Wunderlist PHP SDK

Release

August 11, 2015

Contents

CHAPTER 1

What is it?

This is the unofficial Wunderlist SDK for PHP!!

The Wunderlist API provides REST-based storage and synchronization of a user's lists across multiple platforms and devices. The primary things you'll need to use it are an understanding of our data model, how we version individual entities in a user's data, the formats we use for transmission, and a set of OAuth credentials.

The PHP SDK helps you to interact with this API.

1.1 User guide

1.1.1 Welcome to Wunderlist PHP SDK

What is it?

This is the unofficial Wunderlist SDK for PHP!!

The Wunderlist API provides REST-based storage and synchronization of a user's lists across multiple platforms and devices. The primary things you'll need to use it are an understanding of our data model, how we version individual entities in a user's data, the formats we use for transmission, and a set of OAuth credentials.

The PHP SDK helps you to interact with this API.

Installation

The recommended way to install Wunderlist PHP SDK is with Composer. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project.

```
# Install Composer
curl -sS https://getcomposer.org/installer | php
```

You can add Wunderlist PHP SDK as a dependency using the composer.phar CLI:

```
php composer.phar require italolelis/wunderist
```

Alternatively, you can specify Wunderlist PHP SDK as a dependency in your project's existing composer.json file:

```
 {
   "require": {
      "italolelis/wunderist": "~1.0"
```

```
    }
}
```

After installing, you need to require Composer's autoloader:

```
require 'vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Bleeding edge

During your development, you can keep up with the latest changes on the master branch by setting the version requirement for Wunderlist PHP SDK to `~1.0@dev`.

```
{
   "require": {
      "italolelis/wunderist": "~1.0@dev"
   }
}
```

License

Licensed using the MIT license.

> The MIT License (MIT)
>
> Copyright (c) 2013 italolelis <italolelis@gmail.com>
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contributing

Guidelines

1. Wunderlist PHP SDK follows PSR-0, PSR-1, and PSR-2.
2. Wunderlist PHP SDK is meant to be lean and fast with very few dependencies.
3. Wunderlist PHP SDK has a minimum PHP version requirement of PHP 5.5. Pull requests must not require a PHP version greater than PHP 5.5.

4. All pull requests must include unit tests to ensure the change works as expected and to prevent regressions.

Running the tests

In order to contribute, you'll need to checkout the source from GitHub and install Collection's dependencies using Composer:

```
git clone https://github.com/italolelis/wunderlist.git
cd wunderlist && curl -s http://getcomposer.org/installer | php && ./composer.phar install --dev
```

Wunderlist PHP SDK is unit tested with PHPUnit. Run the tests using the vendored PHPUnit binary:

```
vendor/bin/phpunit
```

1.1.2 Quickstart

This page provides a quick introduction to Wunderlist PHP SDK and introductory examples. If you have not already installed, Wunderlist PHP SDK, head over to the installation page.

The SDK is pretty simple to use, here is an example of how we can access all lists:

```
<?php

use Wunderlist\Entity\WList;
use Wunderlist\ClientBuilder;

// Instanciate wunderlist API manager
$builder = new ClientBuilder();
$wunderlist = $builder->build('yourClientId', 'yourClientSecret', 'http://domain.com/oauth/callback');

//Here we get all lists for the authenticated user
$lists = $wunderlist->getService(WList::class)->all();

//For each list on the lists
$lists->map(function($list) {
    echo $list->getTitle();
});
```

What about all taks for a list?

```
<?php

use Wunderlist\Entity\Task;
use Wunderlist\Entity\WList;

//Here we get all lists for the authenticated user
$lists = $wunderlist->getService(WList::class)->all();

//For each list on the lists
$lists->map(function($list) {
    $tasks = wunderlist->getService(Task::class)->forList($list);
    $tasks->map(function($task){
        echo $task->getTitle();
    });
});
```

Ok, now lets create a task for a list

This is just some simple things you can do with the SDK. Whant more? please just read our [documentation](http://wunderlist.readthedocs.org/)

Requests

By default all requests made to the API are syncronous, this is because we use the *GuzzleAdapter*. But if you want to make asyncronous request you need to change the adapter *AsyncGuzzleAdapter*, this will allow to make calls like this:

```
$service = $wunderlist->getService(WList::class);
$service->all()->done(function($lists){
       $lists->map(function($list){
           echo $list->getTitle();
       });
});
```

To change the default HttpClient adapter just change on *ClientBuilder*:

```
$builder = new Wunderlist\ClientBuilder();
$wunderlist = $builder->setHttpClient(Wunderlist\Http\AsyncGuzzleAdapter::class)
                      ->build();
```

1.1.3 Authentication

The Wunderlist API uses OAuth2 to allow external applications to request authorization to a user's Wunderlist account without directly handling their password. Developers must register their application before getting started. Registration assigns a unique client ID and client secret for your application's use. After you have registered your application, you can let Wunderlist users authorize access to their account information from your application by getting an access token. After a user has authorized your application and you have an access token, you can use it in Wunderlist API requests by setting the X-Client-ID and X-Access-Token HTTP request headers.

Our SDK handles this procedure very well. For this we have a some simple steps to make. Just remeber that this procedure is already done by our SDK, this is just an stand alone service if you want to use it.

Built in Providers

PHPoAuthLib

PHPoAuthLib provides oAuth support in PHP 5.3+ and is very easy to integrate with any project which requires an oAuth client.

oauth2-client

This package makes it stupidly simple to integrate your application with OAuth 2.0 identity providers.

Custom Providers

You can implement any provider by implementing the *AuthenticationInterface*

```
use Wunderlist\OAuth\AuthenticationInterface;
use Symfony\Component\HttpFoundation\Request;

class MyProviderAuthentication implements AuthenticationInterface
{
    /**
     * @return Request
```

```
     */
    public function getRequest()
    {

    }

    /**
     * @param Request $request
     * @return $this
     */
    public function setRequest($request)
    {

    }

    /**
     * @return string
     */
    public function getConsumerId()
    {

    }

    /**
     * @return string
     */
    public function getAccessToken()
    {

    }

    /**
     * @return string
     */
    public function hasAccessToken()
    {

    }

    public function authorize()
    {

    }
}
```

1.1.4 Services

A service is something that can be consumable by our API client. For example, lists, tasks. Any service implements the ServiceInterface interface. All services has these base methods wich can be vey useful:

```
//Gets a entity base on the ID
$service->getID(123456789);

//Gets the base url used to consume the API
$service->getBaseUrl();

//Makes a GET request to the API
```

```php
$service->get('lists');

//Creates an entity at the API
$service->create($entity);

//Updates an entity at the API
$service->update($entity);

//Deletes an entity from the API
$service->delete($entity);

//Performs a GET for a user ID on the resource.
$service->forUser($user);

//Performs a GET for a task ID on the resource.
$service->forTask($task);

//Performs a GET for a list ID on the resource.
$service->forList($list);

//Updates only certain fields at the API
$service->patch(123456789, ['completed' => true]);
```

You can create a service for wunderlist API, you just need to implement *ServiceInterface*, or if you want to have the implementations above, extend from *AbstractService*

If you want to have readonly operations in your service, you need to extend from *AbstractGetOnlyService*

```php
class MyService extends AbstractService
{
    /**
     * The service's base path. For example 'tasks' will become 'https://a.wunderlist.com/api/v1/tas
     * when an HTTP request is made.
     * @var string
     */
    protected $baseUrl = 'avatar';

    /**
     * The service's resource type. For examples 'Task' for services/Tasks
     * @var string
     */
    protected $type = MyService::class;
}
```

1.1.5 Revisions and Sync

Every entity in the Wunderlist API has a read-only revision property. This property is an integer which is updated in response to changes to that entity or any of its children. When the title of a task is changed, that task's revision is updated—as well as the revisions of all of the parent items of that task, including list and root entities.

Updating Entities

In order to guarantee that updates to Wunderlist entities are correctly executed and kept in sync across clients, any changes to an entity through the API must be accompanied by the revision property. The server uses this property to ensure that the client has the most up-to-date version of the entity. If a client makes a request with an out-of-date revision property, the request will fail, indicating that the client needs to fetch the entity's current state and try again.

If an update request fails, you must fetch the current version of the entity, look for attributes that conflict with your local state e.g. content on a note, and do some sort of local conflict resolution before replaying your changes to the API with the current revision.

Sync

You can completely synchronize a local copy of the Wunderlist data model with the Wunderlist API by checking the root revision property, descending if necessary, and repeating the process for each leaf in the tree. When a russian doll sync occurs on a client, the following rules apply: Fetched revision values and data should not be committed to local models and persistence layers unless child resources are successfully fetched. This means you should not update the child-revision of the parent until all child data has been successfully fetched. E.g. you should not apply list data and revision changes unless all tasks were fetched successfully, etc. Deleted items can be found by comparing your local data to the data retrieved during a russian doll sync and comparing for missing ids. However, since tasks may be moved to another list, you should mark a task as missing and only delete it if it is not present in any lists when the russian doll sync has completed successfully. This pattern can be extended to any model type that is "moveable".

1.1.6 Service Manager

The service manager provides methods for easy access to API resources data data. The manager tries to follow the same principle as a ObjectManager, from Doctrine, where you have a manager to the repositories. Instead of a repository we have a service.

Create an instance of the resource service

```
$listsService = $wunderlist->getService(WList::class);
```

Get all records for a resource

```
$lists = $wunderlist->getService(WList::class)->all();
```

Get a specific resource

```
$list = $wunderlist->find(WList::class, 777);
```

Get a specific resource for a user

```
$user = $wunderlist->getService(User::class)->current();
$lists = $wunderlist->forUser(WList::class, $user);
```

Get a specific resource for a list

```
$list = $wunderlist->find(WList::class, 777);
$tasks = $wunderlist->forList(Task::class, $list);
```

Get a specific resource for a task

```
$task = $wunderlist->find(Task::class, 777);
$subtasks = $wunderlist->forTask(Subtask::class, $task);
```

Create a resource

```
$list = new Wunderlist\Entity\WList();
$list->setTitle('Bad Movies');
$wunderlist->save($list);
```

Update a resource

```
$list = $wunderlist->find(WList::class, 777);
$list->setTitle('Good Bad Movies');
$wunderlist->save($list);
```

Delete a resource

```
$list = $wunderlist->find(WList::class, 777);
$wunderlist->delete($list);
```

1.1.7 Lists

Provides specifics methods for easy access to list data.

Get all accepted lists

```
$acceptedLists = $wunderlist->getService(WList::class)->accepted();
```

Make a list public

```
$list = $wunderlist->find(WList::class, 777);
$listsService->makePublic($list);
```

Make a list private

```
$list = $wunderlist->find(WList::class, 777);
$listsService->makePrivate($list);
```

1.1.8 Tasks

Provides specifics methods for easy access to tasks data.

Get today tasks

```
$todayTasks = $wunderlist->getService(Task::class)->today();
```

Get overdue tasks

```
$overdueTasks = $wunderlist->getService(Task::class)->overdue();
```

Get all tasks with its subtasks

```
$tasks = $wunderlist->getService(Task::class)->allWithSubtasks();
```

Filter tasks by date

```
$tasks = $wunderlist->getService(Task::class)->filterByDate(Carbon::now());
```

1.1.9 Reminders

Provides methods for easy access to reminders data.

Create an instance of the Reminder service

```
$remindersService = $wunderlist->getReminders();
```

Get all lists for a user

```
$user = $userService->current();
$lists = $listsService->forUser($user);
```

Get a specific list

```
$list = $listsService->getID(777);
```

Create a list

```
$list = new Wunderlist\Entity\List();
$list->setTitle('Bad Movies');
$listsService->create($list);
```

Update a list

```
$list = $listsService->getID(777);
$list->setTitle('Good Bad Movies');
$listsService->update($list);
```

Delete a list

```
$list = $listsService->getID(777);
$listsService->delete($list);
```

1.1.10 Integrations

Silex

We have a silex provider for you to use with your Silex application. It's important to configure the authenticator in your silex application.

```
php composer.phar require gigablah/silex-oauth
php composer.phar require italolelis/wunderist-provider
```

Lets register *silex-oauth* and *wunderist-provider*:

```
// app.php
$app->register(new Gigablah\Silex\OAuth\OAuthServiceProvider());
$app->register(new Wunderlist\Silex\Provider\WunderlistServiceProvider());
```

Configure both services:

```
// config/prod.php
$app['oauth.services'] = [
    'wunderlist' => [
        'class' => 'Wunderlist\OAuth\Service\Wunderlist',
        'key' => 'yourClientID',
        'secret' => 'yourClientSecret',
        'scope' => array(),
        'user_endpoint' => 'https://a.wunderlist.com/api/v1/user'
    ]
];

// This is not necessary, but it's good for the login with wunderlist funcionality
$app['security.firewalls'] = array(
    'default' => array(
        'pattern' => '^/',
        'anonymous' => true,
```

```
        'oauth' => array(
            //'login_path' => '/auth/{service}',
            //'callback_path' => '/auth/{service}/callback',
            //'check_path' => '/auth/{service}/check',
            'failure_path' => '/',
            'with_csrf' => true
        ),
        'logout' => array(
            'logout_path' => '/logout',
            'with_csrf' => true
        ),
        'users' => new Gigablah\Silex\OAuth\Security\User\Provider\OAuthInMemoryUserProvider()
    )
);

$app['security.access_rules'] = array(
    array('^/auth', 'ROLE_USER')
);
```

In your controllers:

```
$wunderlist = $app['wunderlist'];
$listsService = $wunderlist->getLists();
```

Skeleton App

We develop a skeleton app which is already configured for you. Just run:

```
php composer.phar create-project italolelis/silex-skeleton-wunderlist your/path/
```

Symfony

Use symfony? Don't worry, we have a Bundle for Wunderlist SDK too!

```
php composer.phar require italolelis/wunderist-bundle
```

Just register the bundle and access the SDK:

```
<?php
// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        // ...
        new Wunderlist\WunderlistBundle(),
    );
}
```

Configure:

```
# app/config/config.yml
wunderlist:
    credentials:
        clientId: yourClientID
        clientSecret: yourClientSecret
        redirectUri: http://domain.com/oauth/callback
```

In your controllers:

```
$wunderlist = $this->get('wunderlist');
$listsService = $wunderlist->getLists();
```