Document ORM Mélanie2 PHP

Ce document décrit le fonctionnement et la mise en place de l'ORM Mélanie2 en PHP.

Sommaire

Versions

V0.1	Présentation de la librairie ORM Mélanie2
V0.2	Ajout de nouvelles informations sur le fonctionnement de l'ORM
V0.3	Corrections
V0.4	Modification pour diffusion publique de l'ORM
V0.5	Ajout du paragraphe sur la mise en place côté Base de données
V0.6	Ajout de l'installation par composer Mise à jour des nouveaux éléments

Présentation

Définition ORM

Un ORM (object-relationnal mapping) va permettre de faire le lien entre une base de données et des objets. Ces objets seront formatés de façon à être facilement exploitable par les applications.

Notre librairie est un ORM écrit en PHP 5 Objet qui permet le mapping de la base de données Horde Mélanie2.

Intérêt de cette librairie pour Mélanie2

La base de données Mélanie2 (Horde) a un schéma très spécifique qui n'a pas évolué depuis des années. Or, de plus en plus d'applications de présentation et de synchronisation utilisent cette base de données. L'idée est donc de faciliter le développement de ces applications en proposant des méthodes de développement simple pour l'accès à ces données. De plus des API de type service Web peuvent être proposé afin d'implémenter cette librairie pour d'autres langages que le PHP.

Pré-requis

L'ORM est écrite en PHP et nécessite une version supérieure ou égale à PHP 5.3 pour fonctionner. Les modules PHP nécessaire sont php5-ldap et php5-psql. L'ORM peut utiliser du cache via memcached et le module php5-memcache. Il existe deux méthodes d'installation pour l'ORM, par composer (conseillé) ou par mise en place de l'archive.

Installation par composer

L'ORM peut être installée via composer (voir https://getcomposer.org/) pour une intégration automatique dans l'application.

C'est la méthode la plus simple. Il suffit d'ajouter un

"require": { "messagerie-melanie2/ORM-M2": *version* },

dans le fichier composer.json puis de faire un composer install ou update.
La version actuelle de l'ORM est la 0.4.0.X. Par principe on met "~0.4.0.0" pour récupérer la dernière version de cette branche.

Cette méthode ne nécessite pas de faire d'include de l'ORM dans le code, par contre il faut include l'autoload.php du dossier vendor.

Installation par l'archive

Récupération de la librairie

La librairie ORM Mélanie2 peut être récupérée auprès du PNE Annuaire et Messagerie du MTES ou bien depuis les sources github (https://github.com/messagerie-melanie2/ORM-M2/releases/latest). La version peut ensuite être décompressée dans un répertoire.

Configuration du fichier php.ini

Ajouter dans la configuration du php.ini (cli ou apache2) le chemin vers la librairie dans le champs "include_path".

Exemple:

> ; UNIX: "/path1:/path2"

> include_path = ".:/usr/share/php:/usr/share/libM2/LibrarySqlMelanie2"

Bien inclure le répertoire "LibrarySqlMelanie2/".

Configuration de l'ORM

Par défaut, l'ORM va chercher la configuration dans le répertoire /etc/LibM2. La configuration à positionner dans ce répertoire est à récupérer dans config/default (https://github.com/messagerie-melanie2/ORM-M2/tree/master/config/default).

Le fichier ldap.php va permettre de configurer un ou plusieurs serveurs ldap. Le fichier sql.php va permettre de configurer un ou plusieurs serveur PostgreSQL (seul serveur supporté par l'ORM M2 pour l'instant). Le fichier config.php est plus général. En principe la seule modification à faire dans ce fichier est le nom de l'application.

Il est possible de créer un fichier env.php dans le répertoire /etc/LibM2 pour modifier le fonctionnement de la configuration. S'inspirer du fichier env.php interne à l'ORM pour plus d'informations : https://github.com/messagerie-melanie2/ORM-M2/blob/master/env.php

Mise en production

Pour la production, les fichiers de configuration "config/ldap.php" et "config/sql.php" doivent être paramétrés.

Les répertoires "docs_html/", "documentation/", "tests/" et le fichier "example.php" peuvent être supprimés.

Licence

L'ORM Mélanie2 est distribuée sous licence GPLv3 (http://www.gnu.org/licenses/gpl.html)

Points sensibles

Serveur

La librairie ORM M2 peut nécessiter plusieurs modules suivant l'usage qui en est fait, si les accès LDAP sont utilisés, il faut le module « php5-ldap », pour des accès à une base de données, les drivers PHP du type de base de données sont nécessaires (php5-pgsql pour Postgresql par exemple). Il faut également disposer d'un serveur Apache (ou autre serveur Web) avec PHP >= 5.3 installé. Le PHP 5.3 apporte notamment la gestion des namespaces ainsi que l'implémentation améliorée des méthodes magiques.

Il semblerait qu'une version >= 5.4 de PHP améliore les performances de ces méthodes magiques (qui sont obligatoirement moins performantes que les méthodes classiques).

Version actuelle

La version actuelle en développement est la 0.4.0.x.

Les produits utilisant l'ORM Mélanie2 en production actuellement sont Zpush2, SabreDAV, Pégase et Roundcube.

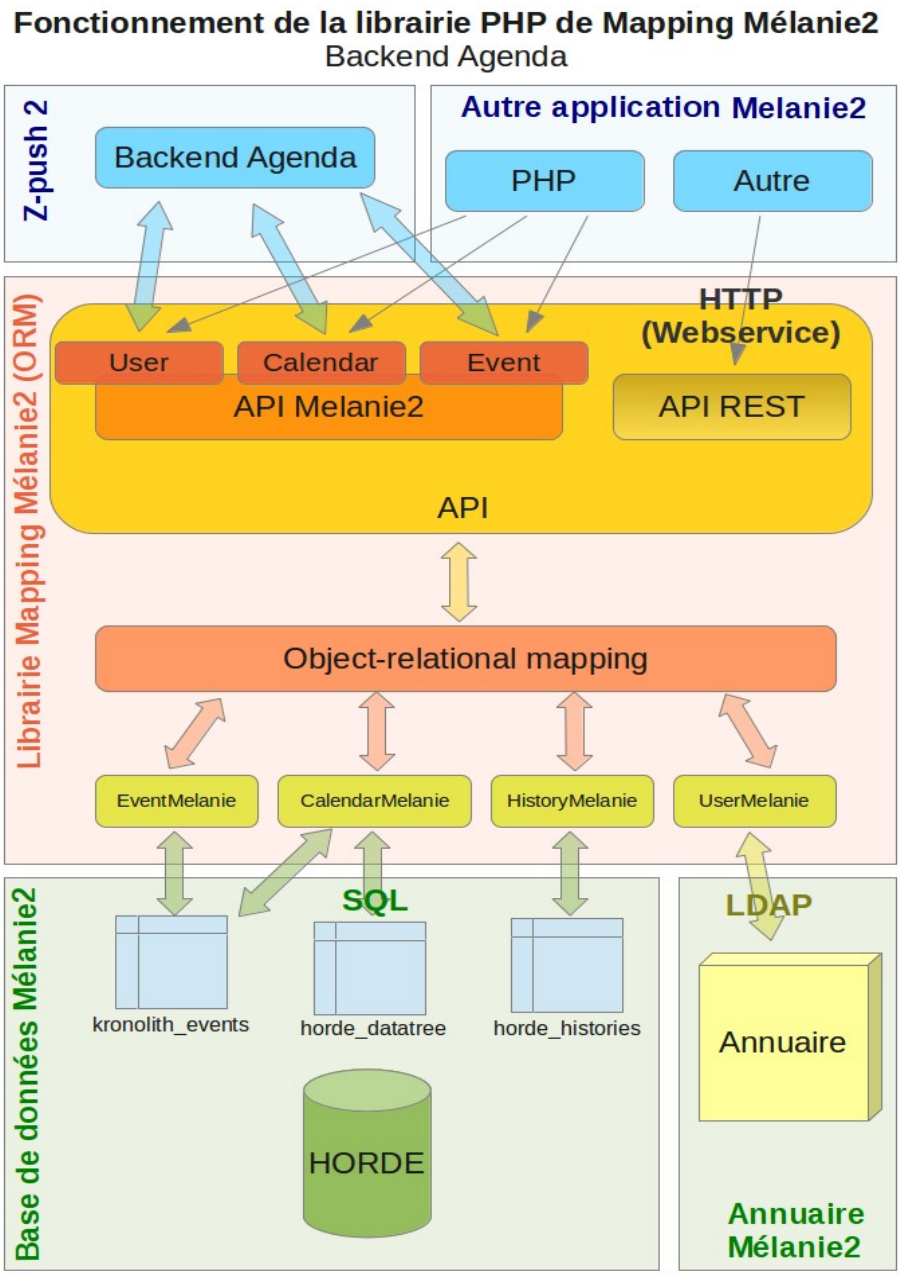
Application

Le but de cette librairie étant de simplifier l'implémentation du backend Mélanie2 dans les applications, il devrait être possible de l'intégrer dans a peu près n'importe quelle application (écrite en PHP ou via des API REST/WS). Néanmoins quelques pré-requis devraient permettre de faciliter l'implémentation :

- La séparation entre le backend de données et la présentation au sein de l'application. Normalement les applications récentes permettent de choisir différents backends pour le stockage des données. Ils sont donc a priori séparés du reste de l'application ce qui permet de simplifier les modifications. Dans le cas contraire, il sera nécessaire d'identifier tous les appels au backend pour les remplacer par des appels à la librairie.
- La gestion des évènements doit se rapprocher au maximum de celle dans horde/kronolith. Actuellement la base de données Mélanie2 ne stocke pas beaucoup d'informations sur la gestion des évènements, notamment au niveau de la récurrence. Un haut niveau de configuration des évènements dans la nouvelle application pourrait être problèmatique sans évolution de la base de données Mélanie2.

Présentation technique

Schéma fonctionnel partie Backend Agenda avec Zpush2



Ce schéma représente l'architecture de l'ORM pour la partie Agenda. Tous les modules sont basés sur le même principe, seul le noms des champs dans la base de données et les noms d'objets sont différents.

Couche d'extraction des données

Les données sont extraites en SQL depuis la base de données ou en LDAP depuis l'annuaire Mélanie2. Les besoins couverts sont donc assez large, allant de l'authentification de l'utilisateur à la liste des événements dans un laps de temps.

La couche d'abstraction des données utilise la librairie native PDO (http://php.net/manual/fr/book.pdo.php). Pour permettre le mapping des données, les données sont extraites via PDO en mode « fetch to class » qui aura pour effet d'instancier une nouvelle classe avec les données reçu de la base.

L'utilisation de PDO va permettre d'implémenter plusieurs de drivers de base de données, PostgreSQL dans le cas de Mélanie2, mais aussi MySQL ou SQLite.

Pour s'éviter d'implémenter une classe par table en définissant exactement le schéma dans celle-ci, on utilise une classe abstraite qui implémente les méthodes magiques de PHP (http://php.net/manual/fr/language.oop5.magic.php). Cette classe (`LibMelanie\Lib\MagicObject`) est abstraite et est étendue par tous les objets et conteneur de la librairie. Un premier mapping au niveau du nom des attributs est alors exécuté au moment de la lecture des données via les méthodes magiques __get et __set.

Couche de mapping des données

Le mapping des valeurs se fait dans un deuxième temps via une deuxième classe abstraite utilisant des méthodes magiques (`LibMelanie\Lib\Melanie2Object`). Ce sont alors les classes d'API Mélanie2 qui implémente cette classe. Cette séparation a été faite pour séparer la couche d'extraction des données de celle de présentation des données aux applications. L'idée sera alors de simplifier le changement vers une nouvelle base de données par exemple.

Configuration du mapping

Le mapping se configure en premier lieu dans le fichier config/mapping.php. Il permet d'associer les noms de propriétés des objets aux nom des champs dans la base de données. Il s'agit du mapping bas niveau effectué par la classe `LibMelanie\Lib\MagicObject`.

Par exemple :

```
// Gestion des évènements : objet EventMelanie
"EventMelanie" => array(
            "uid" => array(self::name => "event_uid", self::type =>
self::string, self::size => 255),

            // DATA
            "title" => array(self::name => "event_title", self::type =>
self::string, self::size => 255, self::defaut => ''),
```

Pour le mapping plus spécifique, nécessitant plus de code notamment pour la conversion des données, il est possible de définir dans les classes des méthodes « setMap<*Property*> »/« getMap<*Property*> » qui seront automatiquement appelées lors de la lecture ou la définition d'une propriété de la classe.

Par exemple :

```
/**
 * Mapping status field
 * @param Event::STATUS_* $status
 */
protected function setMapStatus($status) {
      M2Log::Log(M2Log::LEVEL_DEBUG, get_class($this)."-
>setMapStatus($status)");
      if (!isset($this->objectmelanie)) throw new
Exceptions\ObjectMelanieUndefinedException();
      if (isset(MappingMelanie::$MapStatusObjectMelanie[$status]))
            $this->objectmelanie->status = MappingMelanie::
$MapStatusObjectMelanie[$status];
}
/**
 * Mapping status field
 */
protected function getMapStatus() {
      M2Log::Log(M2Log::LEVEL_DEBUG, get_class($this)."->getMapStatus()");
      if (!isset($this->objectmelanie)) throw new
Exceptions\ObjectMelanieUndefinedException();
      if (isset(MappingMelanie::$MapStatusObjectMelanie[$this->objectmelanie-
>status]))
            return MappingMelanie::$MapStatusObjectMelanie[$this-
>objectmelanie->status];
      else
            return MappingMelanie::
$MapStatusObjectMelanie[self::STATUS_CONFIRMED];
}
```

Gestion de l'autoload

L'utilisation de l'ORM dans une application se fait via un autoload (les fichiers PHP sont chargés en mémoire dynamiquement quand c'est nécessaire), ce qui optimise l'usage mémoire et cpu du serveur. Pour que l'autoload soit fonctionnel, l'ORM utilise des namespaces. Tous les namespaces sont basés sur le même format : LibMelanie\<chemin_vers_le_fichier.php>. Le nom du fichier PHP doit alors correspondre au nom de la classe appelée en minuscule. Pour cela, chaque fichier PHP ne peut contenir qu'une seul classe PHP. L'arborescence de l'ORM doit donc respecter l'arborescence des namespaces.

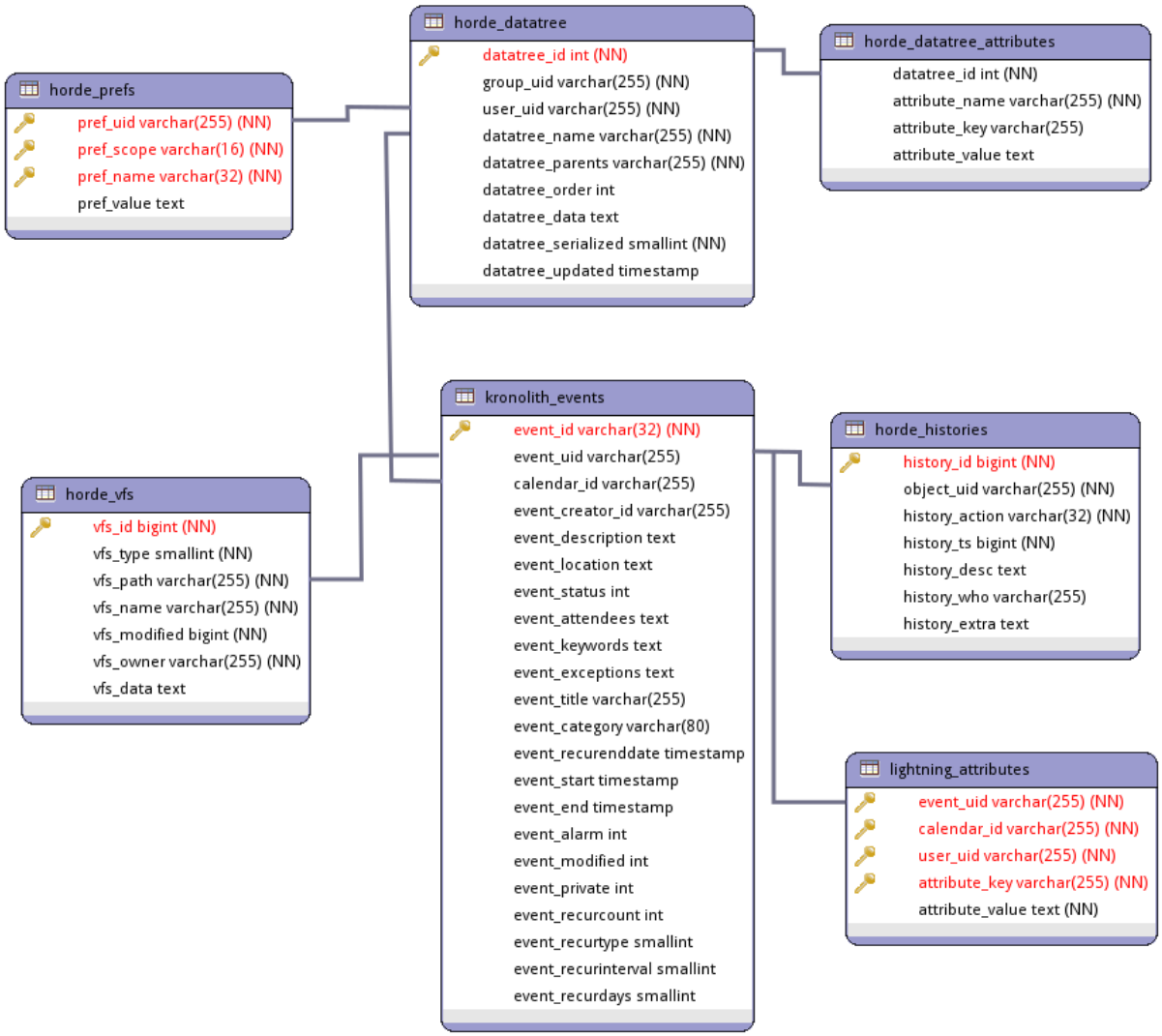
Exemple de configuration avec la base de données Mélanie2

La base de données Mélanie2 étant initialement une base de données Horde, elle possède donc des spécificités liées au produit. Or depuis depuis la mise en place du schéma, les applications collaboratives ont bien évoluées et les schémas de données sont généralement très différents du schéma Mélanie2. L'ORM possède donc une configuration sur-mesure pour extraire les données de la base vers des formats plus standard et adaptés aux applications modernes.

Pour présenter un exemple de configuration de l'ORM avec la base de données Mélanie2, nous n'allons montrer que la partie Agenda. Il s'agit de la partie la plus complexe utilisant toutes les capacités de l'ORM, les autres modules étant plus simples mais basés sur le même principe.

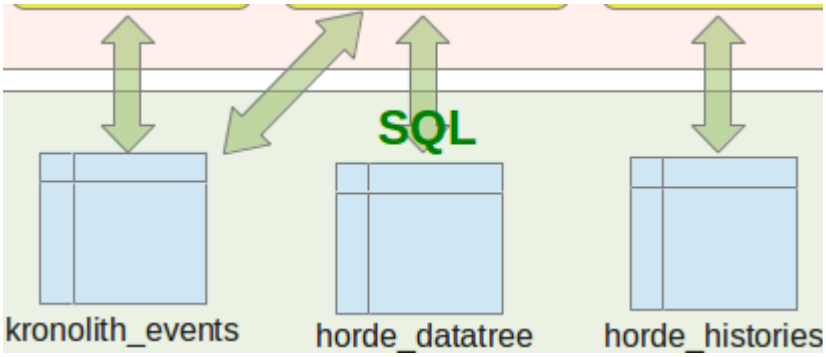
Présentation du schéma de la base de données Mélanie2

Voici la partie du schéma de la base de données Mélanie2 concernant uniquement la gestion des agendas et événements. Pour le schéma complet de la base, se référer au fichier « Melanie2_database_schema.png » du dossier documentation/.



Dans ce schéma, les agendas sont stockés dans la table « horde_datatree », les événements sont stockés dans la table « kronolith_events » et les pièces jointes dans la table « horde_vfs ». Les droits de partages des agendas vers les utilisateurs ou groupes sont enregistrés dans la table « horde_datatree_attributes », les préférences utilisateur (timezone, …) sont stockés dans la table « horde_prefs ». La table « horde_histories » permet d'enregistrer les création, modifications et suppressions d'événements par date et utilisateur, alors que la table « lightning_attributes » ajoute des paramètres aux événements notamment pour les échanges CalDAV avec le client Lightning (Thunderbird).

Configuration des requêtes SQL



L'ORM peut gérer plusieurs sortes de requêtes SQL : des requêtes standards et des requêtes personnalisées. Pour la plupart des objets Mélanie2, les requêtes SQL standards suffisent. Mais pour d'autres objets, plus complexes, on utilise des requêtes personnalisées, par exemple pour les événements ou les pièces jointes.

La configuration des requêtes se fait dans le dossier « sql/ ». Pour les requêtes standards, elles sont déjà implémentées dans le fichier « sqlobjectrequests.php ». Ces requêtes sont alors automatiquement configurées par l'ORM en fonction des objets utilisés par les applications et des valeurs associées.

Exemple de requête standard, pour lister un ou plusieurs objet générique :

```
      /**
       * Récupère un objet générique de Mélanie2
       * @var SELECT
       * @param Replace {fields_list}, {table_name}, {where_clause}
       */
      const getObject = "SELECT {fields_list} FROM {table_name} WHERE
{where_clause};";
```

Il existe 4 requêtes standards pour couvrir la majorité des besoin : getObject, insertObject, updateObject, deleteObject.

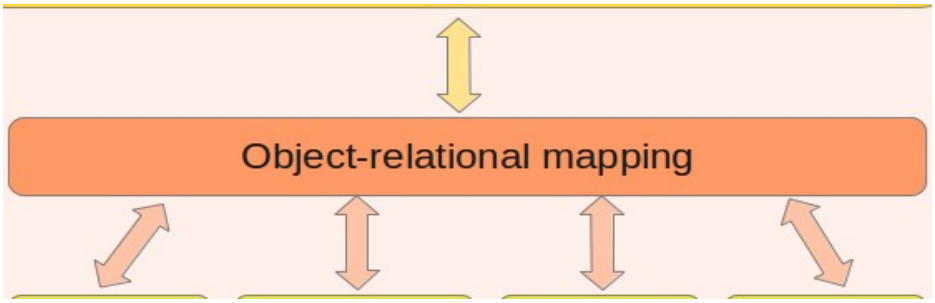
Pour la création de requêtes spécifiques, il faut créer une nouvelle classe dans le dossier « sql/ », cette nouvelle classe pourra alors être utilisée dans l'objet personnalisé pour réaliser les requêtes nécessaires.

Exemple, pour lister les événements d'un calendrier, la classe « SqlCalendarRequests » est implémentée dans le fichier « sqlcalendarrequests.php », avec la requête suivante :

```
/**
       * @var SELECT
       * @param REPLACE {event_range}
       * @param PDO :calendar_id
       */
      const listAllEvents = "SELECT k1.*, k2.event_creator_id as organizer_uid,
k2.event_attendees as organizer_attendees, k2.calendar_id as organizer_calendar
FROM kronolith_events k1 LEFT JOIN kronolith_events k2      ON k1.event_uid =
k2.event_uid AND k2.event_attendees <> '' AND k2.event_attendees <> 'a:0:{}'
WHERE k1.calendar_id = :calendar_id{event_range} ORDER BY event_start;";
```

Chaque classe pouvant contenir autant de requêtes personnalisées que nécessaire.

Configuration du mapping



Une fois que les requêtes sont configurées, il faut passer à la configuration du mapping. La configuration s'effectue dans le fichier « config/mapping.php », la classe MappingMelanie contenant plusieurs tableaux de configuration. Le mapping se fait en 3 étapes :

Configurer dans le tableau « $Table_Name », le nom de la table SQL en fonction de l'objet, exemple :

```
      /**
       * Tables associées aux objets
       * @var array
       */
      public static $Table_Name = array(
            "EventMelanie" => "kronolith_events",
            "HistoryMelanie" => "horde_histories",
            "EventProperties" => "lightning_attributes",
            "AttachmentMelanie" => "horde_vfs",
            "CalendarMelanie" => "horde_datatree",
            "UserPrefs" => "horde_prefs",
            "Share" => "horde_datatree_attributes",
      );
```

Configurer dans le table « $Primary_Keys », la liste des clés primaires par objet. Ce sont des clés primaires au sens ORM, c'est à dire qu'elles doivent correspondre aux valeurs nécessaires pour récupérer un objet (exemple pour Mélanie2 un événement nécessite un uid et un calendar, alors que la clé primaire sql est un id). La configuration se fait sur les valeurs mappées (voir ci-dessous). Exemple de configuration :

```
	/**
	 * Clés primaires des tables Melanie2
	 * @var array
	*/
	public static $Primary_Keys = array(
		"EventMelanie" => array("uid", "calendar"),
		"HistoryMelanie" => array("uid", "action"),
		"EventProperties" => array("event", "calendar", "user", "key"),
		"AttachmentMelanie" => array("path", "name"),
		"CalendarMelanie" => array("id", "owner", "group"),
		"UserPrefs" => array("user", "scope", "name"),
		"Share" => array("object_id", "name"),
	);
```

Configurer le mapping des valeurs, dans le tableau « $Data_Mapping ». C'est là que les champs des objets de l'ORM vont être associés à un champ la table de la base de données. Chaque champ ayant un nom et un type, et peut avoir des valeurs par défaut ou des tailles. L'ORM va alors traiter et formater les données pour qu'elle correspondent à ce qu'attend la base de données.

Exemple de configuration du mapping de valeurs pour une table :

```
      /**
       * Gestion du mapping entre les données et les champs de la base de
données
       * need name, type if != string, format for datetime (user constants)
       * @var array
      */
      public static $Data_Mapping = array(
                  // Gestion de l'historique : objet HistoryMelanie
                  "HistoryMelanie" => array(
                              "id" => array(self::name => "history_id",
self::type => self::integer),
                              "uid" => array(self::name => "object_uid"),
                              "action" => array(self::name =>
"history_action"),
                              "timestamp" => array(self::name => "history_ts",
self::type => self::timestamp, self::defaut => 0),
                              "description" => array(self::name =>
"history_desc"),
                              "who" => array(self::name => "history_who"),
                              "extra" => array(self::name => "history_extra")
                  ),
      );
```

Le reste du fichier « mapping.php » peut contenir des tableau de mapping à utiliser dans les traitements personnalisés.

Création d'un objet personnalisé

Si on ne souhaite pas utiliser un objet standard utilisant des requêtes SQL standard, il faut créer un objet personnalisé. Les objets se trouve dans le répertoire « objects », l'objet standard étant le fichier « objectmelanie.php ».

Chaque objet implémente l'interface « IObjectMelanie », et doivent donc définir ces 4 méthodes :

```
/**
 * Interface pour instancier un objet Melanie
 * @author PNE Messagerie/Apitech
 * @package Librairie Mélanie2
 * @subpackage Interfaces
 *
 */
interface IObjectMelanie {
      /**
```

```
        * Charge l'objet
        * @return boolean
        */
       public function load ();

       /**
        * Sauvegarde l'objet
        * @return boolean
       */
       public function save ();

       /**
        * Supprime l'objet
        * @return boolean
       */
       public function delete();

       /**
        * Si l'objet existe
        * @return boolean
       */
       public function exists ();
}
```

Un objet personnalisé peut également définir de nouvelles méthodes, qui seront automatiquement accessibles dans les applications. Ces nouvelles méthodes peuvent alors utiliser les requêtes SQL personnalisées.

Exemple de l'objet personnalisé CalendarMelanie

Définition de la classes

```
/**
 * Traitement des calendriers Melanie2
 * @author PNE Messagerie/Apitech
 * @package Librairie Mélanie2
 * @subpackage ORM
 */
class CalendarMelanie extends MagicObject implements IObjectMelanie {
}
```

Cet objet défini les 4 méthodes de l'interface, ainsi que plusieurs méthodes pour lister les événements d'un calendrier, par exemple, la méthode « getAllEvents » va lister tous les événements du calendrier via une requête personnalisée (trouvable dans la classes SqlCalendarRequests) :

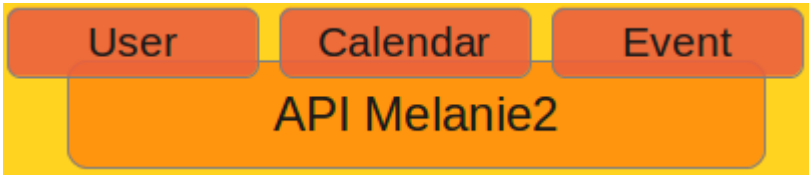
```
       /**
        * Récupère la liste de tous les évènements
        * need: $this->id
        * @return boolean
        */
       function getAllEvents() {
            M2Log::Log(M2Log::LEVEL_DEBUG, $this->get_class."-
>getAllEvents()");
            if (!isset($this->id)) return false;
```

```
            // Params
            $params = [MappingMelanie::$Data_Mapping[$this->objectType]['id']
[MappingMelanie::name] => $this->id];

            // Replace
            $query = str_replace("{event_range}", "",
Sql\SqlCalendarRequests::listAllEvents);

            // Liste les evenements du calendrier
            return Sql\DBMelanie::ExecuteQuery($query, $params,
'LibMelanie\Objects\EventMelanie');
      }
```

Création des API



Les API sont utilisées par les applications pour implémenter l'ORM. Elles doivent donc être relativement standard pour correspondre au maximum d'applications (il est aussi possible d'implémenter des API par application). Chaque API va utiliser un objet standard ou spécifique pour traiter les données. Pour Mélanie2, les API se trouve dans le répertoire « api/melanie2/ ». Chaque fichier PHP correspond à une classe utilisable dans les applications Mélanie2. Il peut exister plusieurs API par objet spécifique, pour notamment affiner le traitement, par exemple l'objet spécifique « EventMelanie » est utilisé par l'API « Event », mais également par les API « Organizer », « Attendee » ou « Recurrence ». De même un objet n'est pas nécessairement associé à une seul API, il peut n'être utilisé que pour un traitement particulier.

Une API peut être très simple, si elle laisse tout le traitement à l'objet spécifique/standard, ou bien complexe, notamment pour des cas de mapping avancé.

Exemple d'implémentation d'une API pour le calendrier

Définition de la classe de l'API « Calendar »

```
/**
 * Classe calendrier pour Melanie2
 *
 * @author PNE Messagerie/Apitech
 * @package Librairie Mélanie2
 * @subpackage API Mélanie2
 * @api
 *
 * @property string $id Identifiant unique du calendrier
 * @property string $owner Identifiant du propriétaire du calendrier
 * @property string $name Nom complet du calendrier
 * @property int $perm Permission associée, utiliser asRight()
 *
 * @method bool load() Charge les données du calendrier depuis la base de
```

```
données
 * @method bool exists() Non implémentée
 * @method bool save() Non implémentée
 * @method bool delete() Non implémentée
 * @method void getCTag() Charge la propriété ctag avec l'identifiant de
modification du calendrier
 * @method void getTimezone() Charge la propriété timezone avec le timezone du
calendrier
 * @method bool asRight($action) Retourne un boolean pour savoir si les droits
sont présents
 */
class Calendar extends Melanie2Object {
}
```

L'utilisation de l'objet spécifique ou standard se fait dans le constructeur de la classe de l'API, par exemple pour utiliser un objet spécifique :

```
	/**
	 * Constructeur de l'objet
	 *
	 * @param UserMelanie $usermelanie
	 */
	function __construct($usermelanie = null) {
		// Définition du calendrier melanie2
		$this->objectmelanie = new CalendarMelanie();
	}
```

Pour un objet standard :

```
	/**
	 * Constructeur de l'objet
	 *
	 * @param UserMelanie $usermelanie
	 * @param EventMelanie $eventmelanie
	 */
	function __construct($usermelanie = null, $eventmelanie = null) {
		// Définition de la propriété de l'évènement
		$this->objectmelanie = new ObjectMelanie('EventProperties');
	}
```

« `'EventProperties'` » correspond alors au nom de l'objet configuré dans le fichier « mapping.php ».

Une fois que les objets sont configurés, les méthodes appelées pour les API seront automatiquement répercutées sur l'objet. Sauf si les méthodes sont définies directement dans l'API, pour un traitement particulier.

Exemple, pour l'appel de la méthode « save() » de l'API « Event », on enregistre en même temps dans l'historique. On effectue donc un traitement particulier sur l'API qui appel la méthode « save() » de l'objet.

```
	/**
	 * Mapping de la sauvegarde de l'objet
	 * Appel la sauvegarde de l'historique en même temps
	 * @ignore
	 */
	function save() {
		// Sauvegarde l'objet
		$insert = $this->objectmelanie->save();
		if (!is_null($insert)) {
			// Sauvegarde des attributs
			$this->saveAttributes();
			// Gestion de l'historique
			$history = new HistoryMelanie();
			$history->uid = ConfigMelanie::CALENDAR_PREF_SCOPE . ":" .
$this->calendar . ":" . $this->realuid;
			$history->action = $insert ? ConfigMelanie::HISTORY_ADD :
ConfigMelanie::HISTORY_MODIFY;
			$history->timestamp = time();
			$history->description = "LibM2/".ConfigMelanie::APP_NAME;
			$history->who = isset($this->usermelanie) ? $this-
>usermelanie->uid : $this->calendar;
			// Enregistrement dans la base
			if (!is_null($history->save())) return $insert;
		}
		return null;
	}
```

Exemple d'implémentation dans une application Mélanie2

Utiliser l'ORM dans le code PHP

Voici un exemple d'implémentation de la librairie ORM M2 dans une application Mélanie2. Cet exemple est basé sur le fichier example.php disponible à la racine du code.

Nom d'application

En cas de configuration multiple de l'ORM (voir configuration du env.php) il faut définir le nom de l'application afin que l'ORM récupère la bonne configuration. Pour cela on ajoute ces lignes (doit être mis en premier, avant les inclusions) :

```
// Configuration du nom de l'application pour l'ORM
if (!defined('CONFIGURATION_APP_LIBM2')) {
      define('CONFIGURATION_APP_LIBM2', '<nom_application>');
}
```

Inclusion des fichiers

Si installation via composer

```
// Inclure le fichier autoload.php
include_once 'vendor/autoload.php';
```

Si installation en mode archive

```
// Inclure le fichier libm2.php
include_once 'includes/libm2.php';
```

Cette inclusion se base sur le fait que la librairie se trouve bien dans le include_path du php.ini

Namespace

La librairie utilise des namespaces, pour pouvoir utiliser les classes il est donc utile d'appeler les bons namespace, quelques exemples :

```
// API
use LibMelanie\Api\Melanie2\User;
use LibMelanie\Api\Melanie2\Calendar;
use LibMelanie\Api\Melanie2\Event;
```

Configuration des logs

La configuration des logs permet d'ajouter les logs de la librairie directement dans les logs de l'application. Il est également possible d'écrire dans un fichier de log indépendant, tout se configure au niveau de l'application.

```
// Log
use LibMelanie\Log\M2Log;

// Configurer les LOG
$log = function ($message) {
```

```
      echo "[LibM2] $message \r\n";
};
M2Log::InitDebugLog($log);
M2Log::InitInfoLog($log);
M2Log::InitErrorLog($log);
```

Il existe 3 niveaux de logs : Info, Debug et Error, d'autres pourront peut être arriver ensuite.

Utiliser les objets de l'ORM pour lire les données

Il est ensuite possible d'instancier des objets conteneurs, par exemple :

```
// Définition d'un utilisateur
$user = new User();
$user->uid = 'julien.test1';

// Définition des conteneurs de l'utilisateur
$calendar = new Calendar($user);

// ID des conteneurs
$calendar->id = 'julien.test1';

// Charger les conteneur
$calendar->load();
```

Chaque conteneur s'instancie exactement de la même façon.

Il est également possible de vérifier les droits pour chaque conteneur.

```
// Vérifier les droits
echo $calendar->asRight(ConfigMelanie::READ) ? "calendar->read OK" : "calendar-
>read NOK";
```

On peut également lister les éléments d'un conteneur

```
// Lister tous les objets d'un conteneur
$events = $calendar->getAllEvents();
```

Les objets suivent le même principe d'implémentation. Par exemple pour le chargement

```
// Récupération d'un objet
$event = new Event($user, $calendar);
$event->uid = 'c2926c54-17af-4753-a614-3589eca2644e';
$event->load();
```

Utiliser les objets de l'ORM pour modifier les données

Pour les modifications/créations d'objets

```
// Modifier un objet
$event->title = "Mon titre d'évènement";
$event->save();
```

Ensuite quelques spécificités liées aux évènements (participants, récurrence)

```
// Gestion d'un evenement
$attendees = $event->attendees;
if ($attendees[0]->response == ConfigMelanie::ACCEPTED) echo "Attendee
accepted";
$event->recurrence->days = array(Recurrence::RECURDAYS_MONDAY,
Recurrence::RECURDAYS_SUNDAY);
```

Toutes les propriétés et méthodes sont visibles depuis la documentation PHPDocs ou accessibles en auto-complétion via un IDE comme eclipse par exemple.