php-common Documentation

Release 1.1.6

Sam Pratt

June 26, 2017

Contents

Contents:

CHAPTER 1

Overview

Requirements

- PHP 5.3+
- *composer*

All composer dependencies are included in the Expressly provided integration releases on GitHub.

Include

Inside your composer.json, you need to include:

```
"require": {
    "expressly/php-common": "2.4.1""
}
```

Or, run the command:

```
composer require expressly/php-common:2.4.1
```

Create a Client

In order to make use of the *DI* container, to include, override, or extend any code we provide, a new instance of the Expressly Client is required.

```
use Expressly;

/*
 * $merchantType is a string to help us identify your system type in logs.
 * Example constants in: Expressly\Entity\MerchantType
 */
$client = new Client($merchantType, array());
$app = $client->getApp();
```

Configuration

All configuration values associated with the repository are contained within *config.yml*.

To run the application in development mode, you can override the Expressly\Client parameters:

```
$client = new Expressly\Client(
    $merchantType,
    array(
        'external' => array(
            'hosts' => array(
                'default' => 'http://dev.expresslyapp.com/api/v2',
                'admin' => 'http://dev.expresslyapp.com/api/admin'
            )
        )
    )
);
```

You can override any part of the container configuration defined in *config.yml* using the above method.

Checklist

For a working Expressly integration, using this library, the following need to be checked off:

- *Include* the project via composer (or alternative).
- Change *Configuration* if needed (for development purposes).
- *Create a Client*.
- Create a localized MerchantProvider: must extend *MerchantProviderInterface*, and register it with the application:

```
use Expressly\Provider\MerchantProviderInterface;

class MyMerchantProvider implements MerchantProviderInterface {
    // your implementation
}

$app['merchant.provider'] = $app->share(function ($app) {
    return new MyMerchantProvider();
});
```

Roadmap

- Dependency drill down: restructure so *Pimple* is the base product instead of *Silex*.

CHAPTER 2

Requests

All dispatchable requests will use one of the following external hosts. Routing, and host definitions are defined in *config.yml*.

Ping

GET /api/admin/ping
Ping the API to see if the server is currently running.

Request Headers

- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "Server": "Live",
    "DB Status": "Live"
}
```

PHP Implementation Example:

```
use Expressly\Event\ResponseEvent;
use Expressly\Subscriber\UtilitySubscriber;

$event = new ResponseEvent();
$app['dispatcher']->dispatch(UtilitySubscriber::UTILITY_PING, $event);

if ($event->isSuccessful()) {

}
```

Register Merchant

POST /api/v2/plugin/merchant

Register a store with the server.

Request JSON Object

- **apiBaseUrl** (*string*) – see *merchant_url*; if there isn't any special base routing the value should be the exact same as the shop url
- **apiKey** (*string*) – api key retrieved from the *Portal* (see <https://buyexpressly.com/#/install>)

Request Headers

- Content-Type – application/json

Status Codes

- 204 No Content – Registered successfully
- 400 Bad Request – Invalid data/request

PHP Implementation Example:

```
use Expressly\Event\MerchantEvent;
use Expressly\Subscriber\MerchantSubscriber;

$event = new MerchantEvent(...);
$app['dispatcher']->dispatch(MerchantSubscriber::MERCHANT_REGISTER, $event);

if ($event->isSuccessful()) {

}
```

Get Campaign Migration Popup

GET /api/v2/migration/ (**string:** *uuid*)

Request the popup to start a campaign migration for the unique user.

Parameters

- **uuid** – Unique campaign migration uuid

Request Headers

- Content-Type – application/json
- Authorization – Basic token

Response Headers

- Content-Type – text/html

Status Codes

- 200 OK – campaign migration found, html for popup returned
- 400 Bad Request – Invalid data/request

PHP Implementation Example:

```
use Expressly\Event\CustomerMigrateEvent;
use Expressly\Subscriber\CustomerMigrationSubscriber;

$event = new CustomerMigrateEvent(...);
$app['dispatcher']->dispatch(CustomerMigrationSubscriber::CUSTOMER_MIGRATE_POPUP, $event);

if ($event->isSuccessful()) {

}
```

Get Campaign Migration Data

GET /api/v2/migration/ (**string:** *uuid*) **/user**
User has accepted popup, or been forced here directly; request, and start data migration.

Request Headers

- Content-Type – application/json
- Authorization – Basic token

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – Successfully returns user information
- 400 Bad Request – Invalid data/request

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "meta": {
        "locale": "UKR",
        "sender": "https://expresslyapp.com/api/v1/migration/{uuid}"
    },
    "data": {
        "email": "john.smith@gmail.com",
        "customerData": {
            "firstName": "John",
            "lastName": "Smith",
            "gender": "M",
            "billingAddress": 0,
            "shippingAddress": 1,
            "company": "Expressly",
            "dob": "1987-08-07",
            "taxNumber": "GB0249894821",
            "onlinePresence": [
                {
                    "field": "website",
                    "value": "http://www.myblog.com"
                }
            ],
            "dateUpdated": "2015-07-10T11:42:00+01:00",
```

```
                "emails": [
                    {
                        "email": "john.smith@gmail.com",
                        "alias": "default"
                    },
                    {
                        "email": "john@smithcorp.com",
                        "alias": "work"
                    }
                ],
                "phones": [
                    {
                        "type": "M",
                        "number": "020734581250",
                        "countryCode": 44
                    },
                    {
                        "type": "L",
                        "number": "020731443250",
                        "countryCode": 44
                    }
                ],
                "addresses": [
                    {
                        "firstName": "John",
                        "lastName": "Smith",
                        "address1": "12 Piccadilly",
                        "address2": "Room 14",
                        "city": "London",
                        "companyName": "WorkHard Ltd",
                        "zip": "W1C 34U",
                        "phone": 1,
                        "alias": "Work address",
                        "stateProvince": "LND",
                        "country": "GBR"
                    },
                    {
                        "firstName": "John C.",
                        "lastName": "Smith",
                        "address1": "23 Sallsberry Ave",
                        "address2": "Flat 3",
                        "city": "London",
                        "companyName": "",
                        "zip": "NW3 4HG",
                        "phone": 0,
                        "alias": "Home address",
                        "stateProvince": "LND",
                        "country": "GBR"
                    }
                ]
            },
            "cart": {
                "productId": "491",
                "couponCode": "20OFF"
            }
        }
    }
```

PHP Implementation Example:

```
use Expressly\Event\CustomerMigrateEvent;
use Expressly\Subscriber\CustomerMigrationSubscriber;

$event = new CustomerMigrateEvent(...);
$app['dispatcher']->dispatch(CustomerMigrationSubscriber::CUSTOMER_MIGRATE_DATA, $event);

if ($event->isSuccessful()) {

}
```

Migration Success

POST /api/v2/migration/ (**string:** *uuid*) **/success**

Tells the server if the migration was successful, or if the user already existed on this store.

Parameters

- **uuid** – Unique campaign migration uuid

Request JSON Object

- **status** (*enum*) – enum to tell server is migration was successful; can be: 'migrated', 'existing_customer'

Request Headers

- Content-Type – application/json
- Authorization – Basic token

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – Migration status acknowledged
- 400 Bad Request – Invalid data/request

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "success": "true",
    "msg": ""
}
```

PHP Implementation Example:

```
use Expressly\Event\CustomerMigrateEvent;
use Expressly\Subscriber\CustomerMigrationSubscriber;

$event = new CustomerMigrateEvent(...);
$app['dispatcher']->dispatch(CustomerMigrationSubscriber::CUSTOMER_MIGRATE_SUCCESS, $event);

if ($event->isSuccessful()) {
```

```
    }
```

Get Campaign Banner

GET /api/v2/banner/ (**string:** *uuid*) **?email=**
string: *email* If banner campaign is setup, get banner for a specified store, and email combination.

Parameters

- **uuid** – Unique banner uuid
- **email** – Email for the currently logged in user

Request Headers

- Content-Type – application/json
- Authorization – Basic token

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – Successfully found valid data for campaign banner
- 400 Bad Request – Invalid data/request

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bannerImageUrl": "https://buyexpressly.com/assets/banner/awesome-banner.jpg",
    "migrationLink": "https://www.myblog.com/expressly/api/3aff1880-b0f5-45bd-8f33-247f55981f2c
}
```

PHP Implementation Example:

```
use Expressly\Event\BannerEvent;
use Expressly\Subscriber\BannerSubscriber;

$event = new BannerEvent(...);
$app['dispatcher']->dispatch(BannerSubscriber::BANNER_REQUEST, $event);

if ($event->isSuccessful()) {

}
```

CHAPTER 3

Routes

All routes are dispatched using the included *RouteResolver*. If you platform has a built in parser, the associated regex is attached (statically) to each route in the Expressly\Routes namespace. The *RouteResolver* will check if the required header(s), method, and route were passed through, and return the valid data (if any) inside a matched *Route* object, or null.

```
$query = '/expressly/api/*';
// @type Expressly\Entity\Route|null
$route = $app['route.resolver']->process($query);
```

Every single expected endpoint will be prefixed with the registered *merchant_url*. All Endpoints that must have hooks created in the mother system have a corresponding *Presenter*.

Ping Store

GET /expressly/api/ping

Route class Expressly\Route\Ping

Simple response message to note that the plugin has been installed.

Request Headers

- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "expressly": "Stuff is happening!"
}
```

Show Popup

GET /expressly/api/ (**string:** *uuid*)

Route class Expressly\Route\CampaignPopup

Start the user migration process. This uri should invoke the *Get Campaign Migration Popup* request. The Popup can be shown over any page you wish, we recommend appending the html to your homepage.

Request Headers

- Authorization – Basic token

Response Headers

- Content-Type – text/html

Migrate User

GET /expressly/api/ (**string:** *uuid*) **/migrate**

Route class Expressly\Route\CampaignMigration

End of the user migration process. This uri should invoke the *Get Campaign Migration Data* request. The method should add all data for the provided user to the store, and if provided, add a product and/or coupon to the users' cart. The user should be logged in directly after this migration, and a welcome email (if not part of your stores' initial flow) should be dispatched.

Request Headers

- Authorization – Basic token

Response Headers

- Content-Type – text/html

Get User

GET /expressly/api/user/ (**string:** *email*)

Route class Expressly\Route\UserData

Returns user, via your application facilities, conforming to our defined *entities*.

Request Headers

- Authorization – Basic token
- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "meta": {
        "locale": "UKR",
        "sender": "https://yourstore.com/",
        "issuerData": []
    },
```

```
        "data": {
            "email": "john.smith@gmail.com",
            "customerData": {
                "firstName": "John",
                "lastName": "Smith",
                "gender": "M",
                "billingAddress": 0,
                "shippingAddress": 1,
                "company": "Expressly",
                "dob": "1987-08-07",
                "taxNumber": "GB0249894821",
                "onlinePresence": [
                    {
                        "field": "website",
                        "value": "www.myblog.com"
                    }
                ],
                "dateUpdated": "2015-07-10T11:42:00+01:00",
                "dateLastOrder": "2015-07-10T11:42:00+01:00",
                "numberOrdered": 5,
                "emails": [
                    {
                        "email": "john.smith@gmail.com",
                        "alias": "default"
                    },
                    {
                        "email": "john@smithcorp.com",
                        "alias": "work"
                    }
                ],
                "phones": [
                    {
                        "type": "M",
                        "number": "020734581250",
                        "countryCode": 44
                    },
                    {
                        "type": "L",
                        "number": "020731443250",
                        "countryCode": 44
                    }
                ],
                "addresses": [
                    {
                        "firstName": "John",
                        "lastName": "Smith",
                        "address1": "12 Piccadilly",
                        "address2": "Room 14",
                        "city": "London",
                        "companyName": "WorkHard Ltd",
                        "zip": "W1C 34U",
                        "phone": 1,
                        "alias": "Work address",
                        "stateProvince": "LND",
                        "country": "GBR"
                    },
                    {
                        "firstName": "John C.",
```

```
                        "lastName": "Smith",
                        "address1": "23 Sallsberry Ave",
                        "address2": "Flat 3",
                        "city": "London",
                        "companyName": "",
                        "zip": "NW3 4HG",
                        "phone": 0,
                        "alias": "Home address",
                        "stateProvince": "LND",
                        "country": "GBR"
                    }
                ]
            }
        }
    }
```

PHP Implementation Example:

```
    $customer = new Customer();
    /*
     * fill in as many applicable setters as possible
     * $customer
     *      ->setFirstName('John')
     *      ->setLastName('Smith');
     */
    $response = new CustomerMigratePresenter($merchant, $customer, $email, $id);
    // display content however your application prefers
    echo json_encode($response->toArray());
```

Invoices for Customer Purchases

`POST /expressly/api/batch/invoice`

Route class Expressly\Route\BatchInvoice

Given a list of date ranges, and emails checks to see if the associated campaign users have had any transactions during the specified period.

Request Headers

- Authorization – Basic token
- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example Request:

```
    POST /expressly/api/batch/invoice
    Host: prod.expresslyapp.com
    Authorization: Basic token

    {
        "customers": [
            {
                "email": "john.smith@gmail.com",
                "from": "2015-07-01T00:00:00+00:00",
```

```
                "to": "2015-08-01T00:00:00+00:00"
            }
        ]
    }
```

Example Response:

```
    HTTP/1.1 200 OK
    Content-Type: application/json

    {
        "invoices": [
            {
                "email": "john.smith@gmail.com",
                "orderCount": 1,
                "preTaxTotal": 100.00,
                "tax": 10.00,
                "orders": [
                    {
                        "id": "ORDER-5321311",
                        "date": "2015-07-10T11:42:00+01:00",
                        "itemCount": 2,
                        "coupon": "",
                        "currency": "GBP",
                        "preTaxTotal": 100.00,
                        "postTaxTotal": 110.00,
                        "tax": 10.00
                    }
                ]
            }
        ]
    }
```

PHP Implementation Example:

```
    use Expressly\Entity\Invoice;
    use Expressly\Entity\Order;
    use Expressly\Presenter\BatchInvoicePresenter;

    $invoices = array();

    foreach ($json->customers as $customer) {
        $invoice = new Invoice();
        $invoice->setEmail($customer->email);

        foreach ($userOrders as $userOrder) {
            $order = new Order();
            $order
                ->setId($userOrder->getId())
                ->setDate(new \DateTime($userOrder->getOrderDate())
                ->setItemCount($userOrder->getQuantity())
                ->setTotal($userOrder->getTotalPreTax(), $userOrder->getTax())
                ->setCoupon($userOrder->getCoupon());

            $invoice->addOrder($order);
        }

        $invoices[] = $invoice;
    }
```

```
    $presenter = new BatchInvoicePresenter($invoices);
    // display content however your application prefers
    echo json_encode($presenter->toArray());
```

Customers on Store

`POST /expressly/api/batch/customer`

Route class Expressly\Route\BatchCustomer

Given a list of emails, checks to see if a user has completed the migration process.

Request Headers

- Authorization – Basic token
- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example Request:

```
POST /expressly/api/batch/customer
Host: prod.expresslyapp.com
Authorization: Basic token

{
    "emails": [
        "john.smith@gmail.com"
    ]
}
```

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "existing": [
        "john.smith@gmail.com"
    ],
    "deleted": [],
    "pending": []
}
```

PHP Implementation Example:

```
use Expressly\Presenter\BatchCustomerPresenter;

$existingUsers = array();
$deletedUsers = array();
$pendingUsers = array();

foreach ($json->emails as $email) {
    // add user to certain sector of array, depending on state
}
```

```
    $presenter = new BatchCustomerPresenter($existingUsers, $deletedUsers, $pendingUsers);
    // display content however your application prefers
    echo json_encode($presenter->toArray());
```

CHAPTER 4

Exceptions

All exceptions caught within our code are logged to a *redis* server hosted by us.

GenericException

src/Exception/GenericException.php (ExpresslyExceptionGenericException)

CHAPTER 5

Flows

For all flows, the following Actors have been used:

SERVER Expressly API server;

STORE The store Expressly is being configured with;

CUSTOMER The customer interacting with the store.

Merchant Registration

Pre-conditions:

- Expressly plugin has been integrated, ticking all boxes of *Checklist*.
- API key exists, and has been created on the *Portal* (see <https://buyexpressly.com/#/install>).

Main Flow:

1. STORE calls *Register Merchant* with appropriate body, and header.
2. SERVER pings store to make sure you have the plugin installed correctly.
3. SERVER returns response successfully to STORE

Alternate Flows:

3-1. SERVER cannot ping STORE, returns error message.

2-2. SERVER received invalid credentials, returns error message.

User Campaign Migration

Pre-conditions:

- *Merchant Registration*.
- Campaign has been created on the *Portal*.

Main Flow:

1. CUSTOMER navigates to provided link with unique uuid attached (*Show Popup*).
2. STORE requests popup for unique uuid (*Get Campaign Migration Popup*).
3. SERVER returns popup html rendered for the given campaign, and CUSTOMER.

4. STORE renders html atop any given store page (e.g. homepage).
5. CUSTOMER accepts terms & conditions, and privacy policy provided by pressing 'ok'.
6. STORE navigates to *Migrate User*, and requests information.
7. SERVER returns information associated with CUSTOMER.
8. STORE adds customer to their store; adds product, and coupon (if provided, and supported) to cart.
9. STORE tells SERVER that CUSTOMER has been migrated correctly (*Migration Success*).
10. STORE logs user in, and navigates to homepage.

Alternate Flows:

7-1. CUSTOMER already exists, STORE tells SERVER that customer has been migrated previously (*Migration Success*). 8-1. STORE adds product, and coupon (if provided, and supported) to cart. 9-1. STORE shows CUSTOMER message that they already exist, asking if they want to go to the login page. 10-1. CUSTOMER accepts confirm message, and is redirected to the STORE login page.

Check Purchases

Pre-conditions:

- *Merchant Registration*.

Main Flow:

1. SERVER requests endpoint (*Invoices for Customer Purchases*) with JSON of emails, and date period to STORE.
2. STORE compares emails, and period to gather purchase information for given CUSTOMERs'.
3. STORE returns compiled data to SERVER.

Check Customer Migration

Pre-conditions:

- *Merchant Registration*.

Main Flow:

1. SERVER requests endpoint (*Customers on Store*) with JSON of emails to STORE.
2. STORE compares emails to determine whether CUSTOMER has been migrated.
3. STORE returns compiled data to SERVER.

Campaign Banner

Pre-conditions:

- *Merchant Registration*.
- Campaign for serving banners has been created on the *Portal*.
- CUSTOMER is logged in.

Main Flow:

1. STORE requests banner from SERVER (*Get Campaign Banner*).
2. SERVER returns image, and url.
3. STORE displays banner on page (in the location it was called from) on page render.
4. Banner clicked on, redirecting to associated route starting flow-migration off-site.

CHAPTER 6

Integrations

WooCommerce

WooCommerce repository

PrestaShop

PrestaShop repository

Magento

Magento repository

Magento 2

Magento 2 repository

OpenCart 1.5

OpenCart 1.5 repository

osCommerce 2.3

osCommerce 2.3 repository

VirtueMart

VirtueMart repository

CHAPTER 7

Need Help?

For any help with integration, questions, or bugs please contact info@buyexpressly.com

CHAPTER 8

License

The MIT License (MIT)

Copyright (c) 2015 expressly

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.

Bibliography

[composer] PHP package manager: https://getcomposer.org/

[config.yml] src/Resources/config/config.yml

[Silex] Silex PHP Microframework: http://silex.sensiolabs.org/

[Pimple] *DI* implementation: http://pimple.sensiolabs.org/

[DI] Dependency Injection

[MerchantProviderInterface] src/Provider/MerchantProviderInterface.php (ExpresslyProviderMerchantProviderInterface)

[config.yml] src/Resources/config/config.yml

[merchant_url] the location to execute/catch our paths; example: https://www.example.com/route?action=/expressly/api/ping

[Portal] Expressly Portal: https://buyexpressly.com/#/portal/login

[RouteResolver] src/Resolver/RouteResolver (namespace ExpresslyResolverRouteResolver)

[merchant_url] the location to execute/catch our paths; example: https://www.example.com/route?action=/expressly/api/ping

[Route] src/Entity/Route (namespace ExpresslyEntityRoute)

[Presenter] src/Presenter (namespace ExpresslyPresenter)

[entities] src/Entity (namespace ExpresslyEntity)

[redis] NoSQL data store: http://redis.io/

[Portal] Expressly management Portal: https://buyexpressly.com/#/portal/login

/api

/expressly