# GearmanBundle

***Release***

**Jul 25, 2017**

# Contents


**1 User Documentation** **3**
- 1.1 Installing/Configuring . . . 3
- 1.2 Configuration . . . 4
- 1.3 Definition of Workers . . . 6
- 1.4 Running your jobs . . . 10
- 1.5 Client . . . 13
- 1.6 Kernel Events . . . 15
- 1.7 Customize . . . 18
- 1.8 Cache . . . 19
- 1.9 Faqs . . . 19

**2 CookBook** **21**
- 2.1 Job Status . . . 21

GearmanBundle is a bundle for Symfony2 intended to provide an easy way to support developers who need to use job queues. For example: mail queues, Solr generation queues or Database upload queues.

CHAPTER 1

# User Documentation

## Installing/Configuring

### Tags

- Use last unstable version ( alias of `dev-master` ) to stay always in last commit
- Use last stable version tag to stay in a stable release.
-

**Note:** As long as Symfony2 versions 2.1 and 2.2 are not maintained anymore, and as long as these branches had same code than master branch, they all have been deleted

### Installing Gearman

To install Gearman Job Server with `apt-get` use the following commands:

```
$ sudo apt-get install gearman-job-server
```

And start server

```
$ service gearman-job-server start
```

Then you need to install **Gearman driver** using the following commands

```
$ pecl install channel://pecl.php.net/gearman-X.X.X
```

You will find all available gearman versions in Pear Repository Finally you need to start php module

```
$ echo "extension=gearman.so" > /etc/php5/conf.d/gearman.ini
```

### Installing GearmanBundle

You have to add require line into you composer.json file

```
"require": {
   "php": ">=5.3.3",
   "symfony/symfony": "2.4.*",

   "mmoreram/gearman-bundle": "dev-master"
}
```

Then you have to use composer to update your project dependencies

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

And register the bundle in your appkernel.php file

```
return array(
   // ...
   new Doctrine\Bundle\DoctrineCacheBundle\DoctrineCacheBundle(),
   new Mmoreram\GearmanBundle\GearmanBundle(),
   // ...
);
```

## Configuration

We must configure our Worker. Common definitions must be defined in config.yml file, setting values for all installed Workers. Also we must config gearman cache, using doctrine cache.

---

**Note:** If `iterations` value is 0, worker will not kill itself never, so thread will be alive as long as needed. The reason to allow workers to kill themselves is just to prevent each process to accumulate a large quantity of memory.

---

```
doctrine_cache:
    providers:
        gearman_cache:
            type: file_system
            namespace: doctrine_cache.ns.gearman

gearman:
   # Bundles will parsed searching workers
   bundles:
      # Name of bundle
      AcmeBundle:

         # Bundle name
         name: MmoreramerinoTestBundle

         # Bundle search can be enabled or disabled
         active: true

         # If any include is defined, Only these namespaces will be parsed
         # Otherwise, full Bundle will be parsed
         include:
```

```
            - Services
            - EventListener

        # Namespaces this Bundle will ignore when parsing
        ignore:
            - DependencyInjection
            - Resources

  # default values
  # All these values will be used if are not overwritten in Workers or jobs
  defaults:

      # Default method related with all jobs
      # do // deprecated as of pecl/gearman 1.0.0. Use doNormal
      # doNormal
      # doBackground
      # doHigh
      # doHighBackground
      # doLow
      # doLowBackground
      method: doNormal

      # Default number of executions before job dies.
      # If annotations defined, will be overwritten
      # If empty, 0 is defined by default
      iterations: 150

      # Default amount of time in seconds required for the execution to run.
      # This is useful if using a tool such as supervisor which may expect a command␣
↪to run for a
      # minimum period of time to be considered successful and avoid fatal␣
↪termination.
      # If empty, no minimum time is required
      minimum_execution_time: null

      # Default maximum amount of time in seconds for a worker to remain idle before␣
↪terminating.
      # If empty, the worker will never timeout
      timeout: null

      # execute callbacks after operations using Kernel events
      callbacks: true

      # Prefix in all jobs
      # If empty name will not be modified
      # Useful for rename jobs in different environments
      job_prefix: null

      # Autogenerate unique key in jobs/tasks if not set
      # This key is unique given a Job name and a payload serialized
      generate_unique_key: true

      # Prepend namespace when callableName is built
      # By default this variable is set as true
      workers_name_prepend_namespace: true

  # Server list where workers and clients will connect to
  # Each server must contain host and port
```

```
   # If annotations defined, will be full overwritten
   #
   # If servers empty, simple localhost server is defined by default
   # If port empty, 4730 is defined by default
   servers:
      localhost:
         host: 127.0.0.1
         port: 4730
```

In development mode you do not want to cache things over more than one request. An easy solution for this is to use the array provider in the dev environment ( Extracted from DoctrineCacheBundle documentation )

```
#config.yml
doctrine_cache:
    providers:
        gearman_cache:
            type: file_system
            namespace: doctrine_cache.ns.gearman
```

In development mode you do not want to cache things over more than one request. An easy solution for this is to use the array cache in the dev environment ( Extracted from DoctrineCacheBundle documentation )

```
#config_dev.yml
doctrine_cache:
    providers:
        gearman_cache:
            type: array
            namespace: doctrine_cache.ns.gearman
```

## Definition of Workers

This Bundle allows you to configure whatever as a Job. It provides you an easy way to execute it with Supervisor, for example. Moreover, it let you call client methods in Symfony2 environment in a really simple and practical way. Job annotations always overwrite work annotations, and work annotations always overwrite environment settings.

```
<?php

namespace Acme\AcmeBundle\Workers;

use Mmoreram\GearmanBundle\Driver\Gearman;

/**
 * @Gearman\Work(
 *     iterations = 3,
 *     minimumExecutionTime = 3,
 *     timeout = 20,
 *     description = "Worker test description",
 *     defaultMethod = "doBackground",
 *     servers = {
 *         { "host": "192.168.1.1", "port": 4560 },
 *         { "host": "192.168.1.2", "port": 4560 },
 *     }
 * )
 */
class AcmeWorker
```

```
{
    /**
     * Test method to run as a job
     *
     * @param \GearmanJob $job Object with job parameters
     *
     * @return boolean
     *
     * @Gearman\Job(
     *     iterations = 3,
     *     minimumExecutionTime = 2,
     *     timeout = 30,
     *     name = "test",
     *     description = "This is a description"
     * )
     */
    public function testA(\GearmanJob $job)
    {
        echo 'Job testA done!' . PHP_EOL;

        return true;
    }

    /**
     * Test method to run as a job
     *
     * @param \GearmanJob $job Object with job parameters
     *
     * @return boolean
     *
     * @Gearman\Job(
     *     defaultMethod = "doLowBackground"
     * )
     */
    public function testB(\GearmanJob $job)
    {
        echo 'Job testB done!' . PHP_EOL;

        return true;
    }
}
```

## Worker annotations

```
/**
 * @Gearman\Work(
 *     name = "MyAcmeWorker",
 *     iterations = 3,
 *     minimumExecutionTime = 3,
 *     timeout = 20,
 *     description = "Acme Worker. Containing multiple available jobs",
 *     defaultMethod = "doHigh",
 *     servers = {
 *         { "host": "192.168.1.1", "port": 4560 },
 *         { "host": "192.168.1.2", "port": 4560 },
 *     }
```

```
 * )
 */
```

- name : Name of work. You can associate a group of jobs with some keyword
- description : Short description about all jobs inside
- iterations : You can overwrite iterations of all jobs inside
- minimumExecutionTime: You can overwrite the main default minimum execution time
- timeout: You can overwrite the main default timeout
- servers : array containing servers providers will connect to offer all jobs
- service : You can use even a service. Must specify callable service name
- defaultMethod : You can define witch method will be used as default in all jobs

## Job annotations

```
/**
 * @Gearman\Job(
 *     name = "doSomething",
 *     iterations = 10,
 *     minimumExecutionTime = 2,
 *     timeout = 30,
 *     description = "Acme Job action. This is just a description of a method that do␣
↪something",
 *     defaultMethod = "doBackground",
 *     servers = { "host": "192.168.1.1", "port": 4560 }
 * )
 */
```

- name : Name of job. You will use it to call job
- description : Short description about this job. Important field
- iterations : You can overwrite iterations of this job.
- minimumExecutionTime: You can overwrite the worker minimum execution time
- timeout: You can overwrite the worker timeout
- servers : array containing servers providers will connect to offer this job
- defaultMethod : You can define witch method will be used as default in this job

## Job as a service

If you want to use your service as a worker, you have to specify service variable in Worker annotation.

```
<?php

namespace Acme\AcmeBundle\Services;

use Mmoreram\GearmanBundle\Driver\Gearman;

/**
 * @Gearman\Work(
```

```
 *      service="myServiceName"
 * )
 */
class AcmeService
{

    /**
     * Test method to run as a job
     *
     * @param \GearmanJob $job Object with job parameters
     *
     * @return boolean
     *
     * @Gearman\Job()
     */
    public function testA(\GearmanJob $job)
    {
        echo 'Job testA done!' . PHP_EOL;

        return true;
    }
}
```

And have this service defined in your dependency injection definition file

```
# /Resources/config/services.yml
bundles:
   Services:
      myServiceName:
         class: Acme\AcmeBundle\Services\AcmeService
         arguments:
            event_dispatcher: @event_dispatcher
            mailer: @mailer
```

## Console output from workers

If you need your worker to output information to the console, you can have your worker class implement *Mmoreram\GearmanBundle\Command\Util\GearmanOutputAwareInterface*.

This interface requires a single method be implemented *public function setOutput(OutputInterface $output);*. To avoid needing to check the output is available, you can by default set it to an instance of *Symfony\Component\Console\Output\NullOutput*.

```
namespace Acme\AcmeBundle\Services;

use Symfony\Component\Console\Output\NullOutput;
use Mmoreram\GearmanBundle\Command\Util\GearmanOutputAwareInterface;
use Mmoreram\GearmanBundle\Driver\Gearman;

/**
 * @Gearman\Work(
 *      iterations = 3,
 *      description = "Worker test description",
 *      defaultMethod = "doBackground"
 * )
 */
```

```
class AcmeWorker implements GearmanOutputAwareInterface
{
    /**
     * @var OutputInterface
     */
    protected $output;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->output = new NullOutput();
    }

    /**
     * @param OutputInterface $output
     */
    public function setOutput(OutputInterface $output)
    {
        $this->output = $output;
    }

    /**
     * Test method to run as a job with console output
     *
     * @param \GearmanJob $job Object with job parameters
     *
     * @return boolean
     *
     * @Gearman\Job(
     *     iterations = 3,
     *     name = "test",
     *     description = "This is a description"
     * )
     */
    public function testA(\GearmanJob $job)
    {
        $this->output->writeln('Job testA done!');

        return true;
    }
}
```

## Running your jobs

Gearman provides a set of commands that will make easier to know all workers settings.

```
$ php app/console
```

A subset of listed commands are Gearman specific.

```
gearman
    gearman:cache:clear     Clears gearman cache data on current environment
    gearman:cache:warmup    Warms up gearman cache data
```

```
    gearman:job:describe    Describe given job
    gearman:job:execute     Execute one single job
    gearman:worker:describe Describe given worker
    gearman:worker:execute  Execute one worker with all contained Jobs
    gearman:worker:list     List all Gearman Workers and their Jobs
```

## Listing workers and jobs

Once all your workers are defined, you can simply list them to ensure all settings are correct.

```
$ php app/console gearman:worker:list

@Worker:  Mmoreramerino\TestBundle\Services\AcmeWorker
callablename:  MmoreramerinoTestBundleServicesMyAcmeWorker
Jobs:
  - #1
      name: testA
      callablename: MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething
```

## Listing worker settings

You can describe full worker using its callableName. This command provides you all information about desired Worker, overwritting custom annotation settings to default config settings. This command also provides you all needed information to work with Supervisord.

```
$ php app/console gearman:worker:describe MmoreramerinoTestBundleServicesMyAcmeWorker

@Worker\className : Mmoreramerino\TestBundle\Services\AcmeWorker
@Worker\fileName : /var/www/projects/myrepo/src/Mmoreramerino/TestBundle/Services/
↪AcmeWorker.php
@Worker\nameSpace : Mmoreramerino\TestBundle\Services
@Worker\callableName: MmoreramerinoTestBundleServicesMyAcmeWorker
@Worker\supervisord : /usr/bin/php /var/www/projects/myrepo/app/console␣
↪gearman:worker:execute MmoreramerinoTestBundleServicesMyAcmeWorker --no-interaction
@worker\iterations : 3
@Worker\#jobs : 1

@worker\servers :

    #0 - 192.168.1.1:4560
    #1 - 192.168.1.2:4560

@Worker\description :

    Acme Worker. Containing multiple available jobs
```

## Listing job settings

You can also describe full job using also its callableName This command provides you all information about desired Job, overwritting custom annotation settings to worker settings. This command also provides you all needed information to work with Supervisord.

```
$ php app/console gearman:job:describe MmoreramerinoTestBundleServicesMyAcmeWorker~
↪doSomething

@Worker\className : Mmoreramerino\TestBundle\Services\AcmeWorker
@Worker\fileName : /var/www/projects/myrepo/src/Mmoreramerino/TestBundle/Services/
↪AcmeWorker.php
@Worker\nameSpace : Mmoreramerino\TestBundle\Services
@Worker\callableName: MmoreramerinoTestBundleServicesMyAcmeWorker
@Worker\supervisord : /usr/bin/php /var/www/projects/myrepo/app/console␣
↪gearman:worker:execute MmoreramerinoTestBundleServicesMyAcmeWorker --no-interaction
@worker\iterations : 3
@Worker\#jobs : 1

@worker\servers :

    #0 - 192.168.1.1:4560
    #1 - 192.168.1.2:4560

@Worker\description :

    Acme Worker. Containing multiple available jobs

@job\methodName : testA
@job\callableName : MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething
@job\supervisord : /usr/bin/php /var/www/projects/myrepo/app/console␣
↪gearman:job:execute MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething --no-
↪interaction
@job\iterations : 10
@job\defaultMethod : doBackground
@job\servers :

    0 - 192.168.1.1:4560

@job\description :

    #Acme Job action. This is just a description of a method that do something
```

## Run a job

You can execute by command line an instance of a worker or a job. The difference between them is that an instance of a worker can execute any of their jobs, without assignning any priority to them, and a job only can run itself.

```
$ php app/console gearman:worker:execute MmoreramerinoTestBundleServicesMyAcmeWorker
$ php app/console gearman:job:execute MmoreramerinoTestBundleServicesMyAcmeWorker~
↪doSomething
```

**Note:** By using callableName you can let Supervisord maintain alive a worker. When the job is executed as times as iterations is defined, will die, but supervisord will alive it again. You can have as many as worker instances as you want. Get some Supervisord info

## Overriding default settings

From the command line you can run the jobs or workers with overridden settings. These include

- iterations
- minimum-execution-time
- timeout

For example:

```
$ php app/console gearman:job:describe MmoreramerinoTestBundleServicesMyAcmeWorker~
↪doSomething --iterations=5 --minimum-execution-time=2 --timeout=20
```

If these options are ommited, then the configuration defaults are used.

### Request job status

With the Handle given if requesting a background job you can request the status of the job. The Method returns a JobStatus object placed in 'MmoreramGearmanBundleModuleJobStatus'

```
$jobStatus = $gearman->getJobStatus($result);
$jobIsKnown = $jobStatus->isKnown();
$jobIsRunning = $jobStatus->isRunning();
$jobIsFinished = $jobStatus->isFinished();

/**
 * Also gives completion data
 */
$completed = $jobStatus->getCompleted();
$completionTotal = $jobStatus->getCompletionTotal();
$completionPercent = $jobStatus->getCompletionPercent();
```

## Client

You can request a Job by using the gearman client.

```
$this
    ->getContainer()
    ->get('gearman');
```

### Servers

```
$gearman
    ->clearServers()
    ->setServer('127.1.1.1', 4677)
    ->addServer('127.1.1.1', 4678)
    ->addServer('127.1.1.1', 4679);
```

- addServer: Add new server to requested client
- setServer: Clean server list and set new server to requested client
- clearServers: Clear server list

---

**Note:**

**By default, if no server is set, gearman will use server defined as** default in config.yml

host: *127.0.0.1* port: *4730*

## Request a job

```
$result = $gearman
    ->doJob('MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething', json_
↪encode(array('value1')));

$returnCode = $gearman->getReturnCode();
```

- doJob: Call the job and wait for the result
- doNormalJob: Call the job and wait for the result ( Only newest gearman versions )
- doHighJob: Call the job and wait for the result on High Preference
- doLowJob: Call the job and wait for the result on Low Preference
- **doBackgroundJob: Call the job without waiting for the result.**
  - It receives a job handle for the submitted job
- **doHighBackgroundJob: Call the job without waitting for the result on High Preference.**
  - It receives a job handle for the submitted job
- **doLowBackgroundJob: Call the job without waitting for the result on Low Preference.**
  - It receives a job handle for the submitted job
- **callJob: Call the job with default method.**
  - Defined in settings, work annotations or the job annotations
- getReturnCode: Retrieve the return code from the last requested job.

## Tasks

```
$gearman
    ->addTask('MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething', 'value1',
↪$context1)
    ->addLowTask('MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething', 'value2',
↪$context2)
    ->addHighBackgroundTask('MmoreramerinoTestBundleServicesMyAcmeWorker~doSomething',
↪ 'value3', $context3)
    ->runTasks();
```

- addTask: Adds a task to be run in parallel with other tasks
- addTaskHigh: Add a high priority task to run in parallel
- addTaskLow: Add a low priority task to run in parallel
- addTaskBackground: Add a background task to be run in parallel
- addTaskHighBackground: Add a high priority background task to be run in parallel
- addTaskLowBackground: Add a low priority background task to be run in parallel

- runTasks: Run a list of tasks in parallel

## Kernel Events

GearmanBundle transforms Gearman callbacks to Symfony2 kernel events.

### Complete Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackCompleteEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setCompleteCallback](http://www.php.net/manual/en/gearmanclient.setcompletecallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.complete,␣
↪method: onComplete }
```

### Created Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackCreatedEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setCreatedCallback](http://www.php.net/manual/en/gearmanclient.setcreatedcallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.created,␣
↪method: onCreated }
```

### Data Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackDataEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setDataCallback](http://www.php.net/manual/en/gearmanclient.setdatacallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.data,␣
↪method: onData }
```

## Exception Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackExceptionEvent* with no methods. For more information about this GearmanEvent, read [GearmanClient::setExceptionCallback](http://www.php.net/manual/en/gearmanclient.setexceptioncallback.php) documentation.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.exception,␣
↪method: onExcept }
```

## Fail Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackFailEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setFailCallback](http://www.php.net/manual/en/gearmanclient.setfailcallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.fail,␣
↪method: onFail }
```

## Status Callback

This event receives as a parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackStatusEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setStatusCallback](http://www.php.net/manual/en/gearmanclient.setstatuscallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.status,␣
↪method: onStatus }
```

## Warning Callback

This event receives as parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackWarningEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setWarningCallback](http://www.php.net/manual/en/gearmanclient.setwarningcallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.warning,␣
↪method: onWarning }
```

## Workload Callback

This event receives as parameter an instance of *MmoreramGearmanBundleEventGearmanClientCallbackWorkloadEvent* with methods *$event->getGearmanTask()* and *&$event->getContext()*. First method returns an instance of *GearmanTask*. For more information about this GearmanEvent, read [GearmanClient::setWorkloadCallback](http://www.php.net/manual/en/gearmanclient.setworkloadcallback.php) documentation. The second method will return *$context* that you could add in the *addTask()* method.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.client.callback.workload,␣
↪method: onWorkload }
```

## Starting Work Event

This event receives as parameter an instanceof *MmoreramGearmanBundleEventGearmanWorkStartingEvent* with one method: *$event->getJobs()* returns the configuration of the jobs.

This event is dispatched before a job starts.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.work.starting, method:␣
↪onWorkStarting }
```

## Execute Work Event

This event receives as parameter an instanceof *MmoreramGearmanBundleEventGearmanWorkExecutedEvent* with three methods: *$event->getJobs()* returns the configuration of the jobs, *$event->getIterationsRemaining()* returns the remaining iterations for these jobs, *$event->getReturnCode()* returns the return code of the last executed job.

This event is dispatched after a job has been completed. After this event is completed, the worker continues with its iterations.

```
services:
    my_event_listener:
        class: AcmeBundle\EventListener\MyEventListener
        tags:
          - { name: kernel.event_listener, event: gearman.work.executed, method:␣
↪onWorkExecuted }
```

# Customize

Some bundle behaviours can be overwritten

## Custom unique job identifier method

If you want a custom method to generate custom unique values for your jobs when not defined ( specified in generate_unique_key ), you only have to extend default UniqueJobIdentifierGenerator class and overwrite generateUniqueKey method, as folowing example.

```php
<?php

/**
 * Gearman Bundle for Symfony2
 *
 * @author Marc Morera <yuhu@mmoreram.com>
 * @since 2013
 */

namespace My\Custom\Namespace;

use Mmoreram\GearmanBundle\Generator\UniqueJobIdentifierGenerator;

/**
 * Gearman execute methods. All Worker methods
 *
 * @author Marc Morera <yuhu@mmoreram.com>
 */
class MyCustomUniqueJobIdentifierGenerator extends UniqueJobIdentifierGenerator
{

    /**
     * Generate unique key if generateUniqueKey is enabled
     *
     * $this->generateUniqueKey can be used as is protected in parent class
     *
     * @param string $name   A GermanBundle registered function to be executed
     * @param string $params Parameters to send to task as string
     * @param string $unique unique ID used to identify a particular task
     * @param string $method Method to perform
     *
     * @return string Generated Unique Key
     */
    public function generateUniqueKey($name, $params, $unique, $method)
    {
        /**
         * Custom generation
         */
    }
}
```

You need also to overwrite in your config.yml the generator class

```
parameters:

    #
```

```
    # Generators
    #
    gearman.unique_job_identifier.class:␣
↪My\Custom\Namespace\MyCustomUniqueJobIdentifierGenerator
```

## Cache

GearmanBundle caches all annotations. You can clear or warmup just gearman cache by using custom commands

```
$ php app/console

gearman
    gearman:cache:clear     Clears gearman cache data on current environment
    gearman:cache:warmup    Warms up gearman cache data
```

Gearman also clear and warmup cache when using Symfony2 cache commands

```
$ php app/console

cache
    cache:clear             Clears the cache
    cache:warmup            Warms up an empty cache
```

## Faqs

CHAPTER 2

# CookBook

## Job Status