Référence développeur
Accès natifs

La classe
Les exemples

.

Version 1.0.0
Septembre 2006

SQLManagerX Team
Firetox@SQLmanagerX.com

1. SOMMAIRE

2. 1 CONVENTIONS

Ce document utilise un certain nombre de conventions typographiques afin de permettre une meilleure compréhension des membres, méthodes et paramètres à utiliser dans la classe à étudier.
Ces conventions sont les suivantes :
– **Mot en caractère gras** : nom d'un membre ou d'une méthode de la classe,
– *Mot en italique* : paramètres à passer à une méthode,
– *[Mot en italique balisé par des crochets]* : paramètres optionels à passer à la méthode.
– Mot en police fixe : exemple de code.

3. PRESENTATION D'UN ACCES [ALTER]NATIF

3.1. 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

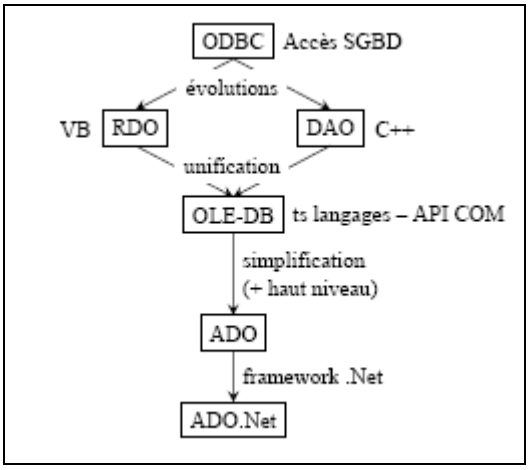


Figure 1: Historique

3.2. 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.

3.3. 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 « **Erreur ! Source du renvoi introuvable.** » page **Erreur ! Signet non défini.**).

3.4. Présentation du package livré pour un accès

Le package livré a souvent une arborescence type :

- La racine contient généralement 2 fichiers :
  -  changelog.txt : trace de toutes le versionning de l'accès
  -  LisezMoi.txt : 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.

3.5. 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,

[1] Gratuit, les deux autres payants

- 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
```

4. 3 MEMBRES

4.1. :mySQLEnDehors

Type	booleen
Utilisation	Permet de savoir si la fin de la sélection est atteinte ou non.
Notes	

4.2. :CLIENT_NORMAL

Type	constante
Utilisation	Utiliser par la méthode **mySQLConnecte()**. C'est le mode de connection par défaut. Ce membre est une constante, donc passée entre parenthèse (passe par valeur).
Notes	

4.3. : CLIENT_COMPRESS

Type	constante
Utilisation	Utiliser par la méthode **mySQLConnecte()** afin de permettre la compression des données entre le client et le serveur lors de l'utilisation d'une ligne à faible débit.Ce membre est une constante, donc passée entre parenthèse (passe par valeur).
Notes	

4.4. : mySQLDebut

Type	constante
Utilisation	Utiliser par la méthode **mySQLTransaction()** afin de débuter une transaction. Ce membre est une constante, donc passée entre parenthèse (passe par valeur).
Notes	

4.5. : mySQLFin

Type	constante
Utilisation	Utiliser par la méthode **mySQLTransaction()** afin de clore une transaction. Ce membre est une constante, donc passée entre parenthèse (passe par valeur).
Notes	

4.6. : mySQLAnnule

Type	constante
Utilisation	Utiliser par la méthode **mySQLTransaction()** afin d'annuler une transaction. Ce membre est une constante, donc passée entre parenthèse (passe par valeur).
Notes	

4.7. : mySQLDebugMode

Type	constante
Utilisation	Permet de créer un moniteur SQL traçant les requêtes envoyées au serveur. Les sorties générées par le moniteur sont envoyées au debugger installé sur le système (Visual C++, debugger système ou le programme DebugView que l'on trouve sur le serveur Unix).
Notes	Ce membre peut recevoir une valeur booléenne (Vrai ou Faux).

5. METHODES

5.1. *Constructeur()*

Utilisation 	Initialise la classe ou c_postgreSQL4WD(). En fait, cette méthode charge la librairie «DLL » en mémoire.
Syntaxe	
Détails 	
Notes	Cette méthode ne retourne aucune information.

5.2. *Destructeur()*

Utilisation 	Clos l'instance de cette classe. Elle libère le fichier «.DLL » » chargé en mémoire.
Syntaxe	
Détails 	
Notes	Cette méthode ne retourne aucune information.

5.3. <objet> :mySQLBloque

Utilisation 	Permet de bloquer les enregistrements retournés par la requête. La requête passée en paramètre n'est pas fermée.
	Exemple : Resultat = ConvSQL:*mySQLBloque(Marequete,1)*
Syntaxe	*<resultat> = <Objet> :mySQLBloque*(*requete, numRequete*)
Détails 	**<resultat> : *booléén*** **vrai** l'action s'est bien déroulée, **faux** le blocage n'a pas pu se faire **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<requete> :** texte de la requete **<numRequete> :** n°identifiant de la requete
Notes	

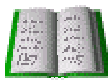
5.4. <objet> : :mySQLConnecte

Utilisation 	Permet de se connecter a une base par l'acces natif
	Exemple : Resultat = ConvSQL:MySQLConnecte("192.6.3.24","Moi","PassFred")
Syntaxe	*<resultat> = <Objet> :mySQLconnecte*(host,user,passwd,db, [port],[socket], [flag])
Détails 	**<resultat> : *booléén*** **vrai** l'action s'est bien déroulée, **faux** la connexion a echouée **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif
Notes	

5.5. <objet> : mySQLDateFormat

Utilisation 	Permet de transformer une date ua format comprehensible par la base SQL
	Exemple : Resultat = ConvSQL:*mySQLDateFormat(datesys())*
Syntaxe	*<resultat> = <Objet> :mySQLDateFormat* (*date*)
Détails 	**<resultat> : *chaine* contenant la date au format de la base** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<date> :** date a transformer
Notes	cette methode peut ne pas etre disponible dans tous les acces natifs

5.6. <objet> : mySQLDeconnecte()

Utilisation 	Déconnecte la session en cours et libère l'instance du serveur SQL. Les requêtes restants en instance sont automatiquement fermées.
	Exemple : Resultat = ConvSQL:*mySQLDeconnecte()*
Syntaxe	*<resultat> = <Objet> : mySQLDeconnecte* ()
Détails 	**<resultat> : *chaine* contenant la date au format de la base** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif
Notes	

5.7. <objet> : mySQLDernierID()

Utilisation 	cette methode permet de recuperer le dernier autoIncrement inseré dans la base SQL. Lors d'un insert si la table contient un autoIncrement alors l'appel a cette fonction permet de recuperer la valeur de cette autoIncrement inséré
	Exemple : Resultat = ConvSQL: `mySQLDernierID ()`
Syntaxe	*<resultat>* = *<Objet> :* mySQLDernierID ()
Détails 	**<resultat> : *chaine* contenant le dernier N° autoIncrement inseré dans la base de donnée** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif
Notes	Cette methode n'est pas presente dans tous les acces

5.8. <objet> : mySQLExec

Utilisation 	Envoie la requête passée en paramètre au moteur de base de donnée SQL. La requête est alors exécutéeet le résultat est retourné au client. *NumRequete* permet d'identifier la requête
	Exemple : Resultat = ConvSQL: `mySQLExec("SELECT * from matable",1)`
Syntaxe	*<resultat> = <Objet> :* mySQLExec(requete, numRequete)
Détails 	**<resultat> : *booléén*** **vrai** l'action s'est bien déroulée, **faux** lea requete n'a pas pu s'executer correctement (il y a une erreur) **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<requete> :** texte de la requete **<numRequete> :** n°identifiant de la requete
Notes	La valeur de *NumRequete* est testée dans cette fonction, elle doit être comprise entre 0 et 4 sinon, une message d'erreur est renvoyé (attention, on ne peut pas identifier plus de 5 requêtes différentes en même temps - utiliser pour cela **mySQLFerme()** et réutilisé le numéro d'identifiant libéré)..

5.9. <objet> : mySQLFerme

Utilisation 	Permet de libérer les ressources utilisées lors de l'appel à la méthode **mySQLExec()**. Il est obligatoire d'appeler cette méthode après chaque **mySQLExec()** même si la requête a retournée un code d'erreur.
	Exemple : Resultat = ConvSQL: `mySQLFerme(1)`
Syntaxe	*<resultat>* = *<Objet> :* mySQLFerme(*numRequete*)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete> :** n°identifiant de la requete
Notes	*NumRequete* permet d'identifier la requête.

5.10. <objet> : mySQLFetch

Utilisation 	Permet de se positionner sur le premier enregistrement de la requête exécutée par **mySQLExec()**. Cette fonction ne récupère qu'un enregistrement (tuple) à la fois, ce qui évite l'utilisation des ressources machine. Cependant, il n'est pas conseillé d'utiliser **mySQLFetch()** pour des traitements longs sur les enregistrements (en effet, les enregistrements liés à la requête sont lockés même s'ils ne sont pas encore en mémoire). Exemple : Resultat = ConvSQL: `mySQLfetch(1)`
Syntaxe	*<resultat> = <Objet> :* mySQLFetch (*numRequete*)
Détails 	**<resultat> : *booleen vrai tant que des lignes restent a lire faux en fin de parcours*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete> :** n°identifiant de la requete
Notes	*NumRequete* permet d'identifier la requête.

5.11. <objet> : mySQLGetErrorMessage

Utilisation 	Permet de récupérer le message d'erreur SQL. Exemple : Resultat = ConvSQL: `mySQLGetErrorMessage()`
Syntaxe	*<resultat> = <Objet> :* mySQLGetErrorMessage ()
Détails 	**<resultat> : *chaine retournée par le serveur SQL lors de l'erreur*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif
Notes	*NumRequete* permet d'identifier la requête.

5.12. <objet> : mySQLGetNumRows

Utilisation 	Permet de récuper le nombre d'enregistrements retourné par une requête.
	Exemple : Resultat = ConvSQL: `mySQLGetNumRows(1)`
Syntaxe	*<resultat> = <Objet> :* mySQLGetNumRows (numRequete)
Détails 	**<resultat> : *entier long*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	Cette fonction ne s'utilisequ'après un **mySQLPremier()** et ne peut s'appliquer à une requête de type fetch.

5.13. <objet> : mySQLLitCol

Utilisation 	Permet de récupérer la valeur du numéro de champs passé en paramètre Exemple : Resultat = ConvSQL: `mySQLLitCol(0,1)`
Syntaxe	*<resultat> = <Objet> :* mySQLLitCol (*numRequete, numChamps*)
Détails 	**<resultat> : *chaine de caractere (contenu de la colonne dans la base SQL)*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant) **<numChamps>** : n°du champs .numero d'ordre dans la requete SQL
Notes	Cette fonction ne s'utilisequ'après un **mySQLPremier()** et ne peut s'appliquer à une requête de type fetch.

5.14. <objet> : mySQLLitColParNom

Utilisation 	Permet de récupérer la valeur du numéro de champs passé en paramètre Exemple : Resultat = ConvSQL: `mySQLLitColParNom (0, « nom »)`
Syntaxe	*<resultat> = <Objet> :* mySQLLitColParNom (*numRequete, nomChamps*)
Détails 	**<resultat> : *chaine de caractere (contenu de la colonne dans la base SQL)*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant) **<nomChamps>** : nom du champs .nom dans la requete dans la requete SQL
Notes	Le nom est celui renvoyé par la requete si vous mettez des alias c'est le nom de l'alias qui doit être mis.

5.15. <objet> : mySQLLitLigne

Utilisation 	Permet de récupérer le contenu d'une ligne complete de la requete. Les colonnes seront separée par des TAB pour facilite le tableAjoute
	Exemple : Resultat = ConvSQL: `mySQLLitLigne (0)`
Syntaxe	*<resultat> = <Objet> :* mySQLLitLigne (numRequete)
Détails 	**<resultat> : *chaine de caractere (contenu de ligne renvoyée par la requete)*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	Cette methode a été faites pour eviter les appels successif dans la dll lors de l'alimentation d'une table par une requete. On recupere toute la ligne avec 1 seul appel dll.

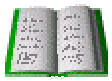
5.16. <objet> : mySQLDecritTable

Utilisation 	Permet de récupérer la structure d'une table. La chaine est formatée d'une certaine façon pour identifier tous les elements important de chaque colonne
	Exemple : Resultat = ConvSQL: `mySQLDecritTable (« maTable »)`
Syntaxe	*<resultat> = <Objet> :* mySQLDecritTable (chaine)
Détails 	**<resultat> : *chaine de caractere (desciption des colonnes d'une table)*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<nomTable>** : chaine de caractere contenant le nom de la table
Notes	La methode renvoi une chaine formatée ainsi : - nom de la colonne /type de la colonne (ex char(5)) /valeur par defaut / la colonne est cle / la colonne est autoIncrement

5.17. <objet> : mySQLLitMemo

Utilisation 	Permet de récupérer le contenu d'un champs BLOB (binaire ou texte) dans le champs passé en paramètre (en général un champs de type image, le nom d'un bouton, d'un onglet, ...). Exemple : Resultat = ConvSQL: `mySQLLitMemo (0, 2,« nomChampsWindev »)`
Syntaxe	*<resultat> = <Objet> :* mySQLLitMemo (numRequete, numChamps,nomChamps)
Détails 	**<resultat> : *chaine de caractere (contenu de la colonne dans la base SQL)*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant) **<numChamps>** : N°du champs : n°ordre de la colonne dans la req uete **<nomChamps>** : nom du champs dans la fenetre windev qui va recevoir le memo
Notes	Le nom est celui renvoyé par la requete si vous mettez des alias c'est le nom de l'alias qui doit être mis.

5.18. <objet> : mySQLPremier

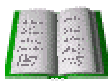
Utilisation 	Permet de se positionner sur le premier enregistrement de la requête exécutée par **mySQLExec()**.S'il n'y a pas d'enregistrement, le membre **mySQLEnDehors** est positionné à vrai. *NumRequete* permet d'identifier la requête. Exemple : Resultat = ConvSQL: `mySQLPremier (1)`
Syntaxe	*<resultat> = <Objet> :* mySQLPremier (numRequete)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	

:

5.19. <objet> : mySQLQuoteString

Utilisation 	Renvoie la chaine de caractère passée en paramètre entre simple quote. Cette fonction doit être utilisée lors de l'utilisation de la clause WHERE.
	Exemple : Resultat = ConvSQL: `mySQLQuoteString (« Machaine »)`
Syntaxe	*<resultat> = <Objet> :* mySQLQuoteString (chaine)
Détails 	**<resultat> : *chaine mise entre quote*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<chaine>** : chaine de caractere a mettre ente quote
Notes	

5.20. <objet> : mySQLSuivant

Utilisation 	Permet de se positionner sur l'enregistrement suivant.
	Exemple : Resultat = ConvSQL: `mySQLSuivant(1)`
Syntaxe	*<resultat> = <Objet> :* mySQLSuivant (numRequete)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	Si la fin de la sélection est atteinte, le membre **mySQLEnDehors** passe à vrai sinon il passe à faux

5.21. <objet> : mySQLTable

Utilisation 	Permet de renseigner le contenu de la table *nomTable* avec le contenu de la requête
	Exemple : Resultat = ConvSQL: `mySQLTable (0, nomtableWindev »)`
Syntaxe	*<resultat> = <Objet> :* mySQLTable (numRequete, nomChamps)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant) **<nomChamps>** : nom du champs table dans la fenetre windev
Notes	Il est nécessaire que la table *nomTable* contienne le même nombre de colonne que la requête ne retourne de champs (elle peut aussi en contenir plus).

:

5.22. <objet> : mySQLTransaction

Utilisation 	Permet de débuter (membre **mySQLDebut**), de clore (membre **mySQLFin**) ou d'annuler (membre**mySQLAnnule**) une transaction.
	Exemple : Resultat=ConvSQL: `mySQLTransaction (convSQL:mysqlDebut, nomtableWindev »)`
Syntaxe	*<resultat> = <Objet> :* mySQLTransaction (mode,numRequete)
Détails 	**<resultat> : *booleen a vrai si la transaction a pue etre faites*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant) **<nomChamps>** : nom du champs table dans la fenetre windev
Notes	Il est nécessaire que la table *nomTable* contienne le même nombre de colonne que la requête ne retourne de champs (elle peut aussi en contenir plus).

5.23. <objet> : mySQLDernier

Utilisation 	Permet de se positionner sur le dernier enregistrement de la requête exécutée par **mySQLExec()**.
	Exemple : Resultat = ConvSQL: `mySQLDernier (1)`
Syntaxe	*<resultat> = <Objet> :* mySQLDernier (numRequete)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	S'il n'y a pas d'enregistrement, le membre **mySQLEnDehors** est positionné à vrai

5.24. <objet> : mySQLPrecedent

Utilisation 	Permet de se positionner sur l'enregistrement précédent.
	Exemple : Resultat = ConvSQL: `mySQLPrecedent (1)`
Syntaxe	*<resultat> = <Objet> :* mySQLPrecedent (numRequete)
Détails 	**<resultat> : *aucun*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<numRequete>** : n°de la requete (dentifiant)
Notes	Si le début de la sélection est atteinte, le membre **mySQLEnDehors** passe à vrai sinon il passe à faux

5.25. <objet> : mySQLEscapeString

Utilisation 	Formate la chaine de caractères passée en paramètre afin d'être compatible avec le format des chaines de caractères du moteur SQL (en fait, elle double les simples quotes incluses dans la chaine, et insère des caractères d'échappement en cas de besoin. Elle retourne la chaine ainsi constituée encadrée par des simples quotes). Cette fonction est obligatoire lors d'appel aux commande INSERT et UPDATE.
	Exemple : Resultat = ConvSQL: `mySQLEscapeString (V_Machaine)`
Syntaxe	*<resultat> = <Objet> :* mySQLEscapeString (chaine)
Détails 	**<resultat> : *chaine resultat de la transformation*** **<Objet> : objet acces natif** nom de l'objet instance de l'acces natif **<chaine>** : chaine de caractere contenant la chaine a transformer
Notes	

6. EXEMPLES

6.1. Exemple de requête simple

```
Local
retCode is boolean
mySQL is
mySQL :mySQLConnecte(« localhost », « root », « mdp », « essai »)
retCode = mySQL :mySQLExec(« SELECT id, nom FROM client », 0)
if (retCode) then
mySQL :mySQLTable(0, « TABLE1 »)
end
mySQL :mySQLFerme(0)
mySQL :mySQLDeconnecte()
```

6.2. Exemple de requête fetch

```
Local
retCode is boolean
mySQL is
client_id, client_nom are strings
mySQL :mySQLConnecte(« localhost », « root », « mdp », « essai »)
retCode = mySQL :mySQLExec(« SELECT id, nom FROM client », 0)
if (retCode) then
while (mySQL :mySQLFetch(0))
client_id = mySQL :mySQLLitCol(0, 1)
client_nom = mySQL :mySQLLitCol(0, 2)
tableajoute(« TABLE1 », client_id + TAB + client_nom)
end
end
mySQL :mySQLFerme(0)
mySQL :mySQLDeconnecte()
```