BindToConfig Documentation

Marcin Dąbrowski

Dec 16, 2018

Contents

1 Code example 3

BindToConfig is a simple and lightweight alternative to **.NET Core Options**, that doesn't require any additional interface implemented by classes. Focused on simplicity and good practices, promotes creating many small, clean, immutable classes that are a representation of application's configuration which lifetime scope and binding to the configuration are set in composition root and not directly by dependent classes.

BindToConfig fills the gap between .NET Core's Configuration and DI, by taking care of mapping and binding objects to the configuration, validating and registering in .NET Core DI. With just a single method call it provides .NET Core applications with an easy way of adding classes representing parts of Configuration.

Built with the simplicity and promotion of best practices in mind

- Interface Segregation Principle - no additional interface is required
- Immutability
- Fail-fast
- Composition root
- Base what is need: `services.AddBoundToConfig<ConfigClass>(..)`

small, lightweight, free and open-source.

CHAPTER 1

Code example

In *ConfigureServices* method, just call *AddFromConfig* or *AddBoundToConfig* on *services*.
Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddMvc();
  // adds SampleConfig1 as Singleton
  services.AddFromConfig<SampleConfig1>(Configuration, "SectionKey1");
  // adds Sample Config2 as Scoped,
  // new scopes return updated Config2 object when configuration changes
  // use it only when You really use live-updates of configuration
  services.AddBoundToConfig<SampleConfig2>(Configuration, "SectionKey2");
  services.AddBoundToConfig<SampleConfig3>(
      Configuration,
      "NonExisting",
      BindOptions.DontThrowIfSectionIsMissingOrEmpty);
}
```

From now on *SampleConfig1* and *SmapleConfig2* are filled with proper values from the configuration and registered in DI, so that You can simply declare it as constructor parameters of any class:

```
public class ValuesController : ControllerBase
{
  ...
  public ValuesController(SampleConfig1 config1, SampleConfig2 config2, SampleConfig3␣
↪config3)
  {
    _config1 = config1;
    _config2 = config2;
    _config3 = config3;
  }
}
```