XRL

Version 3.1.0

mai 12, 2018

Table des matières

1 Vue d'ensemble 3

2 Installation et usage 5

3 Fonctionnalités 9

4 Contributions 15

5 Licence 17

6 Autres ressources 19

XRL (abréviation de XML-RPC Library) est un ensemble de classes PHP facilitant la création de clients et de serveurs XML-RPC.

CHAPITRE 1

Vue d'ensemble

- **Installation facile**—récupérez l'archive PHAR ou ajoutez `fpoirotte/xrl` aux dépendances de votre `composer.json` et vous êtes fin prêt.
- **Syntaxe très intuitive**—écrivez des clients et serveurs XML-RPC comme n'importe quel autre code.
- **Conversion automatique des types**—utilisez les types PHP natifs sans vous préoccuper des particularités liées à XML-RPC.
- **Support pour de nombreuses extensions**—vous utilisez les *capabilities* ? l'"*introspection* ? le *multicall* ? ... oui, *nous les supportons* !

CHAPITRE 2

Installation et usage

2.1 Installation

Avant d'installer XRL, assurez-vous de disposer d'une installation de PHP fonctionnelle.

XRL nécessite PHP 5.3.4 ou plus récent, ainsi que les extensions PHP suivantes :

- XMLReader
- XMLWriter
- libxml
- GMP
- PCRE
- SPL
- Reflection

Note : Utilisez `php -v` et `php -m` pour obtenir des informations sur votre version de PHP et les extensions disponibles.

XRL peut être installé en utilisant *Archive PHAR*, *Composer*, ou depuis les *Sources*. L'utilisation de PHAR est recommandée.

2.1.1 Archive PHAR

Téléchargez la dernière archive PHAR disponible sur https://github.com/fpoirotte/XRL/releases et sauvegardez la sur votre ordinateur.

(Pour les utilisateurs d'Unix/Linux) Si vous le souhaitez, rendez le fichier exécutable.

2.1.2 Composer

XRL peut être installé grâce au gestionnaire de dépendances Composer. Ajoutez simplement `fpoirotte/xrl` aux dépendances de votre fichier file :*composer.json* :

```
$ php composer.phar require fpoirotte/xrl
```

2.1.3 Sources

Pour installer XRL à partir des sources, utilisez **`git`** afin de cloner le dépôt :

```
$ git clone https://github.com/fpoirotte/XRL.git /new/path/for/XRL
```

2.2 Démarrage rapide

En supposant que XRL est correctement *installé* sur votre ordinateur, vous pouvez à présent coder des clients et serveurs XML-RPC.

2.2.1 Coder un client XML-RPC

1. Chargez et enregistrez le chargeur automatique [1]

```
require_once('/path/to/XRL/src/Autoload.php');
\fpoirotte\XRL\Autoload::register();
```

2. Créez le nouveau client en le configurant pour qu'il interroge le serveur XML-RPC distant

```
$client = new \fpoirotte\XRL\Client("http://xmlrpc.example.com/server");
```

3. Appelez une méthode fournie par ce serveur, comme vous le feriez avec n'importe quel autre code

```
// Call the remote procedure named "hello",
// with "world" as its parameter.
$result = $client->hello('world');

// $result now contains the remote procedure's result,
// as a regular PHP type (integer, string, double, array, etc.)
var_dump($result); // string(12) "hello world!"

// Methods with names that are not valid PHP identifiers
// can still be called!
var_dump($client->{'string.up'}('game over')); // string(9) "GAME OVER"
```

2.2.2 Coder un serveur XML-RPC

1. Chargez et enregistrez le chargeur automatique[1]

Les utilisateurs du gestionnaire de dépendances Composer doivent utiliser le chargeur automatique habituel se trouvant dans `vendor/autoload.php` à la place.

```
require_once('/path/to/XRL/src/Autoload.php');
\fpoirotte\XRL\Autoload::register();
```

2. Créez une nouvelle instance de serveur

```
$server = new \fpoirotte\XRL\Server();
```

3. Attachez quelques méthodes à ce serveur
   — Vous pouvez attacher des fonctions anonymes, des *closures*, des fonctions globales, des méthodes publiques d'objets, etc. en utilisant l'opérateur d'accès aux attributs `->`. Vous pouvez même utiliser des objets invocables !

```
class Simpson
{
    private $speech = array(
        'Homer'     => 'Doh!',
        'Marge'     => 'Hmm...',
        'Bart'      => 'Aie, caramba!',
        'Lisa'      => M_PI,
        'Maggie'    => null,
    );
    private $character;

    public function __construct($character)
    {
        if (!array_key_exists($character, $this->speech)) {
            throw new InvalidArgumentException("Who's that?");
        }
        $this->character = $character;
    }

    public function __invoke()
    {
        return $this->speech[$this->character];
    }
}
$server->homer  = new Simpson('Homer');
$server->marge  = new Simpson('Marge');
$server->bart   = new Simpson('Bart');
$server->lisa   = new Simpson('Lisa');
$server->maggie = new Simpson('Maggie');
```

   — Ou bien, vous pouvez utiliser la syntaxe des tableaux `[]` à la place. Il s'agit de l'approche recommandée car elle évite des conflits potentiels avec les attributs internes de XRL et simplifie l'utilisation de noms de méthodes qui ne sont pas des identifiants PHP valides.

```
$server['hello'] = function ($s) { return "Hello $s!"; };
$server['string.up'] = 'strtoupper';
```

4. Traitez les requêtes XML-RPC qui arrivent et publiez les résultats

```
$server->handle()->publish();
```

2.3 HipHop Virtual Machine

> HipHop Virtual Machine (HHVM) est une machine virtuelle basée sur la compilation à la volée (JIT) et servant de moteur d'exécution pour les langages de programmation PHP et Hack.
> Source : http://en.wikipedia.org/wiki/HipHop_Virtual_Machine

XRL devrait être compatible avec HHVM. Pour nous assurer de cela, HHVM est inclus dans les tests de notre processus d'Intégration Continue.

CHAPITRE 3

Fonctionnalités

3.1 Types XML-RPC

3.1.1 Types supportés

XRL supporte l'ensemble des types définis dans la spécification officielle de XML-RPC, c'est-à-dire :

- `int` et `i4` : entier signé sur 32 bits
- `boolean` : le type booléen habituel
- `string` : une simple chaîne de caractères
- `double` : nombre flottant signé à double précision
- `dateTime.iso8601` : date/heure (sans milli-secondes ni fuseau horaire)
- `base64` : données binaires encodées en base64
- `struct` : tableau associatif
- `array` : tableau indexé

XRL accepte également les types suivants, dont l'usage est plutôt répandu, bien qu'ils ne fassent pas partie de la spécification officielle :

- `nil` : valeur null (absence de valeur)
- `i8` : entier signé sur 64 bits

Pour finir, les types suivants appartenant à l'espace de noms défini par la Fondation Apache sont aussi supportés. Veuillez noter que dans ce cas particulier, les types doivent appartenir à l'espace de nom ayant pour URI http://ws.apache.org/xmlrpc/namespaces/extensions pour être correctement interprétés.

- `nil` : valeur null (absence de valeur) ; identique au type sans espace de noms
- `i1` : entier signé sur 8 bits
- `i2` : entier signé sur 16 bits
- `i8` : entier signé sur 64 bits ; identique au type sans espace de noms
- `biginteger` : entier de taille arbitraire
- `dom` : élément du DOM, transmis sous forme de fragment XML
- `dateTime` : date/heure avec milli-secondes et fuseau horaire

Lorsque des types non-standards doivent être envoyés, XRL utilise systématiquement les types appartenant à des espaces de noms. Lisez le chapitre qui suit pour plus d'information.

3.1.2 Conversion des types

Par défaut, XRL convertit automatiquement les valeurs entre les types de PHP et ceux de XML-RPC lorsque c'est nécessaire.

Le tableau qui suit montre comment XRL convertit les types PHP vers les types XML-RPC.

Tableau 1 – Conversion de PHP vers XML-RPC

Type PHP	Type XML-RPC
`null`	`nil` dans un espace de noms
`boolean`	`boolean`
`integer`	`i4` si le nombre tient sur 32 bits, `i8` dans un espace de noms [1] sinon
`double`	`double`
`string`	`string` s'il s'agit d'une séquence UTF-8 valide, `base64` sinon
`array`	`array` pour les tableaux indexés, `struct` pour les tableaux associatifs
ressource `GMP integer` (PHP < 5.6.0)	`i4` si le nombre tient sur 32 bits, `i8` dans un espace de noms[1] s'il tient sur 64 bits, `biginteger` dans un espace de noms[1] sinon
objet `\GMP` (PHP >= 5.6.0)	`i4` si le nombre tient sur 32 bits, `i8` dans un espace de noms[1] s'il tient sur 64 bits, `biginteger` dans un espace de noms[1] sinon
objet `\fpoirotte\XRL\Types\AbstractType`	Type XML-RPC représenté par l'objet
objet `\DateTime`	`dateTime.iso8601` .. note : `The XML-RPC ``dateTime.iso8601`` data␣↪type does not support milliseconds, nor passing timezone␣↪information. Only use this type when the actual␣↪timezone is known beforehand (using contextual information or a pre-↪established convention), or when potential errors in date/time␣↪handling are not an issue.`
objet `\DOMNode`	`dom` dans un espace de noms[1]
objet `\XMLWriter`	`dom` dans un espace de noms[1]
objet `\SimpleXMLElement`	`dom` dans un espace de noms[1]
objet `\Exception`	faute XML-RPC (dérivée de `struct`)

Le tableau qui suit montre comment XRL convertit les types XML-RPC vers les types PHP.

Avec l'espace de noms portant l'URI http://ws.apache.org/xmlrpc/namespaces/extensions pour la compatibilité avec les autres implémentations.

Tableau 2 – Conversion XML-RPC vers PHP

Type XML-RPC	Type PHP
`boolean`	`boolean`
`i4`	`integer`
`int`	`integer`
`double`	`double`
`string`	`string`
`base64`	`string`
`array`	tableau indexé
`struct`	objet `\Exception` si la structure représente une faute [2], tableau associatif sinon
`dateTime.iso8601`	`\DateTime` (en utilisant le fuseau horaire local par défaut)
`nil`	`null`
`nil` dans un espace de noms[1]	`null`
`i1` dans un espace de noms[1]	`integer`
`i2` dans un espace de noms[1]	`integer`
`i8`	`integer` on 64-bit PHP, a `GMP integer` resource (on PHP < 5.6.0) or a `\GMP` object (on PHP >= 5.6.0) .. note : `An exception will be thrown in this case␣ ↪if PHP's native` `` ``integer`` type cannot be used and the␣ ↪GMP extension` `is not available.`
`i8` dans un espace de noms[1]	`integer` on 64-bit PHP, either a `GMP integer` resource (on PHP < 5.6.0) or a `\GMP` object (on PHP >= 5.6.0) .. note : `An exception will be thrown in this case␣ ↪if PHP's native` `` ``integer`` type cannot be used and the␣ ↪GMP extension` `is not available.` otherwise (the GMP extension is required in this case)

`biginteger` dans un espace de noms[1]	`GMP integer` resource (PHP < 5.6.0) or `\GMP` object (PHP >= 5.6.0) .. note :
	`An exception will be thrown in this case␣ ↪if the GMP extension`
	`is not available.`

`dom` dans un espace de noms[1]	objet `\SimpleXMLElement`
`datetime` dans un espace de noms[1]	`\DateTime` object (using local timezone information by default, but can be configured to use another timezone as well)

3.1.3 Sous le capot

Les conversions de types sont gérées par les classes `\fpoirotte\XRL\NativeEncoder` (pour les conversions depuis PHP vers XML-RPC) et `\fpoirotte\XRL\NativeDecoder` (pour les conversions depuis XML-RPC

[2] Un `struct` XML-RPC représentant une faute (c'est-à-dire une erreur) est converti en une exception qui est automatiquement levée.

vers PHP), en s'appuyant sur les classes de l'espace de noms `\fpoirotte\XRL\Types\`.

Vous pouvez surcharger ou désactiver les conversions en passant un autre encodeur/décodeur au constructeur du client ou du serveur XML-RPC.

Note : Si vous changez l'encodeur/décodeur par défaut, vous devrez gérer vous-même les conversions depuis/vers les instances de `\fpoirotte\XRL\Types\AbstractType` utilisées en interne par XRL.

Avertissement : Les fautes XML-RPC sont traitées à part et sont toujours transformées en objets `\fpoirotte\XRL\Exception` avant d'être automatiquement levées. Ce comportement est indépendant du décodeur passé au constructeur du client/serveur.

3.2 Extensions

XRL supporte plusieurs extensions par rapport à la spécification XML-RPC originale. Ces extensions sont connues pour être bien supportées par les autres implémentations et pour ne pas rentrer en conflit avec la spécification originale en général.

3.2.1 Extensions supportées

getCapabilities

L'"extension getCapabilities a été créée pour deux raisons :

- Pour permettre aux serveurs XML-RPC d'annoncer les fonctionnalités (non-standards) qu'ils supportent.
- Pour fournir un moyen simple clients XML-RPC d'adapter leur comportement en fonction des fonctionnalités non-standards supportées par un serveur.

Les serveurs XRL implémentent les méthodes additionnelles suivantes lorsque cette extension est activée :

- `system.getCapabilities`

introspection

L'extension introspection permet à un client d'obtenir des informations à propos d'une procédure distante en interrogeant le serveur XML-RPC qui la fournit.

Les serveurs XRL implémentent les méthodes additionnelles suivantes lorsque cette extension est activée :

- `system.listMethods`
- `system.methodSignature`
- `system.methodHelp`

multicall

L'extension multicall a été créée afin d'attenuer la latence introduite par les allers-retours HTTP lorsque plusieurs appels de méthodes sont envoyés au même serveur XML-RPC.

Les serveurs XRL implémentent les méthodes additionnelles suivantes lorsque cette extension est activée :

- `system.multicall`

faults_interop

L'extension faults_interop contient les spécifications de codes correspondant à des cas d'erreurs standards, afin d'encourager l'interopérabilité entre les implémentations XML-RPC.

Cette extension est toujouts activée et n'ajoute aucune méthode supplémentaire aux serveurs XML-RPC. Un développeur désireux d'utiliser les erreurs interopérables définies dans cette extension peut lever l'exception associée dans l'espace de noms `\fpoirotte\XRL\Faults`.

```
$server->error = function () {
    throw new \fpoirotte\XRL\Faults\SystemErrorException();
};
```

Les exceptions suivantes peuvent être utilisées pour représenter une erreur interopérable :

- `ApplicationErrorException`
- `InternalErrorException`
- `InvalidCharacterException`
- `InvalidParameterException`
- `InvalidXmlRpcException`
- `MethodNotFoundException`
- `NotWellFormedException`
- `SystemErrorException`
- `TransportErrorException`
- `UnsupportedEncodingException`

Par ailleurs, l'exception `ImplementationDefinedErrorException` peut être utilisée pour les erreurs propres à l'implémentation. Notez cependant qu'un code d'erreur conforme à la spécification doit être passé explicitement lors de la création d'une telle erreur.

```
$server->error = function () {
    throw new \fpoirotte\XRL\Faults\ImplementationDefinedErrorException(
        -32000,                 // Implementation-defined error code
        "You're out of memory"  // Implementation-defined error message
    );
};
```

Types Apache

L'extension des types Apache est un peu spéciale. Elle n'ajoute pas de nouvelles méthodes. À la place, elle définit de nouveaux types XML-RPC.

Cette extensions est toujours activée. Lisez également la documentation sur les *types XML-RPC supportés* pour plus d'information sur ces types et la manière dont ils sont utilisés dans XRL.

3.2.2 Activer les extensions

Par défaut, XRL n'active que quelques extensions (plus précisément, les extensions `faults_interop` et `Apache types`).

Pour activer le reste des extensions, vous devez appeler `\fpoirotte\XRL\CapableServer::enable()` sur le serveur :

```
// Create a regular XML-RPC server.
$server = new \fpoirotte\XRL\Server();
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Enable additional extensions (capabilities) for that server.
\fpoirotte\XRL\CapableServer::enable($server);
```

Note : Il n'est pas possible actuellement d'activer séparément chaque extension en utilisant `\fpoirotte\XRL\CapableServer::enable()`. C'est une situation de type « tout ou rien ».

3.3 Client XML-RPC en ligne de commandes

XRL fournit un client XML-RPC en ligne de commandes qui facilite le test d'un serveur XML-RPC distant.

Pour les installations PHAR, le client est intégré à l'archive elle-même. Pour l'utiliser, appelez l'interpréteur PHP sur l'archive PHAR :

```
$ php XRL.phar
Usage: XRL.phar [options] <server URL> <procedure> [args...]

Options:
 -h                Show this program's help.
 [...]
```

Pour les autres formes d'installations, appelez l'interpréteur PHP sur `bin/xrl` :

```
$ php ./bin/xrl
Usage: ./bin/xrl [options] <server URL> <procedure> [args...]

Options:
 -h                Show this program's help.
 [...]
```

CHAPITRE 4

Contributions

4.1 Contribuer à XRL

Il existe plusieurs moyens pour contribuer à XRL.

4.1.1 Essayez le !

Plus il y a de personnes qui l'utilisent, mieux c'est. Cela permet de détecter les bugs et les régressions plus rapidement.

4.1.2 Signalez les problèmes / Suggérez des fonctionnalités

Si vous utilisez XRL et que vous avez identifié un problème, merci de bien vouloir nous en faire part sur GitHub. Essayez de fournir autant de détails que possible sur la manière de reproduire le problème. En règle générale, plus le problème est facile à reproduire, plus il a de chances d'être corrigé rapidement.

4.1.3 Améliorez la documentation

Nous essayons de documenter XRL autant possible. Cependant, comme nous sommes à la fois développeurs et rédacteurs de la documentation, nous avons une vision un peu biaisée des éléments qui nécessitent d'être documentés.

Si vous sentez que certains passages pourraient être mieux explicités, envoyez-nous une pull request avec vos modifications et nous ferons de notre mieux pour la relire rapidement. Toute aide pour améliorer la documentation est toujours la bienvenue !

4.1.4 Clonez et améliorez le code !

Si vous trouvez un bug, que vous connaissez bien PHP et que vous avez un peu de temps libre, récupérez le code et essayez de le corriger.

Si vous possédez un compte GitHub, c'est assez facile :

1. Clonez le dépôt
2. Faites vos modifications
3. Envoyez-nous une pull request pour relecture
4. Recommencez à partir de l'étape 2

4.2 Conventions de codage

L'équipe de XRL suit attentivement les conventions de codage définies par la PSR-2.

De plus, lors du développement, la commande suivante permet de vérifier différents aspects de la qualité du code :

```
$ vendor/bin/phing qa
```

Elle exécute les outils suivants sur le code de XRL pour détecter les éventuels problèmes :

— PHP lint (`php -l`) : vérification de la syntaxe PHP
— PHP_CodeSniffer : vérification du respect des conventions de codage
— PDepend : identification des portions de code ayant un couplage trop important
— PHPMD (PHP Mess Detector) : détection des structures de code posant un risque
— PHPCPD (PHP Copy-Paste Detector) : détection des abus de copier/coller
— PHPUnit : passage des tests unitaires

4.3 Remerciements

Remerciements particuliers pour :

— Thibaud Rohmer, qui a été le tout premier utilisateur de XRL et l'instigateur de ce projet

Merci à :

— David Goodger pour Docutils et reStructuredText
— Georg Brandl pour Sphinx
— La personne qui a créé le thème `haiku` de Sphinx

grâce auxquels l'écriture et la publication de cette documentation ont été rendues possibles.

CHAPITRE 5

Licence

XRL est distribué selon les termes de la licence BSD à 3 clauses.

CHAPITRE 6

Autres ressources

- XRL sur GitHub (suivi de code source et de bugs)
- XRL sur Packagist (dépôt Composer)
- XRL sur Travis-CI (intégration continue)
- XRL sur Read The Docs (documentation en ligne)
- Documentation d'API complète (hébergée sur Read The Docs)