

# Logging

## TABLE OF CONTENTS

- [Quick log to watchdog](#)
- [Log an email notification was sent to the the email address for the site.](#)
- [Logging from a service using dependency injection](#)
- [Another example using the logging via dependency injection](#)
- [Logging exceptions from a try catch block](#)
- [Display a message in the notification area](#)
- [Display a variable while debugging](#)
- [Reference](#)

views 81

## Quick log to watchdog

With the Database Logging (dblog) module enabled, you can easily log messages to the database log (watchdog table)

```
//Class with method name
$method = __METHOD__;

// Function name only.
$function = __FUNCTION__;

// Function name, filename, line number.
$str = __FUNCTION__." in ".__FILE__." at ".__LINE__;

\Drupal::logger('test')->info("method = $method");
\Drupal::logger('test')->info("Something goofed up at $str");
\Drupal::logger('test')->debug("Something goofed up at $str");
\Drupal::logger('test')->critical("Something goofed up at $str");

\Drupal::service('logger.factory')->get('test')->error("This is my error message");
```

The parameter “test” used above is typically the module name. It is stored in the “type” field.

You can call difference methods such as info, warning etc. which populate the severity field with an integer indicating the severity of the issue.

The methods are defined in Drupal\Core\Logger\RfcLoggerTrait:

- emergency(\$message, \$context)
- alert(\$message, \$context)
- critical(\$message, \$context)
- error(\$message, \$context)
- warning(\$message, \$context)
- notice(\$message, \$context)
- info(\$message, \$context)
- debug(\$message, \$context)

More at <https://www.drupal.org/docs/8/api/logging-api/overview>

Log an email notification was sent to the the email address for the site.

```
$email_config = \Drupal::config('system.site');
$to = $email_config->get('mail');
// Display message to screen.
$messenger->addMessage("sent a message to $to");
// Log it.
\Drupal::logger('DIR')->info("Email notification send to $to succeeded");
```

Incidentally, calling `\Drupal::logger` like this

```
\Drupal::logger('my_module')->error('This is my error message');
```

actually does this under the covers:

```
\Drupal::service('logger.factory')->get('hello_world')->error('This is my error message');
```

## Logging from a service using dependency injection

From a controller e.g. `WebSphereAddress.php`

In the `websphere_commerce.services.yml` specify the `@logger.factory` to be passed into the constructor.

```
services:
  websphere_commerce.address:
    class: Drupal\websphere_commerce\WebSphereAddressService
    arguments: [ '@config.factory', '@logger.factory' ]
```

In the `WebSphereAddress.php` file specify use statements:

```
use Drupal\Core\Logger\LoggerChannelFactory;
use Drupal\Core\Logger\LoggerChannelFactoryInterface;
```

Create a protected var to store the logger service:

```
/**
 * @var Drupal\Core\Logger\LoggerChannelFactory
 */
protected $logger;
```

Here is the constructor:

```
/**
 * WebsphereAddress constructor.
 *
 * @param \Drupal\Core\Config\ConfigFactoryInterface $config_factory
 *   Config factory.
 * @param \Drupal\Core\Logger\LoggerChannelFactoryInterface $channel_factory
 *   Logger factory.
 */
public function __construct(ConfigFactoryInterface $config_factory, LoggerChannelFactoryInterface $channel_factory) {
    $this->websphereConfig = $config_factory->get('websphere_commerce.api_settings');
}
```

Log errors.

```
if ($response['status'] == API_ERROR) {  
    $this->logger->get('websphere_commerce')->alert("Error saving Shipping info to Websphere.");  
}
```

## Another example using the logging via dependency injection

From the excellent folks at [symfonycasts.com](https://symfonycasts.com) who have a sweet [Drupal 8 course](#) which is still relevant and worth checking out.

In your `dino_roar.services.yml` file, add the listener and specify the arguments of `[@logger.factory]`

Note. You can find the factory info with Drupal console:

```
$ drupal debug:container | grep log
```

one of the results specifies the factory which you can use below:

logger.factory	Drupal\Core\Logger\LoggerChannelFactory
----------------	---

or with Drush and devel

```
$ drush dcs log  
  
- logger.dblog  
- logger.drupaltdrush  
- logger.factory
```

So `dino_roar.dino_listener` will pass the `logger.factory` service to your `DinoListener` class.

```
dino_roar.dino_listener:  
  class: Drupal\dino_roar\Jurassic\DinoListener  
  arguments: ['@logger.factory']  
  tags:  
    - {name: event_subscriber}
```

in your `DinoListener.php` specify a constructor argument of `LoggerChannelFactoryInterface` and store it.

```

namespace Drupal\dino_roar\Jurassic;

use Drupal\Core\Logger\LoggerChannelFactoryInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpKernel\KernelEvents;

class DinoListener implements EventSubscriberInterface {

  /**
   * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
   */
  private $loggerChannelFactory;

  public function __construct(LoggerChannelFactoryInterface $loggerChannelFactory) {

    $this->loggerChannelFactory = $loggerChannelFactory;
  }

  public function onKernelRequest(GetResponseEvent $event) {
    $request = $event->getRequest();
    $shouldRoar = $request->query->get('roar');
    if ($shouldRoar) {
      $this->loggerChannelFactory->get('default')
        ->debug('Roar Requested ROOOOAAAARRR!');
    }
  }

  public static function getSubscribedEvents() {
    return [
      KernelEvents::REQUEST => 'onKernelRequest',
    ];
  }
}

```

## Logging exceptions from a try catch block

In this controller, the try block calls the test() method which throws an exception. The catch block catches the exception and logs the message (and for fun displays a message in the notification area also.)

```

public function build() {

    try {
        $this->test();
    }
    catch (\Exception $e) {
        watchdog_exception('nuts_connect', $e);
        $messenger->addMessage("No, I got caught!");
    }

    $build['content'] = [

        '#type' => 'item',
        '#markup' => $str,
    ];

    return $build;
}

function test() {
    throw new \Exception("blah", 7);
}

```

## Display a message in the notification area

You can display a message with:

```

$messenger = \Drupal::messenger();
$messenger->addMessage("a message");
$messenger->addError("error message");

```

Or

```

\Drupal::messenger()->addError("migration failed");

\Drupal::messenger()->addMessage($message, $type, $repeat);

```

Use \$repeat = FALSE to suppress duplicate messages.

Specify `MessengerInterface::TYPE_STATUS`, `MessengerInterface::TYPE_WARNING`, or `MessengerInterface::TYPE_ERROR` to indicate the severity.

Don't forget

```

use Drupal\Core\Messenger\MessengerInterface;

```

Note. `addMessage()` adds `class="messages messages--status"` to the div surrounding your message while `addError` adds `class="messages messages--status"`. Use these classes to format the message appropriately.

When you need to display a message in a form, use the `$this->messenger()` that is provided by the `Drupal\Core\Messenger\MessengerTrait`;

```

$this->messenger()->addStatus($this->t("Running in Destructive Mode - Changes ARE committed to the database!"));

```

e.g.

```

\Drupal::messenger()->addMessage("Program pending, please assign team and initialize.", MessengerInterface::TYPE_WARNING);

```

## Display a variable while debugging

You can use `var_dump` and `print_r` but sometimes it is difficult to see where they display.

```
$is_front = \Drupal::service('path.matcher')->isFrontPage();  
$is_front = $is_front == TRUE ? "YEP" : "NOPE";  
$messenger->addMessage("is_front = $is_front");  
var_dump($is_front);  
print_r($is_front);
```

string(4) "NOPE" Yippee:NOPE

# Drush Site-Install

Home

✓ a message  
sent a message to admin@example.com  
is\_front = NOPE

✗ error message

**var\_dump()  
and  
print\_r()**

## Reference

- How to Log Messages in Drupal 8 by Amber Matz of Drupalize.me Updated October 2015 <https://drupalize.me/blog/201510/how-log-messages-drupal-8>
- Logging API updated January 2023 <https://www.drupal.org/docs/8/api/logging-api/overview>
- Drupal APIs <https://www.drupal.org/docs/drupal-apis>

[Back to top](#)

Drupal at your fingertips by [Selwyn Politt](#) is licensed under [CC BY 4.0](#)

Page last modified: Apr 13 2023.

[Edit this page on GitHub](#)

