CloudBlue
https://cloudblue.com

Documentation →



This article has been generated from the online version of the documentation and might be out of date. Please, make sure to always refer to the online version of the documentation for the up-to-date information.

Auto-generated at March 29, 2024

© Ingram Micro, Inc. All Rights Reserved.

The Connect SDK project template for php provides developers an complete skeleton to start their automation project using the Connect Fulfillment API together with the Connect SDK for PHP.

Requirements

In order to use this template you will need an environment capable to run PHP scripts, any version starting PHP 5.6 is supported. Additionally please ensure that composer it's functional.

Installation

You can download the quick start template with Composer **create-project** command in your terminal:

```
composer create-project --prefer-dist apsconnect/connect-quickstart-template project-name
```

Once you have the skeleton project ready you can start developing your custom code in **app/ProductFulfillment.php** file in the **processRequest()** method.

```
<?php

namespace App;

/**
 * Class ProductFulfillment
 * @package App
 */
class ProductFulfillment extends \Connect\FulfillmentAutomation
{

    /**
     * Process each pending request
     * @param \Connect\Request $request
     */
    public function processRequest($request)
    {
        // TODO: Implement processRequest() method.
    }

    /**
     * Run the Product Fulfillment Request Processor
     * @return bool
     * @throws \GuzzleHttp\Exception\GuzzleException
     */
    public function run()
    {
```


© Ingram Micro, Inc. All Rights Reserved.

```
        try {

            /**
             * run the application in custom context, any error
             * handling customization should be done here
             */
            $this->process();
            return true;

        } catch (\Exception $e) {

            $this->logger->error($e->getMessage());
            if (is_callable([$this->logger, 'dump'])) {
                $this->logger->dump();
            }
        }

        return false;
    }
}
```

Testing your project

Test the **ProductFulfillment** class is quite easy, you just need to create your tests in **tests/Feature/** directory by default the skeleton have some common tests.

Mocking the FulFillment API calls

To mock the fulfillment API, and with that test like if a request is retrived, you only have to create a valid json file with the response that you want in the **tests/Feature/** directory with the name **YOUR-TEST-NAME.http.json** where **YOUR-TEST-NAME** is the name of the test file in where you want to use that fake response, for example:

- **ProcessSkipTest.http.json** will be injected in **ProcessSkipTest.php** test.
- **RunAndFailTest.http.json** will be injected in **RunAndFailTest.php** test.

Mocking other services

The skeleton also provides a easy way to mock other services, for this you only need to create a new service provider in **tests/Providers**, this new service provider MUST extends the **MockServiceProvider**. Inside this file you should configure your mocked service using the static property **$scope** to retrieve any auxiliary file like a fake json.

```
<?php

namespace Test\Providers;
```


© Ingram Micro, Inc. All Rights Reserved.

```
use Pimple\Container;

/**
 * Class HttpServiceProvider
 * @package Test\Unit\Runtime\Providers
 */
class HttpServiceProvider extends MockServiceProvider
{
    public function register(Container $container)
    {
        $content = '[]';
        $fakeRequestFile = dirname(__DIR__) . '/Feature/' . self::$scope . '.json';
        if (is_readable($fakeRequestFile)) {
            $content = trim(file_get_contents($fakeRequestFile));
        }

        $body = \Mockery::mock('\Psr\Http\Message\StreamInterface');
        $body->shouldReceive('getContents')
            ->andReturn($content);

        $response = \Mockery::mock('\Psr\Http\Message\ResponseInterface');
        $response->shouldReceive('getStatusCode')
            ->andReturn(200);

        $response->shouldReceive('getBody')
            ->andReturn($body);

        $client = \Mockery::mock('GuzzleHttp\ClientInterface');
        $client->shouldReceive('request')
            ->withAnyArgs()
            ->andReturn($response);

        return $client;
    }
}
```

Finally you need to register your mocked service in the **$providers** property of **tests/TestCase** file.

```
    /**
     * List of Mocked Service Providers
     * @var MockServiceProvider[]
     */
    protected $providers = [
        'logger' => '\Test\Providers\LoggerServiceProvider',
```


© Ingram Micro, Inc. All Rights Reserved.

```
    'http' => '\Test\Providers\HttpServiceProvider'
];
```

The scope id has the format **YOUR-TEST-NAME.serviceId.json** where:

- **YOUR-TEST-NAME** is the filename of the test that is running.
- **serviceId** is the id of the service for example **logger** is the logger service id and http is the default Http service for the FulFillment API.

© Ingram Micro, Inc. All Rights Reserved.