## What Does “Macroable” Mean in Laravel

To start, let’s add the following to our app/routes.php file.

#File: app/routes.phpclass Hello{}

Route::get('testbed', function(){

$hello = new Hello;

$hello->sayHi();

});

In real life we’d never define a class in the app/routes.php file, but it’s convenient for demo/learning purposes.

If we load the testbed route in a browser

http://laravel.example.com/testbed

We’ll get the following PHP error

Call to undefined method Hello::sayHi()

This is unsurprising, as we never defined a sayHi method for the Hello class.

Next, let’s add the MacroableTrait to our class

#File: app/routes.php class Hello

{

use Illuminate\Support\Traits\MacroableTrait;

}

Here we’ve used the full trait name, namespace and all. If you’re not familiar with traits, checkout [last week’s primer](http://alanstorm.com/php_traits). Traits follow the same namespace rules as classes, and using a trait will invoke the PHP autoloader. If we load the page with the above in place, you’ll still see the same error

Call to undefined method Hello::sayHi()

However, now try adding the following code to your route

#File: app/routes.php class Hello

{

use Illuminate\Support\Traits\MacroableTrait;

}Route::get('testbed', function(){

// include '/tmp/test.php';

$hello = new Hello;

Hello::macro('sayHi', function(){

echo "Hello","\n<br>\n";

});

$hello->sayHi();

Hello::sayHi();

});

Here we’ve called the static method macro on Hello. While Hello doesn’t define a static method named macro, it does inherit one from the Illuminate\Support\Traits\MacroableTrait trait.

If you reload the page with the above in place, you’ll see the following.

Hello

Hello

This is what the MacroableTrait does — it allows you to add a method (or, a “macro”) to a PHP class programmatically. By calling the macro method, we’ve effectively added the sayHi method to our Hello class, and all objects instantiated from that class. The above example is a little silly (again, for pedagogical reasons) — in real life you’ll see the MacroableTrait used in some global bootstrap-ish code to make a method available to other programmers. An example you might be familiar with from Laravel 4.2 is the [Form helper macros](http://laravel.com/docs/4.2/html" \l "custom-macros).

## How MacroableTrait Works

The MacroableTrait is simpler than you might think. You can find its definition in the following file

vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.php

The most obvious place to start your investigation is the macro method we called earlier

#File: vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.phppublic static function macro($name, callable $macro)

{

static::$macros[$name] = $macro;

}

This method stashed the anonymous PHP function in a static array property, indexed by $name. If you scroll down a bit in the file, you’ll see our [old friends](https://alanstorm.com/...) \_\_call and \_\_callStatic. The \_\_call method

#File: vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.phppublic function \_\_call($method, $parameters)

{

return static::\_\_callStatic($method, $parameters);

}

Simply passes the method call on to the \_\_callStatic method. The \_\_callStatic method

#File: vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.php public static function \_\_callStatic($method, $parameters)

{

if (static::hasMacro($method))

{

return call\_user\_func\_array(static::$macros[$method], $parameters);

}

throw new \BadMethodCallException("Method {$method} does not exist.");

}

will fetch the anonymous function or [PHP callback](https://alanstorm.com/) added with the call to macro, and then call the anonymous function/callback using PHP’s [call\_use\_func\_array](http://php.net/call_use_func_array) method.

If you didn’t follow that, let’s trace things again, this time with our example code. First, when we called

#File: app/routes.php Route::get('testbed', function(){

//...

Hello::macro('sayHi', function(){

echo "Hello","\n<br>\n";

});

//...

});

we were telling our class to store the anonymous function inside of static::$macros['sayHi'].

Next, here’s where we called sayHi (both statically and via an instance method)

#File: app/routes.php Route::get('testbed', function(){

//...

$hello->sayHi();

Hello::sayHi();

//...

});

Since sayHi isn’t defined on the class Hello, PHP ends up calling \_\_call (for the -> invokation) and \_\_callStatic (for the :: invocation), per the rules of [PHP Magic Methods](http://alanstorm.com/php_magic_methods_and_class_aliases). The \_\_call method simply passes the method call onto \_\_callStatic. This means \_\_callStatic actually executes for both instance and static method calls.

If we put on some x-ray variable specs, the method call looks something like this

#File: vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.php public static function \_\_callStatic('sayHi', $parameters)

{

if (static::hasMacro('sayHi'))

{

return call\_user\_func\_array(static::$macros['sayHi'], $parameters);

}

throw new \BadMethodCallException("Method {'sayHi'} does not exist.");

}

Or, expanding that out even further, like this

#File: vendor/laravel/framework/src/Illuminate/Support/Traits/MacroableTrait.php public static function \_\_callStatic('sayHi', $parameters)

{

if (static::hasMacro('sayHi'))

{

return call\_user\_func\_array(function(){

echo "Hello","\n<br>\n";

}, $parameters);

}

throw new \BadMethodCallException("Method {'sayHi'} does not exist.");

}

By leveraging PHP’s magic methods and static functions, the MacroableTrait trait gives you the ability to dynamically add methods to objects at runtime — a feature usually reserved for “more advanced” dynamic languages like ruby/python.

## Gotchas

While powerful, keep in mind macros (by design) have no knowledge of the other properties and methods of a class/object. Since you’re using an anonymous function (or [PHP callback](http://php.net/callback)) to add your method, this function/callback won’t have access to the usual variables like $this, self, or static. This means the methods you add via a MacroableTrait will never be more than stateless helper methods. If you need access to an object or class’s state, then a macro is the wrong tool for your job.

The other thing you should keep in mind when considering the MacroableTrait is whether you want to add **another** item to the already busy “static method namespace” in your Laravel program. We’ll talk more about this next time, when we dive deeper into the various different things a static method call might do in Laravel.