



Référence développeur

Différence de programmation

La classe

Les exemples



Version 1.0.0
Septembre 2005

SQLManagerX Team
Firetox@SQLmanagerX.com

1. SOMMAIRE

1. SOMMAIRE	2
2. INTRODUCTION	3
3. CONVENTIONS	4
4. SCHEMA DE REPRESENTATION	5
5. PRESENTATION DE SQLMANAGERX	6
5.1. L'ENCAPSULATION DU CODE SQL.....	6
5.2. MULTI-BASES DE DONNEES.....	7
5.3. SQLMANAGERX EST OPEN-SOURCE	7
6. MEMBRES	8
7. METHODES	16

2. INTRODUCTION

La Classe **SQLManagerX** permet de gérer les tables SQL d'une façon simple et rapide en ne manipulant que des objets simples (**une ligne d'une table**) de façon ne pas s'occuper du code SQL qui en découle pour faire les insertions, les modifications, les suppressions et les sélections.

Les tables se manipulent presque comme des fichiers HF, on gère les rubriques de la même façon. Ensuite avec un seul ordre on peut faire un insert dans la base correspondante par exemple.

Cette classe est couplée avec les classes d'accès natif mis a votre disposition sur le net

celles de Rodolphe Jouannet : MySQL4WD, PostGreSL4WD ,

celles d'Emmanuel Lecoester : ORacle4WD, ADO4WD, OTL4WD , FB4WD

celles de Frédéric EMPRIN : SQLite4WD, PHP4WD, MSSQL4WD

Vous pouvez ainsi gérer les tables de ces différentes bases avec un code unique.

SQLManagerX est également compatible Wdscript, il suffit dans le projet de mettre le nom du fichier WDL correspondant aux classe de SQLManagerX de ces Accès et des différentes tables. Ensuite le Script connaît les objets et peut afficher dans une page Web le résultat de votre requête (Voir WDScript en détail pour cela : WWW.WDScript.com)

3. 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 :



Information



Attention particulière



Exemple de code source

Mot en caractère gras : nom d'un membre ou d'une méthode de la classe,

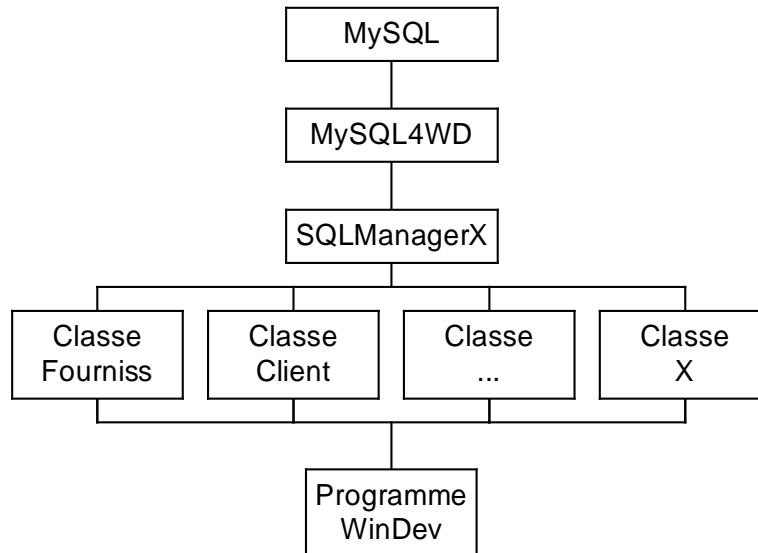
Mot en italique : paramètres à passer à une méthode,

[Mot en italique balisé par des crochets] : paramètres optionnels à passer à la méthode.

Mot en police fixe : exemple de code.

4. SCHEMA DE REPRESENTATION

Le schéma montre comment est fait l'accès du programme la base MySQL. Quelque soit la base, le principe reste valable : la classe MySQL4WD est alors remplacée par la classe d'accès natif correspondante.



On remarquera que le programme WinDev ne manipule que les classes représentant une « image » des tables représentées par des classes.

5. PRESENTATION DE SQLMANAGERX

La Classe SQLManagerX permet de gérer les tables SQL d'une façon simple et rapide en ne manipulant que des objets représentant une ligne¹ d'une table. Cette technique permet de ne pas s'occuper du code SQL qui en découle pour faire les insertions, les modifications, les suppressions. La gestion des tables SQL au travers de la classe s'apparente à celle des fichiers HyperFile. C'est le premier avantage de SQLManagerX : **en capsuler le code SQL lié à la gestion d'une table SQL** .

Deuxième avantage de SQLManagerX : Cette classe est couplée avec les classes d'**accès [alter]natifs** présentés précédemment. Cela vous permet de gérer les tables en provenance de différentes bases (MySQL, Oracle, PostgreSQL, SQLite,...) avec un code unique.

SQLManagerX est également compatible avec le projet [Wdscript](#)², il suffit dans le projet de mettre le nom du fichier WDL correspondant aux classes de SQLManagerX. Ensuite le script connaît les objets et peut afficher dans une page Web le résultat de votre requête.

5.1. L'encapsulation du code SQL

SQLManagerX permet de gérer simplement les tables d'une base SQL : le but initial étant de se rapprocher de la gestion des fichiers HyperFile. Pour cela un système de classe est utilisé. Chacune des classes correspond à une table au niveau de la base de données. Chaque colonne correspondant à un membre de la classe. Les ordres SQL de parcours, mise à jour, filtre et recherche sont alors repris depuis la classe SQLManagerX et utilisées par héritage.



Le développeur saisit ce code :

```
SQLTableVersEcran()  
SQLUpdate()
```

et SQLManagerX envoie comme ordre à la base :

```
UPDATE VILLES set Zone = 1 WHERE CodeP='59000' AND Commune='LILLE'
```

Ce mode de programmation permet de faire oublier au développeur le code SQL généré et envoyé à la base de données. Il peut alors s'orienter sur les parties les plus spécifiques de son développement. Nous sommes à l'air des L4G voire des L5G, il est maintenant important que les programmeurs ne perdent plus de temps sur les actions simples et les codes redondants. Leur temps doit être au maximum utilisé pour les développements qui demandent plus de réflexion, pour respecter les délais de plus en plus courts. Le code standard doit être rapidement écrit et doit être dans la mesure du possible le plus sûr possible.

¹ Un *tuple* pour les puristes.

² **WDScript** est un module de type CGI pour application Web. Celui-ci permet aux utilisateurs de WinDev de créer des sites dynamiques accédant à des bases de données HyperFile (ou autre) en utilisant le *W-Langage* intégré à WinDev.

5.2. Multi-bases de données

Autre point lié au précédent, le code SQL bien que standardisé contient quelques subtilités pour telle ou telle base (on peut citer les fonctions de gestion de dates ou de chaînes de caractères). La gestion au travers SQLManagerX permet d'oublier ces contraintes pour les requêtes SQL.

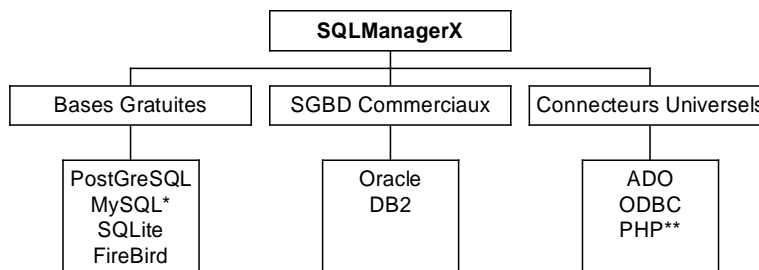


Figure 1: SGBD accédés au travers de SQLManagerX

Dans la vision des choses des auteurs, le code du programme doit être au maximum indépendant du système de base de données utilisé. Il faut arriver à avoir un code de programme indépendant de la source de données. Cela dans un but d'ouverture et de compatibilité avec les différentes bases de données du marché. SQLManagerX répond en tout point à ces préoccupations : *SQLManagerX permet de développer des applications se basant sur des bases de données sans pour autant connaître le langage SQL.*

5.3. SQLManagerX est OPEN-SOURCE

SQLManagerX et sa suite sont OPEN-SOURCE soumis à la licence WD-Libre. Sauf quelques exceptions (DLL Oracle4WD, DLL SQLite4WD, SQLManagerXConvert, SQLManagerXOutil) vous disposez du code source dans son intégralité depuis le code en W-Langage jusqu'au code C permettant de générer la DLL. Vous pouvez modifier et adapter SQLManagerX à vos besoins.

Vous pouvez faire bénéficier à la communauté des développeurs WinDev de vos remarques et/ou de vos ajouts.

6. MEMBRES

SQLManagerX :MySQL

Type	chaîne de caractères
Utilisation	<p>Ce membre est un objet qui permettra de faire tous les accès aux méthodes de la classe de l'accès a la base (ex:CMySQL4WD). lors de l'exécution d'une requête ou pour connaître la structure de la table à gérer.</p> <p>Utiliser pour exécuter la méthode mySQLConnecte() qui nécessite un objet CMySQL4WD, ou autre suivant l'accès que vous utilisez</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :NomTab

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le nom de la table SQL a gérer c'est le nom identique a celui de la base SQL. Il permettra de faire les inserts sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :NomCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le nom de la colonne de la table SQL a gérer c'est le nom identique a celui dans la base SQL.</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX. Il permettra de faire les inserts sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée.</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ValCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne la valeur de la colonne de la table SQL a gérer</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX. Il permettra de faire les inserts sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée. La valeur de la colonne est en correspondance avec l'objet de la table.</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :AncCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne la valeur ancienne de la colonne de la table SQL a gérer Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX. Il permettra de faire les update sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée. La valeur de la colonne est en correspondance avec l'objet de la table. Cette ancienne valeur est stocke afin de faire des requêtes optimisées. Seul les colonne modifiées seront envoyer en update</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :TypCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le type de la colonne de la table SQL a gérer</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX. Il permettra d'avoir pour les inserts sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée, le type de la colonne : IMPORTANT pour faire les SQLQuoteString et SQLFormatdate le cas échéant</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :KeyCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le type de la colonne de la table SQL a gérer</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX. Il permettra de savoir pour les inserts sur ce nom de table, ainsi que toutes les opérations liées à la base SQL et à la table sus nommée, si la Colonne et une clé (primaire ou non). Ce champs est très important pour les requête Update et Delete, pour accéder a l'enregistrement.</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :DefCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le type de la colonne de la table SQL a gérer</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX.</p> <p>Il permettra de connaître pour chaque colonne de la table la valeur par défaut dans SQL. Utile pour optimiser les requêtes, et n'envoyer a SQL que les colonnes ayant une valeur significative a insérer, sinon SQL est capable d'insérer les valeur par défaut.</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ExtCol

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne la partie Extra de la colonne de la table SQL a gérer</p> <p>Ce membre est un tableau dynamique qui aura pour taille celle du nombre de colonnes de la table afin de gérer au mieux la taille des tableaux et éviter de prendre de l'espace mémoire pour rien.</p> <p>Il permet avant tout de faire le lien entre l'objet de la table, et l'objet SQLManagerX.</p> <p>Il permettra de connaître pour chaque colonne de la table si par exemple la colonne est autoIncrement dans SQL. Pour ne pas envoyer de valeurs pour cette colonne la base SQL s'en chargeant</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :Scroll

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le nom de l'ascenseur de la table d'affichage de la table SQL</p> <p>Ce membre, permet la gestion de l'ascenseur lié à une table. Pour des raisons d'efficacité et de rapidité, les tables permettant l'affichage des tables SQL, ne font que des sélections minimales (10 ou 20 lignes): Nous avons donc un accès et affichage identique, sur 1 table contenant 100 lignes et 3 millions de lignes</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ObjTab

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne le nom de l'instance de l'objet.</p> <p>Ce membre permet de faire des test sur l'existence des membres de la classe de la table (ex c_fourniss) et l'existence dans l'objet d'une colonne du même nom. Afin d'avoir toujours une synchronisation exacte entre l'objet de la table et l'objet SQLManagerX correspondant.</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ValTrav

Type	chaîne de caractères
Utilisation	<p>Ce membre est une valeur de travail pour la classe</p> <p>Ce membre permet d'avoir une valeur de travail pour affectation des membre de la classe table et l'objet SQLManagerX. Comme cette synchronisation est faite en compilation dynamique, on ne peut pas utiliser de membre tableau, donc on transfert la valeur dans une variable de travail. afin d'avoir toujours une synchronisation exacte entre l'objet de la table et l'objet SQLManagerX correspondant</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ExtTab

Type	chaîne de caractères
Utilisation	<p>Ce membre renseigne l'extension donnée a l'objet table par rapport a son nom dans la base SQL</p> <p>pour des conventions de dénomination, il est possible de préfixer les nom des objets table. par exemple si dans la base SQL j'ai une table fourniss je peux nommer la classe en c_fourniss le préfixe étant alors "c_".</p>
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :ExtMbr

Type	chaîne de caractères
Utilisation	Ce membre renseigne l'extension donnée aux membres de l'objet table par rapport a son nom (colonne) dans la base SQL pour des conventions de dénomination, il est possible de préfixer les nom des membres objets table. par exemple si dans la base SQL j'ai une table fourniss avec pour colonne nofourm je peux nommer le membre en m_nofourn le préfixe étant alors "m_".
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :SourcePrep

Type	chaîne de caractères
Utilisation	Ce membre renseigne le code source de la requête SQL pour les méthodes prepare et Execute
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :Colonnes

Type	chaîne de caractères
Utilisation	Ce membre permet de connaître les colonnes qui ont été sélectionnées lors du SQLpremier afin de savoir quels membre affecté ou non dans l'objet
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul le constructeur le renseigne ce membre est en privé (inaccessible de l'extérieur)

SQLManagerX :endehors

Type	Booléen
Utilisation	Ce membre est la pour savoir si lors d'un parcours sur SQLpremier suivant , précédent et dernier si la fin de la sélection est atteinte ou non
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :Filtre	
Type	chaîne de caractères
Utilisation	Ce membre renseigne sur la valeur du filtre. C'est la chaîne du where sans le "where"
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :FiltreActif	
Type	Booléen
Utilisation	Ce membre renseigne sur l'activation ou non du filtre qui est contenu dans filtre. le filtre peut très bien être vide (="") et être actif La classe le prendra en compte comme s'il était désactivé.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :FiltreCondition	
Type	chaîne de caractères
Utilisation	Ce membre contient la condition du filtre qui a été activé ou du dernier filtre c'est la condition après le where mais sans le where le filtre peut très bien être vide (="") et être actif La classe le prendra en compte comme s'il était désactivé.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :FiltreOrder	
Type	chaîne de caractères
Utilisation	Ce membre contient le order (« ORDER BY ... ») du filtre qui a été activé ou du dernier filtre le filtre peut très bien être vide (="") et être actif La classe le prendra en compte comme s'il était désactivé.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :NbLig

Type	entier
Utilisation	Ce membre renseigne le nombre de ligne concernées par le filtre (actif ou non) dans le cas d'un filtre non actif ou vide ce membre contiendra le nombre de ligne de la table
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :ErreurSQL

Type	Chaîne de caractère
Utilisation	Ce membre renseigne le message d'erreur si la requête n'a pas pu aboutir.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :ErrorNum

Type	Entier
Utilisation	Ce membre renseigne le numéro d'erreur si la requête n'a pas pu aboutir.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)

SQLManagerX :LienActif

Type	Booléen
Utilisation	ce membre renseigne si un lien est actif et doit être pris en compte.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)



SQLManagerX :DernierID


Type	Entier
Utilisation	Dans le cas d'un identifiant automatique. Lors d'un insert il faut pour récupérer l'identifiant donné par le moteur de base de données exécuter une commande. (select last insert ID) cette commande est faites directement par SQLManagerX si la colonne est en autoIncrement. La valeur recuperer est stockées dans le membre DernierID
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seule la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)




SQLManagerX :Transaction



Type	Booléen
Utilisation	ce membre permet de savoir si les transaction sont actives ou non pour l'objet. Les transactions étant « désactivable » pour pouvoir faire des imbrication d'insert dans une seule transaction ce membre est positionné par les méthodes SQLTransactionActive et SQLtransactionDesactive.
Notes	Ce paramètre est géré en automatique et ne doit pas normalement être modifié. Seul la classe le renseigne ce membre est en public (accessible de l'extérieur) pour consultation voir initialisation)




7. METHODES




SQLManagerX ::Constructeur()	
<p>Utilisation</p> 	<p>Constructeur de la classe. Lors de l'instanciation d'un objet de la classe il faut lui passer certains paramètres</p> <p>Exemple :</p> <p>ADDR1 est un <code>c_work_list(convSql,"work_list","ADDR1")</code></p>
<p>Syntaxe</p>	<p><code>::Constructeur(<Connex>,<NomTab>,<Obj>,<ExtTab>,<Extmbr>)</code></p>
<p>Détails</p> 	<p><Connex> : <i>Objet Acces natif ex MySQL4WD</i> est le nom de l'instance de l'objet accès natif.</p> <p><NomTab> : chaîne de caractères est le nom de la table dans la base SQL (attention suivant si la base se trouve sur un système linux ou unix, la casse est importante).</p> <p><Obj> : chaîne de caractères est le nom de l'instance de l'objet SQLManagerX. Ce paramètre est important car il permet de connaître le nom de l'instance. SQLManagerX utilisant beaucoup la compilation dynamique c'est le seul moyen de pouvoir faire appel a des objets.</p> <p><ExtTab> : chaîne de caractere ce paramètre est le préfixe du nom de la table. Par exemple la table peut s'appeler FOURNISS et le programme avoir une classe représentant cette table nommer C_FOURNISS on indiquera dans ce paramètre alors « C_ »</p> <p><ExtMbr> : chaîne de caractere ce paramètre est le préfixe du nom des membres de la classe table. Par exemple la table peut s'appeler FOURNISS et une colonne s'appeler NOM le programme avoir une classe représentant cette table nommer C_FOURNISS et le membre M_NOM : on indiquera dans ce paramètre alors « M_ »</p>
<p>Notes</p>	<p>La méthode ne revoie aucun résultat</p>




SQLManagerX ::Destructeur()	
Utilisation 	Destructeur de la classe. Exemple :
Syntaxe	<code>::destructeur()</code>
Détails	Aucun
Notes	La méthode ne revoie aucun résultat




SQLManagerX ::SQLInsert ()	
<p>Utilisation</p> 	<p>Méthode permettant l'insertion dans la base d'une ligne correspondante dans la table gérée par l'objet</p> <p>Exemple :</p> <p>Resultat = ADDR1:SQLInsert()</p>
<p>Syntaxe</p>	<p><resultat> = <Objet> :SQLInsert([prepare], [numRequete])</p>
<p>Détails</p> 	<p><resultat> : booléen ou chaîne vrai si l'insert s'est bien passé, faux si l'insert n'a pas pu se faire. Soit parce que la requête contient une erreur soit parce que le serveur de la base n'a pas voulu faire l'insert. Dans le cas de prepare = vrai résultat contiendra la requête qui devait être envoyée au serveur</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><prepare> : booléen [optionnel] booléen signifiant si la requête doit être envoyée ou non. Ce paramètre est utilisé par la méthode SQLPrepare pour récupérer la ligne SQL qui devait être envoyée au serveur.</p> <p><NumRequete> : entier [optionnel] numéro de la requête à utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
	<p>Les membres :ErreurSQL et :numError sont positionnés par rapport à l'erreur renvoyés. Pour Récupérer la requête sans l'envoyée au serveur il faut mettre le paramètre « prepare » a vrai.</p> <p>Par défaut prepare est a faux et numéro de requête vaut 0</p>




SQLManagerX ::SQLRaz()	
Utilisation 	Méthode permettant la mise a blanc des membres de l'objet SQLManagerX Exemple : Resultat = ADDR1:SQLIRaz()
Syntaxe	<code><resultat> = <Objet> :SQLraz()</code>
Détails 	<resultat> : booléen vrai l'action s'est bien déroulée, faux la mise a blanc des membres n'a pas pu se faire <Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes	




SQLManagerX ::SQLDelete ()	
<p>Utilisation</p> 	<p>Méthode permettant la suppression d'une ligne de la table SQL dans la base</p> <p>Exemple :</p> <p>Resultat = ADDR1:SQLDelete()</p>
<p>Syntaxe</p>	<p><resultat> = <Objet>:SQLdelete([NumRequête])</p>
<p>Détails</p> 	<p><resultat> : booléen vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NumRequete> : entier [optionnel] numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
	<p>SQLManagerX refusera le delete si la clé unique de l'objet n'est pas renseignée. Ansi si SQLManagerX se rend compte que votre objet n'est pas complet il refusera de faire le delete pour éviter de supprimer plusieurs lignes. Seules la ligne correspondant a l'objet en cours doit être effacée.</p>




SQLManagerX ::SQLRecherche()	
<p>Utilisation</p> 	<p>Méthode permettant la recherche d'une ligne de la table SQL dans la base</p> <p>Exemple :</p> <pre>Res = ADDR1:SQLRecherche("Noprod=1000") si Res >= 1 alors Info(« J'ai trouve « + res+ » Lignes dans la base ») sinon info(« Erreur : »+ADDR1 :ErreurSQL) fin</pre>
<p>Syntaxe</p>	<p><code><resultat> = <Objet>:SQLrecherche (condition,[NumRequête])</code></p>
<p>Détails</p> 	<p><resultat> : entier</p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p>n : nombre de ligne résultat de la requête</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><condition> : chaîne de caractères condition pour la recherche sans le where. C'est normalement ce qui est mis derriere le where pour effectuer une requêteSQL</p> <p><NumRequete> : entier [optionnel] numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
	<p>SQLRecherche ne renvoie pas que vrai ou faux suivant si la requête a trouvé ou non il renvoie également le nombre de ligne dans le cas ou la recherche renverrait plusieurs lignes.</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p> <p>l'objet SQLManagerX n'est pas affecté par la methode. Les membres ne sont pas mis a jour. Cette méthode permet la recherche d'un éléments sans pour autant se positionner sur la ligne de la table SQL</p>




SQLManagerX ::SQLLitRecherche()	
<p>Utilisation</p> 	<p>Méthode permettant la recherche et lecture d'une ligne de la table SQL dans la base</p> <p>Exemple :</p> <pre> Res = ADDR1:SQLLitRecherche("Noproduct=1000") si Res = 1 alors info(«ADDR1 :NOM») sinon Si res = 0 alors info(« Erreur : »+ADDR1 :ErreurSQL) sinon Info(« J'ai trouve « + res+ » Lignes dans la base ») fin fin </pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>:SQLLitrecherche (condition,[NumRequête])</p>
<p>Détails</p> 	<p><resultat> : entier vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite n : nombre de ligne résultat de la requête</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><condition> : chaîne de caractères condition pour la recherche sans le where. C'est normalement ce qui est mis derriere le where pour effectuer une requêteSQL</p> <p><NumRequete> : entier [optionnel] numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
<p>Notes</p> 	<p>SQLLitRecherche ne renvoie pas que vrai ou faux suivant si la requête a trouvé ou non il renvoie également le nombre de ligne dans le cas ou la recherche renverrait plusieurs lignes. l'objet correspondant ne sera chargé que si la requete contient 1 seule ligne. La recherche est donc a l'identique</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés.</p> <p>L'objet SQLManagerX est affecté par la méthode. Les membres sont mis a jour (seulement si le résultat renvoie vrai ou 1 seule sligne). Attention a bien effacé l'objet avant par un SQLRaz() pour éviter les mélanges de données provenant d'une ligne précédente.</p>

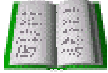


SQLManagerX ::SQLLitBloque()	
<p>Utilisation</p> 	<p>Méthode permettant la lecture bloquante d'une ligne de la table SQL dans la base</p> <p>Exemple :</p> <pre> Res = ADDR1:SQLLitBloque("Noprod=1000") si Res = 1 alors ADDR1 :m_Prenom = « Totot » ADDR1 :SQLUpdate() sinon Si res = 0 alors info(« Erreur : »+ADDR1 :ErreurSQL) sinon Info(« J'ai trouve « + res+ » Lignes dans la base ») fin fin </pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>:SQLLitrecherche (condition,[NumRequête])</p>
<p>Détails</p> 	<p><resultat> : entier</p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p>n : nombre de ligne résultat de la requête</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><condition> : chaîne de caractères condition pour la recherche sans le where. C'est normalement ce qui est mis derriere le where pour effectuer une requêteSQL</p> <p><NumRequete> : entier [optionnel] numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
<p>Notes</p> 	<p>SQLLitBloque ne renvoie pas que vrai ou faux suivant si la requête a trouvé ou non il renvoie également le nombre de ligne dans le cas ou la recherche renverrait plusieurs lignes.</p> <p>l'objet correspondant ne sera chargé que si la requête contient 1 seule ligne. La lecture est donc a l'identique. La ligne correspondante dans la table est bloquée (si type de table le permet innodb pour mySQL) le blocage est fait a la ligne</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membre :ErreurSQL et numError sont positionnés..</p> <p>L'objet SQLManagerX est affecté par la méthode. Les membres sont mis a jour (seulement si le résultat renvoie vrai ou 1 seule ligne). Attention a bien effacé l'objet avant par un SQLRaz() pour éviter les mélanges de données provenant d'une ligne précédente.</p> <p>un lock est generer tout autre action que l'update suivant ce lock liberera le lock : c'est a dire que si on fait un rechercher ou une autre lecture sur le même objet alors le lock est libérer par un rollback. sinon les autres utilisateurs risquent d'être bloques pendant le temps du deadLock de la base.</p>




SQLManagerX ::SQLUpdate()	
<p>Utilisation</p> 	<p>Méthode permettant la mise à jour d'une ligne de la table SQL dans la base</p> <hr/> <p>Exemple :</p> <pre> Res = ADDR1:SQLLitBloque("Noprod=1000") si Res = 1 alors ADDR1 :m_Prenom = « Totot » ADDR1 :SQLUpdate() sinon Si res = 0 alors info(« Erreur : »+ADDR1 :ErreurSQL) sinon Info(« J'ai trouve « + res+ » Lignes dans la base ») fin fin </pre>
<p>Syntaxe</p>	<p><code><resultat> = <Objet>:SQLUpdate ([NumRequête])</code></p>
<p>Détails</p> 	<p><resultat> : <i>booleen</i></p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p> <p><NumRequete> : entier [optionnel]</p> <p>numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
<p>Notes</p> 	<p>L'update est optimisé sur les colonnes réellement modifiées. il n'est donc pas nécessaire de mettre a jour tous les membres de l'objet SQLManagerX avant de faire un update. On ne modifie que les membres qui en ont besoin.</p> <p>De plus la méthode gère seule la condition pour accéder à la ligne exacte par la clé primaire</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés.</p>




SQLManagerX ::SQLXDelete ()	
<p>Utilisation</p> 	<p>Méthode permettant la suppression d'une ou plusieurs ligne de la table SQL dans la base</p> <p>Exemple :</p> <p>Resultat = ADDR1:SQLXDelete("noprod=100")</p>
<p>Syntaxe</p>	<p><code><resultat> = <Objet>:SQLXdelete(condition ,[NumRequête])</code></p>
<p>Détails</p> 	<p><resultat> : booléen</p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p style="padding-left: 20px;">nom de l'objet table SQLManagerX</p> <p><condition> : chaîne de caractères</p> <p style="padding-left: 20px;">condition pour la recherche sans le where. C'est normalement ce qui est mis derriere le where pour effectuer une requêteSQL</p> <p><NumRequete> : entier [optionnel]</p> <p style="padding-left: 20px;">numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
<p>Notes</p> 	<p>le méthode refuse le delete si la condition est vide. Pour éviter la suppression totale des lignes d'une table</p>




SQLManagerX ::SQLXTable ()	
<p>Utilisation</p> 	<p>Méthode permettant d'afficher dans une table windev une ou plusieurs lignes de la table SQL dans la base</p> <p>Exemple :</p> <pre>ADDR1:SQLFiltre("", "order by work_list") ADDR1:SQLXtable("table1" , "Ascenseur1" , "work_list,lnotes" ,Vrai) ADDR1:SQLTableAffiche()</pre>
<p>Syntaxe</p>	<p><i><Objet>:SQLXTable (Nomtable,Ascen,ListeCol,[posAsc])</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NomTable> : chaîne de caractères nom de la table ou sera affiché le resultat</p> <p><Ascen> : chaîne de caractères nom de l'ascenseur de scroll pour la table. L'ascenseur doit être extérieur à la table. Sous l'éditeur de fenêtres de windev il faut donc enlever les ascenseurs de la table et en créer un.</p> <p><listeCol> : chaîne de caractères liste des colonnes à renvoyées dans la tables. On peut spécifier * pour toutes les colonnes.</p> <p><PosAsc> : booléen [optionnel] booléen spécifiant si l'ascenseur doit être repositionné ou non.</p>
<p>Notes</p> 	<p>on peut affecter un filtre avant le SQLXtable et dans ce cas ne rien mettre dans condition. l'ascenseur est repositionné par la méthode par défaut mais si on met le paramètre PosASC à faux alors il ne sera pas positionné. La position sous l'éditeur sera alors conservée.</p> <p>si toutes les colonnes de la tables doivent être renvoyée on peut mettre « * » dans le paramètre listeCol : attention toute fois il faut que la table est assez de colonnes pour pouvoir les afficher sinon un message le spécifiant sera envoyé.</p> <p>SQLXtable est une méthode ne remontant pas toutes les lignes mais seulement celle qui seront affichées dans la table. Donc il faut éviter les loupe et la possibilité de trier les colonnes car elle n'agissent que sur les ligne affichées. L'avantage est que SQLXtable est rapide quelque soit le nombre de lignes dans la Base SQL. Si vous voulez conservé les loupe et les tris il faut plutôt utiliser SQLCtable, plus lente mais qui remonte toutes les lignes de la base (suivant le filtre s'il y en a un)</p>




SQLManagerX ::SQLTableAffiche ()	
Utilisation 	Permet le reaffichage de la table rempli avec SQLXtable Exemple : ADDR1:SQLTableAffiche()
Syntaxe	<code><Objet> :SQLTableAffiche ()</code>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	code généralement a mettre dans le code de l'ascenseur




SQLManagerX ::SQLCTable ()	
<p>Utilisation</p> 	<p>Méthode permettant d'afficher dans une table windev une ou plusieurs lignes de la table SQL dans la base</p> <hr/> <p>Exemple :</p> <pre>ADDR1:SQLFiltre("", "order by work_list") ADDR1:SQLCtable("table1", "work_list,lnotes")</pre>
<p>Syntaxe</p>	<pre><Objet>:SQLCTable (Nomtable,ListeCol,[numReq])</pre>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NomTable> : chaîne de caractères nom de la table ou sera affiché le resultat</p> <p><listeCol> : chaîne de caractères liste des colonnes a renvoyées dans la tables. On peut sépcifier * pour toutes les colonnes.</p> <p>NumRequete> : entier [optionnel] numéro de la requête a utiliser. Par défaut SQLManagerX utilise le numéro 0 mais on peut lui spécifier un autre numéro de requête.</p>
<p>Notes</p> 	<p>On peut affecter un filtre avant le SQLCtable et dans ce cas ne rien mettre dans condition. l'ascenseur est repositionné par la méthode par défaut mais si on met le paramètre PosASC a faux alors il ne sera pas positionné. La position sous l'éditeur sera alors conservée.</p> <p>si toutes les colonnes de la tables doivent être renvoyée on peut mettre « * » dans le paramètre listeCol : attention toute fois il faut que la table est assez de colonnes pour pouvoir les afficher sinon un message le spécifiant sera envoyé.</p> <p>SQLCtable est une méthode remontant toutes les lignes Donc il faut éviter de le faire un SQLCtable sur une table de gros volume (le temps de chargement rique d'atre très long) ou alors affiné le filtre. L'avantage est que SQLCtable est rapide une fois chargé sur les tris et les loupes</p> <p>Si vous voulez un chargement rapide ou un parcours sur une table a gros volume il faut plutôt utiliser SQLXtable, plus rapide , mais ne remontant que des groupe de ligne (attention aux loupes et tris).</p>

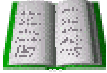


SQLManagerX ::SQLTableVersEcran ()	
<p>Utilisation</p> 	<p>Méthode permettant le transfert des membres de l'objet SQLManagerX vers les champs de l'écran</p> <p>Exemple :</p> <pre> ADDR1:SQLRaz() ADDR1:SQLEcranVersTable() SI ADDR1:SQLLitRecherche ("work_list="100") ALORS ADDR1:SQLTableVersEcran() SINON Info(ADDR1:ErrorNum+" / "+ADDR1:ErreurSQL) FIN </pre>
<p>Syntaxe</p>	<p><Objet>: <i>SQLTableVersEcran(NomFenetre)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NomFenetre> : chaîne de caractères nom de la fenetre windev contenant les champs a remplir</p>
<p>Notes</p> 	<p>SQLTableVersEcran permet de remplir les champs d'une fenêtre avec les valeurs correspondantes de l'objet SQLManagerX.</p> <p>ATTENTION aux noms de champs. Pour pouvoir faire cette action la méthode doit reconnaître les champs et les associer avec les colonnes de la table SQL. Pour nommer un champs il faut donc donné NOMCOL__NOMTABLE. par ex : NOM__FOURNISSE étant la colonne NOM de la table FOURNISSE(avec 2 "_").</p> <p>Le nom de la table n'est pas obligatoire, il le devient que si dans la fenêtre on a plusieurs champs NOM de plusieurs tables SQL différentes (Ceci permet a la méthode de ne pas remplir les champs n'appartenant pas a son objet table)</p> <p>Nom de la fenêtre (fenêtre en cours par défaut) ATTENTION Si le code d'appel a cette méthode se trouve dans le code d'initialisation d'une fenêtre, vous devez absolument spécifier le nom de la fenêtre car currenwin() renvoie sinon la fenêtre précédente</p>




SQLManagerX ::SQLEcranVersTable ()	
<p>Utilisation</p> 	<p>Méthode permettant le transfert des membres de l'objet SQLManagerX vers les champs de l'écran</p> <p>Exemple :</p> <pre>ADDR1:m_work_list = WORK_LIST ADDR1:SQLEcranVersTable() ADDR1:SQLUpdate()</pre>
<p>Syntaxe</p>	<p><Objet>: <i>SQLEcranVersTable(NomFenetre)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NomFenetre> : chaîne de caractères nom de la fenetre windev contenant les champs a remplir</p>
<p>Notes</p> 	<p>SQLEcranVersTable permet de remplir l'objet SQLManagerX. avec les valeurs correspondantes des champs d'une fenêtre</p> <p>ATTENTION aux noms de champs. Pour pouvoir faire cette action la méthode doit reconnaître les champs et les associer avec les colonnes de la table SQL. Pour nommer un champs il faut donc donné NOMCOL__NOMTABLE. par ex : NOM__FOURNISSE étant la colonne NOM de la table FOURNISSE(avec 2 "_").</p> <p>Le nom de la table n'est pas obligatoire, il le devient que si dans la fenêtre on a plusieurs champs NOM de plusieurs tables SQL différentes (Ceci permet a la méthode de ne pas remplir les champs n'appartenant pas a son objet table</p> <p>Nom de la fenêtre (fenêtre en cours par défaut) ATTENTION Si le code d'appel a cette méthode se trouve dans le code d'initialisation d'une fenêtre, vous devez absolument spécifier le nom de la fenêtre car currenwin() renvoie sinon la fenêtre précédente</p>




SQLManagerX ::SQLTableVersEtat ()	
<p>Utilisation</p> 	<p>Méthode permettant le transfert des membres de l'objet SQLManagerX vers les champs d'un etat</p> <p>Exemple :</p> <p>ADDR1:SQLTableVersEtat()</p>
<p>Syntaxe</p>	<p><Objet>: <i>SQLTableVersEtat()</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>SQLTableVersEtat permet de remplir les champs d'un etat avec les valeurs correspondantes de l'objet SQLManagerX.</p> <p>ATTENTION aux noms de champs. Pour pouvoir faire cette action la méthode doit reconnaître les champs et les associer avec les colonnes de la table SQL. Pour nommer un champs il faut donc donné NOMCOL__NOMTABLE. par ex : NOM__FOURNISSE étant la colonne NOM de la table FOURNISSE(avec 2 "_").</p> <p>Le nom de la table n'est pas obligatoire, il le devient que si dans l'etat on a plusieurs champs NOM de plusieurs tables SQL différentes (Ceci permet a la méthode de ne pas remplir les champs n'appartenant pas a son objet table)</p>




SQLManagerX ::GenereClasse()	
Utilisation 	Méthode permettant de générer en fichier txt la déclaration et le constructeur de la classe (objet SQLManagerX) correspondant a la table SQL Exemple : ADDR1:GenereClasse(« MonFic.txt »)
Syntaxe	<i><Objet>: GenereClasse (NomFichier)</i>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX <NomFichier> : Chaîne de caractère nom du fichier a créer : chemin complet
Notes 	le fichier crée contient la déclaration et le constructeur de la classe qui permet de gérer l'objet




SQLManagerX ::SQLPrepare()	
<p>Utilisation</p> 	<p>Méthode permettant de préparer des lignes d'insertion pour envoyer vers le serveur le moins de requêtes possibles</p> <p>Exemple :</p> <pre> POUR i= 1 TO nbenreg ADDR1:m_work_list="MSRG"+NumériqueVersChaine(i,"03d") ADDR1:m_Inotes = "Mon Prenom est "+(i+10) ADDR1:SQLPPrepare() FIN addr1:SQLExecute()</pre>
<p>Syntaxe</p>	<p><Objet>: SQLPrepare()</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>Lors d'un insert de 500 lignes (par exemple) au lieu de faire MaTab:SqlInsert on peut faire Matab:SqlPrepare. la requête est générée mais mise en attente. Sauf dans le cas ou les 64 K sont atteints, dans ce cas la requête est envoyée et on continue la préparation.</p> <p>L'ordre pour exécuter la requête est SQLExecute</p>




SQLManagerX ::SQLExecute ()	
<p>Utilisation</p> 	<p>Méthode permettant de préparer des lignes d'insertion pour envoyer vers le serveur le moins de requêtes possibles</p> <p>Exemple :</p> <pre> POUR i= 1 TO nbenreg ADDR1:m_work_list="MSRG"+NumériqueVersChaine(i,"03d") ADDR1:m_Inotes = "Mon Prenom est "+(i+10) ADDR1:SQLPRepare() FIN ADDR1:SQLExecute()</pre>
<p>Syntaxe</p>	<p><Objet>: SQLExecute()</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>Lors d'un insert de 500 lignes (par exemple) au lieu de faire MaTab:SqlInsert on peut faire Matab:SqlPrepare. la requête est générée mais mise en attente. Sauf dans le cas ou les 64 K sont atteints, dans ce cas la requête est envoyée et on continue la préparation.</p> <p>L'ordre pour exécuter la requête est SQLExecute</p>




SQLManagerX ::SQLPremier ()	
<p>Utilisation</p> 	<p>La méthode permet de se positionner et lire le premier enregistrement de la condition sur l'objet en cours</p> <p>Exemple :</p> <pre>FOUR:SQLPremier() TANT que pas FOUR:endehors Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM) FOUR:SQLSuivant() fin Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM)</pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>: SQLSuivant ()</p>
<p>Détails</p> 	<p><resultat> : <i>booleen</i></p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p style="padding-left: 40px;">nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>La méthode revoie vrai si au moins une ligne a ete trouvée et positionne le membre endehors a vrai ou faux (vrai si au moins une ligne a ete trouvee)</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p>




SQLManagerX ::SQLSuivant ()	
<p>Utilisation</p> 	<p>La méthode permet de se positionner et lire l'enregistrement suivant de la condition sur l'objet en cours</p> <p>Exemple :</p> <pre> FOUR:SQLPremier() TANT que pas FOUR:endehors Trace(FOUR:m_nofourn+ " : "+FOUR:m_NOM) FOUR:SQLSuivant() fin </pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>. SQLpremier()</p>
<p>Détails</p> 	<p><resultat> : <i>booleen</i></p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>La méthode renvoie vrai si au moins une ligne a été trouvée et positionne le membre endehors a vrai ou faux (vrai si au moins une ligne a été trouvée)</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p>




SQLManagerX ::SQLPrecedent ()	
<p>Utilisation</p> 	<p>La méthode permet de se positionner et lire l'enregistrement precedent de la condition sur l'objet en cours</p> <hr/> <p>Exemple :</p> <pre> FOUR:SQLdernier() TANT que pas FOUR:endehors Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM) FOUR:SQLprecedent() fin Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM) </pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>: SQLprecedent()</p>
<p>Détails</p> 	<p><resultat> : <i>booleen</i></p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>La méthode revoie vrai si au moins une ligne a ete trouvée et positionne le membre endehors a vrai ou faux (vrai si au moins une ligne a ete trouvee)</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p>




SQLManagerX ::SQLDernier()	
<p>Utilisation</p> 	<p>La méthode permet de se positionner et lire le dernier 'enregistrement de la condition sur l'objet en cours</p> <p>Exemple :</p> <pre>FOUR:SQLdernier() TANT que pas FOUR:endehors Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM) FOUR:SQLprecedent() fin Trace(FOUR:m_nofourn+" : "+FOUR:m_NOM)</pre>
<p>Syntaxe</p>	<p><resultat> = <Objet>. SQLprecedent()</p>
<p>Détails</p> 	<p><resultat> : <i>booleen</i></p> <p>vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>La méthode revoie vrai si au moins une ligne a ete trouvée et positionne le membre endehors a vrai ou faux (vrai si au moins une ligne a ete trouvee)</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p>




SQLManagerX ::SQLFiltre()	
<p>Utilisation</p> 	<p>La méthode permet de créer un filtre de lecture sur l'objet SQLManagerX</p> <hr/> <p>Exemple :</p> <pre>ADDR1:SQLFiltre(" Inotes <> """, "order by Inotes ") ADDR1:SQLPremier() ADDR1:SQLTableVersEcran()</pre>
<p>Syntaxe</p>	<pre><resultat> = <Objet>.SQLFiltre(condition,order)</pre>
<p>Détails</p> 	<p><resultat> : <i>booleen</i> vrai l'action s'est bien déroulée, faux erreur de la requête ou l'action n'a pas pu être faite</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><condition> : chaîne de caractères condition pour le filtre sans le where. C'est normalement ce qui est mis derriere le where pour effectuer une requêteSQL</p> <p><order> : chaîne de caractères order pour le filtre (ex si on veut order by nom)</p>
<p>Notes</p> 	<p>La méthode renvoie vrai si le filtre a pu être initialiser</p> <p>Dans le cas de faux mais parce que la requête n'a pas pu aboutir les membres :ErreurSQL et numError sont positionnés..</p> <p>La méthode positionne le membre filtreActif a vrai</p> <p>La méthode remplit le membre :NbLig avec le nombre d'enregistrements faisant partis du filtre</p> <p>Si condition est vide alors toutes les lignes de la table seront concerné par les ordres de lecture suivants</p>




SQLManagerX ::SQLActiveFiltre()	
Utilisation 	La méthode permet d'activer le filtre sur l'objet SQLManagerX contenu dans :filtre Exemple : ADDR1:SQLActiveFiltre()
Syntaxe	<i><Objet> :SQLActiveFiltre()</i>
Détails 	<i><Objet></i> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	




SQLManagerX ::SQLDesActiveFiltre()	
Utilisation 	La méthode permet de désactiver le filtre sur l'objet SQLManagerX contenu dans :filtre Exemple : ADDR1:SQLDesActiveFiltre()
Syntaxe	<i><Objet> :SQLDesActiveFiltre()</i>
Détails 	<i><Objet></i> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	




SQLManagerX ::TransactionActive ()	
Utilisation 	La méthode permet d'activer les transactions Exemple : ADDR1:transactionActive ()
Syntaxe	<i><Objet> :TransactionActive ()</i>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	Par défaut SQLManagerX gère tout en transaction (surtout les litBloque et update). On peut cependant désactiver les transactions pour avoir des update imbriquée. Un seul niveau de transaction étant possible on désactive les transaction des objet pour les gérer manuellement




SQLManagerX ::TransactionDesActive()	
Utilisation 	La méthode permet de désactiver les transactions Exemple : ADDR1:TransactionDesActive ()
Syntaxe	<code><Objet> :TransactionDesActive ()</code>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	Par défaut SQLManagerX gère tout en transaction (surtout les litBloque et update). On peut cependant désactiver les transactions pour avoir des update imbriquée. Un seul niveau de transaction étant possible on désactive les transaction des objet pour les gérer manuellement.




SQLManagerX ::EnumToCombo ()	
<p>Utilisation</p> 	<p>Permet le Chargement d'une combo par rapport au valeur de l'Enum de la base SQL</p> <p>Exemple :</p> <p>ADDR1:EnumToCombo("ChoixWork", "MaComboChoixWork")</p>
<p>Syntaxe</p>	<p><i><Objet> :EnumToCombo(colonne,combo,valeurDefaut)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><colonne > : chaîne de caractère nom de la colonne contenant l' ENUM</p> <p><Combo> : chaîne de caractère nom de la combo</p> <p><ValeurDefaut> : Chaîne de caractère Valeur par défaut insérer dans le cas de non selection de la combo ou valeur vide</p>
<p>Notes</p> 	<p>si la colonne de la table SQL est de type ENUM, on recupere dans la base SQL les differentes valeurs de l'enum et les inserer dans une combo pour selection. Les méthodes SQLTableVersEcran et SQLEcranVersTable sont compatoble avec des combo rempli par cette méthode.</p>




SQLManagerX ::SQLChargeCombo()	
<p>Utilisation</p> 	<p>La méthode permet de charger une combo</p> <p>Exemple :</p> <p>ADDR1:SQLchargeCombo("work_list","Select work_list,work_list from work_list")</p>
<p>Syntaxe</p>	<p><i><Objet> :SQLChargeCombo(Combo,requete, [valeurDefaut],[IdVisible])</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><Combo> : chaîne de caractère nom de la combo</p> <p><requete> : chaîne de caractère requête complète comme si on la lançait depuis un éditeur SQL</p> <p><ValeurDefaut> : Chaîne de caractère Valeur par défaut insérer dans le cas de non selection de la combo ou valeur vide</p> <p><idVisible> : booleen booleen spécifiant si l'Id sera affiché ou non</p>
<p>Notes</p> 	<p>permet de remplir une combo par des valeur d'une table.</p> <p>la requête doit avoir 2 colonnes : Un ID et un texte. L'id est généralement le code qui sert a faire la liason avec une autre table. Le libelle est celui affiché dans la combo.</p> <p>Idvisible permet de faire apparaître dans la combo en plus du texte la valeur de l'ID</p> <p>par exemple : un table TVA qui contient un code TVA et un libelle. Dans les table liées TVA contient bien entendu l'indice (ou code) ainsi nous pouvons avoir dans windev une combo affichant le libelle de TVA mais qui contient en réalité le code.</p> <p>La méthode est compatible avec les méthodes SQLTableVersEcran et SQLEcranVersTables. La combo rempli par cette méthode affichera le bon libellé.</p>




SQLManagerX ::SaisieAssitée ()	
<p>Utilisation</p> 	<p>Permet une saisie assistée d'un champs par rapport aux valeur de la base SQL</p> <p>Exemple :</p> <pre>ADDR1:SaisieAssitee("work_list", "WORK_LIST")</pre>
<p>Syntaxe</p>	<p><i><Objet> :saisieAssitee(colonne,champs)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><colonne > : chaîne de caractère nom de lla colonne contenant l' ENUM</p> <p><champs> : chaîne de caractère nom du champs</p>
<p>Notes</p> 	<p>La méthode permet dans un champs de saisie au fur et a mesure de la saisie d'affichée la valeur dans la base la plus proche de la valeur saisie.</p>




SQLManagerX ::SQLFichierExiste ()	
Utilisation 	Teste l'existence de la table dans la Base lors de problèmes. Exemple : ADDR1:SQLFichierExiste()
Syntaxe	<Objet> :SQLFichierExiste()
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	




SQLManagerX ::SQLSauvePosition ()	
<p>Utilisation</p> 	<p>Sauvegarde de la position en cours pour manipulation de l'objet</p> <hr/> <p>Exemple :</p> <p>MaPosition est une chaine MaPosition = ADDR1:SQLSauvePosition() ADDR1 :SQLRaz() ADDR1:RetourPosition (MaPosition)</p>
<p>Syntaxe</p>	<p><resultat> = <Objet> :SQLSauvePosition()</p>
<p>Détails</p> 	<p><resultat> : chaîne de caractere position de l'element sauvegarder</p> <p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p>
<p>Notes</p> 	<p>Cela permet de manipuler 2 lignes d'une table tout en conservant la position précédente pour y revenir par RetourPosition</p>




SQLManagerX ::SQLRetourPosition()	
Utilisation	Retour a la position précédente
	Exemple : MaPosition est une chaine MaPosition = ADDR1:SQLSauvePosition() ADDR1 :SQLRaz() ADDR1:RetourPosition(MaPosition)
Syntaxe	<Objet> :SQLRetourPosition(Position)
Détails	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p> <position> : chaîne de caractère position a restaurer : chaîne de caractère renvoyer par la fonction SQLSauvePosition</p>
Notes	<p>Par défaut SQLManagerX gère tout en transaction (surtout les litBloque et update).</p> <p>On peut cependant désactiver les transactions pour avoir des update imbriquée. Un seul niveau de transaction étant possible on désactive les transaction des objet pour les gérer manuellement</p>
	




SQLManagerX ::SQLLien ()	
<p>Utilisation</p> 	<p>Permet de lier 2 objets SQLManagerX L'objet lié est automatiquement filtré sur la cle du premier</p> <p>Exemple :</p> <pre>si ADDR1:SQLlitrecherche("**", "work_list="+WORK_LIST+"") alors ADDR1:SQLtableversecran() ADDR2:SQLlien("ADDR1") fin</pre>
<p>Syntaxe</p>	<p><Objet> :SQLLien(ObjetLie)</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><ObjetLie> : objet SQLManagerX nom de l'objet table SQLManagerX qui sert de lien</p>
<p>Notes</p> 	<p>Il est possible de lier 2 ou plusieurs objets SQLManagerX.</p> <p>Par exemple j'ai une table enteCommande et ligneCommande il est bien évident que lignecommande a une clé étrangère qui est la clé de entecommande. Et bien en parcourant entecommande on peut dire à SQLManagerX de le faire directement sur ligneCommande.</p> <p>Ainsi a chaque lecture de entecommande le filtre sur lignecommande change pour être directement filtré sur la clé en cours de enteCommande</p> <p>Le lien est en fait un filtre sur l'objet courant avec la clé primaire de l'objet parent, il faut donc activer un filtre avec la clé de l'objet parent pour avoir table1. ID = table2.ID and pour les partie de la clé primaire parent condition a transmettre au filtre en plus id pour order</p> <p>on peut lie ainsi plusieurs objet SQLManagerX. Il peut y avoir qu'un seul parent mais plusieurs enfant (tables liées)</p>




SQLManagerX ::SQLActivelien()	
Utilisation 	La méthode permet d'activer le(s) lien(s) sur l'objet SQLManagerX Exemple : ADDR1:SQLActiveLien()
Syntaxe	<code><Objet> :SQLActiveLien()</code>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	

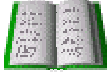


SQLManagerX ::SQLDesActivelien()	
Utilisation 	La méthode permet de désactiver le lien sur l'objet SQLManagerX Exemple : ADDR1:SQLDesActiveLien()
Syntaxe	<i><Objet> :SQLDesActiveLien()</i>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	

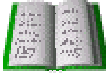


SQLManagerX ::SQLRecupereExec()	
<p>Utilisation</p> 	<p>Permet de récupérer les colonnes d'une requête pour les affecter à l'objet SQLManagerX. La méthode reconnaîtra dans la requête les colonnes concernant son objet et lui donnera les valeurs</p> <p>Exemple :</p> <p>Requete est un chaine</p> <pre>// jointure sur 2 table requete = « SELECT Inotes, work_list.Id, maliste from work_list, liste where work_list.ID = liste.ID » SI convSql:mysqlExec(requete,2) ALORS convSql:mysqlPremier(2) // recuperation des elements de l'objet ADDR1 dans la requete SI ADDR1:SQLRecupereExec(2) ALORS ADDR1:SQLTableVersEcran() FIN //recuperation des elements de l'objet LISTE1 dans la requete SI LISTE1:SQLRecupereExec(2) ALORS LISTE1:SQLTableVersEcran() FIN FIN convSql:mysqlFerme(2)</pre>
<p>Syntaxe</p>	<p><Objet> :SQLRecupereExec(NumRequete)</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NumRequete> : entier numéro de la requête à utiliser pour la récupération</p>
<p>Notes</p> 	<p>SQLManagerX gère les tables de façon autonomes et donc en mode fiche.</p> <p>Si on veut avoir des requêtes complexes (jointure etc....) il faut le faire directement avec l'accès natif. Normalement ensuite il faut faire des SQLLitCol pour récupérer les valeurs.</p> <p>En utilisant SQLRecupereExec il est possible de charger les différents objets SQLManagerX concernés par les valeurs de la requête.</p> <p>Cela évite des variables pour stockés les valeurs de la requêtes. Et évite de programmer tous les SQLLitCol.</p>




SQLManagerX ::SQLLigneSelect()	
Utilisation 	<p>Permet de générer le code SQL comprenant toutes les colonnes de l'objet pour créer une requete. A la place de * permet d'avoir toutes les colonnes nommées</p> <p>Exemple :</p> <p>Requete est un chaine requete = ADDR1 :SQLLigneSelect() // renvoie work_list, lnotes, maliste</p>
Syntaxe	<p><Objet> :LigneSelect()</p>
Détails 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p>
Notes 	




SQLManagerX :: SQLCreeVue ()	
<p>Utilisation</p> 	<p>Permet de créer une « vue » . Principe : ne remonte que les colonnes dont on a besoin dans l'objet. SQLManagerX par défaut remonte toutes les colonnes d'une table et charge l'objet correspondant. avec les vue on peut dire a SQLManagerX de ne s'occuper que de certaines colonnes</p> <p>Exemple :</p> <pre>requete = ADDR1 :SQLCreeVue(" work_list,lnotes ", " worl_list > 10000", "order by lnotes")</pre>
<p>Syntaxe</p>	<p><i><Objet> : SQLCreeVue(p_ListeRubriques , p_condition ,p_order)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><ListeRubrique> : chaine liste des colonnes concernées par la vue</p> <p><condition> : chaine condition a implementer (le where de la requete)</p> <p><order> : chaine tri de la vue</p>
<p>Notes</p> 	<p>en SQL une vue est en fait une extraction de certaine données de la base. SQLManagerX simule cet etat en ne remontant que quelque colonnes et en appliquant un filtre. Cela permet de n'avoir dans le parcours SQLManagerX que les données que l'on désire.</p> <p>attention SQLManagerX ne créer pas dans la base de table temporaire ou de vue à proprement parlé. Cette fonction permet de simulé les vues. Ceci est interessant surtout pour les bases ne gérant pas les vues.</p>




SQLManagerX :: SQLDétruitVue ()	
Utilisation 	<p>Permet de supprimer une « vue » .</p> <p>Principe : ne remonte que les colonnes dont on a besoin dans l'objet. SQLManagerX par défaut remonte toutes les colonnes d'une table et charge l'objet correspondant. avec les vue on peut dire a SQLManagerX de ne s'occuper que de certaines colonnes</p> <p>Exemple :</p> <pre>requete = ADDR1 :SQLDétruitVue()</pre>
Syntaxe	<code><Objet> : SQLDétruitVue()</code>
Détails 	<p><Objet> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p>
Notes 	<p>en SQL une vue est en fait une extraction de certaine données de la base. SQLManagerX simule cet etat en ne remontant que quelque colonnes et en appliquant un filtre. Cela permet de n'avoir dans le parcours SQLManagerX que les données que l'on désire.</p> <p>attention SQLManagerX ne créer pas dans la base de table temporaire ou de vue à proprement parlé. Cette fonction permet de simulé les vues. Ceci est interessant surtout pour les bases ne gérant pas les vues.</p>




SQLManagerX :: <i>SQLDump</i> ()	
<p>Utilisation</p> 	<p>Permet d'exporter les données d'une table vers un fichier .sql (Ecriture de requête SQL)</p> <hr/> <p>Exemple :</p> <pre>ADDR1:SQLDump("c:\essai.sql", "", Vrai)</pre>
<p>Syntaxe</p>	<pre><Objet> :SQLDump(p_FichierDest , p_bEnAppend , p_condition , p_numRequete , p_Drop , p_Jauge)</pre>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_FichierDest > : chaine Chemin et nom complet du fichier de destination</p> <p>< p_bEnAppend > : booleen Vrai = Ouvrir le fichier de destination et ajouter à la fin Faux = Ouvrir le fichier en création (vide)</p> <p>< p_Condition > : chaine Condition SQL à appliquer sur les données à exporter</p> <p>< p_NumRequete > : entier N°Requete à utiliser</p> <p>< p_Drop > : booleen stipulant si le DROP table doit etre mis dans le fichier</p> <p>< p_Jauge > chaine Nom du champ de la barre de progression (optionnel)</p>
<p>Notes</p> 	<p>SQLDump permet de créer un fichier de sauvegarde de la base.</p> <p>On peut avoir un fichier global de la base ou alors créer un fichier par table. On peut également créer des export facilement puisqu'on peut rajouter une condition</p>




SQLManagerX :: SQLDuplicate ()	
<p>Utilisation</p> 	<p>Permet de dupliquer un objet SQLManagerX. Cette fonction est très utile pour avoir 2 objet similaires ayant le même contexte.</p> <hr/> <p>Exemple :</p> <pre> ADDR1:SQLPremier() TANTQUE PAS ADDR1:endehors Trace("ADDR1 : "+ADDR1:m_work_list+ " / "+ADDR1:m_Inotes) ADDR2:SQLDuplicate(ADDR1,2) Trace("ADDR2: "+ADDR2:m_work_list+ " / "+ADDR2:m_Inotes) ADDR1:SQISuivant() FIN </pre>
<p>Syntaxe</p>	<p><i><Objet> :SQLDuplicate(p_ObjetDestination,p_sensAction)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_ObjetDestination > : <objet> nom de l'objet SQLManagerX destination</p> <p>< p_sensAction > : entier</p> <ul style="list-style-type: none"> 2 : copie dans les valeur AncCol 1 : copie dans les valeur ValCol 0 : Copie dans les 2 valeur AnCOI et ValCol
<p>Notes</p> 	<p>avec le sensAction = 0 on duplique complètement tout l'objet. Les 2 objets SQLMANagerX seront identiques en tout points.</p>




SQLManagerX :: SQLAttacheMemo ()	
<p>Utilisation</p> 	<p>charge un fichier image document autre dans une colonne mémo ensuite il restera plus qu'a faire SQLinsert ou SQLupdate</p> <hr/> <p>Exemple :</p> <pre> ADDR1:m_work_list="MSRG"+NumériqueVersChaine(i,"03d") ADDR1:m_inotes = "Mon Prenom est "+(i+10) ADDR1:SQLAttacheMemo("photo","c:\logo.jpg") SI PAS ADDR1:SQLInsert() ALORS Info(ADDR1:ErreurSQL) </pre>
<p>Syntaxe</p>	<p><i><Objet> : SQLAttacheMemo (p_nomColonne ,p_NomFichier)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_ nomColonne > : chaine nom de la colonne dans la base SQL qui contient le memo</p> <p>< p_ NomFichier > : chaine nom du fichier (image ou document) qui sera inserer en tant que memo dans la colonne</p>
<p>Notes</p> 	




SQLManagerX :: SQLChargeMemo ()	
<p>Utilisation</p> 	<p>C'est l'équivalent de SQLLitCol mais pour une colonne de type memo. Si vous executez une requête dans laquelle une colonne de type memo est renvoyée. C'est le seul moyen pour récupérer ce que la base SQL a renvoyée.</p>
<p>Exemple :</p>	<p>ADDR1:SQLChargeMemo("WORD","photo")</p>
<p>Syntaxe</p>	<p><Objet> : SQLChargeMemo (p_nomChamps, p_nomColonne)</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_ nomChamps > : chaine nom du champs de la fenetre qui doit afficher le memo (champs image , html, ou texte)</p> <p>< p_ nomColonne > : chaine nom de la colonne dans la base SQL qui contient le memo</p>
<p>Notes</p> 	




SQLManagerX :: <i>SQLEdit</i> ()	
<p>Utilisation</p> 	<p>SQLEdit est utilisée par les 2 états permettant de faire une édition d'une requête.</p> <hr/> <p>Exemple :</p> <pre>ilprimeEtat(R_SQLEtatP,"SELECT work_list,Inotes,Inotes as L2 FROM work_list","Essai d' Edition","ADDR1")</pre>
<p>Syntaxe</p>	<p><Objet> : <i>SQLEdit</i> (p_ p_numReq)</p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_numReq > : entier numéro de la requête exécutée et a éditer</p>
<p>Notes</p> 	<p>Vous n'avez pas besoin d'utiliser cette méthode directement l'appel a cette méthode est fait par les 2 états SQLEtatP et SQLEtat</p> <p>Cette méthode est également utiliser par SQLTableEdit qui permet d'éditer une table chargée par SQLCtable ou SQLXtable</p> <p>Vous pouvez en nommant vos colonnes intervenir sur le format d'edition ou demander des somme ou rupture. Par exemple select macolone as [option]_nomaffiche</p> <p>Les differentes options :</p> <ul style="list-style-type: none"> C : la colonne sera centree S : la colonne est somme D : cadrage a droite N+chiffre : Format decimal avec chiffre apres la virgule : Ex N3 pour 3 decimales L+nombre : largeur forcee pour la colonne en pixel : ex L50_ R : Demande une rupture sur la colonne. Plusieurs colonnes peuvent servir Y : Somme sur la colonne sur la rupture (la somme sera remise à 0 avec la rupture <p>TOUTES CES OPTIONS SONT CUMULABLES : c'est a dire que je peux les mettre a la suite par exemple select macolone as SN3C_nomAffiche from.... Fera de la colonne une somme, la colonne sera affichee avec 3 decimales et sera centree</p> <p>ATTENTION : S, Y doivent etre positionne avant les formatage par exemple si vous voulez une somme centree il faut faire SC_ et non CS_</p> <p>LES DEUX ETATS R_SQLLETAT.WDE et R_SQLLETATP.WDE sont necessaire au fonctionnement de la methode. Il se trouve dans le projet exemple de SQLManagerX</p>

SQLManagerX :: SQLTableEdit ()	
<p>Utilisation</p> 	<p>Permet d'editer une table prealablement chargée par SQLXtable ou SQLCtable.</p> <hr/> <p>Exemple :</p> <pre>// chargement de la table : TableSupprimeTout(tbl_client) client:SQLFiltre(v_sqlquery,v_order) client:SQLXtable("tbl_client","asc_client",v_colonne,Vrai) client:SQLTableAffiche() // bouton edition client:SQLTableEdit("Liste des Clients")</pre>
<p>Syntaxe</p>	<p><Objet> : <i>SQLTableEdit (p_titre est une chaîne, p_Orientation est une chaîne="")</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p>< p_titre > : chaîne titre de l'edition</p> <p><p_Orientation> : chaîne orientation de l'etat :</p> <ul style="list-style-type: none"> « PORTRAIT » ou vide pour l'edition portrait « PAYSAGE> < LANDSCAPE » : pour le paysage
<p>Notes</p> 	<p>SQLTableEdit utilise les etats SQLetatP et SQLetat ces etats sont generiques mais peuvent etre modifié pour ressembler a vos edition.</p>




SQLManagerX :: <i>GetShema</i> ()	
Utilisation 	Permet d'avoir sous forme de chaîne, la creation de la table et des index dans 2 membres :shemaTable et :shemaIndex Exemple : // chargement de la table : client:GetShema() trace(client :shemaTable) trace(client :shemaIndex)
Syntaxe	<Objet> : <i>GetShema</i> ("")
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	GetShema permet de récupérer les instruction de création de la table et des index pour par exemple faire une procédure de sauvegarde et régénération de la table




SQLManagerX :: <i>GetPrimaryKey ()</i>	
Utilisation	Permet de connaître les colonnes faisant partie de la primary key de la table
	Exemple : <code>trace(client :getPrimaryKey())</code>
Syntaxe	<code><Objet> : getPrimaryKey ()</code>
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	




SQLManagerX :: SetPrimaryKey ()	
Utilisation 	<p>Permet de modifier les colonnes faisant partie de la primary key de la table</p> <hr/> <p>Exemple :</p> <pre>client :SetPrimaryKey("Nom,prenom")</pre>
Syntaxe	<pre><Objet> : setPrimaryKey (p_nomDesColonnes)</pre>
Détails 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><p_nomDesColonnes> : chaîne liste des colonnes devant faire partie de la primary key séparées par une virgule <i>si p_nomDesColonne est vide alors la méthode remettra les colonnes comme dans la base de données. Les colonnes not null seront celle connues du serveur</i></p>
Notes 	<p>Pour PERVASIVE cette fonction est obligatoire dans la déclaration de l'objet. La base ne renvoyant pas les colonnes qui font partie de la primary key il faut alors au moment de la déclaration rajouter une ligne SetPrimaryKey pour que SQLManagerX fonctionne correctement</p>




SQLManagerX :: <i>GetAutoIncrement()</i>	
Utilisation	Permet de connaître les colonnes qui sont autoIncrement dans la table
 Version 4	Exemple : <code>trace(client : <i>GetAutoIncrement</i> ())</code>
Syntaxe	<code><Objet> : <i>GetAutoIncrement</i> ()</code>
 Détails	<code><Objet> : objet SQLManagerX</code> nom de l'objet table SQLManagerX
 Notes	




SQLManagerX :: <i>SetAutoIncrement</i> ()	
Utilisation 	<p>Permet de modifier les colonnes qui sont autoIncrement</p> <hr/> <p>Exemple :</p> <p>client : <i>SetAutoIncrement</i> ("Nom,prenom")</p>
Syntaxe	<p><Objet> : <i>SetAutoIncrement</i> (p_nomDesColonnes)</p>
Détails 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><p_nomDesColonnes> : chaîne liste des colonnes devant être auto Increment séparées par une virgule</p> <p><i>si p_nomDesColonne est vide alors la méthode remettra les colonnes comme dans la base de données. Les colonnes not null seront celle connues du serveur</i></p>
Notes 	<p>SQLManagerX gère les autoIncrement en n'envoyant pas la valeur dans la base (cette dernière faisant l'increment et ajoutant dans la table la valeur) cette valeur est normalement stocke dans :dernierID</p> <p>il n'y a pas vraiment de cas précis ou on a besoin de modifier les autoIncrement. Cette fonction est plutôt la au cas ou SQLManagerX n'arriverai pas pas déterminé que la colonne est auto Increment dans votre base.</p>




SQLManagerX :: <i>GetNotNull</i> ()	
Utilisation 	Permet de connaître les colonnes qui sont not null dans la table Exemple : trace(client : <i>GetNotNull</i> ())
Syntaxe	<Objet> : <i>GetNotNull</i> ()
Détails 	<Objet> : objet SQLManagerX nom de l'objet table SQLManagerX
Notes 	

SQLManagerX :: SetNotNull ()	
<p>Utilisation</p>	<p>Permet de modifier les colonnes qui sont autoIncrement</p>
<p>Version</p> 	<p>Exemple :</p> <p>client : <i>SetNotNull</i> ("Nom,prenom")</p>
<p>Syntaxe</p>	<p><i><Objet></i> : <i>SetNotNull</i> (<i>p_nomDesColonnes</i>)</p> <p><i><p_nomDesColonnes></i> : chaîne</p> <p>liste des colonnes devant être auto Increment séparées par une virgule</p> <p><i>si p_nomDesColonne est vide alors la méthode remettra les colonnes comme dans la base de données. Les colonnes not null seront celle connues du serveur</i></p>
<p>Détails</p> 	<p><i><Objet></i> : objet SQLManagerX</p> <p>nom de l'objet table SQLManagerX</p> <p><i><p_nomDesColonnes></i> : chaîne</p> <p>liste des colonnes devant être non null séparées par une virgule</p>
<p>Notes</p> 	<p>SQLManagerX V5 gerera les valeur non null. Il pourra ainsi verifier que la colonne n'a pas de valeur null avant d'envoyer la requete. Vous pouvez par cette méthode demande que certaines colonnes soient contrôlées par SQLManagerX</p>

SQLManagerX :: <i>SQLAttacheMemo</i> ()	
<p>Utilisation</p>	<p>Permet de lier un memo a une colonne de l'objet</p>
<p>Version</p> 	<p>Exemple :</p> <p>client : <i>SQLAttacheMemo</i> ("memoClient", "c:\maphoto.jpg")</p>
<p>Syntaxe</p>	<p><i><Objet></i> : <i>SQLAttacheMemo</i> (<i>p_nomDesColonne</i>, <i>p_nomFichier</i>)</p> <p><i><p_nomDesColonnes></i> : chaîne nom de la colonne qui recevra le fichier</p> <p><i><p_nomFichier></i> : chaîne <i>nom du fichier (image ou word) a transferer avec le chemin complet</i></p>
<p>Détails</p> 	
<p>Notes</p> 	<p>Pour l'instant SQLManagerX est compatible avec les memos MySQL, SQLite, FB Prévu rapidement PostgreSQL, HF OTL</p>

SQLManagerX :: <i>SQLChargeMemo()</i>	
<p>Utilisation</p>	<p>Permet d'envoyer une colonne memo de la table SQL dans un champs,</p>
<p>Version</p> 	<p>Exemple :</p> <p>client : <i>SQLChargeMemo</i> ("MonChamp ","memoClient")</p>
<p>Syntaxe</p>	<p><i><Objet></i> : <i>SQLChargeMemo</i> (<i>p_non du Camps de la fenetre</i> , <i>p_nomcolonne SQL</i>)</p> <p><i><p_nom du champs de la fenetre ></i> : chaîne nom du champs de la fenetre qui doit afficher le memo (image, html, bouton etc)</p> <p><i><p_nom de la colonne SQL></i> : chaîne nom de la colonne dans la table SQL</p>
<p>Détails</p> 	
<p>Notes</p> 	<p>Pour l'instant SQLManagerX est compatible avec les memos MySQL, SQLite, FB Prévu rapidement PostGreSQL, HF OTL</p>

SQLManagerX :: <i>SQLTableConstruit</i> ()	
<p>Utilisation</p> 	<p>SQLTableConstruit permet de créer une table memoire dynamiquement en fonction du filtres et des colonnes a renvoyées. elle</p> <hr/> <p>Exemple :</p> <p>AVEC SQLCTABLE, affichage des PK et mode saisie exemple o_test un un c_test(consq,"test","o_test") o_test:sqlfiltre("id>100 order by id") o_test:sqltableconstruit("table1","nom,prenom,id","","faux,vrai) charge la table1 avec nom,prenom,id, colorise les PK, et met en saisie</p> <p>AVEC SQLXTABLE, et mode affichage exemple o_test un un c_test(consq,"test","o_test") o_test:sqlfiltre("id>100 order by id") o_test:sqltableconstruit("table1","nom,prenom,id","ascenseur1",faux,faux,faux)</p>
<p>Syntaxe</p>	<p><i><Objet> : SQLTableConstruit(Xlpar_table , xlpar_listeCol ,Xlpar_ascenseur , Xlpar_positionAcscenseur ,Xlpar_key, Xlpar_actif ,Xlpar_memo ,p_numRequete)</i></p>
<p>Détails</p> 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><Xlpar_table> : chaîne nom de la table mémoire dans la fenêtre</p> <p><xlpar_listeCol> : chaîne nom des champs ou *</p> <p><Xlpar_ascenseur> : chaîne nom de l'ascenseur ou "" la chaîne vide "" utilise la méthode Ctable sinon c'est la méthode Xtable qui est utilisée</p> <p><Xlpar_positionAcscenseur> : booleen pour la position de l'ascenseur (vrai : SQLManagerX repositionne l'ascenseur)</p> <p><Xlpar_key> : booleen indiquant si colorise ou pas les PK</p> <p><Xlpar_actif > : booleen indiquant si met en saisie ou pas les colonnes</p> <p><Xlpar_memo> : booleen indiquant si on charge ou pas les blob/memo dans les colonnes</p> <p><p_numRequete> : entier numéro de la requete</p>
<p>Notes</p> 	<p>ATTENTION : le paramètre XLPAr_listeCol est très important. En effet il détermine quelle méthode va être utilisée pour charger la table si ce paramètre est a vide c'est la méthode Ctable qui est utilisée et donc toutes les lignes vont être chargée. Donc pour certaine requête cela risque d'être lent.</p> <p>Il vaut mieux privilégier Xtable en donnant le nom des colonnes ou bien avoir un filtre permettant de ne remonter que quelque lignes dans la table mémoire.</p>

SQLManagerX :: SQLGereFiche()	
Utilisation	<p>SQLgereFiche permet de demander a SQLManagerX de gerer toute la fiche et de generer le code de tous les elements de la fenetres. Ainsi une seule ligne de code permet de gerer la fiche</p> <p>Exemple :</p> <pre>client:SQLgereFiche("w_client_fiche",vrai)</pre>
Syntaxe	<pre><Objet> : SQLGereFiche(NomdeFenetre,GereLesMessage)</pre>
Détails 	<p><Objet> : objet SQLManagerX nom de l'objet table SQLManagerX</p> <p><NomFenetre > : nom de la fenetre fiche a gerer le code etant dans la partie intialisation de la fenetre windev ne connaît le nom a ce moement donc on lui fournit</p> <p><Gere les message> : SQLManagerX peut gerer pour vous les messages. Il verifie que tous les champs non null doit etre saisie, et peut si vou smettez oui dans ce parametre, gerer les messages pour vous.</p>
Notes  	<p>Le constructeur de SQLManagerX recherche toutes les variables nécessaires a la gestion de la V5 dans un fichier .ini (SQLMxLangue.ini) ce fichier a plusieurs section permettant d'avoir les différents types de nom pour les actions dans la section [NAVIGATION]et les différents messages suivant la langue : chaque section est [NATION_ plus le n° de langue windev]</p> <p>Ex en francais :</p> <p>[NATION_5]</p> <p>utiliser si la cle primaire est vide lors des insert et update, delete CLEPRIM=CLE Primaire non auto Increment Vide</p> <p>utiliser par la V5 pour prévenir l'utilisateur de la saisie obligatoire des champs le libelle du champs sera inséré entre le DEBUTMESS et FINMESS le libelle du champs ou le nom de la colonne (pour les champs sans libelle sera inséré entre les deux. si le champs n'a pas de libelle alors</p> <p>; le nom de la colonne de la table SQL sera utilisée DEBUTMESS="Attention" FINMESS="doit être renseigné"</p> <p>Confirmation d'une suppression SUPPMESS=Confirmez la suppression de l'éléments ?</p> <p>Affichage de la navigation quant on fait precedent et qu'on est sur la premier fiche ou suivant et qu'on est sur la derniere fiche. Un message peut etre envoye par SQLManagerX</p> <p>DEBUTPARC=Debut de parcour atteint FINPARC=Fin de parcour atteint</p> <p>SI VOUS VOULEZ UTILISER DU CODE EN PLUS :</p> <p>SQLManagerX va gérer un code spécifique, mais vous pouvez quant meme utiliser du code dans les boutons par défaut ce code sera exécuter après les actions de SQLManagerX.</p> <p>Le membre :CodeUtilisateurAvant permet de spécifier que le code du bouton devra etre executer avant le code de SQLManagerX en mettant le booleen a vrai Par exemple dans l'initialisation de la fenetre avec la ligne géant la V5 vous pouvez avoir</p> <pre>client:SQLgereFiche("w_client_fiche",Vrai) client:CodeUtilisateurAvant = vrai</pre> <p>donnera l'ordre a SQLManagerX d'exécuter le code du bouton avant de faire ses actions.</p> <p>ATTENTION NE METTEZ PAS LE CODE D'INITALISATION DU BOOLEEN DANS LE CODE DU BOUTON. L'AFFECTAION NE FONCTIONNERA PAS ET VOTRE CODE SERA EXECUTER APRES</p>

SQLManagerX :: SQLGereFiche()

Notes



SQLManagerX détecte les actions en utilisant le nom des boutons si vous voulez modifier le nom des boutons standard de SQLManagerX il faut remplacer les valeur suivante

vous pouvez appeler votre bouton par exemple BTN_MODIFIER_CLIENT ce bouton sera reconnu et fera l'action modifier de la classe SQLManagerX

[NAVIGATION]

1 =MODIFIER
2 =AJOUTER
3 =MEMO
4 =VALIDER
5 =MODIFIER_MEMO
6 =SUPPRIMER
7 =VISUALISER_MEMO
8 =QUITTER
9 =PREMIER
10=SUIVANT
11=PRECEDENT
12=DERNIER
13=NOUVEAU

les boutons de la fenetre devront contenir ces mot pour que SQLManagerX reconnaisse l'action a executer lors du clic. Le programme exemple de SQLManagerX a une fenetre faites en V5 « **w_client_fiche** »

si vous progammer dans une autre langue il suffit de changer dans la section correspondante le nom de boutons pour qu'ils soient en correspondance avec votre programmation par exemple en anglais :

[NAVIGATION]

1 =MODIFY
2 =ADD
3 =MEMO
4 =VALIDATE
5 =UPDATE_MEMO
6 =DELETE
7 =VIEW_MEMO
8 =CLOSE
9 =FIRST
10=NEXT
11=PREVIOUS
12=LAST
13=NEW