apai-io

Release

Apr 17, 2017

Contents

CHAPTER 1

Amazon-Product-Adverstising-API library based on PHP using REST and SOAP.

Contents:

Installation

Install via Composer

The easiest way to install apai-io is to use the PHP dependency management tool composer. First you have to add a file named composer.json to your project root Edit this composer.json and add the following content to it

```
{
    "require": {
        "exeu/apai-io": "~1.0"
    }
}
```

Now you have to download composer:

```
$ curl -sS https://getcomposer.org/installer | php
```

Note that there are other ways to get composer. Please read the guide: http://getcomposer.org/download/

After composer is downloaded you can run the following command:

```
$ php composer.phar install
```

After the installation succeeded, composer created a vendor dir where the library is placed and a additional autoload.php which you can include.

For example - create a index.php and add the following:

```php
require_once "vendor/autoload.php";

use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();

$conf
    ->setCountry('com')
    ->setAccessKey('YOUR ACCESS KEY')
    ->setSecretKey('YOUR SECRET KEY')
    ->setAssociateTag('YOUR ASSOCIATE TAG');

$apaiIo = new ApaiIO($conf);
```

Now your done and ready to use apai-io. See Basic usage

Basic usage

Making your first request

After the installation you can use apai-io out of the box.

This guide expects that you use an autoloader in your project!

Creating your configuration First of all you have to create a new configuration object. Apai-IO has one built in configuration class which can be used in nearly every project. Its called GenericConfiguration.

```php
use ApaiIO\Configuration\GenericConfiguration;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey('YOUR ACCESS KEY')
    ->setSecretKey('YOUR SECRET KEY')
    ->setAssociateTag('YOUR ASSOCIATE TAG');
```

You have to set the country, accesskey, secretkey and your associate tag using the setter functions of the Generic-Configuration class.

How to get your accesskey, secretkey and associate tag is documented here: http://docs.aws.amazon.com/AWSECommerceService/latest/DG/becomingAssociate.html

The country could be one of the following: **de, com, co.uk, ca, fr, co.jp, it, cn, es, in, com.br, com.mx, com.au**

For example if you set this to de you will send your requests to the german amazon database (www.amazon.de)

Creating your operation After setting up the configuration, you can go to the next step and create your operation.

In the following example we take the ItemSearch operation: http://docs.aws.amazon.com/AWSECommerceService/latest/DG/ItemSearch.html

```php
use ApaiIO\Operations\Search;

$search = new Search();
$search->setCategory('DVD');
```

```
$search->setActor('Bruce Willis');
$search->setKeywords('Die Hard');
```

Now you have your first operation configured. It searches the DVD-Index, looking for the actor Bruce Willis and for the keywords Die Hard.

There are magic setter functions within this class. You can set additional parameters of your choice! What parameters are possible see the link above.

For example you can set the condition of the item to used:

```
$search->setCondition('used');
```

Making the request The next step is to make the request:

```
use ApaiIO\ApaiIO;

$apaiIo = new ApaiIO($conf);
$response = $apaiIo->runOperation($search);

var_dump($response);
```

If everything worked fine and your configuration (accesskey, secretkey, associatetag) was correct, you will get a xml response as string.

The full example

```
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\Operations\Search;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey('YOUR ACCESS KEY')
    ->setSecretKey('YOUR SECRET KEY')
    ->setAssociateTag('YOUR ASSOCIATE TAG');

$search = new Search();
$search->setCategory('DVD');
$search->setActor('Bruce Willis');
$search->setKeywords('Die Hard');

$apaiIo = new ApaiIO($conf);
$response = $apaiIo->runOperation($search);

var_dump($response);
```

Advanced configuration

RequestFactory Callbacks

Since version 1.3 you are able to set up a callback with the configuration which is called before the internal request factory returns a new requestobject.

With this change you can manipulate the requestobject easily.

```php
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\Operations\Search;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey('YOUR ACCESS KEY')
    ->setSecretKey('YOUR SECRET KEY')
    ->setAssociateTag('YOUR ASSOCIATE TAG')
    ->setRequestFactory(
        function($request) {
            // do what ever you want
            return $request;
        }
    );

$search = new Search();
$search->setCategory('DVD');
$search->setActor('Bruce Willis');
$search->setKeywords('Die Hard');

$apaiIo = new ApaiIO($conf);
$response = $apaiIo->runOperation($search);

var_dump($response);
```

ResponseTransformerFactory Callbacks

Since version 1.3 you are able to set up a callback with the configuration which is called before the internal response-transformer factory returns a new responsetransformerobject.

With this change you can manipulate the responsetransformerobject easily.

```php
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\Operations\Search;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey('YOUR ACCESS KEY')
    ->setSecretKey('YOUR SECRET KEY')
    ->setAssociateTag('YOUR ASSOCIATE TAG')
    ->setResponseTransformerFactory(
        function($responseTransformer) {
            // do what ever you want
            return $responseTransformer;
        }
    );

$search = new Search();
$search->setCategory('DVD');
$search->setActor('Bruce Willis');
$search->setKeywords('Die Hard');

$apaiIo = new ApaiIO($conf);
```

```
$response = $apaiIo->runOperation($search);

var_dump($response);
```

Built in operations

apai-io comes with some built in operations.

ItemSeach

The ItemSearch can be used to search products in the amazon database.

Please see the following documentation for more information.

Finding movies with “Bruce Willis” and keywords like “Die Hard”

```
use ApaiIO\Operations\Search;

$search = new Search();
$search->setCategory('DVD');
$search->setActor('Bruce Willis');
$search->setKeywords('Die Hard');
$search->setResponsegroup(array('Large', 'Images'));

$response = $apaiIo->runOperation($search);
```

Finding books “lord of the rings” and the condition “used”

```
use ApaiIO\Operations\Search;

$search = new Search();
$search->setCategory('Books');
$search->setKeywords('lord of the rings');
$search->setCondition('used');

$response = $apaiIo->runOperation($search);
```

ItemLookup

The ItemLookup can be used to lookup a product in the amazon database.

Please see the following documentation for more information.

Getting the item with asin “B00D6BN9NK” -> link

```
use ApaiIO\Operations\Lookup;

$lookup = new Lookup();
$lookup->setItemId('B00D6BN9NK');
$lookup->setResponseGroup(array('Large')); // More detailed information

$response = $apaiIo->runOperation($lookup);
```

SimilarityLookup

The SimilarityLookup can be used to lookup similar products of the given asin in the amazon database.

Please see the following documentation for more information.

Getting similar products to asin “B00D6BN9NK” -> link

```
use ApaiIO\Operations\SimilarityLookup;

$similaritylookup = new SimilarityLookup();
$similaritylookup->setItemId('B00D6BN9NK');

$response = $apaiIo->runOperation($similaritylookup);
```

BrowseNodeLookup

The BrowseNodeLookup can be used to browse the amazon product nodes.

Please see the following documentation for more information.

Browsing the node “163357” (Comedy)

```
use ApaiIO\Operations\BrowseNodeLookup;

$browseNodeLookup = new BrowseNodeLookup();
$browseNodeLookup->setNodeId(163357);

$response = $apaiIo->runOperation($browseNodeLookup);
```

Creating custom operations

If the built in operations are note enough, you can build your own operations.

For example if you want to add your own custom logic to the operation like validation or more special getter and setter.

apai-io accepts every operation class which implements the following interface:

```
ApaiIO\Operations\OperationInterface
namespace ApaiIO\Operations;

interface OperationInterface
{
    public function getName();

    public function setResponseGroup(array $responseGroup);

    public function getOperationParameter();
}
```

Lets build a custom ItemSearch class

The class

```
namespace Acme\MyApp;

use ApaiIO\Operations\OperationInterface;

class SimpleKeywordSearch implements OperationInterface
{
    private $operationParameter = array(
        'ResponseGroup' => 'Large',
        'SearchIndex' => 'Blended'
    );

    public function getName()
    {
        return "ItemSearch"; // Amazon operation name
    }

    public function setResponseGroup($responseGroup)
    {
        $this->operationParameter['ResponseGroup'] = $responseGroup;
    }

    public function getOperationParameter()
    {
        return $this->operationParameter;
    }

    public function setKeywords($keywords)
    {
        $this->operationParameter['Keywords'] = $keywords;
    }

    public function setCategory($category)
    {
        $this->parameter['SearchIndex'] = $category;
    }
}
```

Running the operation

```
use Acme\MyApp\SimpleKeywordSearch;

$simpleKeywordSearch = new SimpleKeywordSearch();
$simpleKeywordSearch->setKeywords("Bruce Willis, Die Hard");

$response = $apaiIo->runOperation($simpleKeywordSearch);
```

The ResponseTransformer

The Responsetransformer is a nice way to manipulate the response which comes from the amazon api.

For example if you making a REST call, you will get back a xml response which contains all items of the specified responsegroup.

So what to do if you want to change this response? The answer is: The ResponseTransfomer!

ResponseTransfomer under the hood (interface)

```
namespace ApaiIO\ResponseTransformer;

interface ResponseTransformerInterface
{
    /**
     * Transforms the response of the request
     *
     * @param mixed $response
     */
    public function transform($response);
}
```

Every ResponseTransformer implements this simple interface. The function transform takes care about transforming the response which comes from amazon.

Nothing else to do! You have the choice: You can build your own ResponseTransformer or you can use one of the built in.

Built in Responsetransformer

apai-io comes with some build in ResponseTransfomer classes.

ObjectToArray

The ObjectToArray ResponseTransformer transforms Objects (stdClass) to an array. Its basicly used when you decided to make SOAP requests.

You can pass the fully quallified class name to the configuration object which you use:

```
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey(AWS_API_KEY)
    ->setSecretKey(AWS_API_SECRET_KEY)
    ->setAssociateTag(AWS_ASSOCIATE_TAG)
    ->setRequest('\ApaiIO\Request\Soap\Request')
    ->setResponseTransformer('\ApaiIO\ResponseTransformer\ObjectToArray');

$apaiIo = new ApaiIO($conf);

// ... Preparing your operation

$response = $apaiIo->runOperation($operation);
```

If you run this code the $response will be an array containing the whole response from the amazon api.

You now can access the key you want or iterate over it.

Note: This will only work if you are going to make SOAP Requests.

XmlToDomDocument

By default apai-io uses REST request to query the amazon database.

All REST requests having a XML-String as response

The XmlToDomDocument transformer transforms the XML-Response to an instance of DOMDocument.

```
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\ApaiIO;

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey(AWS_API_KEY)
    ->setSecretKey(AWS_API_SECRET_KEY)
    ->setAssociateTag(AWS_ASSOCIATE_TAG)
    ->setResponseTransformer('\ApaiIO\ResponseTransformer\XmlToDomDocument');

$apaiIo = new ApaiIO($conf);

// ... Preparing your operation

$response = $apaiIo->runOperation($operation);
```

Xslt

If you want to transform a XML to a for example usage ready HTML you can use the XSLT Transformer.

What XSLT is, you can see here: Wikipedia :)

```
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\ApaiIO;
use ApaiIO\ResponseTransformer\Xslt;

$xsltResponseTransformer = new Xslt($xstlTemplate); // $xstlTemplate -> String

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey(AWS_API_KEY)
    ->setSecretKey(AWS_API_SECRET_KEY)
    ->setAssociateTag(AWS_ASSOCIATE_TAG)
    ->setResponseTransformer($xsltResponseTransformer);

$apaiIo = new ApaiIO($conf);

// ... Preparing your operation

$response = $apaiIo->runOperation($operation);
```

Creating your own ResponseTransformer

If you need your own ResponseTransformer you can simply achieve this by implementing the ResponseTransformer-Interface and passing the instance of your class or its name to the configuration object.

Let's build our own Transformer which returns all Item Elements via XPath.

```php
namespace Acme\Demo;

use ApaiIO\ResponseTransformer\ResponseTransformerInterface;

class ItemSearchXmlToItems implements ResponseTransformerInterface
{
    public function transform($response)
    {
        $xml =  simplexml_load_string($response);
        $xml->registerXPathNamespace("amazon", "http://webservices.amazon.com/
↪AWSECommerceService/2011-08-01");

        $elements = $xml->xpath('//amazon:ItemSearchResponse/amazon:Items/amazon:Item
↪');

        return $elements;
    }
}
```

Now you have build the class you can use it out of the box:

```php
use ApaiIO\Configuration\GenericConfiguration;
use ApaiIO\ApaiIO;
use Acme\Demo\ItemSearchXmlToItems;

$itemSearchXmlToItems = new ItemSearchXmlToItems();

$conf = new GenericConfiguration();
$conf
    ->setCountry('com')
    ->setAccessKey(AWS_API_KEY)
    ->setSecretKey(AWS_API_SECRET_KEY)
    ->setAssociateTag(AWS_ASSOCIATE_TAG)
    ->setResponseTransformer($itemSearchXmlToItems);

$apaiIo = new ApaiIO($conf);

// ... Preparing your operation

$response = $apaiIo->runOperation($operation);
```

If you dont want to instanciate the object you can pass the fully quallified class name:

```php
$conf->setResponseTransformer('\Acme\Demo\ItemSearchXmlToItems');

$apaiIo = new ApaiIO($conf);

// ... Preparing your operation

$response = $apaiIo->runOperation($operation);
```

Testing

PHPUnit

When you download ApaiIO it comes with some unittests. If you plan to run these tests you can do it simply by running phpunit in the project root dir.

$ cd /path/to/lib $ phpunit Please see http://phpunit.de/manual/current/en/index.html for more information about PHPUnit.

The tests are located in the folder: tests/. Feel free to have a look at them.

Environment variables

If you want to run all tests you need to set up some environment variables. These variables are nescessary to make request to the amazon api.

```
$ export APAI_IO_SECRETKEY=YOUR SECRET KEY
$ export APAI_IO_ACCESSKEY=YOUR ACCESS KEY
```

If these variables are correct, all tests will run instead of skipping some tests.

Index

B