Référenciel de fonctions de FX.php

Introduction à ses fonctions fondamentales.

Dans FX, presque toutes les fonctions peuvent s'appeler dans n'importe quel ordre.
Quelques exceptions, cependant :
1) la déclaration d'un exemple de FX doit venir avant les autres; et
2) une des fonctions de requête doit s'appeler en tout dernier. Une liste des fonctions suit avec des descriptions de chaque paramètre approprié. Veuillez noter que dans le développement original, la classe de FX était antérieurement nommée FMData()

Important: Toutes les fois que FX est utilisé, vous devez appeler FX() et une des fonctions de requête. Certaines des autres fonctions sont aussi exigées, selon le type de requête que vous voulez faire exécuter.

Dans ce document, la description de chacune des fonctions dans le groupe des 'fonctions proches' ('Next Functions') indique si elles sont nécessaires ou facultatives.
Conventions utilisées:
+ $ReturnedData est employé pour indiquer la rangée contenant les données retournées par une requête. Des exemples de la syntaxe pour construire $ReturnedData, se trouvent dans les 'dernières fonctions' ('Last Functions') au sein de ce document (explication_fonction.txt).
+ Lines , là où les descriptions de paramètre commencent et sont indiquées par un signe du genre '>> '.
+ Les signes // indiquent le début de commentaires pour les pages php.

___première fonction à appeler_____

FX ($dataServer, $dataPort=591)
Le premier paramètre, $dataServer, devrait être l'adresse IP (un nom de domaine fonctionne sans problème) de la machine où vous avez le Web Companion de FileMaker Pro.

$dataPort est une indication facultative du paramètre du port d'entrée de FileMaker
(habituellement le port réservé spécialement pour Filemaker Pro est le 591).
Ceci peut être configuré dans FileMaker pro (sous le menu 'édition' dans la plupart des versions, ou menu de 'FileMaker Pro 'dans OS X) sous:

Préférence->Application->Module externe (Plug-Ins)
Puis choisissez 'Web Companion' 'et cliquez sur 'configurer'.
Vous verrez la case où saisir le numéro de port.

Par défaut, "80" doit être inscrit, mais il est stipulé dans l'aide de FileMaker (sous 'Web Companion'), qu'il faut spécifier un numéro pour ce module externe afin de faire fonctionner la publication internet:
FileMaker, Inc. a réservé le numéro de port 591 auprès de l'"Internet Assigned Numbers Authority" (IANA) pour son usage avec le module du Web Companion de FileMaker Pro.

Si ni l'un ni l'autre de ces ports ne fonctionne pour vous, vous pourriez également essayer d'utiliser un des ports entre 49152 et 65535. Selon l'IANA, ces ports sont considérés dynamiques et/ou privés. C'est-à-dire qu'ils ne sont réservés à aucune application spécifique. Typiquement, vous appelleriez ainsi cette fonction:

```
$InstanceName = nouveau FX('127.0.0.1 '); // si vous employez le port 591 ou,
$portNumber = '49494'; // où la valeur 49494 est le port que vous voulez employer
$InstanceName = new FX('127.0.0.1', $portNumber);
```

___ Fonctions suivantes, à appeler _____

Dans ce groupe, les fonctions peuvent s'appeler dans n'importe quel ordre entre elles-mêmes.
Certaines sont facultatives. Certaines peuvent être employées plusieurs fois dans une même requête.

SetDBData ($database, $layout="", $groupSize=50)

SetDBData() est une fonction exigée à moins que l'action appelée soit FMDBNames().

En effectuant une action de FMDBNames(), la fonction de SetDBData() ne devrait pas s'appeler.
Dans tout autre cas, si vous n'indiquez pas les coordonnées de l'endroit où FX peut trouver vos données, vous obtiendrez une erreur en retour.

>> le paramètre $database est exigé pour chaque action excepté pour FMDBNames().
Il devrait contenir le nom de la base de données à consulter. Le paramètre peut être une variable, comme montré ici, ou le nom littéral de la base de données: par exemple. 'livre.fp5'.

>> $layout est techniquement un paramètre facultatif qui indique le nom de la disposition à l'accès.
Il est facultatif parce que FileMaker contient par défaut un modèle interne contenant chaque champ.
Cependant, employer la disposition par défaut aura presque toujours comme conséquence une réduction des performances.

D'autre part, ce paramètre ne devrait pas être passé quand l'action à appeler est FMLayoutNames().
Pour toutes autres actions (par exemple FMFind(), FMAdd(), etc.), je recommande que le développeur crée dans Filemaker les modèles qui contiennent uniquement les champs nécessaires aux requêtes spécifiques (c'est une pratique commune et générale, peu importe votre méthode d'accès aux données de Filemaker).
Ceci aura pour conséquence une amélioration mesurable de l'exécution du Web Companion.

>> le paramètre final, $groupSize, indique le nombre de fiches (record) à renvoyer après une requête.
Comme dans l'exemple plus haut, c'est 50, le nombre de fiche par défaut. La valeur spéciale 'All' (toutes les fiches) peut tout aussi bien être employée.
L'exemple ci-dessous indique la base de données, le modèle, et une taille d'un groupe de 10 fiches:

```
$returnCount = 10;
$InstanceName->SetDBData('MyDatabase', 'web_layout', $returnCount);
```

SetDBPassword ($DBPassword, $DBUser='FX')

SetDBPassword() est une fonction facultative à moins que votre base de données soit protégée par mot de passe . En effet, si une base de données fonctionne avec mot de passe, vous devez établir le mot de passe donnant accès à l'utilisateur pour réaliser les actions qui lui sont attribuées.
(Note de Philippe: consulter le manuel de Filemaker pour comprendre le processus des accès restreints sous Filemaker)

Si vous n'indiquez aucun mot de passe pour une base de données qui est configurée pour en exiger au moins un, vous obtiendrez un message d'erreur en retour.

>> $fmPassword est le seul paramètre pris par SetDBPassword().

Ce peut être le mot de passe littéral pour la base de données, ou une variable contenant le mot de passe. Par mesure supplémentaire de sécurité, ce peut être une bonne idée de garder le dossier contenant votre mot de passe en dehors de votre répertoire web de Filemaker, et de l'inclure par l'intermédiaire de PHP. Un mot de passe littéral serait placé comme ceci:

$InstanceName->SetDBPassword('toc-toc');.

>> $DBUser est uniquement nécessaire dans les cas ou vous utilisez les bases de données spécifiques à la sécurité internet (voir fichiers dans SECURITE_WEB). Noter que parce que **$DBUser** est uniquement requis pour un type très spécifique d'autentification, **$DBUser** est utilisé en deuxième. Les usages et problèmatique de sécurité sont identiques à ceux de **$DBPassword.**
Un mot de passe littéral serait placé comme ceci:

$InstanceName->SetDBPassword('toc-toc', 'usager_web');

AddDBParam ($name, $value, $op="")

La fonction d'AddDBParam n'est pas techniquement exigée (vous ne verrez pas une erreur reliée par requête, si vous l'omettez), mais un appel à FMFind(), FMDelete(), et FMEdit() ne donnera pas grand chose sans lui.

AddDBParam() est la méthode par laquelle des critères de requête sont indiqués. Il n'est pas rare d'employer plus d'une occurrence de cette fonction dans une requête, particulièrement en exécutant une recherche complexe ou en mettant à jour une fiche existante.

>> le paramètre $name est employé pour indiquer sur quelles données vous souhaitez faire une requête. Habituellement, ce sera un nom de champ (si le champ est d'une base de données relationnelle, vous devrez vous assurer que le lien est indiqué avec le nom du champ, par exemple (nom_du_lien::nom_du_champ.)

Les principaux usages de cette fonction seront de ce type. Cependant, il y a quelques cas spéciaux:
1) quand vous exécutez une requête de FMEdit() ou de FMDelete(), vous aurez besoin d'un code semblable à ceci:

$InstanceName->adddbparam('-recid ', $currentRecord);
$currentRecord est le numéro d'identifcation de la fiche dans FileMaker (RecordID) que vous souhaitez mettre à jour.

Le 'RecordID' fait partie de l'information retournée par FX.php pour chaque fiche dans un ensemble trouvé. Si les données retournées ont été stockées dans une rangée appelée $ReturnedData, les valeurs de clef de la sous rangée dans $ReturnedData['data'], contiennent le RecordID et le ModifyID (la prochaine exception qui sera discutée) séparés par une virgule.
Les parties de RecordID et de ModifyID de la clef courante ont pu être extraites comme ceci:

$recordPointers = explode ('.', $key); // où $key est la clef courante de $ReturnedData['data']
$currentRecord = $recordPointers[0]; // voici le numéro de fiche: RecordID
$currentModified = $recordPointers[1]; // et voici le numéro de modification: ModifyID

2) en exécutant une requête de FMEdit(), il pourrait y avoir des cas où vous voulez vous assurer que

les données soumises sont en effet plus récentes que celles qui sont dans la base de données. Ceci peut être fait en ajoutant un exemple d'AddDBParam() où $name est '-modid' (comme mentionné en même temps que le -'recid' ci-dessus.) Comme ceci:

$InstanceName->AddDBParam('-modid', $currentModified);

Si la fiche dans la base de données est plus récente que les données sur lesquelles la mise à jour courante est fondée, $ReturnedData['errorCode'] contiendra la valeur 306. Cette condition a pu être vérifiée de cette manière:

si ($ReturnedData['errorCode '] == 306)
{ echo "désolé les données soumises sont plus anciennes que les données existantes."; }

La documentation complète sur les codes d'erreur pouvant être retournés à FX par le Web Companion peut être téléchargée à: http://www.filemaker.com/downloads/pdf/xml_Appendix_C.pdf (lien en anglais)

3) les recherches par défaut sont par la logique 'and' (et).
Ceci signifie que si vous passez dans des requêtes multiples d'AddDBParam(), le résultat renvoyé sera seulement celui correspondant à tous les critères.

$InstanceName->AddDBParam('-lop', 'or');

Si vous vouliez par contre exécuter une recherche qui renvoie les fiches qui répondent à n'importe lequel des critères indiqués, vous pouvez placer une logique 'or' (ou) en utilisant '-lop' le paramètre spécial de $name ':

$InstanceName->AddDBParam('-lop', 'or');.

Ce paramètre spécial doit seulement être employé en demandant une logique de recherche 'or' (ou).
La logique 'and' (et) est toujours employée, par défaut. (cela se comprend si on considère les manières dont les logiques 'ou' et 'et' des recherches sont exécutées dans FileMaker).

4) Il pourrait y avoir des cas où vous voudriez exécuter un script de FileMaker sur un jeu de données retourné par la base. Dans ces cas-là, employez le paramètre de '-script' $name:
'-script', '-script.prefind', ou '-script.presort'.

Quand ces paramètres sont employés, le paramètre $value devrait être le nom du script de FileMaker que vous souhaitez voir exécuter. L'utilisation en est assez explicite.

Quand '-script' est employé, le script indiqué dans FileMaker sera exécuté après que la recherche courante et le tri (si c'est le cas) soient exécutés.

Employer '- script.prefind 'fait exécuter le script indiqué dans FileMaker avant que la recherche courante et le tri soient exécutés.

le paramètre '-script.presort' $name fait fonctionner le script indiqué dans FileMaker après la recherche en cours, mais avant que le tri en cours soit exécuté. L'utilisation de ces paramètres serait semblable aux exemples suivants:

$InstanceName->AddDBParam('-script', 'Adresses De Re-lookup ');
$InstanceName->AddDBParam('-script.prefind', 'Effacement des fiches expirées');

$InstanceName->AddDBParam('-script.presort', 'Omet les duplicatas');

>> $value contient la valeur qu'une question devrait localiser dans le champ indiqué près $name.
Dans le cas d'une question de FMEdit(), $value est employé pour tenir les nouvelles valeurs à assigner aux champs dans la fiche indiquée avec le rapport d'AddDBParam() de '-recid', comme décrit dans les cas spéciaux ci-dessus.

$value peut également tenir des valeurs correspondantes aux cas spéciaux comme précédemment décrits.
Des caractères d'ambiguité (joker) peuvent être employés dans les variables $value de la même manière que dans FileMaker.
Si vos caractères d'ambiguité (joker) semblent se comporter étrangement, essayez d'employer la fonction de l'urlencode() de PHP sur $value.

>> le troisième paramètre dans des rapports d'AddDBParam est $op.
Ce paramètre est facultatif et sans lui, la recherche par défaut 'begins with' ('commence par') est exécuté.
Ce paramètre ne devrait pas être employé avec les cas spéciaux décrits ci-dessus.
$op est l'opérateur que vous souhaitez employer pour comparer des données au niveau d'un champ (rubrique).

Une valeur pour $op doit être indiquée pour chaque champ que vous souhaitez rechercher en dehors de la recherche par défaut. Les valeurs possibles pour $op vont comme suit (la syntaxe est la même pour tout ces derniers; voyez les exemples à l'extrémité de cette section):

'eq' - une recherche de type 'égal'
'cn' - une recherche de type 'contient'
'bw' - une recherche de type 'commence avec'
'ew' - une recherche de type 'se termine avec'
'gt' - une recherche de type 'plus grand que' (supérieur à)
'gte' - une recherche de type 'plus grand que ou égal'
'lt' - une recherche de type 'plus petit que' (inférieur à).'lte' - une recherche de type 'plus petit que ou égal'
'neq' - une recherche de type 'pas égal à'

Pour information complémentaire, voir la page B-7 dans la documentation suivante de FileMaker:
http://www.filemaker.com/downloads/pdf/xml_Appendix_B.pdf

AddDBParam() pourrait être employé pour une requête de FMFind() comme ceci:

$InstanceName->AddDBParam('prenom', 'John');
$InstanceName->AddDBParam('nom', 'Smith', 'eq');

D'autre part, les déclarations d'AddDBParam() pour une requête de FMEdit() pourraient ressembler à quelque chose comme suit:

$InstanceName->AddDBParam('-recid', $currentRecord);
$InstanceName->AddDBParam('Nom', $Nom);
$InstanceName->AddDBParam('Adresse', $Adresse);

Veuillez noter que des opérateurs ne sont pas employés pour des requête de FMEdit().
Autrement AddDBParam() (dans le paramètre de '-recid') est employé de la même manière dans ces

exemples.
Dans tous les cas, des variables ou des valeurs littérales peuvent être employées comme paramètres.

AddSortParam ($field, $ortOrder = "")

AddSortParam() est employé pour déterminer comment FileMaker fera le tri des fiches trouvées avant de renvoyer les données à FX. Le mode de fonctionnement est plus ou moins identique à celui de FileMaker Pro. Cette fonction est toujours facultative.

>> $field indique sur quel champ vous souhaitez faire votre tri.
>> $sortOrder peut contenir une des trois valeurs: Montant, descendant, ou à déterminer.

Si AddSortParam() est appelé avec seulement un paramètre $field, le tri montant est présumé, par défaut. Le tri à déterminer se fait de la même manière que dans FileMaker (c-à-d. il emploie la liste de valeur composée pour le champ indiqué sur le modèle courant, comme indiqué par SetDBData() dans le cas de FX.) La syntaxe est comme suit:

$InstanceName->AddSortParam('FirstName', 'Descend'); FMSkipRecords ($skipSize)
exemple traduit:
$InstanceName->addsortparam('Nom', 'descendant'); FMSkipRecords ($skipSize)

Cette fonction est employée pour indiquer quelle fiche dans l'ensemble trouvé devrait être d'abord retournée par FX.

FMSkipRecords() est facultatif.

>> $skipSize est le seul paramètre pris par FMSkipRecords().
En incrémentant ou en décrémentant cette valeur, des fiches dans l'ensemble trouvé pourraient être paginées à travers les pages par groupes dont la taille a été indiquée avec le paramètre $groupSize dans SetDBData(). Par exemple:

$skipSize = $skipSize + $groupSize;
$InstanceName->fmskiprecords($skipsize);

Ceci incrémenterait le nombre de fiches à outrepasser par la valeur stockée dans $groupSize. Cela permettrait de passer ce paramètre à FX de sorte que le nombre ($skipSize) de fiches soit réellement outrepassé et qu'ainsi le prochain ($groupSize) nombre de fiches soit retourné par la requête courante.

FMPostQuery ($isPostQuery = true)

Cette fonction facultative est employée pour changer la méthode d'accès de FX à FileMaker d'un HTTP GET à un HTTP POST. D'une façon générale, les deux méthodes sont interchangeables et FX utilise GET par défaut..Cependant, parce que le nombre de caractères contenu dans un URL est limité, FMPostQuery() devrait être employé lors de travaux avec de grandes quantités de données (plus d'une centaine de caractères causeront des problèmes avec un HTTP GET.)

IMPORTANT: FMPostQuery() détermine la méthode par laquelle des données sont envoyées de votre serveur au Web Companion de FileMaker, PAS la méthode par laquelle des données sont envoyées du

browser d'un utilisateur à votre serveur (ceci est presque toujours fait dans le code HTML.)

>> $isPostQuery a une valeur égale à vrai par défaut, ainsi un appel à FMPostQuery() ressemblera habituellement à ceci:
$InstanceName - > FMPostQuery(); // configure la requête courante au Web Companion comme un POST

___ Dernières fonctions a appeler_____

Les fonctions suivantes doivent être employées seules pour chaque requête avec FX.php.
Appeler une de ces fonctions est ce qui cause réellement l'interaction avec FileMaker Pro.
Le résultat de la recherche courante (si approprié) peut alors être stocké dans une rangée, si désiré.

>> Que ces données soient retournées par la fonction, ou pas,est déterminé par le seul paramètre qui est passé à chacune de ces fonctions:
$returnDataSet.
Le réglage $returnDataSet configurer à "vrai" fera renvoyer à FX les données qu'il a reçues de FileMaker.
Dans les exemples où des données sont seulement envoyées à FileMaker, $returnDataSet peut être placé à "faux" (false). (le non renvoi des données augmentera l'exécution légèrement).

>> Pour vérifier si les données sont vraiment renvoyées, la confirmation peut se faire en déterminant une valeur **$returnData.** En inscrivant **$returnData** à la valeur 'basic' cela renverra les fiches à l'exception de celle déjà trouvées.

FMDelete ($returnDataSet = false, $returnData = 'basic')

Cette fonction supprime la fiche indiquée par un appel d'AddDBParam() en utilisant le paramètre spécial de '-recid' documenté ci-dessus. Puisqu'une fiche est supprimée, les données ne sont pas retournées par défaut.

```
$InstanceName->AddDBParam('-recid', $currentRecord);
$InstanceName->FMDelete(true);
```

FMEdit ($returnDataSet = true, $returnData = 'full')

La fonction utilisée pour mettre à jour le contenu d'une fiche donnée.
FMEdit() exige aussi un appel d'AddDBParam() qui place le paramètre spécial de '-recid'.
Des données sont retournées par défaut pour cette fonction (après tout, vous voulez vous assurer que les changements ont bien été faits, n'est-ce pas ?)

```
// Dans aucunes requêtes, FX() et SetDBData() au complets, ne sont absents...
$InstanceName->AddDBParam('-recid', $currentRecord);
$InstanceName->AddDBParam('FirstName', $firstName);
$InstanceName->AddDBParam('Nom', $Nom);
$InstanceName->AddDBParam('Titre', $titre);
$InstanceName->AddDBParam('Adresse', $adresse);
$InstanceName->AddDBParam('Bureau', 'Montreal, 'QC');
$ReturnedData = $InstanceName->FMEdit(); // conserve les fiches mises à jour dans $ReturnedData
```

FMFind ($returnDataSet = true)

Effectue une recherche dans FileMaker Pro comme indiqué par les paramètres précédemment passés. Par défaut, cette fonction renverra des données -- un groupe de fiches dont la taille a été indiquée dans SetDBData().. Dans aucune requête, FX() et SetDBData() au complet, ne sont absents...

```
$InstanceName->AddDBParam('FirstName', $firstName);
$InstanceName->AddDBParam('LastName', $lastName, 'eq');
$InstanceName->AddDBParam('Office', 'Austin, TX');
$ReturnedData = $InstanceName->FMFind(); // emmagasine les jeux de fiches trouvées dans
//$ReturnedData
```

FMFind ($returnDataSet = true, $returnData = 'full')

Equivalent à "Show All Records" (voir toutes les fiches) dans FileMaker Pro. Par défaut, cette fonction renverra des données. Seule le nombre de fiches spécifié par le paramètre $groupSize du SetDBData() sera renvoyé. Cependant, le reste des fiches peut être retourné en employant la fonction de FMSkipRecords() comme documenté ci-dessus.

```
$InstanceName = new FX('127.0.0.1');
$InstanceName->SetDBData('CDList', 'web_view', 'All');
$ReturnedData = $InstanceName->FMFindAll(); //emmagasine toutes les fiches dans $ReturnedData
```

FMFindAny ($returnDataSet = true, $returnData = 'full')

Renvoie une fiche aléatoire. Par défaut des données sont retournées.

```
$InstanceName = new FX('127.0.0.1');
$InstanceName->SetDBData('CDList', 'web_view');
$ReturnedData = $InstanceName->FMFindAny(); //enmagasine une fiche aléatoire dans $ReturnedData
```

FMNew ($returnDataSet = true, $returnData = 'full')

Crée une nouvelle fiche dans FileMaker Pro avec des champs saisis ayant les valeurs indiquées par la fonction d'AddDBParam(). Par défaut les données sont retournées.--- la fiche vient d'être créée.
//Note: requêtes incomplètes dans l'exemple, FX() et SetDBData() complets sont absents...

```
$InstanceName->AddDBParam('FirstName', $firstName);
$InstanceName->AddDBParam('LastName', $lastName);
$InstanceName->AddDBParam('Office', 'Austin, TX');
$ReturnedData = $InstanceName->FMNew(); // stocke la nouvelle fiche dans $ReturnedData
```

FMView ($returnDataSet = true, $returnData = 'full')

Cette fonction est quelque peu différente des autres dans ce groupe parce qu'elle renvoie des informations sur le modèle, et non sur un ensemble de fiches. Des données sont retournées par défaut, et je vois peu l'utilité de cette fonction if $returnDataSet is set to 'false' (si $returnDataSet est déclaré comme Faux). S'il y

a des champs qui sont composés en tant que menus (drop-down) sur le modèle courant (comme indiqué par SetDBData()), l'exemple suivant montrera un menu (drop-down) sur la page Web pour chacun des modèles indiqués dans la base de données:

```
$InstanceName = new FX('127.0.0.1');
$InstanceName->SetDBData('CDList', 'web_view');
$ReturnedData = $InstanceName->FMView();      //emmagasine l'information du modèle dans
                                                //$ReturnedData
foreach ($ReturnedData['valueLists'] as $key => $value) {
   foreach ($value as $key1 => $value1) {
      $valuelistData[$key] .= "\t<option value=\"$value1\">$value1</option>\n";
   }
   echo "<select name=\"$key\">\n";
   echo $valuelistData[$key];
   echo "</select>\n";
   echo "<br />\n";
}
```

FMDBNames ($returnDataSet = true, $returnData = 'full')

FMDBNames() est semblable à FMView() parce qu'il renvoie des informations sur l'environnement de FileMaker sur le serveur utilisé, plutôt qu'un groupe de fiches. À la différence de chaque autre fonction d'action, la fonction de SetDBData() ne devrait pas être appelée en même temps.que cette fonction. Une fois appelé, FMDBNames() renvoie les noms de toutes les bases de données ouvertes et partagées par l'intermédiaire du Web Companion sur la machine ciblée. Les données sont retournées par défaut. Comme précédemment, le réglage $returnDataSet mis sur False (faux) serait peu pratique. Une requête pour montrer les bases de données actuellement actives et disponibles par l'intermédiaire du Web Companion, pourrait ressembler à ceci:

```
$serverIPAddress = '127.0.0.1';
$InstanceName = new FX($serverIPAddress);
$ReturnedData = $InstanceName->FMDBNames();
// stocke l'information du modèle dans in $ReturnedData
        echo " Bases de données ouverte sur $serverIPAddress \n";
        foreach ($ReturnedData['data'] as $value)
                { echo $value['DATABASE_NAME'][0] . " \n"; }
```

FMLayoutNames ($returnDataSet = true, $returnData = 'full')

Comme les deux fonctions précédentes, FMLayoutNames() renvoie des informations sur l'environnement de
FileMaker sur le serveur utilisé, plutôt qu'un groupe de fiches. Comme pour les autres cas, les données sont retournées par défaut. Et comme eux encore, le réglage $returnDataSet mis sur 'False! (faux) serait peu pratique. Cependant, à la différence des autres fonctions d'action, la fonction de SetDBData() devrait être appelée, mais aucun modèle ne serait spécifié. FMLayoutNames() retourne les noms des modèles disponibles dans la base de données et spécifiés par la fonction SetDBData().

Une requête pour montrer les modèles dans une base de données ressemblerait à ceci:

```
$serverIPAddress = '127.0.0.1';
```

```
$currentDatabase = 'Ma_base_de_données.fp5';
$InstanceName = new FX($serverIPAddress);
$InstanceName->SetDBData($currentDatabase);
$ReturnedData = $InstanceName->FMLayoutNames();
        echo " Modèle dans $currentDatabase \n";
        foreach ($ReturnedData['data'] as $value)
                { echo $value['LAYOUT_NAME'][0] . " \n"; }
```

___Lecture additionnelles_____

Pour plus d'informations sur FX.php (incluant la dernière version anglaise):
http://www.iviking.org/

Pour plus d'informations sur FileMaker et le XML:
http://www.filemaker.com/downloads/pdf/xml_Chapter_7.pdf

Pour plus d'informations sur les paramètres acceptés par FileMaker et le Web Companion :
http://www.filemaker.com/downloads/pdf/xml_Appendix_B.pdf

Pour plus d'informations sur les codes d'erreurs renvoyés par FileMaker et le Web Companion:
http://www.filemaker.com/downloads/pdf/xml_Appendix_C.pdf

En conclusion, il est impossible de couvrir ici toutes les possibilités que PHP offre.
Heureusement, Internet est une riche source d'informations, entre autres, le manuel en ligne de PHP qui est excellent. On peut le trouver à: www.php.net/manual/fr/ .

___Notes additionnelles_____

Les documents relatifs à FX.php (version française) sont disponibles à:
http://le-nomade.com/francais2/fx.php
http://www.iviking.org/
http://www.file-making.com/files/