Bienvenue dans le projet MacaronDB.
MacaronDB est une petite librarie écrite en PHP fournissant un jeu de classes destinées à faciliter le travail avec les bases de données DB2 pour IBM i, DB2 Express C, et MySQL 5.
J'utilise cette librarie au sein de projets personnels et professionnels, elle est donc entièrement testée et complètement opérationnelle.
Le projet est publié sous licence New BSD.

Welcome in the MacaronDB project.
MacaronDB is a small library of PHP code which provides a set of classes for working with several databases like DB2 for IBM i, DB2 Express C, and MySQL 5.
I use this library in personal and professional projects, so it is completely tested and functional.
The project is published under the New BSD license.
The documentation of the project is available only in french for the moment.

Sommaire

1. Présentation du projet MacaronDB

1.1 – Introduction

Ce chapitre d'introduction a pour but d'expliquer les raisons d'être de ce projet.

Je m'appelle Grégory Jarrige et suis le développeur de la librairie MacaronDB. Je totalise un peu plus de 20 ans d'expérience dans le développement et la modernisation de logiciels de gestion, principalement sur la plateforme IBM i. Expert Zend PHP 5 certifié, j'axe actuellement mes recherches sur les « design patterns » (PHP et SQL) et les meilleures pratiques en matière de modernisation de systèmes d'information.

Je suis Consultant et « Lead Developer » chez Six-Axe Consultants sur les projets IBM i nécessitant la mise en oeuvre de PHP et/ou l'utilisation de fonctions avancées de DB2 tel que le langage PL/SQL (procédures stockées).

Je forme régulièrement des développeurs IBM i (pour la plupart ayant une solide expérience en RPG ou en Cobol) au langage PHP, et j'ai souvent noté leur frustration de ne pas disposer de librairie simple à mettre en oeuvre, qui pourrait les aider à démarrer rapidement leurs projets PHP sur plateforme IBM i. Quand ils retournent dans leurs entreprises respectives après quelques jours de formation, ils replongent dans les problèmes du quotidien, et ils n'ont tout simplement pas le temps de développer une librairie telle que MacaronDB. Le fait de publier MacaronDB en open source est la manière que j'ai trouvée pour répondre à leur besoin. Je suis néanmoins conscient que MacaronDB – comme toute librairie qui se respecte – nécessite une période d'apprentissage, et peut être complexe à appréhender pour des développeurs qui démarrent leur premier projet. C'est la raison pour laquelle j'ai mis l'accent, dans cette première version de la documentation du projet, sur les fonctionnalités qui me semblent les plus importantes pour démarrer rapidement.

Si vous êtes intéressés par MacaronDB, mais que vous estimez avoir besoin d'un accompagnement pour pouvoir l'utiliser de manière optimale, vous pouvez prendre contact avec Six-Axe Consultants pour bénéficier de journées de transfert de compétences (facturables), journées durant lesquelles je vous fournirai des exemples détaillés d'utilisation, en les adaptant à vos propres besoins. Le nombre de journées qui peuvent être nécessaires pour une bonne prise en main de MacaronDB dépend directement de vos besoins, et de votre niveau d'expertise sur PHP. Cela peut donc aller d'une journée, à quatre ou cinq. Si vous avez déjà suivi une formation sur PHP, ou si vous avez une certaine expérience de ce langage, j'estime qu'une à deux journées peuvent suffire pour que vous puissiez tirer parti des possibilités de cette librairie. Deux à trois jours supplémentaires peuvent être nécessaires pour vous permettre de développer des modules de type CRUD (*), si c'est là votre besoin.

1.2 - Historique et conception du projet

Le projet MacaronDB ne s'est pas fait en un jour, mais est le fruit d'une lente maturation. La première version de ce projet - qui ne s'appelait pas encore "MacaronDB" - se présentait sous la forme d'une classe PHP abstraite, que j'avais appelée PDOWrapper (car elle s'appuyait exclusivement sur PDO). Elle regroupait quelques unes des méthodes essentielles du projet,

telles que "selectOne()", "selectBlock()", etc... Pour développer cette première version, je m'étais inspiré d'un "wrapper" proposé par Jack D. Herrington, dans son livre "PHP Hacks" (éditions O'Reilly). Le "wrapper" proposé par Jack était écrit pour la librarie PearDB, mais son adaptation à PDO a été facile à faire. En travaillant sur mes propres projets, j'ai progressivement enrichi ma classe PDOWrapper, en lui ajoutant de nouvelles méthodes, au fur et à mesure de mes besoins (executeCommand(), export2XML(), etc...). Je n'ai pas publié "PDOWrapper" en open source, faute de temps, et parce que – faute de recul - je n'étais pas sûr que ce projet puisse intéresser d'autres développeurs. La classe PDOWrapper contenait des méthodes statiques qui s'utilisait de la façon suivant :

```
$data = PDOWrapper::selectBlock($connecteur, $requete_sql, $parametres) ;
```

Il est encore possible d'utiliser certaines classes du projet MacaronDB de cette manière, mais l'on verra qu'il est possible d'utiliser une autre manière, qui me semble aujourd'hui plus pratique, au moins pour traiter les opérations courantes.

Lorsque j'ai commencé à utiliser PHP sur des bases de données DB2 (et en particulier DB2 pour IBM i), je me suis rendu compte que disposer d'une librarie équivalente à celle que j'avais écrite pour mes projets personnels, était indispensable. Mais je ne voulais pas utiliser mon propre code, car je pensais - peut être un peu naïvement - que je trouverais un projet open source adapté à mes besoins. J'ai vite déchanté, et ce pour plusieurs raisons :

- si plusieurs librairies orientées « bases de données » proposent des connecteurs pour un certain nombre de bases de données, la base DB2 est très souvent oubliée,

- les rares librairies proposant un support de DB2 proposent un support incomplet, dédié à DB2 pour Windows/Linux, et ne prenant pas en compte les spécificités de DB2 pour IBM i,

- certaines librairies sont encore écrites selon une approche objet correspondant au PHP 4, et je les ai éliminées d'office de ma recherche.

- la plupart des librairies que j'ai étudiées sont souvent brillamment développées. On sent en lisant leur code que les développeurs qui les ont écrites maîtrisent à la perfection certaines techniques avancées de la programmation orientée objet (dont les « design patterns »), mais on a parfois l'impression que leurs concepteurs ne les utilisent pas eux-mêmes, ou qu'ils ne travaillent pas au développement d'applications d'entreprise avec ces librairies, car il y manque souvent la plupart des fonctionnalités dont j'estime avoir besoin pour développer des applications « métier ». Nous y reviendrons au travers des différents exemples que je propose dans la seconde partie de ce document.

Déçu par le résultat de mes recherches, et après réflexion, j'ai décidé d'adapter ma classe PDOWrapper à DB2. Or tout allait bien tant que j'utilisais MacaronDB sur un Zend Server pour Windows, en attaquant la base de données DB2 pour IBM i via PDO et le "iSeries Access ODBC Driver". Par contre cela s'est corsé quand j'ai voulu porter MacaronDB sur un Zend Server pour IBM i. Car j'ai eu la désagréable surprise de constater que l'implémentation de PDO pour DB2 sur serveur IBM i était incomplète, et donc inutilisable. J'ai alors révisé l'architecture de MacaronDB de manière à ne plus être exclusivement dépendant de PDO, ce qui m'a conduit à créer un jeu de classes complémentaires adaptées à l'extension "ibm_db2".

Par ailleurs, il m'est apparu nécessaire de ne plus travailler exclusivement avec une classe abstraite et un jeu de méthodes statiques. Travailler avec une classe instanciable, plutôt qu'avec une classe abstraite, offre quelques avantages indéniables qu'il serait trop long de développer ici. Mais cela m'a conduit à ajouter une nouvelle classe à MacaronDB, classe que j'ai appelée "DBInstance". Grâce à cette nouvelle classe, l'exemple de tout à l'heure peut s'écrire de la façon suivante :

```
$data = $connecteur->selectBlock( $requete_sql, $parametres ) ;
```

C'est cette méthode que je décrirai en détails dans les exemples de la seconde partie de cette documentation.

1.3 - Evolution du projet

En environnement IBM i, MacaronDB m'a permis de répondre jusqu'ici très efficacement à toutes les problématiques que j'avais à traiter, tels que :

- développement d'écrans de consultation et de recherche, y compris sur de gros volumes de données,
- développement d'écrans de gestion de type CRUD (*). Le développement d'écrans de type CRUD m'a conduit à incorporer une implémentation du design pattern "Active Record", implémentation que j'avais initialement écrite pour mon vieux PDOWrapper, et que j'ai légèrement adaptée pour prendre en compte certaines spécificités de DB2. Je reviendrai dans la documentation sur la manière d'utiliser le design pattern "Active Record".

Si pour l'instant MacaronDB ne sait adresser que les bases de données DB2 et MySQL, j'envisage de l'ouvrir à d'autres bases de données dans un avenir... je l'espère assez proche.

MacaronDB est l'un de mes outils de travail quotidiens. J'insiste sur ce point, car je crois que c'est important : je suis développeur mais aussi utilisateur du projet. Cela me permet de connaître les forces et les faiblesses du produit, et de le faire évoluer en fonction de mes besoins. En tant qu'utilisateur du projet, je suis particulièrement vigilant sur 2 points :

- corriger rapidement les bugs que je peux identifier.
- maintenir une compatibilité ascendante entre les différentes versions du produit, ceci afin d'éviter tout risque de régression par rapport aux projets utilisant MacaronDB que j'ai développés jusqu'ici. Je ne me lancerai donc pas dans un projet de réécriture qui remettrait en cause la manière d'utiliser les classes et méthodes du projet. Si vous avez besoin de composants stables pouvant servir de socle pour vos applications, MacaronDB est peut être l'un de ces composants.

MacaronDB est un projet open source publié sous licence New BSD. Vous pouvez donc le télécharger librement et l'utiliser dans vos propres projets, sous réserve de bien respecter les termes de la licence New BSD (**)

AVERTISSEMENT : j'utiliserai dans cette documentation le terme "Zend Server" pour désigner l'environnement d'exécution de PHP, car c'est l'environnement PHP que j'utilise dans les projets d'entreprise sur lesquels je travaille au quotidien. Je vous recommande d'ailleurs de l'utiliser, car il est très complet, facile à installer. Ce projet est développé par la société Zend, dont la société Six-Axe Consultants est partenaire. Il existe une version gratuite de Zend Server (la version "Community Edition") qui est parfaite pour se "faire la main" (il ne lui manque que quelques fonctions qui ne sont pas indispensables pour un simple environnement de développement). La version de Zend Server dédiée à la plateforme IBM i est le seul « stack » PHP opérationnel pour cette plateforme. Si vous ne travaillez pas en environnement IBM i, et que vous avez déjà vos habitudes avec un environnement LAMP ou WAMP différent, "macaronDB" fonctionnera très bien, à condition que vous ayiez pris soin d'activer les extensions bases de données que vous souhaiter utiliser via MacaronDB (sujet sur lequel nous reviendrons ultérieurement).

Pour tous renseignements, vous pouvez me contacter à l'adresse email suivante :

gjarrige ((at)) six-axe.fr

Pour tous renseignements sur la société Six-Axe Consultants, je vous invite à visite le site :

http://www.six-axe.fr

Pour tous renseignements sur la société Zend et les produits qu'elle commercialise (Zend Server, Zend Studio, etc…), je vous invite à visiter le site :

http://www.zend.com/fr/

(*) CRUD (Create Retrieve Update Delete) : désigne les 4 opérations de base d'un module de gestion classique. Par extension, désigne tout module offrant les fonctions de recherche, consultation, modification, création et suppression. Avec l'aide de MacaronDB, le développement d'un CRUD de gestion d'une table SQL composée d'une dizaine de colonnes peut se faire en moins d'une heure.

(**) Pour comprendre le fonctionnement de la licence "New BSD", je vous invite à lire la définition du terme "licence BSD" dans Wikipédia.

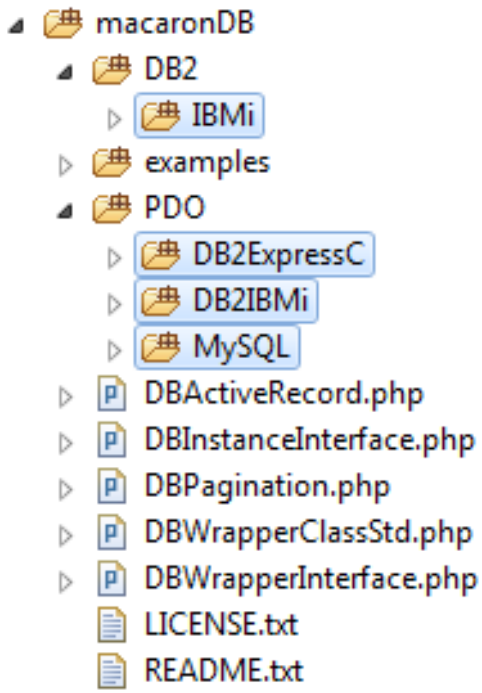
2. Mode d'emploi du projet MacaronDB

2.1 - Liste des connecteurs bases de données

MacaronDB propose plusieurs connecteurs bases de données adaptés à différents contextes d'exécution :

- si votre projet doit s'exécuter sur un Zend Server pour IBM i, en utilisant la base de données DB2 pour IBM i, alors vous utiliserez le connecteur DB2_IBMi
- si votre projet doit s'exécuter sur un Zend Server pour Windows/Linux, en utilisant une base de données DB2 hébergée sur un serveur IBM i, alors vous utiliserez le connecteur PDO_DB2IBMi. Cela présuppose que vous ayiez installé sur votre serveur Windows ou Linux le logiciel "iSeries Navigator " d'IBM (ou « System i Navigator » selon les versions), car le connecteur s'appuie sur le "Iseries Access ODBC Driver", qui est fourni en standard avec le logiciel "iSeries Navigator".
- si votre projet doit s'exécuter sur un Zend Server pour Windows/Linux, en utilisant une base de données DB2 Express C, alors vous utiliserez le connecteur PDO_DB2ExpressC.
- si votre projet doit s'exécuter sur un Zend Server (peu importe la plateforme), en utilisant la base de données MySQL, alors vous utiliserez le connecteur PDO_MySQL.

Si l'on ouvre le répertoire de MacaronDB via l'explorateur de Zend Studio par exemple, les connecteurs précités apparaissent clairement dans l'arborescence du projet :



2.2 - Définition d'une connexion

On place généralement la déclaration du (ou des) connecteur(s) bases de données dans un script particulier qui sera appelé au démarrage de l'application. On parle souvent de « script d'amorçage » ou de « script de configuration » pour désigner ce type de script. Le script de configuration sera appelé par les autres scripts de l'application via la fonction PHP require_once(), pour éviter de rappeler plusieurs fois ce script par erreur.

On commence par regarder si la variable $cnx_db01 n'a pas été déjà déclarée. Ce test n'est pas absolument indispensable, surtout si vous avez pris soin de bien charger ce script via la fonction require_once(). On initialise ensuite un tableau $options contenant quelques directives propres à la base DB2 pour IBM i, tel que le mode de « nommage » utilisé par les requêtes (qui peut être « système » ou « SQL »), la liste des bibliothèques où se situent les objets DB2 utilisés par l'application, ainsi que le mode de retour des noms de colonnes renvoyés par SQL dans les jeux de données (result sets). Dans l'exemple, les noms des colonnes seront renvoyés systématiquement en majuscule.

Le test surligné en jaune permet d'identifier « à la volée » si l'application s'exécute sur un serveur IBM i ou pas, on adapte donc le choix du connecteur au contexte d'exécution.

Je vous propose de lire le code, puis les explications complémentaires qui suivent :

```
if (! isset ( $cnx_db01 )) {
	/*
	 * Tableau des options de configuration de la base DB2
	 */
	$options = array ();
	$options['i5_naming'] = true ;
	$options['i5_libl'] = array('mabib1', 'mabib2', 'mabib3') ;
	$options['DB2_ATTR_CASE'] = 'UPPER'  ;

	/*
	 * Si la plateforme est de type IBMi alors on se connecte à la base de
	 * données avec DB2_connect,
	 * sinon on se connecte à la base de données avec PDO
	 */
	if (php_uname('s') == 'OS400' || PHP_OS == "AIX" || PHP_OS == "OS400") {

		/*
		 * Ouverture d'une connexion BD sur un serveur IBM i, avec
		 * DB2 Connect
		 */
		require_once 'DB2/IBMi/DBWrapper.php';
		require_once 'DB2/IBMi/DBConnex.php';
		require_once 'DB2/IBMi/DBInstance.php';
		$cnx_db01 = new DB2_IBMi_DBInstance('*LOCAL', '', '', $options ) ;
	} else {
		/*
		 * Connexion BD sur un serveur Windows ou Linux avec PDO
		 */
		require_once 'PDO/DB2IBMi/DBWrapper.php';
		require_once 'PDO/DB2IBMi/DBConnex.php';
		require_once 'PDO/DB2IBMi/DBInstance.php';
		/*
```

```
             * définition de l'adresse IP, du profil et du mot de passe
             * de connexion au serveur IBM i
             */
            $ipa = 'adresse IP de votre serveur IBM I' ;
            $usr = 'votre profil' ;
            $pwd = 'mot de passe correspondant au profil' ;
            $cnx_db01 = new PDO_DB2IBMi_DBInstance($ipa, $usr, $pwd, $options ) ;
        }
}
```

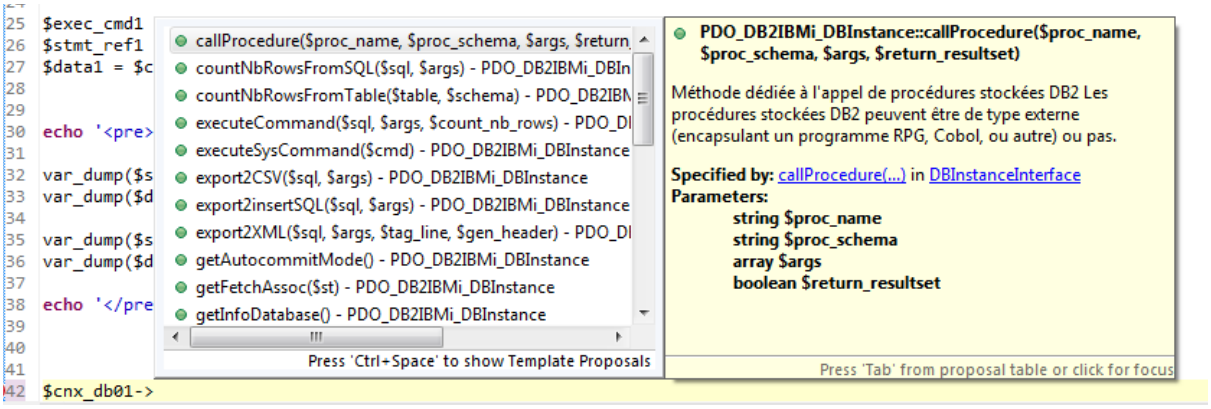
Dans l'exemple qui précède, j'ai choisi de montrer une méthode permettant de rendre votre code PHP indépendant de la plateforme d'exécution. En effet, si vous souhaitez que votre application puisse s'exécuter tantôt sur un Zend Server pour Windows, tantôt sur un Zend Server pour IBM i, et ce en modifiant le moins de code possible, alors vous pouvez le faire via le code de la page précédente.

En revanche, si votre application est destinée à ne « tourner » que sur un serveur IBM i, alors vous pouvez ne conserver qu'une partie du code précédent, soit :

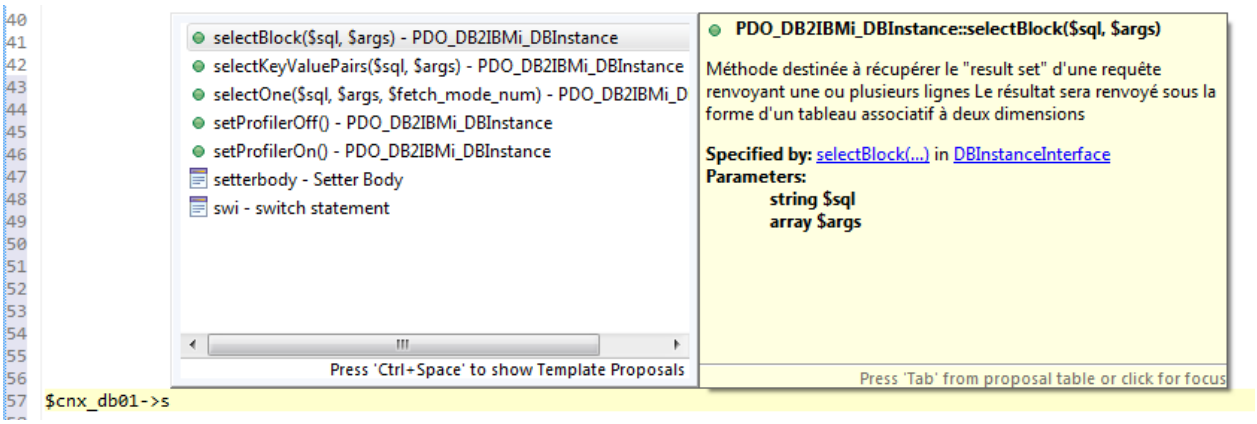
```
$options = array ();
$options['i5_naming'] = true ;
$options['i5_libl'] = array('mabib1', 'mabib2', 'mabib3') ;
$options['DB2_ATTR_CASE'] = 'UPPER'  ;
require_once 'DB2/IBMi/DBWrapper.php';
require_once 'DB2/IBMi/DBConnex.php';
require_once 'DB2/IBMi/DBInstance.php';
$cnx_db01 = new DB2_IBMi_DBInstance('*LOCAL', '', '', $options ) ;
```

A partir du moment où votre connexion est établie, vous pouvez bénéficier du mécanisme d'auto-complétion de Zend Studio, ce qui rend l'utilisation de MacaronDB simple et conviviale.

Exemple d'auto-complétion :



Autre exemple, après avoir tapé la touche « s » :



L'auto-complétion permet de connaître la liste des méthodes disponibles, et pour chaque méthode, affiche dans le cadre de droite un court descriptif sur l'utilisation de la méthode sélectionnée.

Les différentes méthodes à votre disposition sont les suivantes :

```
function selectOne($sql, $args = array(), $fetch_mode_num = false);

function selectBlock($sql, $args = array());

function selectKeyValuePairs($sql, $args = array());

function executeCommand($sql, $args = array(), $count_nb_rows = true);

function executeSysCommand ($cmd) ;

function callProcedure($proc_name, $proc_schema, &$args = array(),
          $return_resultset = false);

function getStatement($sql, $args = array());

function getFetchAssoc($st);

function getPagination($sql, $args, $offset, $nbl_by_page, $order_by = '');

function getScrollCursor($sql, $args, $offset, $nbl_by_page, $order_by = '' ) ;

function export2CSV($sql, $args = array());

function export2XML($sql, $args = array(), $tag_line = '', $gen_header=true) ;

function export2insertSQL($sql, $args = array());

function getLastInsertId($sequence = '');

function valueIsExisting($table, $nomcol, $valcol, $where_optionnel = '');

function valueIsExistingOnOtherRecord($table, $nomcol, $valcol, $idencours,
          $where_optionnel = '');
```

```
function getInfoDatabase();

function countNbRowsFromTable($table, $schema = '');

function countNbRowsFromSQL($sql, $args = array());
```

Pour pouvoir incorporer cette librairie dans vos projets, il vous suffit de télécharger et dézipper le répertoire "macaronDB", et de l'installer dans l'un des sous-répertoires de votre projet. Par exemple, si l'on considère que votre application utilise l'arborescence suivante :

```
/votreprojet/
/votreprojet/inc/
/votreprojet/inc/class/
/votreprojet/inc/macaronDB/
/votreprojet/inc/model/
etc…
```

Une solution relativement simple à mettre en oeuvre consiste à définir le chemin d'accès vers les répertoires « stratégiques » de votre application (dont celui de MacaronDB). Ce chemin d'accès sera utilisé par l'interpréteur PHP à chaque fois que votre application devra incorporer des scripts via les méthodes « require() », « require_once() », « include() » ou « include_once() ».

La définition du chemin d'accès doit se faire de préférence à l'intérieur d'un script de configuration (ou d'amorçage), dans lequel vous prendrez soin d'insérer les lignes suivantes :

```
// stockage du Path standard, en supposant que le script courant se trouve dans le répertoire « inc »
define('APP_PATH_STD', realpath(dirname(dirname(__FILE__))));

// directory setup and class loading
set_include_path('.'
                . PATH_SEPARATOR . APP_PATH_STD
                . PATH_SEPARATOR . APP_PATH_STD . DIRECTORY_SEPARATOR. 'inc' .DIRECTORY_SEPARATOR
                . PATH_SEPARATOR . APP_PATH_STD . DIRECTORY_SEPARATOR. 'inc' .DIRECTORY_SEPARATOR .
                  'class'. DIRECTORY_SEPARATOR
                . PATH_SEPARATOR . APP_PATH_STD . DIRECTORY_SEPARATOR. 'inc' .DIRECTORY_SEPARATOR .
                  'macaronDB'. DIRECTORY_SEPARATOR
                . PATH_SEPARATOR . APP_PATH_STD . DIRECTORY_SEPARATOR. 'inc' .DIRECTORY_SEPARATOR .
                  'model'. DIRECTORY_SEPARATOR
                . PATH_SEPARATOR . APP_PATH_STD . DIRECTORY_SEPARATOR. 'pear' .DIRECTORY_SEPARATOR
                . PATH_SEPARATOR . get_include_path());
```

2.3 - Exécution de requêtes SQL

Pour exécuter une requête SQL ne renvoyant qu'une seule ligne, vous pouvez écrire ceci :

```
$sql = 'select code, description from mytable where code1 = ?
        and code2 = ?';
$params = array ($value1, $value2 ) ;

$data = $cnx_db01->selectOne ( $sql, $params);
```

Dans l'exemple ci-dessus, si la requête $sql a abouti, alors la variable $data contient un tableau associatif à une dimension contenant les données retournées par SQL. En revanche, si la requête n'a pas abouti, la variable $data contiendra le booléen "false".

Pour information, les 2 paramètres de la méthode selectOne() sont la requête SQL à exécuter, et un tableau PHP optionnel contenant les valeurs qui vont se substituer aux points d'interrogation dans la requête SQL. La méthode selectOne() exécute la requête en s'appuyant sur les méthode prepare() et execute() du connecteur base de données sous-jacent.

Il est important de noter que toutes les requêtes exécutées par les méthodes de MacaronDB sont exécutées selon ce principe, qui garantit la meilleure sécurité possible contre les attaques dites par "injection SQL". Je vous déconseille formellement de créer vos requêtes SQL en concaténant les éléments variables de vos clauses WHERE avec le code de la requête. Ce faisant, vous rendriez vos requêtes vulnérables aux attaques dites "par injection SQL".

Si vous souhaitez récupérer un tableau avec des postes numérotés (plutôt qu'un tableau associatif), alors il vous suffit d'ajouter le booléen "true" en troisième paramètre (optionnel) de la méthode selectOne(). Exemple :

```
$data = $cnx_db01->selectOne ( $sql, array ($value1, $value2 ), true );
```

Si votre requête est susceptible de renvoyer de 1 à X lignes, alors vous devez utiliser la méthode « selectBlock » (plutôt que « selectOne »). Exemple :

```
$sql = 'select code, description from mytable where code1 = ?
        and code2 = ?';

$data = $cnx_db01->selectBlock ( $sql, array ($value1, $value2 ) );
```

Dans l'exemple ci-dessus, la variable $data contiendra – si tout s'est bien passé - un tableau associatif à 2 dimensions (que vous pourrez « parcourir » via un foreach par exemple). Si en revanche la requête a échoué, alors $data contiendra « false ».

Si vous souhaitez récupérer un jeu de données sous la forme d'un tableau à deux dimensions, facile à utiliser pour la génération de champs de formulaire de type "select" (liste déroulante), alors vous apprécierez sûrement la méthode selectKeyValuePairs() qui s'utilise de la façon suivante :

```
$sql = 'select code, description from mytable';

$data = $cnx_db01->selectKeyValuePairs( $sql );
```

Si vous souhaitez exécuter une requête de type INSERT, UPDATE ou DELETE, alors vous devez recourir aux services de la méthode executeCommand(), qui fonctionne sur le même principe que les méthodes vues précédemment. Exemple :

```
$sql = 'update mytable set col1 = ? where id = ?';
```

```
$data = $cnx_db01->executeCommand( $sql, array($value, $id) );
```

Si vous souhaitez exécuter une commande système IBM i, par exemple pour déclencher un DSPPGMREF destiné à alimenter une table DB2, vous devez utiliser la méthode executeSysCommand(). Exemple :

```
$cmd = 'DSPPGMREF …';

$data = $cnx_db01->executeSysCommand( $cmd );
```

Les méthodes “export2CSV()”, “export2XML()” et “export2insertSQL()” permettent de générer très simplement des données aux formats indiqués dans le nom des méthodes. Ces méthodes sont déjà opérationnelles, mais elles sont adaptées au traitement de volumes de données limités. Pour traiter de gros volumes de données, on pourra s’inspirer du code utilisé dans ces méthodes, mais il sera certainement souhaitable d’écrire des scripts PHP dédiés écrivant directement sur disque pour éviter de surcharger la mémoire du serveur d’exécution. La méthode “export2insertSQL()” peut présenter quelques faiblesses si elle est exécutée via le connecteur PDO car dans ce cas les données exportées dans le script SQL sont toutes encadrées par des apostrophes, même si elles sont numériques. Cette faiblesse sera corrigée dans une prochaine version du projet MacaronDB.

2.4 – Appel de procédures stockées

Les procédures stockées DB2 se classent en 2 catégories :

- Les procédures « full SQL » écrites en PL/SQL
- Les procédures « externes », servant d’intermédiaire pour l’appel de programme écrits en langage RPG, CL, Cobol, Java…

Nous allons en revanche nous attarder sur les procédures stockées externes, car elles constituent la voie royale pour rendre accessibles aux applications web, vos composants « métier », souvent riches et complexes écrits en RPG ou Cobol.

La documentation de PHP propose des exemples d’utilisation assez complets :

- Pour « ibm_db2 » : http://fr.php.net/manual/fr/function.db2-bind-param.php
- Pour PDO : http://fr.php.net/manual/fr/pdo.prepared-statements.php

Comme point de départ pour tester cette fonctionnalité, il nous faut une procédure stockée, aussi je vous propose de créer un programme RPG que nous appellerons PGMREFPRC – que nous encapsulerons ensuite dans une procédure stockées externe - et dont voici le code source :

```
 ******************************************************
 * DESCRIPTION DSPPGMREF d'un pgm reçu en paramètre
 * NOM DU PROGRAMME PGMREFPRC
 ******************************************************
h usrprf(*owner) datfmt(*iso)
d Requete         s            300
d Commande        s            200
d PGMREF          PR                  Extpgm('QCMDEXC')
d  String                     1000    Const
d                                     Options(*Varsize)
d  Len                         15P 5 Const
c     *entry        plist
c                   parm                    CODBIB           10
c                   parm                    CODPGM           10
/free
    Commande = 'DSPPGMREF PGM(' + %trim(CODBIB) + '/' + %trim(CODPGM)
    + ') OUTPUT(*OUTFILE) OBJTYPE(*ALL)'
    + ' OUTFILE(QTEMP/PGMREF1) OUTMBR(*FIRST *REPLACE) ' ;
    CALLP(E) PGMREF(Commande:%len(Commande));

    if not(%error()) ;
      Requete = 'SELECT WHFNAM, WHLNAM, WHSNAM, WHRFNO, '
        + ' WHFUSG, WHRFNM, WHRFSN, WHRFFN, WHOBJT '
        + ' FROM QTEMP/PGMREF1 WHERE WHOBJT = ''F''' ;
      EXEC SQL
        PREPARE REQ1 FROM :Requete ;
      EXEC SQL
        DECLARE C1 CURSOR FOR REQ1 ;
      EXEC SQL
        OPEN C1 ;
      EXEC SQL
        SET RESULT SETS CURSOR C1 ;
   endif ;
   *InLR = *On;
 /End-Free
```

Explications : Ce programme reçoit un nom de programme en paramètre, effectue un DSPPGMREF sur ce programme pour générer une table temporaire, puis exploite cette table temporaire via SQL pour en extraire un jeu de données qu'il va renvoyer en sortie, sous la forme d'un « result set » directement exploitable par la procédure stockée appelante. On pouvait faire plus simple, mais le mélange des techniques est ici intéressant d'un point de vue pédagogique.

Maintenant que nous avons notre programme RPG, nous pouvons l'encapsuler dans une procédure stockées externe, dont le code source SQL de création est le suivant :

```
CREATE PROCEDURE YOURLIB/PGMREFPRCS (
IN CODBIB CHAR(10) ,
IN CODPGM CHAR(10) )
DYNAMIC RESULT SETS 1
LANGUAGE RPGLE
SPECIFIC YOURLIB/PGMREFPRCS
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
EXTERNAL NAME 'YOURLIB/PGMREFPRC'
```

```
PARAMETER STYLE SQL ;

COMMENT ON SPECIFIC PROCEDURE YOURLIB/PGMREFPRCS IS 'Appel pgm PGMREFPRC' ;
```

A noter : le logiciel « System i Navigator » d'IBM fournit d'excellents assistants pour créer des procédures stockées « full SQL » ainsi que des procédures stockées externes telle que celle ci-dessus.

Il nous reste à écrire un peu de code pour afficher le contenu du « result set » renvoyé par la procédure stockée :

```
// paramètres d'appel de la procédure cataloguée
echo 'Liste des fichiers utilis&eacute;s par le programme : ' . trim ( $codbib ) .
'/' . trim ( $codpgm ) . '<p/>';
echo <<<EOM
<table border="1" width="100%" cellspacing="0" cellpadding="5" >
<tr class="header-row">
<td>WHFNAM</td>
<td>WHLNAM</td>
<td>WHSNAM</td>
<td>WHRFNO</td>
<td>WHFUSG</td>
<td>WHRFNM</td>
<td>WHRFSN</td>
<td>WHRFFN</td>
<td>WHOBJT</td>
</tr>
EOM;
echo '<table>'.PHP_EOL ;
// requête d'appel de la procédure stockée
$parms = array($codbib, $codpgm) ;
$datas = $cnx_db01->callProcedure ('PGMREFPRCS', 'M3DHSAPPGM', $parms, true );
foreach ($datas as $data) {
      echo <<<EOM
<tr>
<td>{$data['WHFNAM']}</td>
<td>{$data['WHLNAM']}</td>
<td>{$data['WHSNAM']}</td>
<td align="right">{$data['WHRFNO']}</td>
<td align="right">{$data['WHFUSG']}</td>
<td>{$data['WHRFNM']}</td>
<td>{$data['WHRFSN']}</td>
<td align="right">{$data['WHRFFN']}</td>
<td align="center">{$data['WHOBJT']}</td>
</tr>
EOM;

}
echo '</table>'.PHP_EOL ;
```

2.5 – Techniques de pagination

Deux méthodes sont à votre disposition pour gérer la pagination sur une requête SQL :

- getPagination() : pagination pilotée par une technique "full SQL"

- getScrollCursor() : pagination pilotée par la technique dite du « scroll cursor »

Les deux méthodes produisent des résultats sensiblement équivalents en termes de performances, mais il semble que les performances se dégradent avec la technique « full SQL », lorsque l'on pagine sur une table SQL de grande taille.

Les techniques de pagination nécessitent, pour être correctement présentées, de faire appel à des notions d'architecture applicative, notamment en ce qui concerne la manière de faire interagir champs de recherche, gestion de l'offset, ainsi que la gestion de la barre de pagination qui accompagne inévitablement cette technique. Ces explications pourront être fournies aux personnes intéressées sous la forme de sessions de transfert de compétences, sessions durant lesquelles je vous fournirai des exemples complètement fonctionnels, et vous expliquerai comment les adapter à vos besoins.

Si vous vous sentez d'attaque pour vous lancer sans attendre, voici un exemple de script PHP que vous pouvez utiliser comme base de travail pour gérer vos listes paginées :

```
$max_lines_by_page = 10 ; // nombre de lignes maxi par page
$nb_lignes_total = $cnx_db01->countNbRowsFromSQL ( $sql_query, array() );

$offset = intval($_GET['offset']) ;
if ($offset <= 0 || $offset > 1000000) {
	$offset = 1 ;
}

$params = array() ; // paramètres conservés pour transmission à la fonction de pagination
$params ['offset'] = $offset;

if (is_null($nb_lignes_total)) {
	echo 'ERREUR : requête inopérante<br/>';
} else {
	if ($nb_lignes_total <= 0) {
		echo 'La requête fonctionne mais ne renvoie aucune donnée<br/>';
	} else {
		$datas = $cnx_db01->getScrollCursor( $sql_query, array(), $offset,
				$max_lines_by_page );
		if (is_array($datas) && count($datas)>0) {
			echo '<table border="1" width="100%" cellspacing="0" cellpadding="5" >';
			echo '<tr class="header-row">';
			foreach ($datas[0] as $key=>$value) {
				echo '<td>'.trim($key).'</td>';
			}
			echo '</tr>'.PHP_EOL ;

			foreach ( $datas as $data ) {
				echo '<tr>';
				foreach ($data as $key=>$value) {
					if (is_int($value) || is_float($value)) {
						// cet alignement ne sera pas pris en compte avec PDO
						// qui renvoie systématiquement des données de type
						// string, par contre il fonctionnera avec DB2_Connect
						// qui renvoie des données correctement typées
						$align = 'right';
					} else {
						$align = 'left';
```

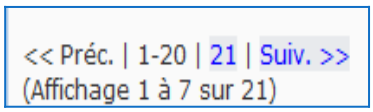
```
                                $value = trim($value) ;
                            }
                            echo '<td align="'.$align.'">'.$value.'</td>';
                        }
                        echo '</tr>';
                    }
                    echo '</table>';
                }
                echo '<br/>';
                // Appel de la fonction générant la barre de pagination
                DBPagination::pcIndexedLinks ( $nb_lignes_total, $offset,
                   $max_lines_by_page, $_SERVER ['PHP_SELF'], $params );
                echo '<br/>';
                echo "(Affichage " . $offset . " à " . ($offset + $lastRowNumber - 1)
                    . " sur " . $nb_lignes_total . ")";
        }
}
```

Exemple de pagination effectuée sur une table DB2, en l'occurrence il s'agit de la table système QSYS2/SYSTABLES :

TABLE_SCHEMA	TABLE_NAME	TABLE_TEXT
QSYS2	AUTHORIZATIONS	
QSYS2	CATALOG_NAME	
QSYS2	CHARACTER_SETS	
QSYS2	CHARACTER_SETS_S	
QSYS2	CHECK_CONSTRAINTS	
QSYS2	COLUMNS	
QSYS2	COLUMNS_S	
QSYS2	CONDENSEDINDEXADVICE	
QSYS2	DBMON_QUERIES	
QSYS2	INFORMATION_SCHEMA_CATALOG_NAME	
QSYS2	PARAMETERS	
QSYS2	PARAMETERS_S	
QSYS2	PROCEDURES	
QSYS2	QASQSRC	QSYS2/QASQRESL LF DDS
QSYS2	QASQTYPE	QASQTYPE

<< Préc. | 1-15 | 16-30 | 31-45 | 46-60 | ... | 106-116 | Suiv. >>
(Affichage 1 à 15 sur 116)

A noter : la classe DBPagination qui est fournie dans le projet MacaronDB, permet de générer très simplement une barre de pagination telle que celle présentée dans les exemples ci-dessus et ci-dessous :



2.6 – Gestion des erreurs

Si une requête n'aboutit pas suite à une erreur, la log des erreurs de PHP est automatiquement alimentée avec différentes informations qui sont précieuses pour le débogage. Ces informations sont les suivantes :

- le code et le message d'erreur renvoyés par le connecteur base de données
- la requête SQL en erreur
- le tableau des arguments transmis à la requête (

```
[6-Apr-2012 10:23:58] PHP Warning:  db2_prepare() [<a href='function.db2-
prepare'>function.db2-prepare</a>]: Statement Prepare Failed in
/www/zendsvr/htdocs/HermesDB/inc/macaronDB/DB2/IBMi/DBWrapper.php on line 61
[6-Apr-2012 10:23:59] code SQL   -> SELECT COUNT(*) AS NB_ROWS FROM (SELECT A.TABLE_SCHEMA,
A.TABLE_NAME, A.TABLE_TEXT
 FROM QSYS2.SYSTABLESx A where A.TABLE_SCHEMA = 'QSYS2') AS FOO
[6-Apr-2012 10:23:59] getMessage -> Erreur sur selectOne/db2_prepare :
DB2 Error Code : 42704 //
DB2 Error Msg : SYSTABLESX de type *FILE dans QSYS2 non trouvé. SQLCODE=-204 //

[6-Apr-2012 10:23:59] arguments -> array(0) {
}
```

L'emplacement du fichier de log est différente selon la version du Zend Server utilisée :

- sur IBM i :
  - /usr/local/zendsvr/var/log/php.log
- sur Windows :
  - C:\Program Files (x86)\Zend\ZendServer\logs\php\php_error.log

Il est également possible de consulter cette log directement à partir du tableau de bord du Zend Server, mais la lecture de la log n'est pas toujours facile dans ce mode d'affichage.

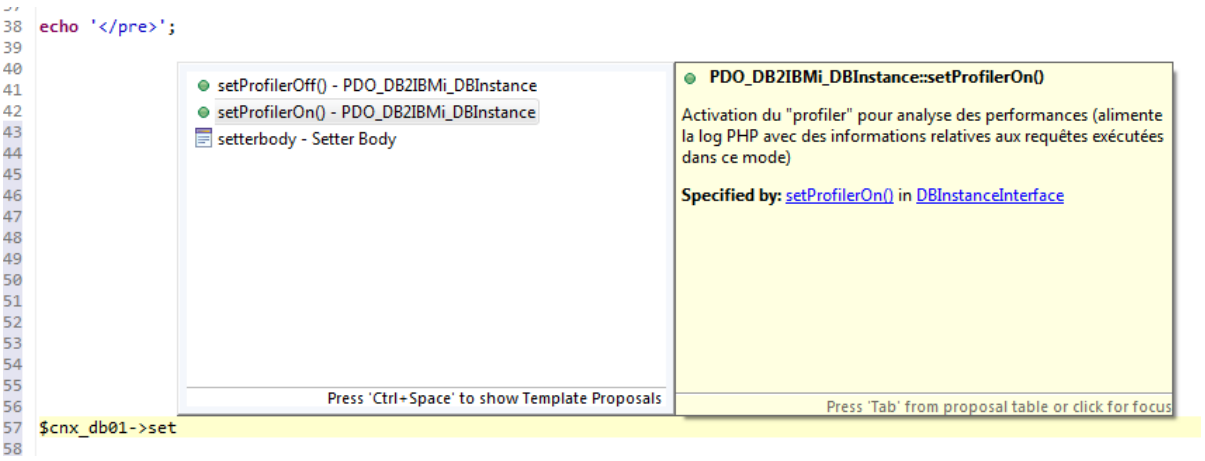
Si l'erreur est déclenchée sous PDO, la log est alimentée avec des informations équivalentes, même si elles sont présentées différemment :

```
[6-Apr-2012 08:32:57 UTC] array(7) {
  ["SQL_code"]=>
  string(147) "SELECT COUNT(*) AS NB_ROWS FROM (SELECT A.TABLE_SCHEMA, A.TABLE_NAME,
A.TABLE_TEXT
 FROM QSYS2/SYSTABLESx A where A.TABLE_SCHEMA = 'QSYS2') AS FOO"

 ["Message"]=>
  string(205) "SQLSTATE[42S02]: Base table or view not found: 0 [IBM][Pilote ODBC System i
Access][DB2 for i5/OS]SQL0204 - SYSTABLESX de type *FILE dans QSYS2 non trouvé.
(SQLPrepare[0] at ext\pdo_odbc\odbc_driver.c:206)"

  ["Trace"]=>
  string(703) "#0 C:\Program Files (x86)\Zend\Apache2\htdocs\DBToolbox\inc\macaronDB\
DBWrapperClassStd.php(55): PDO->prepare('SELECT COUNT(*)...')
```

```
#1 C:\Program Files (x86)\Zend\Apache2\htdocs\DBToolbox\inc\macaronDB\
DBWrapperClassStd.php(601): DBWrapperClassStd::selectOne(Object(PDO_DB2IBMi_DBInstance),
'SELECT COUNT(*)...', Array, true)
#2 C:\Program Files (x86)\Zend\Apache2\htdocs\DBToolbox\inc\macaronDB\PDO\DB2IBMi\
DBInstance.php(149): DBWrapperClassStd::countNbRowsFromSQL(Object(PDO_DB2IBMi_DBInstance),
'SELECT A.TABLE_...', Array)
#3 C:\Program Files (x86)\Zend\Apache2\htdocs\DBToolbox\application\
dbrequet_saisie.php(149): PDO_DB2IBMi_DBInstance->countNbRowsFromSQL('SELECT A.TABLE_...',
Array)
#4 {main}"

  ["Code"]=>
  string(5) "42S02"

  ["File"]=>
  string(88) "C:\Program Files (x86)\Zend\Apache2\htdocs\DBToolbox\inc\macaronDB\
DBWrapperClassStd.php"

  ["Line"]=>
  int(55)

  ["Arguments"]=>
  string(0) ""
}
```

2.7 – Analyse des performances

Il est possible de mesurer le temps d'exécution d'une requête SQL, en activant le profiler via la méthode setProfilerOn() :



La désactivation se fait au moyen de la méthode setProfilerOff().

L'activation du profiler a pour effet d'envoyer – pour toutes les requêtes exécutées – des informations dans la log telles que :

```
[6-Apr-2012 08:39:24 UTC] array(6) {
  ["Profiler type"]=>
  string(3) "SQL"

  ["time"]=>
  float(0.1254551410675)

  ["SQL"]=>
  string(105) "SELECT A.TABLE_SCHEMA, A.TABLE_NAME, A.TABLE_TEXT
 FROM QSYS2/SYSTABLES A where A.TABLE_SCHEMA = 'QSYS2'"

  ["Arguments"]=>
  array(0) {
  }

  ["nb_rows"]=>
  int(20)

  ["Status"]=>
  string(4) "good"
}
```

Attention : Ne pas oublier de supprimer l'appel de la méthode setProfilerOn(), une fois que vous n'avez plus besoin de mesurer les performances de vos requêtes SQL.

2.8 – Techniques complémentaires

Toutes les méthodes proposées par MacaronDB n'ont pas été intégralement décrites dans cette première version de la documentation de MacaronDB. Cette documentation va évoluer rapidement pour présenter les fonctioonalités du projet de manière plus exhaustive. Mais les explications présentées dans ce document devraient vous permettre de tirer parti de MacaronDB rapidement (du moins je l'espère).

Un des points importants qui n'ont pas présentés ici se situe au niveau de la classe ActiveRecord (dérivée du « design pattern » de même nom), qui est destinée à faciliter le développement de modules de mise à jour. Mais l'utilisation de cette classe nécessite des connaissances relativement approfondies en programmation objet, et c'est un sujet qu'il est beaucoup plus facile d'aborder lors de sessions de formation, ou de sessions de transfert de compétences. J'essaierai néanmoins de présenter les fonctionnalités de cette classe dans une prochaine version de cette documentation.