[The make Method](https://laravel.com/docs/10.x/container" \l "the-make-method)

You may use the **make method** to resolve a class instance from the container. The **make method** accepts the name of the class or **interface** you wish to resolve:

***use App\Services\Transistor;***

***$transistor = $this->app->make(Transistor::class);***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If some of your class's dependencies are not resolvable via the container, you may inject them by passing them as an associative array into the **makeWith** method. For example, we may manually pass the **$id** constructor argument required by the **Transistor service**:

***use App\Services\Transistor;***

***$transistor = $this->app->makeWith(Transistor::class, ['id' => 1]);***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The **bound method** may be used to determine if a class or interface has been explicitly bound in the container:

***if ($this->app->bound(Transistor::class)) {}***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you are outside of a service provider in a location of your code that does not have access to the **$app variable**, you may use the App [facade](https://laravel.com/docs/10.x/facades) or the app [helper](https://laravel.com/docs/10.x/helpers#method-app) to resolve a class instance from the container:

***use App\Services\Transistor;***

***use Illuminate\Support\Facades\App;***

***$transistor = App::make(Transistor::class);***

***$transistor = app(Transistor::class);***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you would like to have the Laravel container instance itself injected into a class that is being resolved by the container, you may type-hint the Illuminate\Container\Container class on your class's constructor:

***use Illuminate\Container\Container;***

***public function \_\_construct(***

***protected Container $container***

***) {}***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[Automatic Injection](https://laravel.com/docs/10.x/container#automatic-injection)

Alternatively, and importantly, you may type-hint the dependency in the constructor of a class that is resolved by the container, including [controllers](https://laravel.com/docs/10.x/controllers), [event listeners](https://laravel.com/docs/10.x/events), [middleware](https://laravel.com/docs/10.x/middleware), and more. Additionally, you may type-hint dependencies in the handle method of [queued jobs](https://laravel.com/docs/10.x/queues). In practice, this is how most of your objects should be resolved by the container.

***use App\Repositories\UserRepository;***

***use App\Models\User;***

***class UserController extends Controller***

***{***

***/\*\****

***\* Create a new controller instance.***

***\*/***

***public function \_\_construct(***

***protected UserRepository $users,***

***) {}***

***/\*\****

***\* Show the user with the given ID.***

***\*/***

***public function show(string $id): User***

***{***

***$user = $this->users->findOrFail($id);***

***return $user;***

***}***

***}***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[Method Invocation & Injection](https://laravel.com/docs/10.x/container#method-invocation-and-injection)

Sometimes you may wish to invoke a method on an object instance while allowing the container to automatically inject that method's dependencies. For example, given the following class:

***<?php***

***namespace App;***

***use App\Repositories\UserRepository;***

***class UserReport***

***{***

***public function generate(UserRepository $repository): array***

***{***

***return [***

***// ...***

***];***

***}***

***}***

You may invoke the generate method via the container like so:

***use App\UserReport;***

***use Illuminate\Support\Facades\App;***

***$report = App::call([new UserReport, 'generate']);***

The call method accepts any PHP callable. The container's call method may even be used to invoke a closure while automatically injecting its dependencies:

***use App\Repositories\UserRepository;***

***use Illuminate\Support\Facades\App;***

***$result = App::call(function (UserRepository $repository) {***

***});***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[Container Events](https://laravel.com/docs/10.x/container#container-events)

The service container fires an event each time it resolves an object. You may listen to this event using the **resolving method**:

***use App\Services\Transistor;***

***use Illuminate\Contracts\Foundation\Application;***

***$this->app->resolving(Transistor::class, function (Transistor $transistor, Application $app) {***

***// Called when container resolves objects of type "Transistor"...***

***});***

***$this->app->resolving(function (mixed $object, Application $app) {***

***// Called when container resolves object of any type...***

***});***

As you can see, the object being resolved will be passed to the callback, allowing you to set any additional properties on the object before it is given to its consumer.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[PSR-11](https://laravel.com/docs/10.x/container#psr-11)

Laravel's service container implements the [PSR-11](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-11-container.md) interface. Therefore, you may type-hint the ***PSR-11 container interface*** to obtain an instance of the Laravel container:

***use App\Services\Transistor;***

***use Psr\Container\ContainerInterface;***

***Route::get('/', function (ContainerInterface $container) {***

***$service = $container->get(Transistor::class);***

***});***

An exception is thrown if the given identifier can't be resolved. The exception will be an instance of **Psr\Container\NotFoundExceptionInterface** if the identifier was never bound. If the identifier was bound but was unable to be resolved, an instance of **Psr\Container\ContainerExceptionInterface** will be thrown.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*