PHP4WD
Prise en main

Rappels

Revue du package

Version 2.0.3
Décembre 2005

SQLManagerX Team
Firetox@SQLmanagerX.com

1. Sommaire

2. Rappels

2.1. Rappel général

3. Comment lire ce document

Tout dépend de ce que vous devez faire. Nous avons recensé ci dessous les principaux cas de figure. Néanmoins, nous partons du principe que vous avez un minimum de bagages informatiques.

3.1. Rappels Typographiques

Cette icône annonce une information ou un rappel. La teneur de ce point est présentée en italique.

Cette icône annonce un point important. La teneur de ce point important est présentée en italique.

CETTE ICONE ANNONCE UN EXEMPLE DE CODE SOURCE. CE CODE EST PRESENTE EN PETITES MAJUSCULES.

3.2. Je ne connais pas WinDev depuis longtemps

Dans votre cas il sera parfois difficile de suivre ce document. Certains aspects (utilisation des classes, héritage, utilisation de DLL) vont vous paraître bien compliqués. Nous vous conseillons de bien lire la documentation sur ces points avant de suivre ce guide.

4. Présentation d'un accès [alter]natif

4.1. Rappel : les méthodes existantes standardisées

Il existe différentes méthodes pour accéder à une base de données tierce (entendons par tierce tout autre SGBD (Système de Gestion de Bases de Données) que HyperFile) au travers d'une application WinDev :

- **ODBC** (Open Database Connectivity) fournit une interface API (Application Program Interface - Interface de programmation) que différents éditeurs de bases de données implémentent par l'intermédiaire de pilotes ODBC spécifiques à un système SGBD particulier. Votre application utilise cette API pour appeler le gestionnaire de pilotes ODBC, qui transmet les appels au pilote approprié. Le pilote, à son tour, interagit avec le SGBD par l'intermédiaire de SQL.
- **DAO** (Data Access Objects) utilise le moteur de bases de données Microsoft Jet pour fournir un ensemble d'objets d'accès aux données. Il est optimisé autour du moteur de bases de données Microsoft Jet (Access !).
- **OLE DB** (Object Linking and Embeding DataBases) est une api COM qui permet un accès unifié à toutes sortes de sources de données. Des fournisseurs OLE DB existent pour la plupart des serveurs de bases de données, Active Directory, les feuilles de calcul Excel, les fichiers XML ou les fichiers texte (.txt ou .csv).
- **ADO** (ActiveX Data Object) est un composant ActiveX permettant d'accéder aux bases de données de façon beaucoup plus facile que les méthodes précédentes sans se soucier de tout ce qui est allocation des environnements de travail (cf. programmation avec la couche basse d'ODBC). ADO fournit des objets (les principaux sont *Connection*, *Command* et *Recordset*) qui permettent de se connecter à une base et de réaliser des requêtes SQL (Structured Query Language – langage structuré de requête) sur cette base.

Mais quel est le lien entre toutes ces méthodes d'accès :

- ODBC a été avant tout conçue par Microsoft pour répondre aux besoins des programmeurs C/C++.
- Microsoft introduit RDO pour faciliter le travail des programmeurs en Visual Basic (RDO se base sur l'ODBC).
- Microsoft introduit DAO pour accéder à Access en pleine évolution.
- Microsoft unifie tout cela et développe OLE-DB mais son exploitation est particulièrement difficile aussi bien en C++ qu'en VB.
- Microsoft met alors au point ADO et tout récemment ADO.net

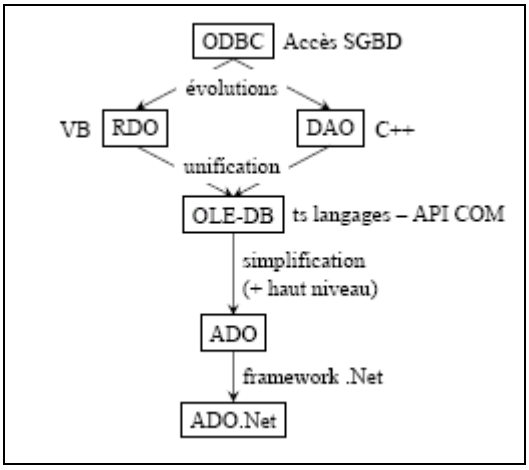

Figure 1: Historique

4.2. Méthodes pour WinDev (inspirées de la documentation)

WinDev propose différents modes d'exécution des requêtes SQL en fonction du type d'accès effectué à la base de données.

Accès à une base de données HyperFile (diffusion libre et gratuite avec vos applications WinDev)

Aucune contrainte d'installation.

Les fonctions SQL et HyperFile (HLitxxx, ...) peuvent être utilisées avec ce type d'accès.

Accès par un driver ODBC direct

Les caractéristiques de la connexion à la base de données doivent être définies dans l'administrateur ODBC de Windows. Seules les fonctions de type SQL sont utilisables pour ce type d'accès. Les fonctions HyperFile (HLitxxx, ...) ne sont pas utilisables.

Accès ODBC via le provider OLE DB

Ce type d'accès utilise un provider OLE DB spécifique. Ce type d'accès est déconseillé car plus lent qu'un accès par un driver ODBC. En effet, les performances sont moins bonnes que par un driver ODBC direct car l'accès se fait à la fois par le driver ODBC et par le provider OLE DB.

Les fonctions HyperFile (HLitxxx, ...) et SQL peuvent être utilisées avec ce type d'accès.

Il est nécessaire de définir les caractéristiques de la connexion à la base de données dans l'administrateur ODBC de Windows. Le provider ainsi que le MDAC 2.6 (ou supérieur) doivent être installés sur le poste.

Accès par un provider OLE DB

Ce type d'accès utilise un provider OLE DB. Le provider ainsi que le MDAC 2.6 (ou supérieur) doivent être installés sur le poste.

Les fonctions SQL et HyperFile (HLitxxx, ...) peuvent être utilisées avec ce type d'accès.

Accès par un accès natif : Accès natif Oracle, SQL Server, AS/400, ...

Pour chaque type d'accès natif, il est nécessaire de posséder un module complémentaire (et payant sauf pour MySQL NDLR) à WinDev. Il permet d'accéder à la base sans drivers externes depuis un programme en W-Langage.

L'accès est direct sur base sans passer par une couche intermédiaire : MDAC inutile, OLE DB inutile, ODBC inutile. Seule installation nécessaire : la couche client sur le poste de l'utilisateur.

La structure de la base peut être récupérée dans l'analyse WinDev.

Le RAD permet de générer du code avec les fonctions Hxxx (HLitSuivant, ...) ou SQLxxx.

L'outil visionneur de données (WDMAP) est utilisable sur la base de données.

Les fonctions SQL et HyperFile (HLitxxx, …) peuvent être utilisées avec ce type d'accès. Ce type d'accès est plus rapide que les accès par ODBC ou par un provider OLE DB.

4.3. L'accès [alter]natif

Un accès [alter]natif s'appuie sur les accès bas niveau fournis par l'éditeur du SGBD accédé (pas de couches intermédiaires utilisées).

Les accès bas niveau plus communément appelés API sont fournis bien souvent par un jeu de fichiers écrits en C (par exemple *OCI*, Oracle Call Interface pour Oracle, *DBLIB* pour SQL Server, *Jet* pour Access). Nous devons avoir un « lien » entre ces API et notre application WinDev. Pour se faire, une dll (Dynamic Link Library - Bibliothèque de Liens Dynamique) est l'élément le plus simple à mettre en œuvre. On se base donc sur les API pour créer une dll à l'aide d'un logiciel spécifique (entre autre : Dev-Cpp[1], Visual Studio, Borland C++) tout en répondant à nos besoins fonctionnels (se connecter à la base, effectuer des ordres SQL,…). L'utilisation d'une dll présente un double intérêt :

- Une dll est plus facilement exploitable par WinDev (ou tout autre application Windows)
- Une dll est plus facilement intégrable dans un projet WinDev

Pour terminer une classe WinDev encapsule les différents appels à la dll pour faciliter le développement.

Le tout permet :

- Une faible consommation de ressource mémoire et CPU avec une seule dll chargée.
- Une intégration dans votre projet facilitée.
- Un mode de programmation facile à maîtriser.
- Une génération de type RAD (à venir) grâce aux fichiers MDL et MDE de WinDev.

Le tout ne permet pas :

- De récupérer la structure des tables dans une analyse WinDev.
- D'utiliser WDMap car l'analyse est inexistante (voir la FAQ « J'utilise régulièrement WDMap, maintenant je fais comment ? » page 15).

[1] Gratuit, les deux autres payants

-

4.4. Présentation du package livré pour un accès

Le package livré a souvent une arborescence type :

- La racine contient généralement 2 fichiers :
  - 

: trace de toutes le versionning de l'accès
  - 

: reprend la licence WD-Libre
- **dll** contient les sources de la dll. Leur mise en forme peut varier en fonction de l'outil utilisé pour générer la DLL (Dev-cpp, VC++, Borland C++)
- **Windev7** contient un projet type.

Par défaut, le projet exemple fourni utilise la base de données créée par défaut avec votre SGBD.

4.5. Comment l'intégrer dans un projet existant

Partie technique

Pour utiliser l'accès que vous venez de récupérer, vous avez besoin de :

- Copier la DLL présente dans le répertoire Windev7/Exe du projet type dans le répertoire Exe de votre projet,
- Copier la classe présente dans le répertoire Windev7 dans le répertoire hébergeant les classes de votre projet. Certaines classes utilisent la classe générique *c_log4WD.wdc* (Celle-ci permet de tracer les ordres envoyés à la base), il est nécessaire de la copier aussi.

Vous avez donc maintenant les fichiers nécessaires au bon fonctionnement de l'accès. Il faut maintenant importer la (ou les) classe(s) dans votre projet (clic droit dans le kouglof et importer une classe).

Par soucis de vérification immédiate, une compilation totale du projet apparaît intéressante.

La DLL de l'accès devra être livrée dans le package de l'installation à destination de l'utilisateur final.

Partie développement

Pour pouvoir exploiter l'accès, il faut ouvrir une connexion à la base de données. Ensuite, vous pouvez vous baser sur les nombreux exemples fournis dans le projet type pour commencer vos développements.

Les exemples fournis ci-dessous sont liés à l'accès *Oracle4WD*. Ils seront repris et expliqués en détail dans les pages suivantes.

Déclaration :

```
// Instance de la classe accès natif
MonOracle est c_Oracle4WD
```

Connexion :

```
//Gère la connexion à la base Oracle avec comme paramètre la chaîne
//de connexion au format user/pwd@base

SI PAS MonOracle:mySQLConnecte("user/pwd@base") ALORS
   Info (MonOracle:mySQLErreur+ "/"+MonOracle:mySQLGetErrorMessage())
FIN
```

Affichage dans une table mémoire :

```
// Remplir une table mémoire avec une requête
TableSupprimeTout(TABLE1)

// Exécution avec ouverture implicite du curseur 0
retCode = MonOracle:mySQLExec("SELECT work_list, lnotes FROM work_list ",0)

SI (retCode =1) ALORS
   MonOracle:mySQLTable(0, "TABLE1")
SINON
   Info (MonOracle:mySQLErreur+ "/"+MonOracle:mySQLGetErrorMessage())
FIN

// Fermeture du curseur 0
MonOracle:mySQLFerme(0)
```

4.6. Fonctionnement de PHP4WD

Le but de cet accès est **d'accéder à un serveur MySQL chez un hébergeur public sur Internet au travers du langage PHP**. L'idée a été lancée en 2002 sur le site wdforge (rbesset à l'époque). Notre équipe a développé ce concept simple de prime abord mais demandant des connaissances PHP en sus.

Rappels du besoin

Les serveurs MySQL des hébergeurs publics ne sont évidement pas disponibles directement, par soucis de sécurité ceux-ci bloque le port standard 3306. Ces hébergeurs proposent souvent le couple PHP/Apache. La page PHP est elle capable d'accéder au serveur de donnée (HTTP → PHP → MySQL).

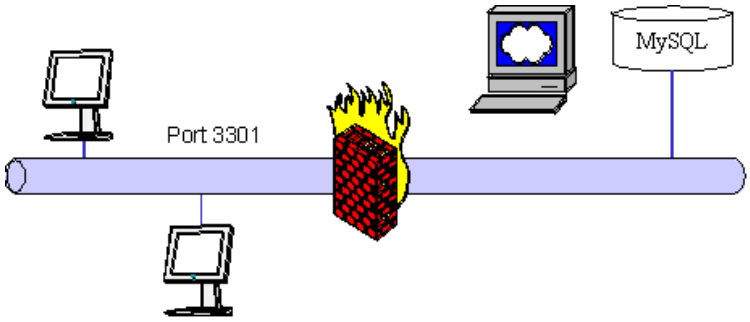

Figure 2: Constatation avec un serveur MySQL sur le port 3301

Pourquoi ne pas alors utiliser un script PHP qui reçoit en paramètre la requête SQL à exécuter sur le serveur de donnée et retourne le résultat ? Le PHP s'occupe de la connexion avec le serveur de donnée, exécute la requête et peut même proposer un retour en HTML.

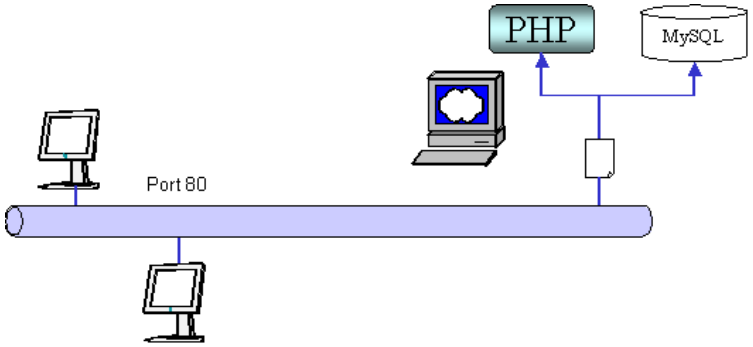

Figure 3: Principe de contournement

Architecture proposée

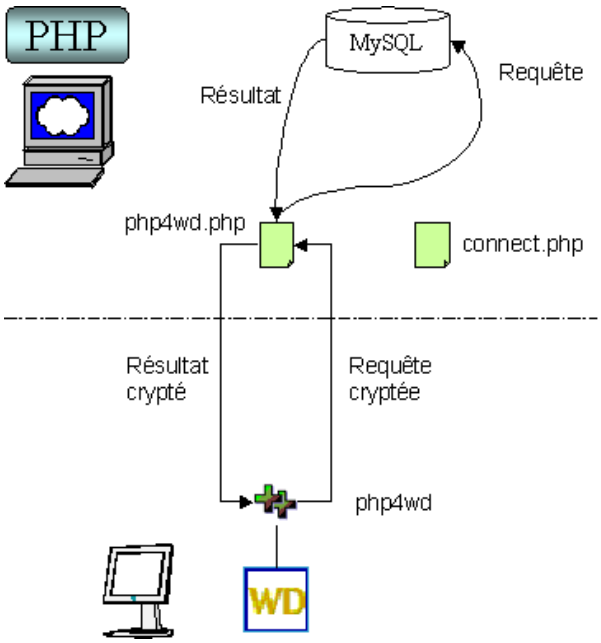

Figure 4: Principe mis en œuvre

La classe php4wd envoie la requête cryptée par une clé publique et privée au script PHP se trouvant sur le site Web (chez votre hébergeur).

Le script php4wd.php exécute la requête, récupère le résultat, crypte celui-ci et le transmet à la classe par HTTP.

La classe lit le résultat et conserve celui-ci en mémoire (ce que font les dll des accès natifs). Ensuite la classe php4Wd exploite ce résultat au même titre qu'un autre accès.

La clé publique est celle qui sert pour l'élaboration de la clé privée, qui permet le cryptage et le décryptage.

Seule la dépose des deux fichiers PHP est nécessaire dans la mise en œuvre. Il suffit de modifier les paramètres de connexion.

On peut donc venir interfacer une application WinDev sur un site WEB existant !

Attention, utilisez vos propres clés. Modifiez les clés dans les fichiers pour vous assurez un chiffrement personnel.

Un petit exemple

```
monAccess is c_Php4WD()

v_retCode = MonAccess:mySQLConnecte("http://monsite.com/php4wd.php")
IF (v_retCode = false) THEN
   Erreur("Impossible de se connecter à la base de donnée 'test'.", …
   "Erreur n° " + monAccess:mySQLErreur, monAccess:mySQLGetErrorMessage())
END
retCode = monAccess:mySQLExec("SELECT *  FROM client", 0)
IF (retCode=1) THEN
   monAccess:mySQLPremier(0)
   WHILE (NOT monAccess:mySQLEnDehors)
      Trace( monAccess:mySQLCol(0,1)+"/"+monAccess:mySQLCol(0,2))
      monAccess:mySQLSuivant(0)
   END
ELSE
   Erreur("Erreur n° " + monAccess:mySQLErreur, …
   monAccess:mySQLGetErrorMessage())
END
monAccess:mySQLFerme(0)
```

Etant donné que la classe communique avec la page PHP, et que c'est la page PHP qui communique avec la base de données, on remarque que toute base pouvant être accédée par le PHP le sera aussi par la classe.

4.7. Puis-je utiliser les accès (sous licence WD-Libre) dans un soft commercial ? Dois-je rétribuer l'auteur ?

Oui, vous pouvez intégrer les différentes ressources autour de SQLManagerX dans un produit même à finalité commerciale. En aucun cas une redevance ne vous sera exigée. La seule obligation « morale » concerne un remerciement au travers de la fenêtre « A propos de » dans votre application.

4.8. Concernant les dll, où vaut-il mieux les placer ?

Lorsque l'on installe une application, on se demande où placer les dll. Deux possibilités s'offrent à nous : dans le répertoire exe de l'application, dans le répertoire system32 du poste client :

- L'avantage dans system32 est une non multiplication des dll sur le poste, l'inconvénient est que toutes les applications nécessitant cette dll vont utiliser cette dll unique. Tout va bien si toutes les applications impactées utilisent bien la même version de la dll sinon de gros problèmes apparaissent (la majorité des cas).
- L'avantage dans le répertoire exe est que l'application fonctionnera toujours vu son indépendance vis-à-vis des autres. L'inconvénient comme vous vous en doutez à la multiplication des dll.

4.9. La grande question : quelle base pour mon projet ?

Plutôt que de reprendre une analyse déjà effectuée, je vous renvoie sur une page de developpez.net : comparatif réalisé par développez.com (http://fadace.developpez.com/sgbdcmp) sur les différents SGBD du marché.

4.10. Pourquoi utilisez-vous des tables mémoires ?

Tout simplement parce que les tables fichier sont directement liées sur un fichier ou une requête HyperFile. Nous ne pouvons donc pas utiliser cette fonctionnalité.

4.11. J'utilise régulièrement WDMap, maintenant je fais comment ?

Comme HyperFile, tout éditeur de SGBD propose une application permettant de consulter et piloter les données contenues en base.

En plus de cela, il existe des **produits commerciaux** (TOAD pour Oracle, IB-Expert pour FireBird,…), des **programmes en open-source** (TORA pour Oracle, Sqlyog pour MySQL,…) ou bien encore des outils au travers de page **web** (le plus connu phpMyAdmin pour MySQL, IBWebAdmin pour FireBird, SQLiteManager pour SQLite, phpPgAdmin pour PostgreSQL,...).

4.12. Pourquoi toutes les méthodes disponibles commencent-elles par MySQL ?

C'est tout simplement lié à un historique. MySQL4WD a été le premier accès [alter]natif créé juste avant SQLManagerX. Par un souci de compatibilité de nommage des méthodes entre les différents accès (surtout au niveau SQLManagerX), les méthodes ont toutes le même nom de méthode au niveau de la classe. Sinon la légende veut aussi que Rodolphe soit un peu possessif et qu'il ait débuté ses méthodes par MySQL pour « MonSQL ».

4.13. Accès natifs : Quels sont les numéros de requêtes permis?

SQLManagerX utilise généralement le numéro de requête 0 (et également 1 dans SQLedit). Un nombre limité d'identifiant (généralement 5) de requêtes simultanées vous est proposé.

4.14. Accès natifs : Pourquoi dois-je faire MysqlFerme(X)?

A chaque exécution d'une requête, vous lui affectez un identifiant (Cet identifiant est transparent avec SQLManagerX). Afin de pouvoir réutiliser cet identifiant, il est nécessaire de fermer la requête précédemment ouverte afin de libérer les ressources allouées en mémoire pour celle-ci. Dans les accès [alter]natifs, cela est fait de manière explicite par le développeur avec la commande MysqlFerme(X).

4.15. Y-a-t-il compatibilité ascendante entre les versions successives des accès natifs ?

Dans la mesure du possible oui. Seule exception est faite si l'API de l'éditeur change et nous oblige à ajouter dans paramètre dans les méthodes existantes. Dans tous les cas, une mention spéciale vous est communiquée lors de la mise à disposition de l'accès.