Quick answers to common problems

.NET Framework 4.5 Expert Programming Cookbook

Over 50 engaging recipes for learning advanced concepts of .NET Framework 4.5

A.P. Rajshekhar

[PACKT] PUBLISHING enterprise professional expertise distilled

.NET Framework 4.5 Expert Programming Cookbook

Over 50 engaging recipes for learning advanced concepts of .NET Framework 4.5

A.P. Rajshekhar

[PACKT] PUBLISHING enterprise professional expertise distilled

BIRMINGHAM - MUMBAI

.NET Framework 4.5 Expert Programming Cookbook

First published: January 2013

Production Reference: 1110113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84968-742-3

www.packtpub.com

Cover Image by David Gutierrez (bilbaorocker@yahoo.co.uk)

Credits

Author

A.P. Rajshekhar

Reviewers

Wei Chung, Low

Jason De Oliveira

Acquisition Editor

James Jones

Lead Technical Editor

Ankita Shashi

Technical Editors

Jalasha D'costa

Worrell Lewis

Varun Pius Rodrigues

Copy Editors

Brandt D'Mello

Aditya Nair

Laxmi Subramanian

Ruta Waghmare

Project Coordinators

Priya Sharma

Abhishek Kori

Proofreader

Lawrence A. Herman

Indexer

Rekha Nair

Production Coordinator

Shantanu Zagade

Cover Work

Shantanu Zagade

About the Author

A.P. Rajshekhar, Senior Developer with Red Hat, has more than 7 years of experience in IT, having worked on applications ranging from enterprise-level web applications and game development to android applications. His endeavors include development of Learning Management System, Health Systems, Supply Management Solution, and Xbox-based games. He has extensive knowledge of different technologies (SOA, Portal, Java Persistence, and .NET Persistence) and platforms (Sharepoint and JBoss EAP). He is also the author of *Building Dynamic Web 2.0 Websites with Ruby on Rails*, *Packt Publishing*, that was in Amazon's top 50 in Web 2.0 for 6 months. Apart from that he has also contributed to DevShed Portal on topics ranging from server-side development (JEE/.NET/RoR) to mobile (Symbian/Android-based development) and game development (SDL and OpenGL) with a total readership of more than 3 million. He is currently ranked among the top 10 authors on DevShed. You can find out about his interests on his blogs – `http://aprajshekhar.wordpress.com` and `http://sententiasupervicis.wordpress.com`.

Authoring a book is not an easy feat. However, the help and guidance from my family and friends helped me to author this book. First, I would like to thank Packt Publishing for providing me an opportunity to work on such an exciting project. I would like to thank my parents for their constant encouragement. Special thanks to my friends Shrikant Khare and Sormita Chakraborty for their support, encouragement, and initial research on the topics to be covered.

About the Reviewers

Wei Chung, Low is a Business Intelligence Manager, a .NET developer, and a MCT, MCPD, MCITP, MCTS, MCSD.NET. He works with IPG MediaBrands (NYSE: IPG) at its Kuala Lumpur, Malaysia campus. He is also a member of PMI, certified as PMP. He started working on Microsoft .NET early in his career and has been involved in development, consultation, and corporate training in the areas of business intelligence, system integration, and virtualization. He has worked for the Bursa Malaysia (formerly Kuala Lumpur Stock Exchange) and Shell IT International previously, which gave him rich integration experiences across different platforms.

He strongly believes that a good system implementation delivers precious value to businesses, and integration of various systems across different platforms shall always be a part of it, just as diverse people from different cultures live together in harmony in most of the major cities.

Jason De Oliveira works as CTO for Cellenza (`http://www.cellenza.com`), an IT Consulting company specializing in Microsoft technologies and Agile methodology in Paris, France. He is an experienced Manager and Senior Solutions Architect, with advanced skills in Software Architecture and Enterprise Architecture.

Jason works for big companies and helps them to realize complex and challenging software projects. He frequently collaborates with Microsoft and you can quite often find him at the Microsoft Technology Center (MTC) in Paris.

He loves sharing his knowledge and experience via his blog, by speaking at conferences, writing technical books, writing articles in the technical press, giving software courses as a Microsoft Certified Trainer (MCT), and coaching co-workers in his company.

Microsoft has awarded him the Microsoft® Most Valuable Professional (MVP C#) Award since 2011 for his numerous contributions to the Microsoft community. Microsoft seeks to recognize the best and brightest from technology communities around the world with the MVP Award. These exceptional and highly respected individuals come from more than 90 countries, serving their local online and offline communities and have an impact worldwide. Jason is very proud to be one of them.

Please feel free to contact him via his blog if you need any technical assistance or want to exchange information on technical subjects (`http://www.jasondeoliveira.com`).

Jason has worked on the following books:

- *WCF 4.5 Multi-tier Services Development with LINQ to Entities*, *Packt Publishing*
- *Visual Studio 2012 - Développez pour le web, ENI Publishing*

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Instant Updates on New Packt Books

Get notified! Find out when new books are published by following `@PacktEnterprise` on Twitter, or the *Packt Enterprise* Facebook page.

Table of Contents

Preface

.NET is an architecture-neutral programming language and agnostic framework that caters to the varying requirements from desktop application, to business solutions, to multiplayer online three-dimensional games. The Version 4.5 added many new features and enhanced the existing ones that help in the development of robust and user-friendly solutions more easily. *.NET Framework 4.5 Expert Programming Cookbook* takes a hands-on approach in teaching you how to use the new as well as advanced features of the .NET Framework 4.5. Each topic will teach you how to use a specific feature of .NET to solve a real world problem or scenario.

This is a concise and practical cookbook with recipes which demonstrates advanced concepts with all the new functionality of the .NET Framework 4.5.

What this book covers

Chapter 1, *Core .NET Recipes*, will cover the core concepts in .NET, which include metadata programming, reflection, asynchronous I/O, and dynamic programming.

Chapter 2, *Application Events and Windows Forms*, covers topics such as event handling, dynamically generating controls, and layouts as well as creating video players using Managed DirectX.

Chapter 3, *Threading and Parallel Programming*, will cover multi-threading, thread-safety, and the parallel framework extensions to avoid threading pitfalls in your Windows Forms, WPF, and Silverlight applications.

Chapter 4, *ASP.NET Recipes – I*, explains the new features of ASP.NET applications including strongly-typed controls, HTML 5 controls, and client-side storage as well as passing data between Silverlight and the ASPX page.

Chapter 5, *ADO.NET Recipes*, covers saving and retrieving files of big size (BLOB) in SQL Server, managing transactions, and using DataSet to operate upon XML data.

Chapter 6, *WCF Recipes*, explains uploading files using streamed mode, implementing REST services, handling exceptions using FaultContract, implementing custom binding, and securing services using role-based security.

Chapter 7, *WPF Recipes*, will cover design patterns that include MVVM, repository pattern, and Data Mapper as well as new controls such as Ribbon control and live data shaper. It will also cover creating a video player using WPF Media API.

Chapter 8, *ASP.NET Recipes – II*, covers the new features of ASP.NET websites such as enabling Google/Facebook, SSO-based authentication, adding unobtrusive validation, embedding maps in websites, and protecting against cross-server scripting attacks.

Chapter 9, *Silverlight Recipes*, explains the new pivot control, accessing webcams, and client-side storage.

Chapter 10, *Entity Framework Recipes*, will cover using LINQ to join multiple entities, calling stored procedures using Entity Framework, handling long-running tasks, and using table-valued functions of MS SQL Server.

What you need for this book

You will need Visual Studio 2012, MS SQL Server 2008 or higher, Windows 7 or higher, and hardware compatible with Windows 7 or higher. A webcam will be required to run certain recipes.

Who this book is for

This book is for those who have basic to intermediate knowledge about the .NET Framework and want to understand its advanced features and features new to .NET 4.5. Each advanced feature covered in this book assumes that the reader has the basic knowledge of the concept being discussed.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning:

Code words in text are shown as follows: "The core of the validation logic lies in the `IsUsernameUnique` method of the `MockRespository` class."

A block of code is set as follows:

```
public interface IRepository
{
  void AddUser(User user);
  bool IsUsernameUnique(string userName);
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
foreach (string filename in Directory.EnumerateFiles(sourceDir))
{
  using (FileStream sourceStream = File.Open(filename, FileMode.Open))
  {
  }
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Then click on the **Copy** button to start copying."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title through the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, or added to any list of existing errata, under the Errata section of that title.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Core .NET Recipes

In this chapter we will cover:

- Implementing the validation logic using the Repository pattern
- Creating a custom validation attribute by extending the validation data annotation
- Using XML to generate a localized validation message
- Extending the validation attribute for localization
- Creating custom attributes
- Processing custom attributes via reflection
- Using asynchronous file I/O for directory-to-directory copy
- Accessing JSON using dynamic programming

Introduction

This chapter will cover recipes related to core concepts in .NET, which will include the following:

- **Metadata-driven programming**: The first six recipes will cover how to use attributes as metadata for specific purposes such as validation and localization.
- **Reflection**: The *Processing custom attributes via reflection* recipe will tell you how to use reflection to create metadata processors such as applications or libraries that can understand custom attributes and provide the output based on them.
- **Asynchronous file I/O**: This is a new feature for file input/output introduced in .NET 4.5. The *Using asynchronous file I/O for directory-to-directory copy* recipe will cover this feature.

- **Dynamic programming**: .NET 4.0 introduced the concept of dynamic programming, in which blocks of code marked as dynamic will be executed directly, bypassing the compilation phase. We will look at this in the last recipe, *Accessing JSON using dynamic programming*.

Implementing the validation logic using the Repository pattern

The **Repository pattern** abstracts out data-based validation logic. It is a common misconception that to implement the Repository pattern you require a relational database such as MS SQL Server as the backend. Any collection can be treated as a backend for a Repository pattern. The only point to keep in mind is that the business logic or validation logic must treat it as a database for saving, retrieving, and validating its data. In this recipe, we will see how to use a generic collection as backend and abstract out the validation logic for the same.

The validation logic makes use of an entity that represents the data related to the user and a class that acts as the repository for the data allowing certain operations. In this case, the operation will include checking whether the user ID chosen by the user is unique or not.

How to do it...

The following steps will help check the uniqueness of a user ID that is chosen by the user:

1. Launch Visual Studio .NET 2012. Create a new project of **Class Library** project type. Name it `CookBook.Recipes.Core.CustomValidation`.
2. Add a folder to the project and set the folder name to `DataModel`.
3. Add a new class and name it `User.cs`.
4. Open the `User` class and create the following public properties:

Property name	Data type
`UserName`	`String`
`DateOfBirth`	`DateTime`
`Password`	`String`

Use the automatic property functionality of .NET to create the properties. The final code of the `User` class will be as follows:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    /// <summary>
    /// Contains details of the user being registered
```

```
    /// </summary>
    public class User
    {

        public string UserName { get; set; }
        public DateTime DateOfBirth { get; set; }
        public string Password { get; set; }

    }
}
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

5. Next, let us create the repository. Add a new folder and name it `Repository`.
6. Add an interface to the `Repository` folder and name it `IRepository.cs`.
7. The interface will be similar to the following code snippet:

```
public interface IRepository
    {
    }
```

8. Open the `IRepository` interface and add the following methods:

Name	Description	Parameter(s)	Return Type
`AddUser`	Adds a new user	User object	`Void`
`IsUsernameUnique`	Determines whether the username is already taken or not	string	`Boolean`

After adding the methods, `IRepository` will look like the following code:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    public interface IRepository
    {
        void AddUser(User user);
        bool IsUsernameUnique(string userName);
    }
}
```

9. Next, let us implement `IRepository`. Create a new class in the `Repository` folder and name it `MockRepository`.

10. Make the `MockRepository` class implement `IRepository`. The code will be as follows:

```
namespace CookBook.Recipes.Core.Data.Repository
{
   public  class MockRepository:IRepository
    {
       #region IRepository Members
        /// <summary>
        /// Adds a new user to the collection
        /// </summary>
        /// <param name="user"></param>
        public void AddUser(User user)
        {

        }
        /// <summary>
        /// Checks whether a user with the username already
        ///exists
        /// </summary>
        /// <param name="userName"></param>
        /// <returns></returns>
        public bool IsUsernameUnique(string userName)
        {

        }

        #endregion
    }
}
```

11. Now, add a private variable of type `List<Users>` in the `MockRepository` class. Name it `_users`. It will hold the registered users. It is a `static` variable so that it can hold usernames across multiple instantiations.

12. Add a constructor to the class. Then initialize the `_users` list and add two users to the list:

```
public  class MockRepository:IRepository
    {
        #region Variables
        Private static List<User> _users;

        #endregion
        public MockRepository()
        {
            _users = new List<User>();
            _users.Add(new User() { UserName = "wayne27",
DateOfBirth = new DateTime(1950, 9, 27), Password = "knight"
});
```

```
            _users.Add(new User() { UserName = "wayne47",
DateOfBirth = new DateTime(1955, 9, 27), Password = "justice"
});
        }
        #region IRepository Members
        /// <summary>
        /// Adds a new user to the collection
        /// </summary>
        /// <param name="user"></param>
        public void AddUser(User user)
        {
        }
        /// <summary>
        /// Checks whether a user with the username already exists
        /// </summary>
        /// <param name="userName"></param>
        /// <returns></returns>
        public bool IsUsernameUnique(string userName)
        {

        }

        #endregion
 }
```

13. Now let us add the code to check whether the username is unique. Add the following statements to the `IsUsernameUnique` method:

```
bool exists = _users.Exists(u=>u.UserName==userName);
return !exists;
```

The method turns out to be as follows:

```
public bool IsUsernameUnique(string userName)
{
     bool exists =
     _users.Exists(u=>u.UserName==userName);
     return !exists;
}
```

14. Modify the `AddUser` method so that it looks as follows:

```
  public void AddUser(User user)
  {
            _users.Add(user);
  }
```

How it works...

The core of the validation logic lies in the `IsUsernameUnique` method of the `MockRespository` class. The reason to place the logic in a different class rather than in the attribute itself was to decouple the attribute from the logic to be validated. It is also an attempt to make it future-proof. In other words, tomorrow, if we want to test the username against a list generated from an XML file, we don't have to modify the attribute. We will just change how `IsUsernameUnique` works and it will be reflected in the attribute. Also, creating a **Plain Old CLR Object** (**POCO**) to hold values entered by the user stops the validation logic code from directly accessing the source of input, that is, the Windows application.

Coming back to the `IsUsernameUnique` method, it makes use of the **predicate** feature provided by .NET. Predicate allows us to loop over a collection and find a particular item. Predicate can be a static function, a delegate, or a lambda. In our case it is a lambda.

```
bool exists = _users.Exists(u=>u.UserName==userName);
```

In the previous statement, .NET loops over `_users` and passes the current item to `u`. We then make use of the item held by `u` to check whether its username is equal to the username entered by the user. The `Exists` method returns true if the username is already present. However, we want to know whether the username is unique. So we flip the value returned by `Exists` in the return statement, as follows:

```
return !exists;
```

Creating a custom validation attribute by extending the validation data annotation

.NET provides **data annotations** as a part of the `System.ComponentModel.DataAnnotation` namespace. Data annotations are a set of attributes that provides out of the box validation, among other things. However, sometimes none of the in-built validations will suit your specific requirements. In such a scenario, you will have to create your own validation attribute. This recipe will tell you how to do that by extending the validation attribute. The attribute developed will check whether the supplied username is unique or not. We will make use of the validation logic implemented in the previous recipe to create a custom validation attribute named `UniqueUserValidator`.

How to do it...

The following steps will help you create a custom validation attribute to meet your specific requirements:

1. Launch Visual Studio 2012. Open the `CustomValidation` solution.
2. Add a reference to `System.ComponentModel.DataAnnotations`.

3. Add a new class to the project and name it `UniqueUserValidator`.
4. Add the following using statements:

```
using System.ComponentModel.DataAnnotations;
using CookBook.Recipes.Core.CustomValidation.MessageRepository;
using CookBook.Recipes.Core.Data.Repository;
```

5. Derive it from `ValidationAttribute`, which is a part of the `System.ComponentModel.DataAnnotations` namespace. In code, it would be as follows:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    public class UniqueUserValidator:ValidationAttribute
    {

    }
}
```

6. Add a property of type `IRepository` to the class and name it `Repository`.
7. Add a constructor and initialize `Repository` to an instance of `MockRepository`. Once the code is added, the class will be as follows:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    public class UniqueUserValidator:ValidationAttribute
    {
        public IRepository Repository {get;set;}

        public UniqueUserValidator()
        {
            this.Repository = new MockRepository();
        }
    }
}
```

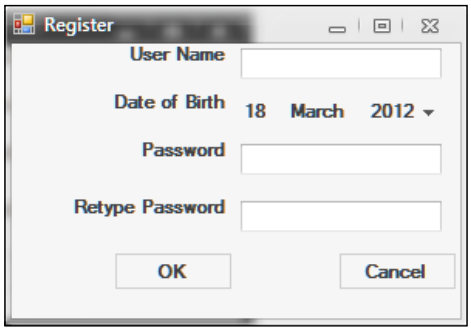
8. Override the `IsValid` method of `ValidationAttribute`. Convert the `object` argument to `string`.
9. Then call the `IsUsernameUnique` method of `IRepository`, pass the string value as a parameter, and return the result. After the modification, the code will be as follows:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    public class UniqueUserValidator:ValidationAttribute
    {
        public IRepository Repository {get;set;}
```

```
            public UniqueUserValidator()
            {
                this.Repository = new MockRepository();
            }
            public override bool IsValid(object value)
            {
                string valueToTest = Convert.ToString(value);
                return this.Repository.IsUsernameUnique(valueToTest);
            }
        }
    }
```

We have completed the implementation of our custom validation attribute. Now let's test it out.

10. Add a new **Windows Forms Application** project to the solution and name it `CustomValidationApp`.
11. Add a reference to the `System.ComponentModel.DataAnnotations` and `CustomValidation` projects.
12. Rename `Form1.cs` to `Register.cs`.
13. Open `Register.cs` in the design mode. Add controls for username, date of birth, and password and also add two buttons to the form. The form should look like the following screenshot:



14. Name the input control and button as given in the following table:

Control	Name
Textbox	`txtUsername`
Button	`btnOK`

Since we are validating the `User Name` field, our main concern is with the textbox for the username and the **OK** button. I have left out the names of other controls for brevity.

15. Switch to the code view mode. In the constructor, add event handlers for the Click event of `btnOK` as shown in the following code:

```
public Register()
{
 InitializeComponent();
 this.btnOK.Click += new EventHandler(btnOK_Click);
}

void btnOK_Click(object sender, EventArgs e)
{
}
```

16. Open the `User` class of the `CookBook.Recipes.Core.CustomValidation` project. Annotate the `UserName` property with `UniqueUserValidator`. After modification, the `User` class will be as follows:

```
namespace CookBook.Recipes.Core.CustomValidation
{
    /// <summary>
    /// Contains details of the user being registered
    /// </summary>
    public class User
    {
        [UniqueUserValidator(ErrorMessage="User name already
exists")]
        public string UserName { get; set; }
        public DateTime DateOfBirth { get; set; }
        public string Password { get; set; }

    }
}
```

17. Go back to `Register.cs` in the code view mode.
18. Add the following `using` statements:

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using CookBook.Recipes.Core.CustomValidation;
using CookBook.Recipes.Core.Data.Repository;
```

19. Add the following code to the event handler of `btnOK`:

```
//create a new user
User user = new User()
{
UserName = txtUsername.Text, DateOfBirth=dtpDob.Value
};
//create a validation context for the user instance
ValidationContext context = new ValidationContext(user, null,
null);
//holds the validation errors
IList<ValidationResult> errors = new List<ValidationResult>();

if (!Validator.TryValidateObject(user, context, errors, true))
{
 foreach (ValidationResult result in errors)
      MessageBox.Show(result.ErrorMessage);
}
else
{
  IRepository repository = new MockRepository();
  repository.AddUser(user);
  MessageBox.Show("New user added");
}
```

20. Hit *F5* to run the application. In the textbox add a username, say, `dreamwatcher`. Click on **OK**. You will get a message box stating **User has been added successfully**.
21. Enter the same username again and hit the **OK** button. This time you will get a message saying **User name already exists**. This means our attribute is working as desired.
22. Go to **File** | **Save Solution As...**, enter `CustomValidation` for **Name**, and click on **OK**.

We will be making use of this solution in the next recipe.

How it works...

To understand how `UniqueUserValidator` works, we have to understand how it is implemented and how it is used/called. Let's start with how it is implemented. It extends `ValidationAttribute`. The `ValidationAttribute` class is the base class for all the validation-related attributes provided by data annotations. So the declaration is as follows:

```
public class UniqueUserValidator:ValidationAttribute
```

This allowed us to make use of the public and protected methods/attribute arguments of `ValidationAttribute` as if it is a part of our attribute. Next, we have a property of type `IRepository`, which gets initialized to an instance of `MockRepository`. We have used the interface-based approach so that the attribute will only need a minor change if we decide to test the username against a database table or list generated from a file. In such a scenario, we will just change the following statement:

```
this.Repository = new MockRepository();
```

The previous statement will be changed to something such as the following:

```
this.Repository = new DBRepository();
```

Next, we overrode the `IsValid` method. This is the method that gets called when we use `UniqueUserValidator`. The parameter of the `IsValid` method is an object. So we have to typecast it to string and call the `IsUniqueUsername` method of the `Repository` property. That is what the following statements accomplish:

```
string valueToTest = Convert.ToString(value);
return this.Repository.IsUsernameUnique(valueToTest);
```

Now let us see how we used the validator. We did it by decorating the `UserName` property of the `User` class:

```
[UniqueUserValidator(ErrorMessage="User name already exists")]
public string UserName {get; set;}
```

As I already mentioned, deriving from `ValidatorAttribute` helps us in using its properties as well. That's why we can use `ErrorMessage` even if we have not implemented it.

Next, we have to tell .NET to use the attribute to validate the username that has been set. That is done by the following statements in the **OK** button's Click handler in the `Register` class:

```
ValidationContext context = new ValidationContext(user, null, null);
//holds the validation errors
IList<ValidationResult> errors = new List<ValidationResult>();
if (!Validator.TryValidateObject(user, context, errors, true))
```

First, we instantiate an object of `ValidationContext`. Its main purpose is to set up the context in which validation will be performed. In our case the context is the `User` object. Next, we call the `TryValidateObject` method of the `Validator` class with the `User` object, the `ValidationContext` object, and a list to hold the errors. We also tell the method that we need to validate all properties of the `User` object by setting the last argument to true. That's how we invoke the validation routine provided by .NET.