Access to global application settings

Asked 16 years, 4 months ago Modified 11 years, 11 months ago

Viewed 681 times

2

A database application that I'm currently working on, stores all sorts of settings in the database. Most of those settings are there to customize certain business rules, but there's also some other stuff in there.

The app contains objects that specifically do a certain task, e.g., a certain complicated calculation. Those non-UI objects are unit-tested, but also need access to lots of those global settings. The way we've implemented this right now, is by giving the objects properties that are filled by the Application Controller at runtime. When testing, we create the objects in the test and fill in values for testing (not from the database).

This works better, in any case much better than having all those objects need some global *Settings* object --- that of course effectively makes unit testing impossible :) Disadvantage can be that you sometimes need to set a dozen of properties, or that you need to let those properties 'percolate' into sub-objects.

So the general question is: how do you provide access to global application settings in your projects, without the need for global variables, while still being able to unit test

your code? This must be a problem that's been solved 100's of times...

(Note: I'm not too much of an experienced programmer, as you'll have noticed; but I love to learn! And of course, I've already done research into this topic, but I'm really looking for some first-hand experiences)

language-agnostic oop

Share

Improve this question

Follow

asked Aug 15, 2008 at 11:12

onnodb

5,261 ● 1 ● 33 ● 41

4 Answers

Sorted by: Highest score (default)

1

You could use Martin Fowlers ServiceLocator pattern. In php it could look like this:

```
class ServiceLocator {
  private static $soleInstance;
  private $globalSettings;

  public static function load($locator) {
    self::$soleInstance = $locator;
  }

  public static function globalSettings() {
    if (!isset(self::$soleInstance-
>globalSettings)) {
      self::$soleInstance->setGlobalSettings(new
GlobalSettings());
```

```
    }
    return self::$soleInstance->globalSettings;
  }
}
```

Your production code then initializes the service locator like this:

```
ServiceLocator::load(new ServiceLocator());
```

In your test-code, you insert your mock-settings like this:

```
ServiceLocator s = new ServiceLocator();
s->setGlobalSettings(new MockGlobalSettings());
ServiceLocator::load(s);
```

It's a repository for singletons that can be exchanged for testing purposes.

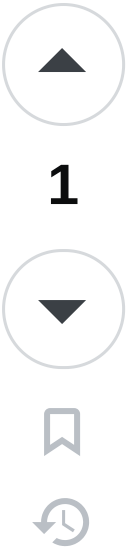
Share Improve this answer

Follow

answered Aug 15, 2008 at 12:25

Magnar

28.8k ● 8 ● 61 ● 65



I like to model my configuration access off of the Service Locator pattern. This gives me a single point to get any configuration value that I need and by putting it outside the application in a separate library, it allows reuse and testability. Here is some sample code, I am not sure what language you are using, but I wrote it in C#.

First I create a generic class that will models my ConfigurationItem.

```
public class ConfigurationItem<T>
{
    private T item;

    public ConfigurationItem(T item)
    {
        this.item = item;
    }

    public T GetValue()
    {
        return item;
    }
}
```

Then I create a class that exposes public static readonly variables for the configuration item. Here I am just reading the ConnectionStringSettings from a config file, which is just xml. Of course for more items, you can read the values from any source.

```
public class ConfigurationItems
{
    public static
ConfigurationItem<ConnectionStringSettings>
```

```
ConnectionSettings = new
ConfigurationItem<ConnectionStringSettings>
(RetrieveConnectionString());

    private static ConnectionStringSettings
RetrieveConnectionString()
    {
        // In .Net, we store our connection string
in the application/web config file.
        // We can access those values through the
ConfigurationManager class.
        return
ConfigurationManager.ConnectionStrings[Configuration
    }
}
```

Then when I need a ConfigurationItem for use, I call it like this:

```
ConfigurationItems.ConnectionSettings.GetValue();
```

And it will return me a type safe value, which I can then cache or do whatever I want with.

Here's a sample test:

```
[TestFixture]
public class ConfigurationItemsTest
{
    [Test]
    public void
ShouldBeAbleToAccessConnectionStringSettings()
    {
        ConnectionStringSettings item =
ConfigurationItems.ConnectionSettings.GetValue();
        Assert.IsNotNull(item);
    }
}
```

Hope this helps.

Share Improve this answer

Follow

answered Aug 15, 2008 at 12:57

Dale Ragan

18.3k ● 3 ● 55 ● 71

0

Usually this is handled by an ini file or XML configuration file. Then you just have a class that reads the setting when neeed.

.NET has this built in with the ConfigurationManager classes, but it's quite easy to implement, just read text files, or load XML into DOM or parse them by hand in code.

Having config files in the database is ok, but it does tie you to the database, and creates an extra dependancy for your app that ini/xml files solve.

Share Improve this answer

Follow

answered Aug 15, 2008 at 11:44

JamesSugrue

15k ● 10 ● 62 ● 93

0

I did this:

```
public class MySettings
{
    public static double Setting1
        { get { return
SettingsCache.Instance.GetDouble("Setting1"); } }
```

```
    public static string Setting2
        { get { return
SettingsCache.Instance.GetString("Setting2"); } }
}
```

I put this in a separate infrastructure module to remove any issues with circular dependencies.
Doing this I am not tied to any specific configuration method, and have no strings running havoc in my applications code.

Share Improve this answer

Follow

edited Jan 22, 2013 at 10:22

NoNaMe

6,212 ● 31 ● 85 ● 114

answered Aug 15, 2008 at 12:20

Lars Mæhlum

6,102 ● 3 ● 29 ● 32