By https://andreiabohner.org

Messenger

Symfony 4.4

Install

```
$ composer require symfony/messenger
```

HEADS UP!

Middleware are called twice:
- *when a message is dispatched*
- *when the worker receives a message from the transport, it passes that message back into the bus and the middleware are called again*

Middleware

Implement MiddlewareInterface

Can access the message and its wrapper (the envelope) while it's dispatched through the bus

You can create your own middleware!

Built-in middlewares:

`SendMessageMiddleware`	Enables async processing, logs the processing of messages if a logger is passed.
`HandleMessageMiddleware`	Calls the registered handler(s).

Envelope

Add metadata or some config to the message

Wrap messages into the message bus, allowing to add useful info inside through envelope stamps

When you pass your message to the bus, internally, it gets wrapped inside an Envelope

Put a message in a Envelope

```
$envelope = new Envelope($message, [
    new DelayStamp(9000)
]);

$messageBus->dispatch($envelope);
```

Add Stamps

Stamp

Implement StampInterface

Attach extra config to the envelope

Piece of info to attach to the message: any sort of metadata/config that the middleware or transport layer may use.

To see on the web debug toolbar which stamps have been applied to the envelope, use dump():

```
dump($messageBus->dispatch($envelope));
```

Message Bus

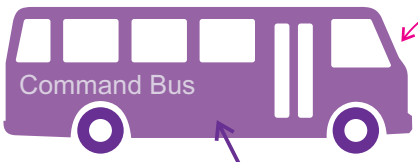
Dispatch messages *to the*

Handler

a PHP callable

Handle messages using the business logic you defined.
Called by `HandleMessageMiddleware`

The bus is a collection of middleware



Can be:
- *Command Bus*
  Usually don't provide any results. Should have exactly one handler.
- *Query Bus*
  Used to get info back from the handler. Rarely async.
- *Event Bus*
  Dispatched after something happens. Can have zero to many handlers.

Dispatch the Envelope (with the message) back to the message bus

Execute middleware again!

Can have multiple middleware

Transport

Consume messages from a transport (reads a message off of a queue)

Transport's serializer transforms into an Envelope object with the message object inside

Used to send & receive messages that will not be handled immediatly. Usually a queue & will be responsible for communicating with a message broker or 3rd parties

Worker

```
$ php bin/console messenger:consume async
```

Built-in Stamps

`SerializerStamp`	configure the serialization groups used by the transport
`ValidationStamp`	configure the validation groups used when the validation middleware is enabled
`ReceivedStamp`	added when a message is received from a transport
`SentStamp`	marks the message as sent by a specific sender. Allows accessing the sender FQCN & the alias if available from the SendersLocator
`SentToFailureTransportStamp`	applied when a message is sent to the failure transport
`HandledStamp`	marks the message as handled by a specific handler. Allows accessing the handler returned value and the handler name
`DelayStamp`	delay delivery of your message on a transport
`RedeliveryStamp`	applied when a messages needs to be redelivered
`BusNameStamp`	used to identify which bus it was passed to
`TransportMessageIdStamp`	added by a sender or receiver to indicate the id of this message in that transport

Messenger

Run async code with queues & workers

By https://andreiabohner.org

Symfony 4.4

Create a Message

Class that holds data

HEADS UP!

Pass always the smallest amount of info to the message!

E.g.: if you need to pass a Doctrine entity in a message, pass the entity's primary key (or any other relevant info, e.g. email)

```php
// src/Message/SmsNotification.php
namespace App\Message;

class SmsNotification
{
    private $content;

    public function __construct(string $content, int $userId)
    {
        $this->content = $content;
        $this->userId = $userId;
    }

    public function getUserId(): int
    {
        return $this->userId;
    }

    public function getContent(): string
    {
        return $this->content;
    }
}
```

Create a Handler

Class that will be called when the message is dispatched. Read the message class & perform some task.

```php
// src/MessageHandler/SmsNotificationHandler.php
namespace App\MessageHandler;

use App\Message\SmsNotification;
use App\Repository\UserRepository;
use Symfony\Component\Messenger\Handler\MessageHandlerInterface;

class SmsNotificationHandler implements MessageHandlerInterface
{
    private $userRepository;

    public function __construct(UserRepository $userRepository)
    {
        $this->userRepository = $userRepository;
    }

    public function __invoke(SmsNotification $message)
    {
        $user = $this->userRepository->find($message->getUserId());

        // ...
    }
}
```

Tells Symfony that this is a message handler

The handler should:

1 - implement MessageHandlerInterface
2 - create an __invoke() method with one argument that's type-hinted with the message class (or interface)

Follow these 2 rules & Symfony will find & register the handler automatically

Query for a fresh object here, on the handler

Manually Configuring a Handler

```yaml
# config/services.yaml
services:
   App\MessageHandler\SmsNotificationHandler:
       tags: [{ name: messenger.message_handler, bus: command.bus }]

       # or configure with options
       tags:
           -
               name: messenger.message_handler
               handles: App\Message\SmsNotification
               bus: command.bus

       autoconfigure: false
```

Restrict this handler to the command bus

Options:
- bus
- from_transport
- handles
- method
- priority

Only needed if can't be guessed by type-hint

Prevent handlers from being registered twice

By https://andreiabohner.org

Messenger

Symfony 4.4

HEADS UP!

By default all messages are sent sync (i.e: as soon as they are dispatched)

unless you configure a route/transport for it so you can send async/queued messages

Dispatch the Message

```php
use Symfony\Component\Messenger\MessageBusInterface;

class DefaultController extends AbstractController
{
    public function index(MessageBusInterface $bus)
    {
        $bus->dispatch(new SmsNotification('A message!'));

        // or use the shortcut
        $this->dispatchMessage(new SmsNotification('A message!'));
        // ...
```

Call the bus: inject the message_bus service (via MessageBusInterface), e.g. in a controller

Call SmsNotificationHandler

If you pass a raw message here, by default, the dispatch() method wraps it in an Envelope

Route Messages to a Transport

For async/queued messages!

A transport is registered using a "DSN" in your .env file

```
#.env
# MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
# MESSENGER_TRANSPORT_DSN=doctrine://default
# MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
```

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            async: '%env(MESSENGER_TRANSPORT_DSN)%'
            async_priority_high:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
                options:
                    queue_name: high

                    #exchange:
                    #    name: high
                    #queues:
                    #    messages_high: ~
                    # or redis try "group"
            async_priority_low:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
                options:
                    queue_name: low

        routing:
            'App\Message\SmsNotification': async

            'App\Message\AbstractAsyncMessage': async
            'App\Message\AsyncMessageInterface': async

            'My\Message\ToBeSentToTwoSenders': [async, async_high_priority]
```

If you want the message not to be sent immediately configure a transport to send to it (a transport tells where to send/read messages)

Could be any desired name

queue_name is specific to the doctrine transport

For amqp send to a separate exchange then queue

Send messages to transports & its handler(s)

Route all messages that extend this base class to async transport

Add all your message class that need to be async here

Route all messages that implement this interface to async transport

Send messages to multiple transports

Consume Messages (Run the Worker)

A command that "handles" messages from a queue is called a "worker"

By default, the command will run forever: looking for new messages on your transport and handling them

This is your worker!

```
$ php bin/console messenger:consume async -vv
```

-vv show details about what is happening

Name of the transport you defined

This is how to Prioritize Transports

```
$ php bin/console messenger:consume async_priority_high async_priority_low
```

A worker can read from one or many transports

Instruct the worker to handle messages in a priority order:
The worker will always first look for messages waiting on async_priority_high
If there are none, then it will consume messages from async_priority_low

Use CQRS - Command Query Responsibility Segregation

Messenger

By https://andreiabohner.org

Symfony 4.4

Messenger Configuration

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        failure_transport: failed
        default_bus: command.bus
        buses:
            command.bus:
                middleware:
                    - doctrine_ping_connection
                    - doctrine_close_connection
                    - doctrine_transaction
                    - doctrine_clear_entity_manager
            query.bus:
                middleware:
                    - validation
            event.bus:
                default_middleware: allow_no_handlers
                middleware:
                    - validation

        routing:
            'App\Message\MyMessage': amqp

        transports:
            amqp: enqueue://default
            async:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
            failed:
                dsn: 'doctrine://default?queue_name=failed'
            async_priority_high:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
                retry_strategy:
                    max_retries: 3
                    delay: 1000
                    multiplier: 2
                    max_delay: 0
                    # service: null

        serializer: messenger.transport.symfony_serializer

    serializer:
        default_serializer: messenger.transport.symfony_serializer
        symfony_serializer:
            format: json
            context: { }
```

Send failed messages to the transport defined here (for later handling)

The bus that is going to be injected when injecting `MessageBusInterface`

autowireable with the `MessageBusInterface` type-hint (because this is the `default_bus`)

In case of error, a new connection is opened

The connection is open before your handler & closed immediately afterwards instead of keeping it open forever

Wrap your handler in a single Doctrine transaction so your handler doesn't need to call flush(). An error will rollback automatically

Cleans the entity manager before sending it to your handler

autowireable with `MessageBusInterface $queryBus`

autowireable with `MessageBusInterface $eventBus`

validate the message object itself through Symfony's validator

Route messages that have to go through the message queue

Supports multiple transports, e.g.:
- doctrine
- inmemory
- redis
- sync

Send messages (e.g. to a queueing system) & receive them via a worker

If handling a message fails 3 times (default max_retries), it will then be sent to the `failed` transport

Configuring Retries & Failures for this transport

If a message fails it's retried multiple times (max_retries) & then will be discarded

Milliseconds delay

Causes the delay to be higher before each retry. E.g. 1 second delay, 2 seconds, 4 seconds

Override all of this with a service that implements `Symfony\Component\Messenger\Retry\RetryStrategyInterface`

Define the serializer for this transport

Define the global serializer. When messages are sent/received to a transport, they're serialized using PHP's native serialize() & unserialize() functions by default

Built-in service that uses the Serializer component

Handling Messages Synchronously using a Transport

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            # ... other transports
            sync: 'sync://'
        routing:
            App\Message\SmsNotification: sync
```

Messenger

send & receive messages to/from other apps or via message queues

By https://andreiabohner.org

Symfony 4.4

Transports

Each transport has a number of different connection options and there are 2 ways to pass them:

1 - via the DSN, as query parameters

```
# .env
MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages

# or with options:
MESSENGER_TRANSPORT_DSN=redis://password@localhost:6379/messages/symfony/consumer?
auto_setup=true&
serializer=1&
stream_max_entries=0&
dbindex=0
```

2 - via the options key under the transport in messenger.yaml

```
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            async: "%env(MESSENGER_TRANSPORT_DSN)%"
                dsn: "%env(MESSENGER_TRANSPORT_DSN)%"
                options:
                    auto_setup: true
                    serializer: 1
                    stream_max_entries: 0
                    dbindex: 0
```

HEADS UP!

Options defined under "options" key take precedence over ones defined in the DSN

AMPQ

Need the AMQP PHP extension

```
# .env
MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
```

```
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            async:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
                options:
                    exchange:
                        name: messages
                        type: direct
                        default_publish_routing_key: normal
                queues:
                    messages_normal:
                        binding_keys: [normal]
```

It's possible to configure AMQP-specific settings on your message by adding AmqpStamp to your Envelope:

```
use Symfony\Component\Messenger\Transport\AmqpExt\AmqpStamp;
// ...

$attributes = [];
$bus->dispatch(new SmsNotification(), [
    new AmqpStamp('routing-key-name', AMQP_NOPARAM, $attributes)
]);
```

AmqpStamp

Options

host	Hostname of the AMQP service.
port	Port of the AMQP service.
vhost	Virtual Host to use with the AMQP service.
user	Username to connect the the AMQP service.
password	Password to connect to the AMQP service.
auto_setup	Enable or not the auto-setup of queues & exchanges. (Default: true)
prefetch_count	Set channel prefetch count.
queues:	
queues[name]	An array of queues, keyed by the name.
binding_keys	The binding keys (if any) to bind to this queue.
binding_arguments	Arguments to be used while binding the queue.
flags	Queue flags. (Default: AMQP_DURABLE)
arguments	Extra arguments.
exchange:	
name	Name of the exchange. (Default: messages)
type	Type of exchange. Possible types: *fanout, direct, topic, header.* (Default: fanout)
default_publish_routing_key	Routing key to use when publishing, if none is specified on the message.
flags	Exchange flags. Possible flags: *AMQP_DURABLE, AMQP_PASSIVE, AMQP_AUTODELETE, ...* (Default: AMQP_DURABLE)
arguments	Extra arguments.
delay:	
queue_name_pattern	Pattern to use to create the queues. (Default: "delay_%exchange_name%_%routing_key%_%delay%")
exchange_name	Name of the exchange to be used for the delayed/retried messages. (Default: "delays")

Doctrine

Use to store messages in a database table

```
# .env
MESSENGER_TRANSPORT_DSN=doctrine://default

# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            async_priority_high: '%env(MESSENGER_TRANSPORT_DSN)%?queue_name=high_priority'
            async_normal:
                dsn: '%env(MESSENGER_TRANSPORT_DSN)%'
                options:
                    queue_name: normal_priority
```

Connection name

When the transport is first used it will create a table named messenger_messages

Options

table_name	Name of the table. (Default: messenger_messages)
queue_name	Name of the queue (a column in the table, to use one table for multiple transports). (Default: default)
redeliver_timeout	Timeout before retrying a message that's in the queue but in the "handling" state (if a worker died for some reason, this will occur, eventually you should retry the message) - in seconds. (Default: 3600)
auto_setup	If the table "messenger_messages" should be created automatically during send/get. (Default: true)

By https://andreiabohner.org

Messenger

Symfony 4.4

Redis

Use streams to queue messages. Need the Redis PHP extension (>=4.3) & a Redis server (^5.0)

```
# .env
MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
# Full DSN Example
MESSENGER_TRANSPORT_DSN=redis://password@localhost:6379/messages/symfony/consumer?
                        auto_setup=true&
                        serializer=1&
                        stream_max_entries=0&
                        dbindex=0
```

Options

stream	Redis stream name. (Default: messages)
group	Redis consumer group name. (Default: symfony)
consumer	Consumer name used in Redis. (Default: consumer)
auto_setup	Create the Redis group automatically? (Default: true)
auth	Redis password.
serializer	How to serialize the final payload in Redis (Redis::OPT_SERIALIZER option). (Default: Redis::SERIALIZER_PHP)
stream_max_entries	Maximum number of entries which the stream will be trimmed to. Set it to a large enough number to avoid losing pending messages (which means "no trimming").
dbindex	

In-memory

Useful for tests!

Doesn't actually delivery messages. Instead, it holds them in memory during the request. All in-memory transports will be reset automatically after each test in test classes extending KernelTestCase or WebTestCase.

Use this config only in test environment

```
# config/packages/test/messenger.yaml
framework:
    messenger:
        transports:
            async_priority_normal: 'in-memory://'
```

```
use Symfony\Component\Messenger\Transport\InMemoryTransport;

class ImagePostControllerTest extends WebTestCase
{
    public function testCreate()
    {
        $transport = self::$container->get('messenger.transport.async_priority_normal');
        $this->assertCount(1, $transport->get());
    }
}
```

Method to call on a transport to get the sent, or "queued" messages

Sync

Useful when developing!

Instead of sending each message to an external queue, it just handles them immediately. They're handled synchronously.

Use this config only in dev environment

```
# config/packages/dev/messenger.yaml
framework:
    messenger:
        transports:
            async: 'sync://'
            async_priority_high: 'sync://'
```

Console

```
$ php bin/console messenger:consume async
```

Run the consumer (worker). Fetch & deserialize each message back into PHP, then pass it to the message bus to be handled.

--time-limit=3600	Run the command for xx minutes and then exit.
--memory-limit=128M	Exit once its memory usage is above a certain level.
--limit =20	Run a specific number of messages and then exit.

```
$ php bin/console debug:messenger
```

List available messages and handlers per bus.

```
$ php bin/console messenger:setup-transports
```

Configure the transports. E.g. table name in doctrine transport.

```
$ php bin/console messenger:stop-workers
```

Sends a signal to stop any messenger:consume processes that are running. Each worker command will finish the message they are currently processing and then exit.

Worker commands are **not** automatically restarted.

Failed transport

These commands are available when the "failure_transport" is configured

```
$ php bin/console messenger:failed:show
```

See all messages in the failure transport.

```
$ php bin/console messenger:failed:show 20 -vv
```

See details about a specific failure.

```
$ php bin/console messenger:failed:retry -vv
```

View and retry messages one-by-one.

```
$ php bin/console messenger:failed:retry -vv --force
```

View and retry messages one-by-one without asking.

```
$ php bin/console messenger:failed:retry 20 30 --force
```

Retry specific messages.

```
$ php bin/console messenger:failed:remove 20
```

Remove a message without retrying it.

Messenger

Symfony 4.4

Install Enqueue

```
$ composer require sroze/messenger-enqueue-transport
```

Other Transports (Available via Enqueue)

Enable Enqueue

```
// config/bundles.php
return [
    // ...
    Enqueue\MessengerAdapter\Bundle\EnqueueAdapterBundle::class => ['all' => true],
];
```

Enable the bundle

```
enqueue://default
    ?queue[name]=queue_name
    &topic[name]=topic_name
    &deliveryDelay=1800
    &delayStrategy=Enqueue\AmqpTools\RabbitMqDelayPluginDelayStrategy
    &timeToLive=3600
    &receiveTimeout=1000
    &priority=1
```

Enqueue extra options

Default configuration

```
# config/packages/enqueue.yaml
enqueue:
    transport:                  # Accept a string DSN, an array with DSN key, or null
        dsn:                    ~
        connection_factory_class: ~     # Should implement "Interop\Queue\ConnectionFactory"
        factory_service:        ~       # Should implement "Enqueue\ConnectionFactoryFactoryInterface"
        factory_class:          ~
    consumption:
        receive_timeout:        10000   # Time in milliseconds queue consumer waits for a message (default: 100ms)
    client:
        traceable_producer:     true
        prefix:                 enqueue
        separator:              .
        app_name:               app
        router_topic:           default
        router_queue:           default
        router_processor:       null
        redelivered_delay_time: 0
        default_queue:          default
        driver_options:         []      # Contains driver specific options
    monitoring:                 # Accept a string DSN, an array with DSN key, or null.
        dsn:                    ~
        storage_factory_service: ~      # Should implement "Enqueue\Monitoring\StatsStorageFactory"
        storage_factory_class: ~
    async_commands:
        enabled:                false
        timeout:                60
        command_name:           ~
        queue_name:             ~
    job:
        enabled:                false
    async_events:
        enabled:                false
    extensions:
        doctrine_ping_connection_extension: false
        doctrine_clear_identity_map_extension: false
        doctrine_odm_clear_identity_map_extension: false
        doctrine_closed_entity_manager_extension: false
        reset_services_extension: false
        signal_extension:       true
        reply_extension:        true
```

MQ broker DSN, E.g. amqp, sqs, gps, ...

Stats storage DSN. Schemes supported: "wamp", "ws", "influxdb"

Messenger

Symfony 4.4

Amazon SQS

Install Enqueue SQS

```
$ composer require enqueue/sqs
```

```
// .env
SQS_DSN=sqs:?key=<SQS_KEY>&secret=<SQS_SECRET>&region=<SQS_REGION>
SQS_QUEUE_NAME=my-sqs-queue-name
```

Add the SQS DSN

Add the queue name as an env var

```
// config/packages/messenger.yaml
framework:
    messenger:
        transports
            async:
                dsn: 'enqueue://default'
                options:
                    receiveTimeout: 20
                    queue:
                        name: '%env(resolve:SQS_QUEUE_NAME)%'
```

Add custom settings for SQS transport

Options

key	*AWS credentials.* If no credentials are provided, the SDK will attempt to load them from the environment. (Default: null)
secret	*AWS credentials.* If no credentials are provided, the SDK will attempt to load them from the environment. (Default: null)
token	*AWS credentials.* If no credentials are provided, the SDK will attempt to load them from the environment. (Default: null)
region	*(string, required)* Region to connect to. See http://docs.aws.amazon.com/general/latest/gr/rande.html for a list of available regions. (Default: null)
retries	*(int)* Configures the maximum number of allowed retries for a client (pass 0 to disable retries). (Default: 3)
version	*(string, required)* The version of the webservice to utilize. (Default: '2012-11-05')
lazy	Enable lazy connection. (Default: true)
endpoint	*(string)* The full URI of the webservice. This is only required when connecting to a custom endpoint e.g. localstack (Default: null)
queue_owner_aws_account_id	The AWS account ID of the account that created the queue.

Google Pub/Sub

Install Enqueue Google Pub/Sub

```
$ composer require enqueue/gps
```

```
// .env
MESSENGER_TRANSPORT_DSN=enqueue://gps?projectId=projdev&emulatorHost=http%3A%2F%2Fgoogle-pubsub%3A8085
```

```
// config/packages/messenger.yaml
framework:
    messenger:
        transports
            async:
                dsn: 'enqueue://gps'
                options:
                    projectId: '%env(GOOGLE_PROJECT_ID)%'
                    keyFilePath: '%env(GOOGLE_APPLICATION_CREDENTIALS)%'
```

Options

projectId	The project ID from the Google Developer's Console.
keyFilePath	The full path to your service account credentials.json file retrieved from the Google Developers Console.
retries	Number of retries for a failed request. (Default: 3)
scopes	Scopes to be used for the request.
emulatorHost	The endpoint used to emulate communication with GooglePubSub.
lazy	The connection will be performed as later as possible, if the option set to true.

Apache Kafka

Install Enqueue Kafka

```
$ composer require enqueue/rdkafka
```

```
// .env
MESSENGER_TRANSPORT_DSN=enqueue://default
KAFKA_BROKER_LIST=node-1.kafka.host:9092,node-2.kafka.host:9092,node-3.kafka.host:9092
```