AMQP PHP Consumer Documentation

Release 1.0

reBuy reCommerce GmbH

Aug 03, 2018

Contents

This is the documentation for the amqp-php-consumer library.

This library allows you to define consumers for AMQP with doctrine annotations. For consuming messages the AMQP-Library videlalvaro/php-amqplib is used.

Contents:

CHAPTER 1

Getting started

1.1 Installation

The amqp-php-consumer library is available on Packagist. You can install it using Composer:

```
$ composer require rebuy/amqp-php-consumer
```

Note: This library follows Semantic Versioning. Except for major versions, we aim to not introduce BC breaks in new releases. You should still test your application after upgrading though. What is a bug fix for somebody could break something for others when they where (probably unawares) relying on that bug.

1.2 Configuration

There are two things you need to do to get started:

1. *create one ore more consumer*
2. *create a consumer manager*

CHAPTER 2

Creating Consumers

Let's assume you have an amqp message which will be published when an order has been created. This message has the routing key `order-created` with the payload `{"order_id": $SOME_ID}`. In this example we create a consumer which sends an email to the customer when this message will be published.

First of all, you have to create a PHP class which represents this message:

```
namespace My\Consumer;

use JMS\Serializer\Annotation\Type;
use Rebuy\Amqp\Consumer\Message\MessageInterface;

class OrderCreatedMessage implements MessageInterface
{
    /**
     * @Type("integer")
     *
     * @var int
     */
    public $orderId;

    public static function getRoutingKey()
    {
        return 'order-created';
    }
}
```

Note: Since this library uses the jms/serializer component to deserialize the payload for all messages, we have to define a `@Type` for the property `$orderId`.

With this message we are able to create our consumer which will send an email to the customer:

```
class OrderConsumer
{
```

(continues on next page)

(continued from previous page)

```
    private $orderService;
    private $emailService;

    public function __construct($orderService, $emailService)
    {
        $this->orderService = $orderService;
        $this->emailService = $emailService;
    }

    /**
     * @Consumer(name="order-created-send-email")
     */
    public function sendMail(OrderCreatedMessage $message)
    {
        $order = $this->orderService->loadOrder($message->orderId);
        $this->emailService->sendOrderCreatedEmail($order);
    }
}
```

Note: You can create multiple consumers which consume the same message, but they must use a different name, otherwise an `ConsumerException` is thrown.

Now that you have created a consumer, you can go on to the next section and create the *consumer manager*

CHAPTER 3

Create and Configure the ConsumerManager

The consumer manager is responsible for registering a consumer and starting the consuming process.

3.1 Create AMQP Connection

You need to create an AMQP connection with an AMQP channel which will then be used by the comsuner manager:

```
$connection = new PhpAmqpLib\Connection\AMQPSocketConnection('localhost', 5672,
↪'username', 'password');
$channel = $connection->channel();

$passive = false;
$durable = true;
$autoDelete = false;
$type = 'topic';

$channel->exchange_declare('your-exchange-name', $type, $passive, $durable,
↪$autoDelete);
```

If you need other values than the ones defined, feel free to adjust them, but it is necessary to declare the exchange before you can go on.

3.2 Create a JMS Serializer

In order to deserialize the payload of an AMQP message we have to create a Serializer object The easiest way to do so is by using the `SerializerBuilder` from the JMS library:

```
use Rebuy\Amqp\Consumer\Serializer\JMSSerializerAdapter;
use JMS\Serializer\SerializerBuilder;

```

(continues on next page)

(continued from previous page)

```
$serializer = SerializerBuilder::create()->build();
$serializerAdapter = new JMSSerializerAdapter($serializer);
```

If you'd rather want to use the symfony serializer, do the following:

```
use Rebuy\Amqp\Consumer\Serializer\SymfonySerializerAdapter;
use Symfony\Component\Serializer\Serializer;
use Symfony\Component\Serializer\Encoder\XmlEncoder;
use Symfony\Component\Serializer\Encoder\JsonEncoder;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;

$encoders = [new XmlEncoder(), new JsonEncoder()];
$normalizers = [new ObjectNormalizer()];

$serializer = new Serializer($normalizers, $encoders);
$serializerAdapter = new SymfonySerializerAdapter($serializer);
```

3.3 Create the Annotation Parser

The annotation parser is responsible for parsing all the consumer annotations and creating a ConsumerContainer. The container is an abstraction of the consumer method and holds all information which are necessary to consume the message:

```
$reader = new Doctrine\Common\Annotations\AnnotationReader();
$parser = new Rebuy\Amqp\Consumer\Annotation\Parser($reader);
```

Tip: You can also use a FileCacheReader instead of the AnnotationReader. Example: `$reader = new FileCacheReader(new AnnotationReader(), '/path/to/cache');`

3.4 Tying it all together

We have now everything we need to create the consumer manager, register consumers and start the consuming process:

```
$manager = new Rebuy\Amqp\Consumer\ConsumerManager($channel, $exchangeName,
↪$serializerAdapter, $parser);
$manager->registerConsumer(new MyConsumer());

$manager->wait()
```

Caution: The consuming process might stop under the following conditions:

- An exception in one of the consumers is thrown
- No message has been processed in the last 900 seconds (this value can be altered with the method `ConsumerManager#setIdleTimeout`)

Note: The `wait` method is a blocking process. This method waits for new messages and passes every message to its desired consumer (if one exists).

CHAPTER 4

Events

There are currently two events dispatched when consuming a message:

- `Rebuy\Amqp\Consumer\ConsumerEvents::PRE_CONSUME`: Before the message is consumed
- `Rebuy\Amqp\Consumer\ConsumerEvents::POST_CONSUME`: After the message has been consumed

These events are dispatched by an symfony2 event dispatcher. If you want to listen to one of these events, you have to create a subsriber/listener, add it to the event dispatcher and set the dispatcher to the manager:

```
$dispatcher = new Symfony\Component\EventDispatcher\EventDispatcher();
$dispatcher->addListener(Rebuy\Amqp\Consumer\ConsumerEvents::PRE_CONSUME,
↪$myListener);
$dispatcher->addSubscriber(new MySubscriber());

$manager = new Rebuy\Amqp\Consumer\ConsumerManager(...);
$manager->setEventDispatcher($dispatcher);
```

4.1 Implemented Subscriber

Some useful subscribers are already shipped with this library:

- *TimingSubscriber*: Uses symfony/stopwatch and league/statsd for writing timing metrics to statds
- *LogSubscriber*: Uses a LoggerInterface to log a debug message for every consumed message

CHAPTER 5

Error Handlers

You can register several error handlers which will be called when an exception in the consuming process is thrown. Every error handler must implement the interface `Rebuy\Amqp\Consumer\Handler\ErrorHandlerInterface`, this interface only requires one method `handle(ConsumerContainerException $ex)`.

An error handler can be registered in the following way:

```
$manager = new Rebuy\Amqp\Consumer\ConsumerManager(...);
$manager->registerErrorHandler(new MyErrorHandler());
```

Danger: As soon as one error handler is registered, the consuming of the message is considered successful. If you want to stop the consuming process, you must throw the passed exception (or an own exception) by yourself.

5.1 Implemented error handlers

Currently there are two error handlers implemented in this library:

- *RequeuerHandler*: Requeues the message so it can be processed at a later time
- *LoggerHandler*: Uses a LoggerInterface to log a warning message (this handler is only useful in combination with the *RequeuerHandler*)

CHAPTER 6

Contributing

We are happy for contributions. Before you invest a lot of time however, best open an issue on github to discuss your idea. Then we can coordinate efforts if somebody is already working on the same thing.

6.1 Testing the Library

This chapter describes how to run the tests that are included with this library.

First clone the repository, install the vendors, then run the tests:

```
$ git clone https://github.com/rebuy/amqp-php-consumer.git
$ cd amqp-php-consumer
$ composer install --dev
$ bin/phpunit
```

6.2 Building the Documentation

First install Sphinx, then build the docs:

```
$ cd doc
$ make html
```