Standalone applications

There are several ways of mounting a Nest application. You can create a web app, a microservice or just a bare Nest **standalone application** (without any network listeners). The Nest standalone application is a wrapper around the Nest **IoC container**, which holds all instantiated classes. We can obtain a reference to any existing instance from within any imported module directly using the standalone application object. Thus, you can take advantage of the Nest framework anywhere, including, for example, scripted **CRON** jobs. You can even build a **CLI** on top of it.

Getting started

To create a Nest standalone application, use the following construction:

```
@@filename()
async function bootstrap() {
  const app = await NestFactory.createApplicationContext(AppModule);
  // application logic...
}
bootstrap();
```

The standalone application object allows you to obtain a reference to any instance registered within the Nest application. Let's imagine that we have a `TasksService` in the `TasksModule`. This class provides a set of methods that we want to call from within a CRON job.

```
@@filename()
const app = await NestFactory.createApplicationContext(AppModule);
const tasksService = app.get(TasksService);
```

To access the `TasksService` instance we use the `get()` method. The `get()` method acts like a **query** that searches for an instance in each registered module. Alternatively, for strict context checking, pass an options object with the `strict: true` property. With this option in effect, you have to navigate through specific modules to obtain a particular instance from the selected context.

```
@@filename()
const app = await NestFactory.createApplicationContext(AppModule);
const tasksService = app.select(TasksModule).get(TasksService, { strict:
true });
```

Following is a summary of the methods available for retrieving instance references from the standalone application object.

`get()`	Retrieves an instance of a controller or provider (including guards, filters, and so on) available in the application context.

`select()`	Navigates through the module's graph to pull out a specific instance of the selected module (used together with strict mode as described above).

> info **Hint** In non-strict mode, the root module is selected by default. To select any other module, you need to navigate the modules graph manually, step by step.

If you want the node application to close after the script finishes (e.g., for a script running CRON jobs), add `await app.close()` to the end of your `bootstrap` function:

```
@@filename()
async function bootstrap() {
  const app = await NestFactory.createApplicationContext(AppModule);
  // application logic...
  await app.close();
}
bootstrap();
```

Example

A working example is available here.