeables et découvrir l'intérêt de la sérialisation automatique assurée par Flash Remoting.

A retenir

- La méthode `call` de l'objet `NetConnection` permet d'appeler une méthode distante.
- Si l'appel réussit, la fonction `succes` passée en paramètre à l'objet `Responder` est exécutée.
- Dans le cas contraire, la fonction `echec` est déclenchée.
- Flash Remoting gère l'encodage de caractères spéciaux automatiquement.

Echanger des données primitives

Comme nous l'avons expliqué précédemment, Flash Remoting se charge de la sérialisation et désérialisation des données.

Afin de nous rendre compte de ce comportement, nous modifions la méthode `premierAppel` en la renommant `echangesTypes` :

```
<?php  
class Echanges  
{  
    function Echanges ( )  
    {  
    }  
    function echangeTypes ( $pDonnees )  
    {  
        return gettype ( $pDonnees );  
    }  
}  
?>
```

Celle-ci accepte un paramètre et retourne désormais son type grâce à la méthode PHP `getType`. Celle-ci est similaire à l'instruction `typeof` d'ActionScript 3.

Cette méthode va nous permettre de savoir sous quel type les données envoyées depuis Flash arrivent côté PHP.

Nous modifions les paramètres passés à la méthode `call` en spécifiant le nouveau nom de méthode ainsi que le valeur booléenne `true` :

```
connexion.call ("org.bytearray.test.Echanges.echangeTypes", retourServeur, true);
```

La fonction `succes` reçoit alors la chaîne `boolean` :

```
function succes ( pRetour:* ):void
{
    // affiche : boolean
    trace ( pRetour );
}
```

Nous voyons que la passerelle AMFPHP a automatiquement conservé le type booléen entre ActionScript et PHP, on dit alors que les données échangées sont *supportées*.

Mais que se passe t-il si AMFPHP ne trouve pas de type équivalent ?

Dans ce cas, AMFPHP interprète le type de données et convertit les données en un type équivalent.

Dans le code suivant, nous passons la valeur 5 de type `int` :

```
connexion.call ("org.bytearray.Echanges.echangeTypes", retourServeur, 5);
```

La méthode distante `echangeTypes` renvoie alors le type reçu, la fonction `succes` affiche les données retournées :

```
function succes ( pRetour:* ):void
{
    // affiche : double
    trace ( pRetour );
}
```

Nous voyons que la passerelle AMFPHP a automatiquement convertit le type `int` en `double`, c'est-à-dire son équivalent PHP.

La valeur 5 est envoyée au sein d'un paquet AMF à la méthode distante, AMFPHP déséréalise automatiquement le paquet AMF et convertit le type `int` en un type compatible PHP. On dit alors que les données échangées sont *interprétées*.

Le tableau ci-dessous regroupe les différents types primitifs supportés et interprétés par AMFPHP :

Type ActionScript	Type PHP	Conversion automatique	Version AMF
-------------------	----------	------------------------	-------------

null	NULL	Oui	AMF0 / AMF3
int	double	Oui	AMF3
uint	double	Oui	AMF3
Number	double	Oui	AMF0 / AMF3
Boolean	boolean	Oui	AMF0 / AMF3
String	string	Oui	AMF0 / AMF3

Tableau 2. Tableau des types primitifs supportés et interprétés.

Cette conversion automatique permet d'échanger des données primitives sans aucun travail de notre part. AMFPHP se charge de toute la sérialisation dans les deux sens, ce qui nous permet de nous concentrer sur d'autres parties plus importantes de l'application telles que l'interface ou autres.

Sans Flash Remoting, nous aurions du sérialiser les données sous la forme d'une chaîne de caractères, puis désérialiser les données côté serveur manuellement.

Grâce à AMFPHP, ce processus est simplement supprimé.

Bien entendu, dans la plupart des applications dynamiques, nous n'échangeons pas de simples données comme celle-ci.

Si nous souhaitons concilier l'avantage des deux technologies, nous pouvons faire transiter une chaîne de caractères compressée au sein d'AMF puis transformer celle-ci en objet XML au sein de Flash.

Nous ajoutons une nouvelle méthode `recupereMenu` :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function exchangeTypes ( $pDonnees )
    {
```



```

        return gettype ( $pDonnees );
    }

    function recupereMenu ()
    {
        $menuXML = '<MENU>
            <RUBRIQUE titre = "Nouveautés" id="12">
                <RUBRIQUE titre = "CD" id="487"/>
                <RUBRIQUE titre = "Vinyls" id="540"/>
            </RUBRIQUE>
            <RUBRIQUE titre = "Concerts" id="25">
                <RUBRIQUE titre = "Funk" id="15"/>
                <RUBRIQUE titre = "Soul" id="58"/>
            </RUBRIQUE>
        </MENU>';

        return $menuXML;
    }
}

?>

```

En appelant la méthode `recupereMenu`, nous recevons la chaîne de caractère que nous transformons aussitôt en objet XML :

```

// création de la connexion
var connexion:NetConnection = new NetConnection ();

// connexion à la passerelle amfphp
connexion.connect ( "http://localhost/echanges/gateway.php" );

// création des fonctions de gestion de retour serveur
function succes ( pRetour:* ):void
{
    try
    {
        // conversion de la chaîne en objet XML
        var menu:XML = new XML ( pRetour );

        /* affiche :
        <RUBRIQUE titre="Concerts" id="25">
            <RUBRIQUE titre="Funk" id="15"/>
            <RUBRIQUE titre="Soul" id="58"/>
        </RUBRIQUE>
        */
        trace(menu.RUBRIQUE.(@id == 25) );

        //affiche : Funk
        trace(menu..RUBRIQUE.(@id == 15).@titre );

    } catch ( pErreur:Error )
    {

```

```
        trace( "erreur d'interprétation du flux XML");
    }
}

function echec ( pErreur:* ):void
{
    trace("echec de l'appel");
}

// création d'un objet de gestion de l'appel
var retourServeur:Responder = new Responder (succes, echec);

// appel de la méthode recupereMenu distante, récupération du flux xml du menu
connexion.call ("org.bytearray.test.Echanges.recupereMenu", retourServeur);
```

En utilisant cette approche, nous conservons :

- La puissance de débogage de Flash Remoting.
- L'appel de méthodes distantes directement depuis Flash.
- La simplicité du format XML.
- La puissance de la norme E4X.

Même si la conversion de la chaîne en objet XML par ActionScript pourrait ralentir les performances de l'application si le flux XML devient important, cela resterait minime dans notre exemple.

Il peut être nécessaire d'échanger des données plus complexes, nous allons voir que Flash Remoting va à nouveau nous faciliter les échanges.

A retenir

- Grâce à Flash Remoting les données primitives sont automatiquement sérialisées et désérialisées.
- Lorsqu'AMFPHP trouve un type correspondant, on dit que les données sont supportées.
- Dans le cas contraire, AMFPHP interprète les données en un type équivalent.
- Il est tout à fait possible de concilier un format de représentation de données tel XML avec Flash Remoting.
- Le compromis XML et AMF est une option élégante à prendre en considération.

Echanger des données composites

Dans la plupart des applications dynamiques, nous souhaitons généralement échanger des données plus complexes, comme des tableaux ou des objets associatifs.

Voici le tableau des différents types composites supportés et interprétés par AMFPHP :

Type ActionScript	Type PHP	Conversion automatique	Version AMF
Object	associative array	Oui	AMF0 / AMF3
Array	array	Oui	AMF0 / AMF3
XML (E4X)	string	Non	AMF3
XMLDocument	string	Non	AMF0 / AMF3
Date	double	Non	AMF0 / AMF3
RecordSet	Ressource MySQL	Oui (Uniquement de MySQL vers Flash)	AMF0 / AMF3
ByteArray	ByteArray (classe AMFPHP)	Oui	AMF3

Tableau 3. Tableau des types composites supportés et interprétés.

Dans le code suivant nous envoyons à la méthode distante un objet

`Date` :

```
connexion.call ("org.bytearray.test.Echanges.echangeTypes", retourServeur, new Date());
```

L'objet `Date` est transmis, AMFPHP ne trouvant pas de type équivalent convertit alors l'objet `Date` en timestamp Unix compatible.

Ainsi, nous recevons côté PHP un `double` :

```
function succes ( pRetour:* ):void
{
    // affiche : double
    trace( pRetour );
}
```

Très souvent, nous avons besoin d'envoyer au serveur un objet contenant différentes informations.

Dans le code suivant, nous envoyons au serveur un tableau associatif contenant les informations liées à un utilisateur :

```
// création d'un joueur fictif
var joueur:Object = new Object();

joueur.nom = "Groove";
joueur.prenom = "Bob";
joueur.age = 29;
joueur.ville = "Paris";
joueur.points = 35485;

// appel de la méthode distante, le joueur est envoyé à la méthode distante
connexion.call ("org.bytearray.test.Echanges.echangeTypes", retourServeur, joueur);
```

Nous modifions la méthode distante en renvoyant simplement les données passées en paramètre :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function echangeTypes ( $pDonnees )
    {
```

```
        return $pDonnees;
    }
}
?>
```

L'objet est retourné à Flash sous sa forme originale :

```
function succes ( pRetour:* ):void
{
    /* affiche :
    nom   : Groove
    prenom : Bob
    age   : 29
    ville : Paris
    points : 35485
    */
    for ( var p:String in pRetour )
    {
        trace( p, " : " + pRetour[p] );
    }
}
```

Nous remarquons donc que la sérialisation automatique est opérée dans les deux sens et nous permet de gagner un temps précieux.

Coté PHP nous accédons aux propriétés de l'objet de manière traditionnelle. Nous pouvons par exemple augmenter l'âge et le score du joueur passé en paramètre :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function exchangeTypes ( $pDonnees )
    {
        $pDonnees["age"] += 10;
        $pDonnees["points"] += 500;

        return $pDonnees;
    }
}
?>
```

L'objet est retourné modifié au lecteur Flash :

```
function succes ( pRetour:* ):void
{
    /* affiche :
    nom : Groove
    ville : Paris
    prenom : Bob
    age : 39
    points : 35985
    */
    for ( var p:String in pRetour )
    {
        trace( p, " : " + pRetour[p] );
    }
}
```

Grâce au format AMF3 et la version 1.9 d'AMFPHP, il est possible d'échanger des objets de types `ByteArray`.

Dans le code suivant, nous créons un tableau d'octets vierge, puis nous écrivons des données texte :

```
// création d'un tableau d'octets vierge
var fluxBinaire:ByteArray = new ByteArray();

// écriture de données texte
fluxBinaire.writeUTFBytes("Voilà du texte encodé en binaire !");
```

L'instance de `ByteArray` est ensuite transmise à la méthode distante :

```
// création d'un tableau d'octets vierge
var fluxBinaire:ByteArray = new ByteArray();

// écriture de données texte
fluxBinaire.writeUTFBytes("Voilà du texte encodé en binaire !");

// appel de la méthode distante, le ByteArray est envoyé au serveur
connexion.call ( "org.bytearray.test.Echanges.echangeTypes", retourServeur,
fluxBinaire);
```

Lorsque la fonction `succes` est exécutée, l'objet `ByteArray` retourné par le serveur est intact et les données préservées :

```
function succes ( pRetour:* ):void
{
    // affiche : true
    trace( pRetour is ByteArray );

    // affiche : Voilà du texte encodé en binaire !
    trace( pRetour.readUTFBytes ( pRetour.bytesAvailable ) );
}
```

Bien que l'intérêt puisse paraître limité, cette fonctionnalité s'avère extrêmement puissante. Nous reviendrons très bientôt sur l'intérêt d'échanger des instances de `ByteArray`.

Malheureusement, certains objets ne peuvent être sérialisés au format AMF, c'est le cas des objets de type `DisplayObject`.

En modifiant la méthode `exchangeTypes` nous pourrions penser pouvoir renvoyer l'objet graphique transmis :

```
<?php
class Echanges
{
    function Echanges ( )
    {
    }

    function exchangeTypes ( $pDonnees )
    {
        return $pDonnees;
    }
}
?>
```

Si nous tentons de passer une instance de `MovieClip` à cette même méthode :

```
// création d'une instance de MovieClip
var animation:MovieClip = new MovieClip();

// appel de la méthode distante, un objet graphique est passé à la méthode distante
connexion.call ("org.bytearray.test.Echanges.exchangeTypes", retourServeur,
animation);
```

La sérialisation AMF échoue, la méthode distante `exchangeTypes` renvoie la valeur `null` :

```
function succes ( pRetour:* ):void
{
    // affiche : null
    trace( pRetour );

    // affiche : true
    trace( pRetour == null )
}
```

Cette limitation n'est pas due à la passerelle remoting AMFPHP mais au format AMF qui ne gère pas au jour d'aujourd'hui la sérialisation et désérialisation d'objets graphiques.

Dans certains cas, nous pouvons avoir besoin de transmettre un flux XML. En passant un objet de type XML, celui-ci est alors aplati sous la forme d'une chaîne de caractères UTF-8 :

```
// création d'un objet XML
var donneesXML:XML = <SCORE><JOUEUR id='25' score='15888' /></SCORE>;

// appel de la méthode echangeTypes distante, nous transmettons un objet XML
connexion.call ("org.bytearray.test.Echanges.echangeTypes", retourServeur,
donneesXML);
```

Le flux XML revient sous la forme d'une chaîne de caractères :

```
function succes ( pRetour:* ):void
{

    // l'objet XML a été converti sous forme de chaîne de caractères
    trace( pRetour is String );

}
```

AMFPHP préfère conserver une chaîne et nous laisser gérer la manipulation de l'arbre XML.

A retenir

- Grâce à Flash Remoting les données composites sont automatiquement sérialisées et désérialisées.
- Lorsqu'AMFPHP trouve un type correspondant, on dit que les données sont supportées.
- Dans le cas contraire, AMFPHP interprète les données en un type équivalent.
- Les objets graphiques ne peuvent ne sont pas compatibles avec le format AMF.

Echanger des données typées

Au cas où nous souhaiterions échanger des types personnalisés avec la passerelle Remoting, AMFPHP intègre un mécanisme approprié.

Une instance de classe personnalisée de type **Utilisateur** peut par exemple être définie côté Flash et transmise au service distant, en conservant côté PHP le type **Utilisateur**.

De la même manière, une méthode distante peut renvoyer à Flash des instances de classes en assurant une conservation des types personnalisés.

Pour plus d'informations, rendez vous à l'adresse suivante :

<http://amfphp.org/docs/classmapping.html>

Passons maintenant de la théorie à la pratique.

Envoyer un email avec Flash Remoting

Au cours du chapitre 14 intitulé *Chargement et envoi de données* nous avons développé un formulaire permettant d'envoyer un email depuis Flash.

Nous allons reprendre l'application et remplacer les échanges réalisés à l'aide la classe `URLLoader` par un échange par Flash Remoting.

La figure 19-10 illustre le formulaire :

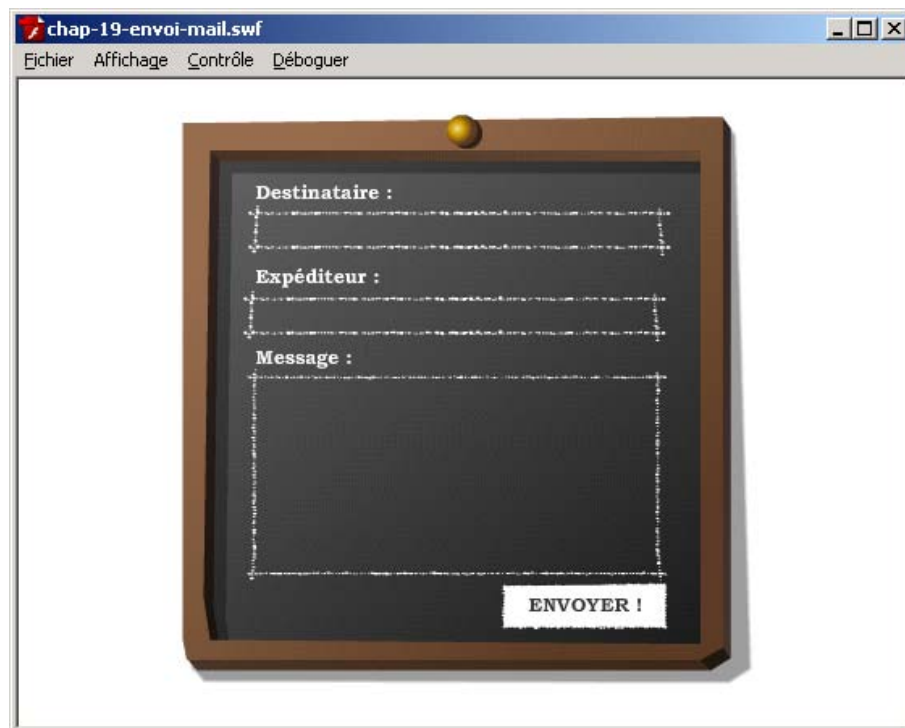


Figure 19-10. Formulaire d'envoi d'email.

La classe de document suivante est associée :

```
package org.bytearray.document  
{  
    import flash.text.TextField;  
    import flash.display.SimpleButton;  
    import org.bytearray.abstrait.ApplicationDefault;  
  
    public class Document extends ApplicationDefault
```

```
    {  
  
        public var destinataire:TextField;  
        public var sujet:TextField;  
        public var message:TextField;  
        public var boutonEnvoi:SimpleButton;  
  
        public function Document ()  
        {  
  
        }  
  
    }  
}
```

Afin de pouvoir envoyer notre email, nous devons tout d'abord prévoir une méthode d'envoi. Pour cela nous ajoutons une méthode `envoiMessage` à notre service distant :

```
<?php  
class Echanges {  
  
    function exchangeTypes ( $pDonnees )  
    {  
  
        return $pDonnees;  
  
    }  
  
    function envoiMessage ( $pInfos )  
    {  
  
        $destinaire = $pInfos["destinataire"];  
        $sujet = $pInfos["sujet"];  
        $message = $pInfos["message"];  
  
        return @mail ( $destinaire, $sujet, $message );  
  
    }  
}  
?>
```

La méthode `envoiMessage` reçoit un tableau associatif contenant les informations nécessaires et procède à l'envoi du message.

La valeur renvoyée par la fonction PHP `mail` indiquant le succès ou l'échec de l'envoi est retournée à Flash.

Afin d'appeler la méthode `envoiMessage` nous ajoutons les lignes suivantes à la classe de document définie précédemment :

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

```
{

    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.SecurityErrorEvent;
    import flash.events.AsyncErrorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetConnection;
    import flash.text.TextField;
    import flash.display.SimpleButton;
    import org.bytearray.abstrait.ApplicationDefault;

    public class Document extends ApplicationDefault

    {

        public var destinataire:TextField;
        public var sujet:TextField;
        public var message:TextField;
        public var boutonEnvoi:SimpleButton;
        private var connexion:NetConnection;

        public function Document ()

        {

            //création de la connexion
            connexion = new NetConnection();

            // écoute des différents événements
            connexion.addEventListener( NetStatusEvent.NET_STATUS,
            erreurConnexion );
            connexion.addEventListener( IOErrorEvent.IO_ERROR,
            erreurConnexion );
            connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
            erreurConnexion );
            connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
            erreurConnexion );

        }

        private function erreurConnexion ( pEvt:Event ):void

        {

            trace( pEvt );

        }

    }

}
```

Puis nous ajoutons une propriété constante **PASSERELLE** contenant l'adresse de la passerelle et nous nous connectons à celle-ci :

```
package org.bytearray.document

{

    import flash.events.Event;
```

```
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.events.AsyncErrorEvent;
import flash.events.NetStatusEvent;
import flash.text.TextField;
import flash.display.SimpleButton;
import flash.net.NetConnection;
import org.bytearray.abstrait.ApplicationDefault;

public class Document extends ApplicationDefault
{
    public var destinataire:TextField;
    public var sujet:TextField;
    public var message:TextField;
    public var boutonEnvoi:SimpleButton;
    private var connexion:NetConnection;

    // adresse de la passerelle AMFPHP
    private static const PASSERELLE:String =
"http://localhost/echanges/gateway.php";

    public function Document ()
    {
        //création de la connexion
        connexion = new NetConnection();

        // écoute des différents événements
        connexion.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
        connexion.addEventListener( IOErrorEvent.IO_ERROR,
erreurConnexion );
        connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
        connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );

        // connexion à la passerelle
        connexion.connect ( Document.PASSERELLE );
    }

    private function erreurConnexion ( pEvt:Event ):void
    {
        trace( pEvt );
    }
}
```

Nous ajoutons une instance de **Responder** afin de gérer les retours serveurs :

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

```

{

import flash.events.Event;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.events.AsyncErrorEvent;
import flash.events.NetStatusEvent;
import flash.net.Responder;
import flash.text.TextField;
import flash.display.SimpleButton;
import flash.net.NetConnection;
import org.bytearray.abstrait.ApplicationDefault;

public class Document extends ApplicationDefault
{

    public var destinataire:TextField;
    public var sujet:TextField;
    public var message:TextField;
    public var boutonEnvoi:SimpleButton;
    private var connexion:NetConnection;
    private var retourServeur:Responder;

    // adresse de la passerelle AMFPHP
    private static const PASSERELLE:String =
"http://localhost/echanges/gateway.php";

    public function Document ()
    {

        //création de la connexion
        connexion = new NetConnection();

        // création de l'objet de gestion des retours serveur
        retourServeur = new Responder ( succes, echec );

        // écoute des différents événements
        connexion.addEventListener( NetStatusEvent.NET_STATUS,
ecouteurCentralise );
        connexion.addEventListener( IOErrorEvent.IO_ERROR,
ecouteurCentralise );
        connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
ecouteurCentralise );
        connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
ecouteurCentralise );

        // connexion à la passerelle
        connexion.connect ( Document.PASSERELLE );

    }

    private function succes ( pRetour:* ):void
    {

        trace ( pRetour );

    }

    private function echec ( pErreur:* ):void

```

```
        {  
            trace ( pErreur );  
        }  
        private function ecouteurCentralise ( pEvt:Event ):void  
        {  
            trace( pEvt );  
        }  
    }  
}
```

Puis nous appelons la méthode distante `envoiMessage` lorsque le bouton `boutonEnvoi` est cliqué :

```
package org.bytearray.document  
{  
    import flash.events.Event;  
    import flash.events.MouseEvent;  
    import flash.events.IOErrorEvent;  
    import flash.events.SecurityErrorEvent;  
    import flash.events.AsyncErrorEvent;  
    import flash.events.NetStatusEvent;  
    import flash.net.Responder;  
    import flash.text.TextField;  
    import flash.display.SimpleButton;  
    import flash.net.NetConnection;  
    import org.bytearray.abstrait.ApplicationDefaut;  
  
    public class Document extends ApplicationDefaut  
    {  
        public var destinataire:TextField;  
        public var sujet:TextField;  
        public var message:TextField;  
        public var boutonEnvoi:SimpleButton;  
        private var connexion:NetConnection;  
        private var retourServeur:Responder;  
        private var infos:Object;  
  
        // adresse de la passerelle AMFPHP  
        private static const PASSERELLE:String =  
        "http://localhost/echanges/gateway.php";  
  
        public function Document ()  
        {  
            //création de la connexion  
            connexion = new NetConnection();  
  
            // création de l'objet de gestion des retours serveur
```

```
        retourServeur = new Responder ( succes, echec );

        // écoute des différents événements
        connexion.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
        connexion.addEventListener( IOErrorEvent.IO_ERROR,
erreurConnexion );
        connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
        connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );

        // connexion à la passerelle
        connexion.connect ( Document.PASSERELLE );

        boutonEnvoi.addEventListener ( MouseEvent.CLICK, envoiMail );
    }

    private function erreurConnexion ( pEvt:Event ):void
    {
        trace( pEvt );
    }

    private function succes ( pRetour:* ):void
    {
        trace ( pRetour );
    }

    private function echec ( pErreur:* ):void
    {
        trace ( pErreur );
    }

    private function envoiMail ( pEvt:MouseEvent ):void
    {
        infos = new Object();

        infos.destinataire = destinataire.text;
        infos.sujet = sujet.text;
        infos.message = message.text;

        connexion.call ( "org.bytearray.Echanges.envoiMessage",
retourServeur, infos );
    }
}
}
```

En testant le code précédent, la méthode écouteur `succes` est déclenchée et reçoit la valeur `true` du serveur indiquant que le mail a bien été envoyé.

Afin de finaliser cet exemple, nous pouvons utiliser la classe `OutilsFormulaire` développée au cours du chapitre 14 et l'importer :

```
| import org.bytearray.ouutils.FormulaireOutils;
```

Puis nous modifions la méthode `envoiMail` afin de tester la validité de l'adresse saisie :

```
| private function envoiMail ( pEvt:MouseEvent ):void
| {
|     if ( FormulaireOutils.verifieEmail ( destinataire.text ) )
|     {
|         infos = new Object();
|
|         infos.destinataire = destinataire.text;
|         infos.sujet = sujet.text;
|         infos.message = message.text;
|
|         connexion.call ( "org.bytearray.Echanges.envoiMessage", retourServeur,
| infos );
|     } else destinataire.text = "Email non valide !";
| }
| }
```

Afin de valider l'envoi du message au sein de l'application nous ajoutons la condition suivante au sein de la méthode de retour `succes` :

```
| private function succes ( pRetour:* ):void
| {
|     if ( pRetour ) message.text = "Message bien envoyé !";
|
|     else message.text = "Erreur d'envoi du message";
| }
| }
```

Ainsi, nous travaillons de manière transparente avec le script serveur sans avoir à nous soucier de sérialiser et désérialiser les données envoyées et reçues.

Exporter une image

Nous avons vu précédemment qu'il était possible de transmettre une instance de `ByteArray` par Flash Remoting.

Comme nous le verrons au cours du chapitre 21 intitulé *ByteArray* il est possible de générer en ActionScript 3 un flux binaire grâce à la classe *ByteArray*.

Afin de pouvoir exploiter ce flux binaire, nous pouvons le transmettre au service distant afin que celui-ci le sauvegarde sur le serveur.

Nous allons reprendre l'application de dessin développée au cours du chapitre 9 intitulé *Etendre les classes natives* et ajouter une fonctionnalité d'export de notre dessin sous la forme d'une image PNG.

Nous définissons une classe de document associée :

```
package org.bytearray.document
{
    import flash.display.SimpleButton;
    import flash.display.Sprite;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Document extends ApplicationDefaut
    {
        private var dessin:Sprite;
        private var stylo:Stylo;
        public var boutonEnvoi:SimpleButton;

        public function Document ()
        {
            // création du conteneur de tracés vectoriels
            dessin = new Sprite();

            // ajout du conteneur à la liste d'affichage
            addChild ( dessin );

            // création du symbole
            stylo = new Stylo( .1 );

            // passage du conteneur de tracés
            stylo.affecteToile ( dessin );

            // ajout du symbole à la liste d'affichage
            addChild ( stylo );

            // positionnement en x et y
            stylo.x = 250;
            stylo.y = 200;
        }
    }
}
```

Nous ajoutons un bouton `boutonEnvoi`, permettant d'exporter le dessin sous forme bitmap.

La figure 19-11 illustre l'application :

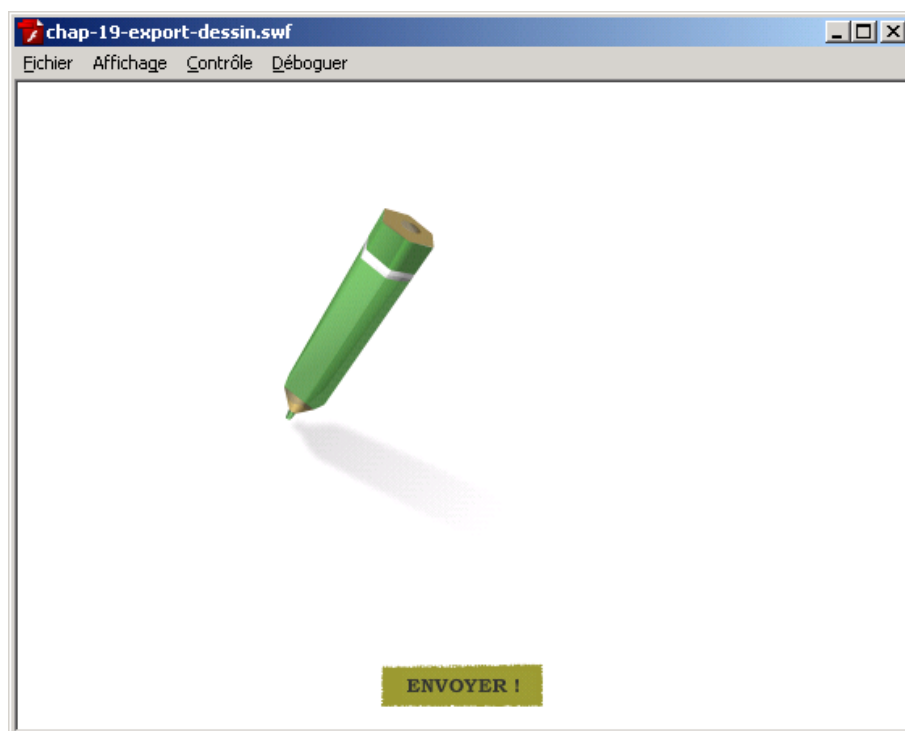


Figure 19-11. Application de dessin.

Lors du clic bouton nous devons générer une image bitmap du dessin vectoriel, pour cela nous allons utiliser une classe d'encodage d'image `EncodeurPNG`.

Nous écoutons l'événement `MouseEvent.CLICK` du bouton d'export :

```
package org.bytearray.document
{
    import flash.display.SimpleButton;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import org.bytearray.abstrait.ApplicationDefault;

    public class Document extends ApplicationDefault
    {
        private var dessin:Sprite;
        private var stylo:Stylo;
        private var renduBitmap:BitmapData;

        public function Document ()
```

```
{  
  
    // création du conteneur de tracés vectoriels  
    dessin = new Sprite();  
  
    // ajout du conteneur à la liste d'affichage  
    addChild ( dessin );  
  
    // création du symbole  
    stylo = new Stylo( .1 );  
  
    // passage du conteneur de tracés  
    stylo.affecteToile ( dessin );  
  
    // ajout du symbole à la liste d'affichage  
    addChild ( stylo );  
  
    // positionnement en x et y  
    stylo.x = 250;  
    stylo.y = 200;  
  
    boutonEnvoi.addEventListener ( MouseEvent.CLICK, exportBitmap );  
  
}  
  
private function exportBitmap ( pEvt:MouseEvent ):void  
{  
  
    trace("export bitmap");  
  
}  
  
}
```

Afin d'encoder notre dessin vectoriel en image PNG nous devons tout d'abord rendre sous forme bitmap les tracés vectoriels.

Souvenez-vous, nous avons découvert lors du chapitre 12 intitulé *Programmation bitmap* qu'il était possible de rasteriser un élément vectoriel grâce à la méthode `draw` de la classe `BitmapData`.

Nous modifions la classe de document afin d'intégrer une rasterisation du dessin vectoriel lorsque le bouton d'export est cliqué :

```
package org.bytearray.document  
{  
  
    import flash.display.BitmapData;  
    import flash.display.SimpleButton;  
    import flash.display.Sprite;  
    import flash.events.MouseEvent;  
    import flash.utils.ByteArray;  
    import org.bytearray.abstrait.ApplicationDefaut;  
    import org.bytearray.encodage.images.EncodeurPNG;  
  
    public class Document extends ApplicationDefaut
```

```
{

    private var dessin:Sprite;
    private var stylo:Stylo;
    private var renduBitmap:BitmapData;
    public var boutonEnvoi:SimpleButton;

    public function Document ()
    {

        // création du conteneur de tracés vectoriels
        dessin = new Sprite();

        // ajout du conteneur à la liste d'affichage
        addChild ( dessin );

        // création du symbole
        stylo = new Stylo( .1 );

        // passage du conteneur de tracés
        stylo.affecteToile ( dessin );

        // ajout du symbole à la liste d'affichage
        //addChild ( stylo );

        // positionnement en x et y
        stylo.x = 250;
        stylo.y = 200;

        boutonEnvoi.addEventListener ( MouseEvent.CLICK, exportBitmap );

    }

    private function exportBitmap ( pEvt:MouseEvent ):void
    {

        // création d'une image bitmap vierge
        renduBitmap = new BitmapData ( stage.stageWidth,
stage.stageHeight );

        // rasterisation des tracés
        renduBitmap.draw ( dessin );

        // encodage de l'image bitmap au format PNG
        var fluxBinaire:ByteArray = EncodeurPNG.encode ( renduBitmap );

        // affiche : 1488
        trace( fluxBinaire.length );

    }

}
```

La variable `fluxBinaire` représente un objet `ByteArray` contenant l'image encodée au format PNG, nous devons à présent transmettre le flux de l'image.

Nous ajoutons une nouvelle méthode `sauveImage` à notre service distant :

```
<?php

class Echanges
{

    function Echanges ( )

    {

    }

    function exchangeTypes ( $pDonnees )

    {

        return $pDonnees;

    }

    function sauveImage ( $pFluxImage )

    {

        //le ByteArray est placé au sein de la propriété data de l'objet reçu
        $flux = $pFluxImage->data;

        // nous décompressons le flux compressé coté Flash
        $flux = gzuncompress($flux);

        $nomImage = "capture.png";

        // sauvegarde de l'image
        $fp = @fopen("../../../images/".$nomImage, 'wb');
        $ecriture = @fwrite($fp, $flux);
        @fclose($fp);

        return $ecriture !== FALSE;

    }

}

?>
```

La méthode `sauveImage` reçoit le flux binaire en paramètre sous ma forme d'une instance de la classe `ByteArray` intégrée à AMFPHP.

Le flux est accessible par la propriété `data` de l'instance de `ByteArray`, nous sauvegardons le flux à l'aide des fonctions d'écriture PHP `fopen` et `fwrite`.

Attention, veuillez à créer un répertoire nommé `images`, afin d'accueillir les futures images sauvées. Dans notre exemple, ce dernier est placé au même niveau que le répertoire `services`.

Nous modifions la méthode `exportBitmap` afin de transmettre l'image à la méthode `sauveImage` :

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

```
{

import flash.display.BitmapData;
import flash.display.SimpleButton;
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.events.AsyncErrorEvent;
import flash.events.NetStatusEvent;
import flash.utils.ByteArray;
import flash.net.Responder;
import flash.net.NetConnection;
import org.bytearray.abstrait.ApplicationDefault;
import org.bytearray.encodage.images.EncodeurPNG;

public class Document extends ApplicationDefault

{

    private var dessin:Sprite;
    private var stylo:Stylo;
    private var renduBitmap:BitmapData;
    public var boutonEnvoi:SimpleButton;

    private var connexion:NetConnection;
    private var retourServeur:Responder;

    // adresse de la passerelle AMFPHP
    private static const PASSERELLE:String =
"http://localhost/echanges/gateway.php";

    public function Document ()
    {

        //création de la connexion
        connexion = new NetConnection();

        // création de l'objet de gestion des retours serveur
        retourServeur = new Responder ( succes, echec );

        // écoute des différents événements
        connexion.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
        connexion.addEventListener( IOErrorEvent.IO_ERROR,
erreurConnexion );
        connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
        connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );

        // connexion à la passerelle
        connexion.connect ( Document.PASSERELLE );

        // création du conteneur de tracés vectoriels
        dessin = new Sprite();

        // ajout du conteneur à la liste d'affichage
        addChild ( dessin );
    }
}
```

```

        // création du symbole
        stylo = new Stylo( .1 );

        // passage du conteneur de tracés
        stylo.affecteToile ( dessin );

        // ajout du symbole à la liste d'affichage
        addChild ( stylo );

        // positionnement en x et y
        stylo.x = 250;
        stylo.y = 200;

        boutonEnvoi.addEventListener ( MouseEvent.CLICK, exportBitmap );
    }

    private function erreurConnexion ( pEvt:Event ):void
    {
        trace( pEvt );
    }

    private function succes ( pRetour:* ):void
    {
        if ( pRetour ) trace("image sauvegardée !");
        else trace("erreur d'enregistrement");
    }

    private function echec ( pErreur:* ):void
    {
        trace( pErreur );
    }

    private function exportBitmap ( pEvt:MouseEvent ):void
    {
        // création d'une image bitmap vierge
        renduBitmap = new BitmapData ( stage.stageWidth,
        stage.stageHeight );

        // rasterisation des tracés
        renduBitmap.draw ( dessin );

        // encodage de l'image bitmap au format PNG
        var fluxBinaire:ByteArray = EncodeurPNG.encode ( renduBitmap );

        // compression zlib du flux
        fluxBinaire.compress();

        // transmission du ByteArray par Flash Remoting
        connexion.call ( "org.bytearray.test.Echanges.sauveImage",
        retourServeur, fluxBinaire );
    }

```

```

    }
}

```

Nous dessinons quelques tracés, puis nous cliquons sur le bouton d'export comme l'illustre la figure 19-12 :



Figure 19-12. Dessin.

Voici les différentes étapes lorsque nous cliquons sur le bouton d'export :

- Les tracés vectoriels sont rasterisés sous la forme d'un objet `BitmapData`.
- Une image PNG est générée à l'aide des pixels accessibles depuis l'objet `BitmapData`.
- Le flux de l'image est envoyé au service distant qui se charge de la sauvegarder sur le serveur.

Lorsque la méthode `succes` est déclenchée, l'image est sauvegardée sur le serveur. En accédant au répertoire `images` nous découvrons l'image sauvegardée comme l'illustre la figure 19-13 :


Nom	Taille	Type
 capture.jpg	4 Ko	Image JPEG

Figure 19-13. Image sauvegardée.

L'image pourrait aussi être sauvee directement en base de données au sein d'un champ de type **BLOB**. L'objet **ByteArray** pourrait ainsi être récupéré plus tard et affiché grâce à la méthode **loadBytes** de l'objet **Loader**.

Il serait aussi envisageable de transmettre uniquement les pixels de l'image à l'aide de la méthode **getPixels** de la classe **BitmapData**. Puis de générer n'importe quel type d'image côté serveur à l'aide d'une librairie spécifique telle GD ou autres.

Se connecter à une base de données

Flash Remoting prend tout son sens lors de la manipulation d'une base de données.

Nous allons intégrer Flash Remoting au menu développé au cours du chapitre 7 intitulé *Interactivité*.

Pour cela nous devons définir une base de données, nous utiliserons dans cet exemple une base de données MySQL.

*Figure 19-14. Base de donnée MySQL.*

Nous créons une base de données intitulée **maBase** puis nous créons une table associée **menu**.

Afin de créer celle-ci nous pouvons exécuter la requête MySQL suivante :

```
--  
-- Structure de la table `menu`  
--  
  
CREATE TABLE `menu` (  
  `id` int(11) NOT NULL auto_increment,  
  `intitule` varchar(30) NOT NULL default '',  
  `couleur` int(11) NOT NULL default '0',
```

```

PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=6 ;

--
-- Contenu de la table `menu`
--

INSERT INTO `menu` (`id`, `intitule`, `couleur`) VALUES (1, 'Accueil',
16737536),
(2, 'Nouveautés', 16737536),
(3, 'Photos', 16737536),
(4, 'Liens', 16737536),
(5, 'Contact', 16737536);

```

Quelques données sont présentes dans la table **menu** :

← T →			id	intitule	couleur
<input type="checkbox"/>			1	Accueil	16737536
<input type="checkbox"/>			2	Nouveautés	16737536
<input type="checkbox"/>			3	Photos	16737536
<input type="checkbox"/>			4	Liens	16737536
<input type="checkbox"/>			5	Contact	16737536

Figure 19-15. Données de la table menu.

Avant de récupérer les données de la table, notre service distant doit au préalable se connecter à la base MySQL.

Nous ajoutons la connexion à la base au sein du constructeur du service distant :

```

<?php
class Echanges
{
    function Echanges ( )
    {
        // connexion au serveur MySQL
        mysql_connect ("localhost", "thibault", "20061982");
        // sélection de la base
        mysql_select_db ("maBase");
    }
}
?>

```

Puis nous ajoutons une méthode **recupereMenu** :

```

<?php
class Echanges

```

```
{  
  
    function Echanges ( )  
  
    {  
  
        // connexion au serveur MySQL  
        mysql_connect ( "localhost", "bobgroove", "20061982");  
        // sélection de la base  
        mysql_select_db ( "maBase");  
  
    }  
  
    function recupereMenu ( )  
  
    {  
  
        return mysql_query ( "SELECT * FROM menu");  
  
    }  
  
}  
  
?>
```

En nous rendant auprès de l'explorateur de services nous pouvons tester directement la méthode `recupereMenu`.

A l'aide de l'onglet *RecordSet view* voyons directement au sein d'une liste le résultat de notre requête.

La figure 19-16 illustre le résultat :

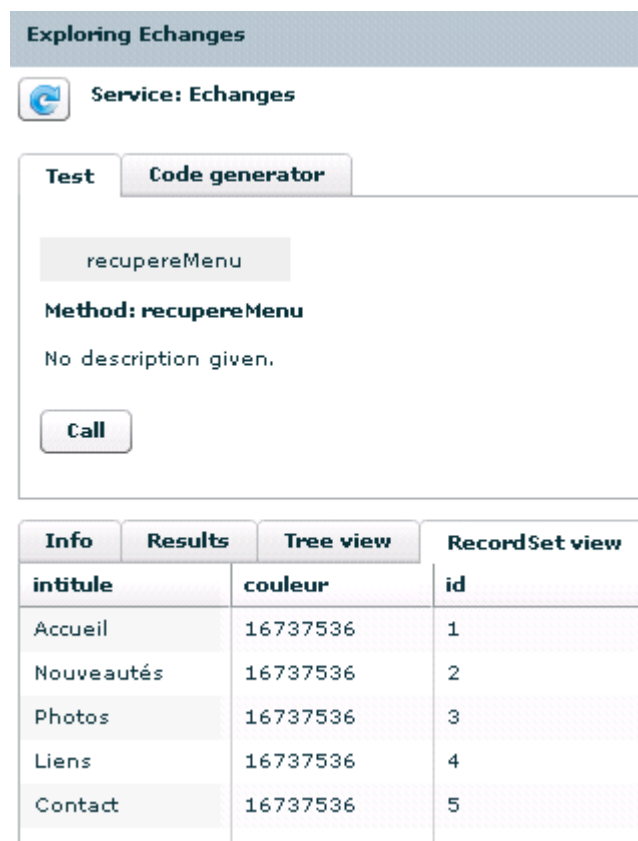


Figure 19-16. Aperçu en direct du résultat de la requête.

Chaque champ de colonne correspond au champ de la table `menu`, grâce à ce mécanisme les données sont présentées en une seule ligne de code PHP.

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

```
package org.bytearray.document
{
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.SecurityErrorEvent;
    import flash.events.AsyncErrorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.Responder;
    import flash.net.NetConnection;
    import org.bytearray.abstrait.ApplicationDefault;

    public class Document extends ApplicationDefault
    {
        private var connexion:NetConnection;
```

```

        private var retourServeur:Responder;

        // adresse de la passerelle AMFPHP
        private static const PASSERELLE:String =
"http://localhost/echanges/gateway.php";

        public function Document ()
        {

            //création de la connexion
            connexion = new NetConnection();

            // création de l'objet de gestion des retours serveur
            retourServeur = new Responder ( succes, echec );

            // écoute des différents événements
            connexion.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
            connexion.addEventListener( IOErrorEvent.IO_ERROR,
erreurConnexion );
            connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
            connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );

            // connexion à la passerelle
            connexion.connect ( Document.PASSERELLE );

            // appel de la méthode distante recupereMenu
            connexion.call ( "org.bytearray.test.Echanges.recupereMenu",
retourServeur );

        }

        private function erreurConnexion ( pEvt:Event ):void
        {

            trace( pEvt );

        }

        private function succes ( pRetour:* ):void
        {

            // affiche : [object Object]
            trace( pRetour );

        }

        private function echec ( pErreur:* ):void
        {

            trace( pErreur.description );

        }

    }

```

```
| }
```

Dès l'initialisation de l'application nous appelons la méthode distante `recupereMenu`. AMFPHP retourne les données issues de la requête `MySQL` sous la forme d'un objet couramment appelé jeu d'enregistrements (*`RecordSet`*).

Un objet `RecordSet` possède une propriété `serverInfo` contenant les propriétés suivantes :

- `totalCount` : le nombre d'enregistrements renvoyés.
- `columnNames` : un tableau contenant le nom des champs de la table.
- `initialData` : un tableau de tableaux, contenant les données de la table.
- `Id` : identifiant de la session en cours créée par AMFPHP.
- `Version` : la version de l'objet `RecordSet`.
- `cursor` : point de départ de la lecture des données.
- `serviceName` : nom du service distant.

Si nous itérons au sein de l'objet `serverInfo` nous découvrons chacune des propriétés et leur valeurs :

```
private function succes ( pRetour:* ):void
{
    /* affiche :
       serviceName : PageAbleResult
       columnNames : id,intitule,couleur
       id : 952b03b6b26e39ff52418985866801ab
       initialData :
1,Accueil,16737536,2,Nouveautés,16737536,3,Photos,16737536,4,Liens,16737536,5,C
ontact,16737536
       totalCount : 5
       version : 1
       cursor : 1
    */
    for ( var p in pRetour.serverInfo )
    {
        trace( p, " : " + pRetour.serverInfo[p] );
    }
}
```

En utilisant AMFPHP à l'aide du framework Flex ou AIR, les ressources `MySQL` sont converties sous la forme d'objet *`ArrayCollection`* permettant un accès facilité aux données.

Au sein de Flash CS3, aucune classe n'est prévue pour gérer l'interprétation des `RecordSet`, nous allons donc devoir développer une petite fonction de reorganisation des données.

Nous ajoutons une méthode `conversion` :

```
private function conversion ( pSource:Object ):Array
{
    var donnees:Array = new Array();
    var element:Object;

    for ( var p:String in pSource.initialData )
    {
        element = new Object();

        for ( var q:String in pSource.columnNames )
        {
            element[pSource.columnNames[q]] = pSource.initialData[p][q];
        }

        donnees.push ( element );
    }

    return donnees;
}
```

Lorsque les données sont chargées nous construisons un tableau d'objets à l'aide de la méthode `conversion` :

```
private function succes ( pRetour:* ):void
{
    // le RecordSet est converti en tableau d'objets
    var donnees:Array = conversion ( pRetour.serverInfo );
}
```

L'idéal étant d'isoler cette méthode afin de pouvoir la réutiliser à tout moment, dans n'importe quel projet. Nous pourrions imaginer une méthode `conversion` au sein d'une classe `OutilsRemoting`.

Nous allons à présent intégrer le menu développé au cours du chapitre 7. Assurez vous d'avoir bien importé le bouton que nous avons créé associé à la classe `Bouton` :

```
package org.bytearray.document
{

```

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.events.AsyncErrorEvent;
import flash.events.NetStatusEvent;
import flash.events.MouseEvent;
import flash.net.Responder;
import flash.net.NetConnection;
import fl.transitions.Tween;
import fl.transitions.easing.Elastic;
import org.bytearray.abstrait.ApplicationDefault;

public class Document extends ApplicationDefault
{
    private var connexion:NetConnection;
    private var retourServeur:Responder;
    private var conteneurMenu:Sprite;

    // adresse de la passerelle AMFPHP
    private static const PASSERELLE:String =
"http://localhost/echanges/gateway.php";

    public function Document ()
    {
        conteneurMenu = new Sprite();

        conteneurMenu.x = 140;
        conteneurMenu.y = 120;

        addChild ( conteneurMenu );

        conteneurMenu.addEventListener ( MouseEvent.ROLL_OVER,
survolBouton, true );
        conteneurMenu.addEventListener ( MouseEvent.ROLL_OUT,
quitteBouton, true );

        //création de la connexion
        connexion = new NetConnection();

        // création de l'objet de gestion des retours serveur
        retourServeur = new Responder ( succes, echec );

        // écoute des différents événements
        connexion.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
        connexion.addEventListener( IOErrorEvent.IO_ERROR,
erreurConnexion );
        connexion.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
        connexion.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );

        // connexion à la passerelle
        connexion.connect ( Document.PASSERELLE );

        // appel de la méthode distante recupereMenu
```



```

        connexion.call ( "org.bytearray.test.Echanges.recupereMenu",
retourServeur );

    }

    private function erreurConnexion ( pEvt:Event ):void

    {

        trace( pEvt );

    }

    private function succes ( pRetour:* ):void

    {

        // le RecordSet est converti en tableau d'objets
        var donnees:Array = filtre ( pRetour.serverInfo );

        var lng:int = donnees.length;
        var monBouton:Bouton;

        var angle:int = 360 / lng;

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

        {

            // instantiation du symbole Bouton
            monBouton = new Bouton();

            // activation du comportement bouton
            monBouton.buttonMode = true;

            // désactivation des objets enfants
            monBouton.mouseChildren = false;

            // affectation du contenu
            monBouton.maLegende.text = donnees[i].intitule;

            // disposition des occurrences
            monBouton.tween = new Tween ( monBouton, "rotation",
Elastic.easeOut, 0, angle * (i+1), 2, true );

            // on crée un objet Tween pour les effets de survol
            monBouton.tweenSurvol = new Tween ( monBouton.fondBouton,
"scaleX", Elastic.easeOut, 1, 1, 2, true );

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

        }

    }

    private function echec ( pErreur:* ):void

    {

        trace( pErreur.description );

    }

```

```
    }

    function survolBouton ( pEvt:MouseEvent ):void
    {
        var monTween:Tween = pEvt.target.tweenSurvol;
        monTween.continueTo (1.1, 2);
    }

    function quitteBouton ( pEvt:MouseEvent ):void
    {
        var monTween:Tween = pEvt.target.tweenSurvol;
        monTween.continueTo (1, 2);
    }

    private function filtre ( pSource:Object ):Array
    {
        var donnees:Array = new Array();
        var element:Object;

        for ( var p:String in pSource.initialData )
        {
            element = new Object();

            for ( var q:String in pSource.columnNames )
            {
                element[pSource.columnNames[q]] =
pSource.initialData[p][q];
            }

            donnees.push ( element );
        }

        return donnees;
    }
}
```

En testant le code précédent, nous obtenons un menu dynamique connecté à notre base de données.

La figure 19-17 illustre le résultat :

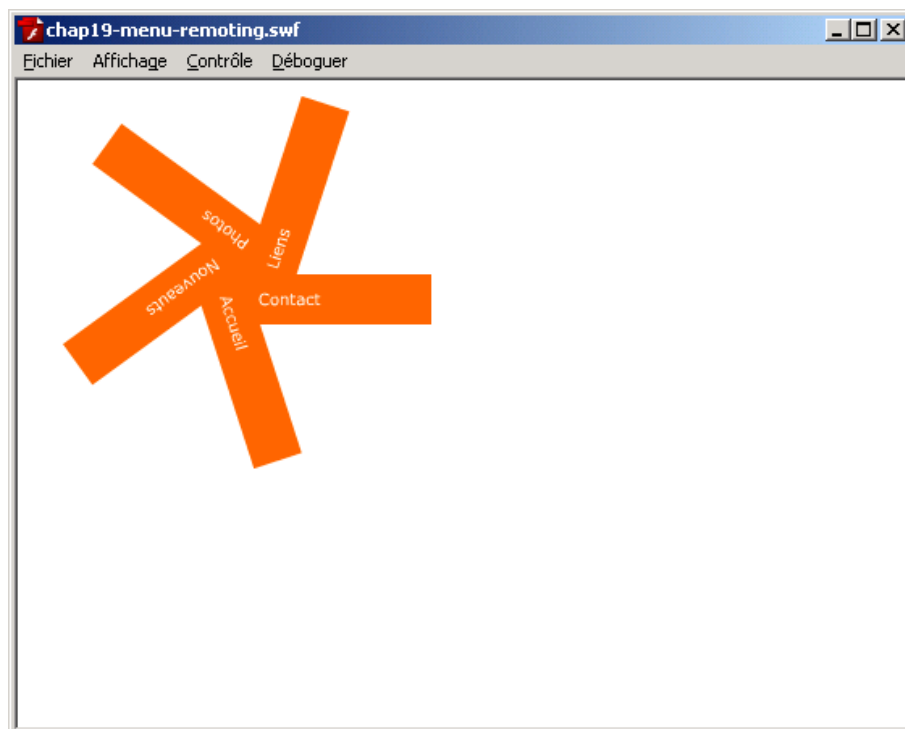


Figure 19-17. Menu dynamique connecté.

Nous avons un champ de la base inutilisé pour le moment au sein de notre menu. Nous allons remédier à cela en liant la couleur de chaque bouton au champ couleur associé :

```
private function succes ( pRetour:* ):void
{
    // le RecordSet est converti en tableau d'objets
    var donnees:Array = filtre ( pRetour.serverInfo );

    var lng:int = donnees.length;
    var monBouton:Bouton;
    var transformationCouleur:ColorTransform;

    var angle:int = 360 / lng;

    for ( var i:int = 0; i < lng; i++ )
    {
        // instantiation du symbole Bouton
        monBouton = new Bouton();

        // activation du comportement bouton
        monBouton.buttonMode = true;

        // désactivation des objets enfants
        monBouton.mouseChildren = false;

        // affectation du contenu
        monBouton.maLegende.text = donnees[i].intitule;
    }
}
```

```

// récupération de l'objet de transformation de couleur
transformationCouleur = monBouton.fondBouton.transform.colorTransform;

// affectation d'une couleur dynamique
transformationCouleur.color = donnees[i].couleur;

// mise à jour de la couleur
monBouton.fondBouton.transform.colorTransform = transformationCouleur;

// disposition des occurrences
monBouton.tween = new Tween ( monBouton, "rotation", Elastic.easeOut,
0, angle * (i+1), 2, true );

// on crée un objet Tween pour les effets de survol
monBouton.tweenSurvol = new Tween ( monBouton.fondBouton, "scaleX",
Elastic.easeOut, 1, 1, 2, true );

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

```

En modifiant les champs couleur de chaque rubrique nous obtenons le résultat illustré par la figure 19-18 :

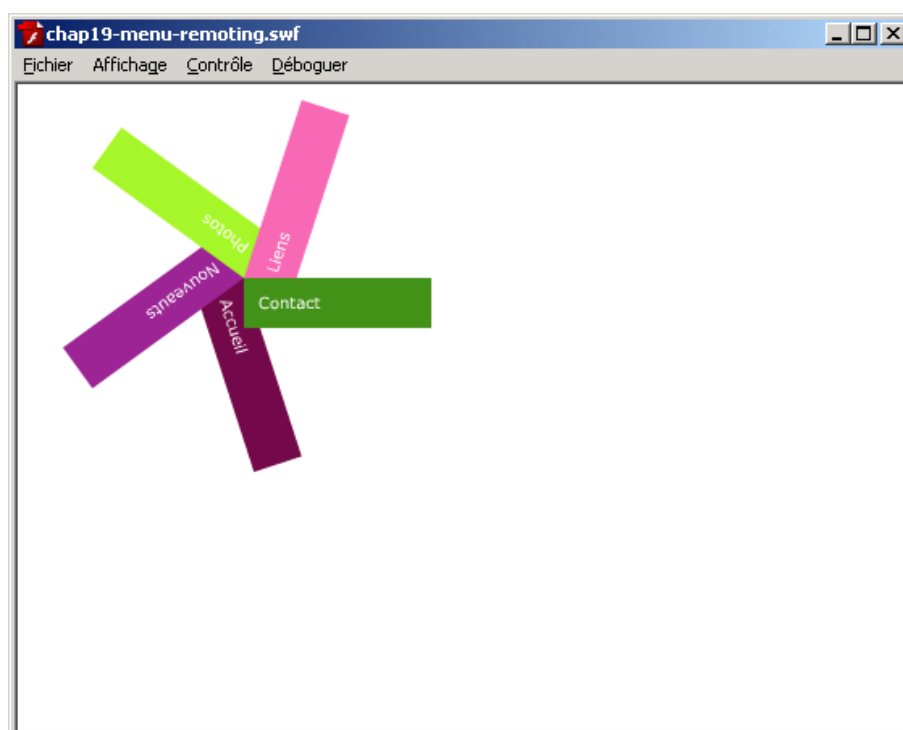


Figure 19-18. Menu connecté avec couleurs dynamiques.

Grâce à Flash Remoting, l'échange de données est réellement simplifié, nous pourrions ajouter de nombreuses fonctionnalités à notre menu.

A vous de développer le gestionnaire d'administration de ce menu à l'aide de méthodes de mise à jour de la base !

A retenir

- Flash Remoting permet de retourner directement des ressources MySQL à Flash.
- La ressource MySQL est convertie en un objet appelé RecordSet.
- Dans Flash CS3, cet objet n'est pas exploitable directement, nous devons réorganiser les données manuellement.
- Pour cela, un script peut être développé et réutilisé pour d'autres projets Flash Remoting.

Sécurité

Une fois le développement de notre service distant terminé et notre application déployée, nous devons impérativement supprimer l'explorateur de services. Cela évite de laisser à disposition l'accès aux méthodes distantes auprès d'une personne mal intentionnée.

Pour supprimer l'explorateur de services, nous supprimons le répertoire `browser`.

Garder à l'esprit que les paquets AMF échangés peuvent être décodés très facilement. Comme nous l'avons vu précédemment, chaque paquet AMF contient de nombreuses informations.

Parmi celles-ci nous retrouvons le nom de la méthode distante à exécuter, l'adresse de la passerelle ainsi que les données à transmettre. De nombreux logiciels de déboguage tels *ServiceCapture* ou *Charles* permettent de lire ces informations.

Ces derniers sont disponibles à l'adresse suivante :

- Service Capture : <http://kevinlangdon.com/serviceCapture/>
- Charles : <http://www.xk72.com/charles/>

Une personne mal intentionnée pourrait être tentée de récupérer l'adresse de notre passerelle AMFPHP et de s'y connecter en appelant les méthodes distantes affichées par un logiciel de capture de paquets AMF tel *Service Capture*.

Afin de supprimer les éventuels messages émis par la méthode `trace` de la classe `NetDebug`, nous pouvons activer le mode « production » d'AMFPHP.

Pour activer cette fonctionnalité il convient de passer la constante `PRODUCTION_SERVER` à `true` au sein du fichier `gateway.php` :

```
|define("PRODUCTION_SERVER", true);
```

Si l'application Flash tentant d'appeler les méthodes de notre service est hébergée sur un autre serveur, l'appel de la méthode `call` de l'objet `NetConnection` entraîne une erreur de sécurité. Attention, si nous avons placé sur notre serveur un fichier de régulation autorisant tous les domaines, l'appel réussira et rendra notre service vulnérable.

Le risque peut donc venir d'une personne se connectant à notre service distant depuis l'environnement auteur de Flash, car aucune restriction de sécurité n'est appliquée dans ce cas.

Les versions 1.9 et ultérieures d'AMFPHP intègrent une nouvelle fonctionnalité permettant d'interdire toute connexion au service distant depuis le lecteur Flash autonome. L'activation de la constante `PRODUCTION_SERVER` active automatiquement ce contrôle et améliore la sécurité de notre service distant.

Pour une sécurité optimale nous pouvons utiliser la méthode `addHeader` de la classe `NetConnection` afin d'authentifier l'utilisateur en cours au sein d'AMFPHP.

Pour plus d'informations à ce sujet, rendez-vous à l'adresse suivante :

<http://www.amfphp.org/docs/authenticate.html>

A retenir

- Une fois l'application Flash Remoting en production il faut impérativement supprimer l'explorateur de services.
- Il est fortement recommandé d'activer le mode « production » d'AMFPHP grâce à la constante `PRODUCTION_SERVER` du fichier `gateway.php`.

La classe Service

Bien que Flash Remoting s'avère extrêmement pratique, l'utilisation de la classe `NetConnection` s'avère peu souple.

La méthode distante à appeler doit être passée sous forme de chaîne de caractères à la méthode `call` ce qui s'avère peu pratique.

Nous allons développer dans la partie suivante, une classe `Service` permettant de représenter le service distant, comme si ce dernier était un objet `ActionScript`.

En d’autres termes, la classe `Service` nous permettra de représenter sous forme d’objet `ActionScript` le service distant.

Pour appeler la méthode distante `recupereMenu`, nous écrirons :

```
// création du service coté Flash
var monService:Service = new Service();

// appel de la méthode distante
monService.recupereMenu();
```

Tous les mécanismes internes liés à l’objet `NetConnection` seront totalement transparents, rendant le développement d’applications `Flash Remoting` encore plus simple.

Au sein d’un paquetage `org.bytearray.remoting` nous définissons la classe `Service` suivante :

```
package org.bytearray.remoting
{
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;
    import flash.events.Event;
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    public dynamic class Service extends Proxy implements IEventDispatcher
    {
        private var diffuseur:EventDispatcher;

        public function Service ()
        {
            diffuseur = new EventDispatcher();
        }

        override flash_proxy function hasProperty(name:*) :Boolean
        {
            return false;
        }

        override flash_proxy function getProperty(name:*) :*
        {
            return undefined;
        }

        public function toString ( ):String
```

```
        {
            return "[object Service]";
        }

        public function addEventListener( type:String, listener:Function,
useCapture:Boolean=false, priority:int=0, useWeakReference:Boolean=false ):void
        {
            diffuseur.addEventListener( type, listener, useCapture, priority,
useWeakReference );
        }

        public function dispatchEvent( event:Event ):Boolean
        {
            return diffuseur.dispatchEvent( event );
        }

        public function hasEventListener( type:String ):Boolean
        {
            return diffuseur.hasEventListener( type );
        }

        public function removeEventListener( type:String, listener:Function,
useCapture:Boolean=false ):void
        {
            diffuseur.removeEventListener( type, listener, useCapture );
        }

        public function willTrigger( type:String ):Boolean
        {
            return diffuseur.willTrigger( type );
        }
    }
}
```

Dans un premier temps nous remarquons que la classe `Service` étend la classe `flash.utils.Proxy` et permet donc d'intercepter l'appel de méthodes inexistantes.

Le code suivant illustre le concept :

```
import org.bytearray.remoting.Service;

var monService:Service = new Service();

monService.methodeInexistante();
```

En testant le code précédent, l'erreur suivante est levée à l'exécution :

```
Error: Error #2090: La classe Proxy ne met pas en oeuvre callProperty. Elle
doit être remplacée par une sous-classe.
```

Afin de pouvoir intercepter les méthodes ou appels de propriétés nous devons définir une méthode `callProperty` surchargeante :

```
override flash_proxy function callProperty ( nomMethode:*, ...parametres:* ):*
```



```
        trace ( nomMethode );  
  
        return null;  
    }  
}
```

Lorsqu'une méthode est appelée sur l'instance de `Service`, la méthode `callProperty` est exécutée et renvoie en paramètre le nom de la propriété référencée.

Rappelez-vous qu'une méthode est considérée dans le modèle objet d'ActionScript telle une fonction référencée par une propriété. La fonction s'exécutant dans le contexte de l'instance. C'est la raison pour laquelle la méthode interceptant les appels est appelée `callProperty`.

En appelant à nouveau une méthode inexistante, nous voyons que la méthode interne `callProperty` est exécutée et affiche le nom de la méthode appelée :

```
import org.bytearray.remoting.Service;  
  
var monService:Service = new Service();  
  
// affiche : methodeInexistante  
monService.methodeInexistante();
```

Nous allons profiter de ce comportement pour capturer le nom de la méthode et déléguer son appel auprès de l'objet `NetConnection`.

Ainsi, pour appeler la méthode distante `envoiMessage` nous n'écrirons plus le code peu digeste suivant :

```
connexion.call ("org.bytearray.Echanges.envoiMessage", retourServeur, infos );
```

Nous préférons l'écriture simplifiée suivante :

```
monService.envoiMessage( infos );
```

Pour mettre en place ce mécanisme, un objet `NetConnection` est enveloppé au sein de l'objet `Service` afin de lui déléguer les méthodes appelées :

```
package org.bytearray.remoting  
{  
  
    import flash.events.EventDispatcher;  
    import flash.events.IEventDispatcher;  
    import flash.events.Event;  
    import flash.events.NetStatusEvent;  
    import flash.events.IOErrorEvent;  
    import flash.events.SecurityErrorEvent;  
    import flash.events.AsyncErrorEvent;  
    import flash.net.NetConnection;  
  
}
```

```
import flash.utils.Proxy;
import flash.utils.flash_proxy;

public dynamic class Service extends Proxy implements IEventDispatcher
{
    private var diffuseur:EventDispatcher;
    private var connection:NetConnection;
    private var cheminService:String;
    private var passerelle:String;

    public function Service ( pService:String, pPasserelle:String,
pEncodage:int=0 )
    {
        diffuseur = new EventDispatcher();

        connection = new NetConnection();

        connection.objectEncoding = pEncodage;
        connection.client = this;
        cheminService = pService;
        passerelle = pPasserelle;

        connection.addEventListener( NetStatusEvent.NET_STATUS,
ecouteurCentralise );
        connection.addEventListener( IOErrorEvent.IO_ERROR,
ecouteurCentralise );
        connection.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
ecouteurCentralise );
        connection.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
ecouteurCentralise );

        connection.connect( passerelle );
    }

    private function ecouteurCentralise ( pEvt:Event ):void
    {
        diffuseur.dispatchEvent ( pEvt );
    }

    override flash_proxy function callProperty ( nomMethode:*,
...parametres:* ):*
    {
        trace ( nomMethode );

        return null;
    }

    override flash_proxy function hasProperty(name:*) :Boolean
    {
        return false;
    }
}
```

```
        override flash_proxy function getProperty(name:*):*
        {
            return undefined;
        }

        public function toString ( ):String
        {
            return "[object Service]";
        }

        public function addEventListener( type:String, listener:Function,
        useCapture:Boolean=false, priority:int=0, useWeakReference:Boolean=false ):void
        {
            diffuseur.addEventListener( type, listener, useCapture, priority,
        useWeakReference );
        }

        public function dispatchEvent( event:Event ):Boolean
        {
            return diffuseur.dispatchEvent( event );
        }

        public function hasEventListener( type:String ):Boolean
        {
            return diffuseur.hasEventListener( type );
        }

        public function removeEventListener( type:String, listener:Function,
        useCapture:Boolean=false ):void
        {
            diffuseur.removeEventListener( type, listener, useCapture );
        }

        public function willTrigger( type:String ):Boolean
        {
            return diffuseur.willTrigger( type );
        }
    }
}
```

Nousinstancions la classe `Service` en passant les paramètres nécessaires :

```
import org.bytearray.remoting.Service;

// création de la connexion
var monService:Service = new Service( "http://localhost/echanges/gateway.php",
"org.bytearray.test.Echanges", ObjectEncoding.AMF3 );
```

L'instance de la classe `Service` doit à présent appeler la méthode distante grâce à l'objet `NetConnection` interne.

La classe `AppelProcedure` se charge d'appeler la méthode distante :

```
package org.bytearray.remoting
{
    import flash.events.EventDispatcher;
    import flash.net.NetConnection;
    import flash.net.Responder;

    public final class ProcedureAppel extends EventDispatcher
    {
        private var recepateur:Responder;
        private var connexion:NetConnection;
        private var requete:String;
        private var parametresFinal:Array;
        private var parametres:Array;

        public function ProcedureAppel( pConnection:NetConnection,
pRequete:String, pParametres:Array )
        {
            recepateur = new Responder ( succes, echec );

            connexion = pConnection;
            requete = pRequete;
            parametres = pParametres;

        }

        private function succes ( pEvt:Object ):void
        {
        }

        private function echec ( pEvt:Object ):void
        {
        }

        internal function appel ():void
        {
            parametresFinal = new Array( requete, recepateur );

            connexion.call.apply ( connexion,
parametresFinal.concat(parametres) );

        }

    }
}
```

Afin que la procédure renseigne à notre service le résultat du serveur, nous définissons deux classes événementielles permettant de faire transiter les résultats.

La classe `EvenementResultat` est définie au sein du paquetage `org.bytearray.remoting.evenements` :

```
package org.bytearray.remoting.evenements
{
    import flash.events.Event;
```

```
public final class EvenementResultat extends Event
{
    public var resultat:Object;

    public static const RESULTAT:String = "resultat";

    public function EvenementResultat (pType:String, pDonnees:Object)
    {
        super (pType, false, false);

        resultat = pDonnees;
    }
}
```

Nous définissons une classe **EvenementErreur** au sein du même paquetage :

```
package org.bytearray.remoting.evenements
{
    import flash.events.Event;

    public final class EvenementErreur extends Event
    {
        public var erreur:Object;

        public static const ERREUR:String = "erreur";

        public function EvenementErreur (pType:String, pFault:Object)
        {
            super (pType, false, false);

            erreur = pFault;
        }
    }
}
```

La classe **ProcedureAppel** est modifiée afin de diffuser deux événements **EvenementResultat.RESULTAT** et **EvenementErreur.ERREUR** :

```
package org.bytearray.remoting
{
    import flash.events.EventDispatcher;
    import flash.net.NetConnection;
    import flash.net.Responder;

    import org.bytearray.remoting.evenements.EvenementResultat;
    import org.bytearray.remoting.evenements.EvenementErreur;

    public final class ProcedureAppel extends EventDispatcher
    {
```

```

        private var recepateur:Responder;
        private var connexion:NetConnection;
        private var requete:String;
        private var parametresFinal:Array;
        private var parametres:Array;

        public function ProcedureAppel( pConnection:NetConnection,
pRequete:String, pParametres:Array )
        {

            recepateur = new Responder ( succes, echec );

            connexion = pConnection;
            requete = pRequete;
            parametres = pParametres;

        }

        private function succes ( pEvt:Object ):void
        {

            dispatchEvent ( new EvenementResultat (
EvenementResultat.RESULTAT, pEvt ) );

        }

        private function echec ( pEvt:Object ):void
        {

            dispatchEvent ( new EvenementErreur ( EvenementErreur.ERREUR,
pEvt ) );

        }

        internal function appel ():void
        {

            parametresFinal = new Array( requete, recepateur );

            connexion.call.apply ( connexion,
parametresFinal.concat(parametres) );

        }

    }
}

```

La classe `Service` instancie donc un objet `ProcedureAppel` à chaque appel de méthode et appelle la méthode `appel` :

```

override flash_proxy function callProperty ( nomMethode:*, ...parametres:* ):*
{

    appelEnCours = cheminService + "." + nomMethode;

    procedureAppel = new ProcedureAppel ( connexion, appelEnCours,
parametres );

    procedureAppel.appel();

    return procedureAppel;
}

```

```
}
```

Lorsque nous appelons une méthode de l'objet `Service`, celui-ci délègue l'appel à l'objet `NetConnection` interne, la méthode est appelée grâce à l'instance de `ProcedureAppel`.

Nous retournons celle-ci afin de pouvoir écouter l'événement `EvenementResultat.RESULTAT` et `EvenementErreur.ERREUR`.

Nous pouvons à présent intégrer la classe `Service` à notre menu dynamique créée auparavant :

```
package org.bytearray.document

{

    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.SecurityErrorEvent;
    import flash.events.AsyncErrorEvent;
    import flash.events.NetStatusEvent;
    import flash.events.MouseEvent;
    import flash.geom.ColorTransform;
    import flash.net.Responder;
    import flash.net.NetConnection;
    import flash.net.ObjectEncoding;
    import fl.transitions.Tween;
    import fl.transitions.easing.Elastic;
    import org.bytearray.abstrait.ApplicationDefault;
    import org.bytearray.remoting.Service;
    import org.bytearray.remoting.ProcedureAppel;
    import org.bytearray.remoting.evenements.EvenementResultat;
    import org.bytearray.remoting.evenements.EvenementErreur;

    public class Document extends ApplicationDefault

    {

        private var service:Service;
        private var conteneurMenu:Sprite;

        // adresse de la passerelle AMFPHP
        private static const PASSERELLE:String =
            "http://www.bytearray.org/tmp/echanges/gateway.php";
        private static const SERVICE_DISTANT:String =
            "org.bytearray.test.Echanges";

        public function Document ()

        {

            conteneurMenu = new Sprite();

            conteneurMenu.x = 140;
            conteneurMenu.y = 120;

            addChild ( conteneurMenu );
```

```

        conteneurMenu.addEventListener ( MouseEvent.ROLL_OVER,
survolBouton, true );
        conteneurMenu.addEventListener ( MouseEvent.ROLL_OUT,
quitteBouton, true );

        //création de la connexion
        service = new Service( Document.PASSERELLE,
Document.SERVICE_DISTANT, ObjectEncoding.AMF3 );

        // appel de la méthode distante
        var procedureAppel:ProcedureAppel = service.recupereMenu();

        // écoute du retour du serveur
        procedureAppel.addEventListener ( EvenementResultat.RESULTAT,
succes );
        procedureAppel.addEventListener ( EvenementErreur.ERREUR, echec
);

        // écoute des différents événements
        service.addEventListener( NetStatusEvent.NET_STATUS,
erreurConnexion );
        service.addEventListener( IOErrorEvent.IO_ERROR, erreurConnexion
);
        service.addEventListener( SecurityErrorEvent.SECURITY_ERROR,
erreurConnexion );
        service.addEventListener( AsyncErrorEvent.ASYNC_ERROR,
erreurConnexion );
    }

    private function erreurConnexion ( pEvt:Event ):void
    {
        trace( pEvt );
    }

    private function succes ( pRetour:EvenementResultat ):void
    {
        // le RecordSet est converti en tableau d'objets
        var donnees:Array = filtre ( pRetour.resultat.serverInfo );

        var lng:int = donnees.length;
        var monBouton:Bouton;
        var transformationCouleur:ColorTransform;

        var angle:int = 360 / lng;

        for ( var i:int = 0; i< lng; i++ )
        {
            // instantiation du symbole Bouton
            monBouton = new Bouton();

            // activation du comportement bouton
            monBouton.buttonMode = true;

            // désactivation des objets enfants

```



```

        monBouton.mouseChildren = false;

        // affectation du contenu
        monBouton.maLegende.text = donnees[i].intitule;

        // récupération de l'objet de transformation de couleur
        transformationCouleur =
monBouton.fondBouton.transform.colorTransform;

        // affectation d'une couleur dynamique
        transformationCouleur.color = donnees[i].couleur;

        // mise à jour de la couleur
        monBouton.fondBouton.transform.colorTransform =
transformationCouleur;

        // disposition des occurrences
        monBouton.tween = new Tween ( monBouton, "rotation",
Elastic.easeOut, 0, angle * (i+1), 2, true );

        // on crée un objet Tween pour les effets de survol
        monBouton.tweenSurvol = new Tween ( monBouton.fondBouton,
"scaleX", Elastic.easeOut, 1, 1, 2, true );

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

private function echec ( pErreur:EvenementErreur ):void
{
    trace( pErreur.erreur.description );
}

function survolBouton ( pEvt:MouseEvent ):void
{
    var monTween:Tween = pEvt.target.tweenSurvol;

    monTween.continueTo (1.1, 2);
}

function quitteBouton ( pEvt:MouseEvent ):void
{
    var monTween:Tween = pEvt.target.tweenSurvol;

    monTween.continueTo (1, 2);
}

private function filtre ( pSource:Object ):Array
{

```

```
        var donnees:Array = new Array();
        var element:Object;

        for ( var p:String in pSource.initialData )
        {
            element = new Object();

            for ( var q:String in pSource.columnNames )
            {
                element[pSource.columnNames[q]] =
                pSource.initialData[p][q];
            }

            donnees.push ( element );
        }

        return donnees;
    }
}
```

Désormais les écouteurs de l'objet `ProcedureAppel` reçoivent un objet événementiel de type `EvenementResultat` contenant les données au sein de la propriété `resultat`.

De la même manière si une erreur lors de l'appel est détectée, l'objet `ProcedureAppel` diffuse un événement `EvenementErreur` contenant les informations liées à l'erreur au sein de sa propriété `erreur`.

La classe `Service` peut ainsi être réutilisée dans n'importe quel projet nécessitant une connexion Flash Remoting optimisée. Cet exemple illustre une des limitations de l'héritage et met en avant la notion de composition.

A retenir

- La classe `Proxy` permet d'intercepter l'accès à une propriété.
- Grâce à la composition, nous délégons les méthodes appelées auprès de l'instance de la classe `Service` à l'objet `NetConnection` interne.