**[[](http://afana.me/)Nadeem Afana](http://afana.me/)**

[Home](http://afana.me/)

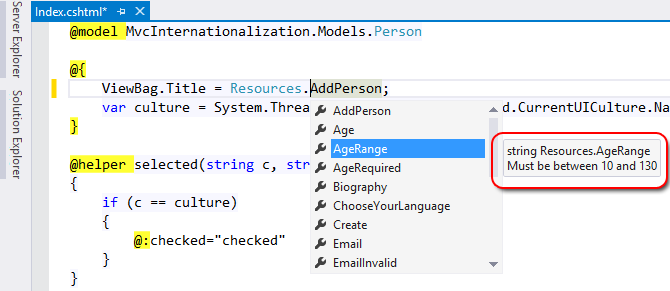
© 2017. All rights reserved.

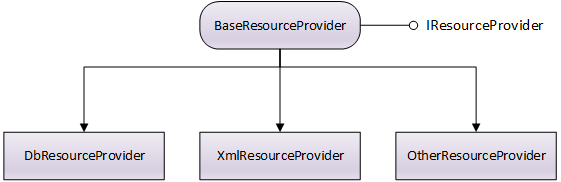
**ASP.NET MVC 5 Internationalization · How to Store Strings in a Database or Xml**

01 Nov 2013

[**Download Code**](http://afana.me/attachments/posts/archived/aspnet-mvc-internationalization-store-strings-in-database-or-xml.zip)

In the previous post, [ASP.NET MVC 5 Internationalization](http://afana.me/post/aspnet-mvc-internationalization.aspx), I showed how to store the localization strings in ResX files. There is no technical reason why you cannot store the localization strings in a database or an XML file, or even in a text file located over a network.

In this post, I will provide a simple yet flexible architecture that allows for storing localization strings in any source (Database, Xml, or any). The provider will also support data annotations and generate a strongly typed resource class. 

Note that you can see a summary in the tooltip box when you highlight a resource. This allows you to preview a resource without going to the data source (Database, xml, etc) 

IResourceProvier simply has one method called GetResource.

public interface IResourceProvider

{

object GetResource(string name, string culture);

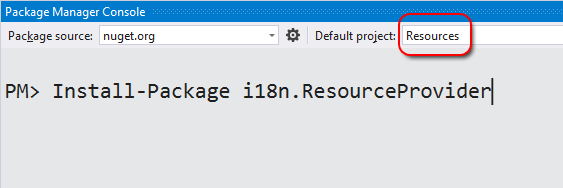
}

This is the most basic requirement to have resources stored in a different data source. BaseResourceProvider implements this interface and provides some benefits such as caching (which is recommended for performance). BaseResourceProvider has two methods to be implemented: ReadResource and ReadResources. If you plan to have caching always turned on, then you only need to implement ReadResources.

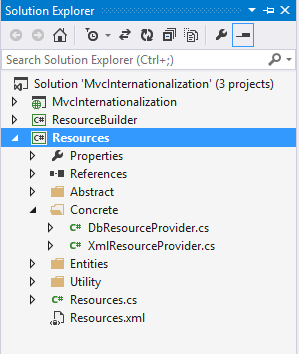
The demo in this post is based on the solution in the previous post [ASP.NET MVC 5 Internationalization](http://afana.me/post/aspnet-mvc-internationalization.aspx)

To add these classes to your project, open Package Manager Console, select your resources project and type

Install-Package i18n.ResourceProvider



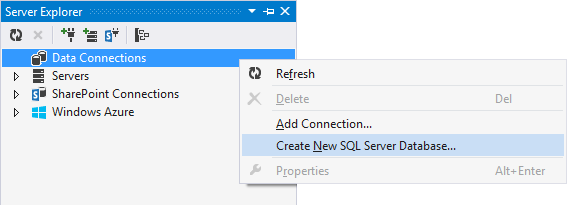
This will also install two providers for you: DbResourceProvider and XmlResourceProvider:

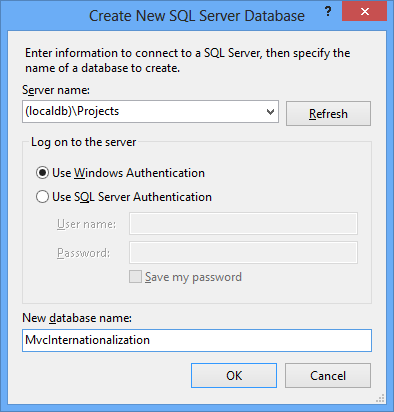


**How to store resources in a Database**

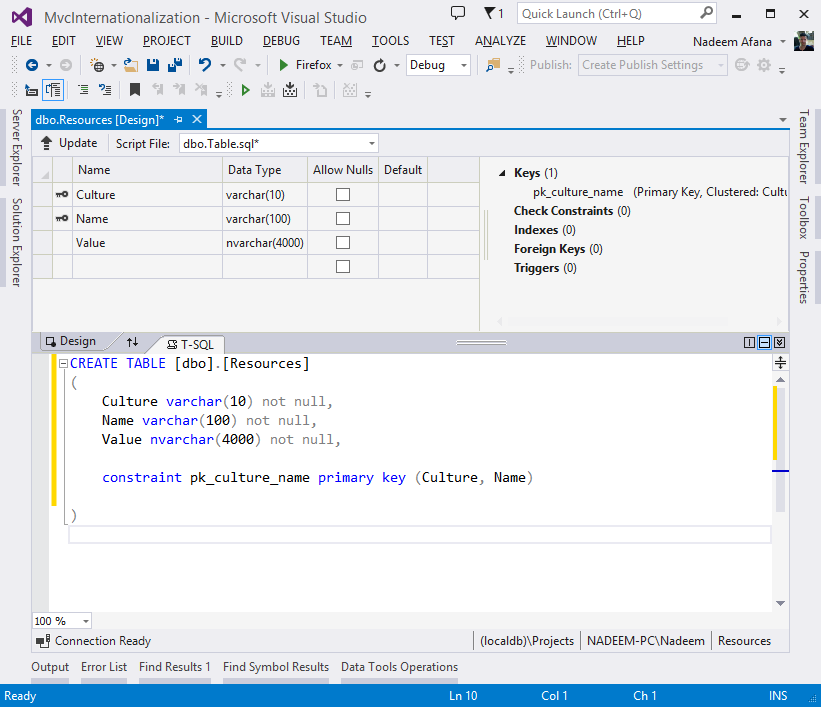
First, we will use the database provider. The first thing to do is add the resource strings into our database.

Open **Visual Studio Server Explorer**, Right click on **Create New SQL Server database**.





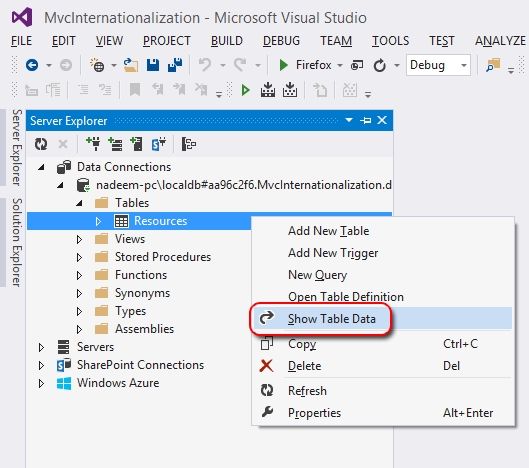
Create a new table called Resources.

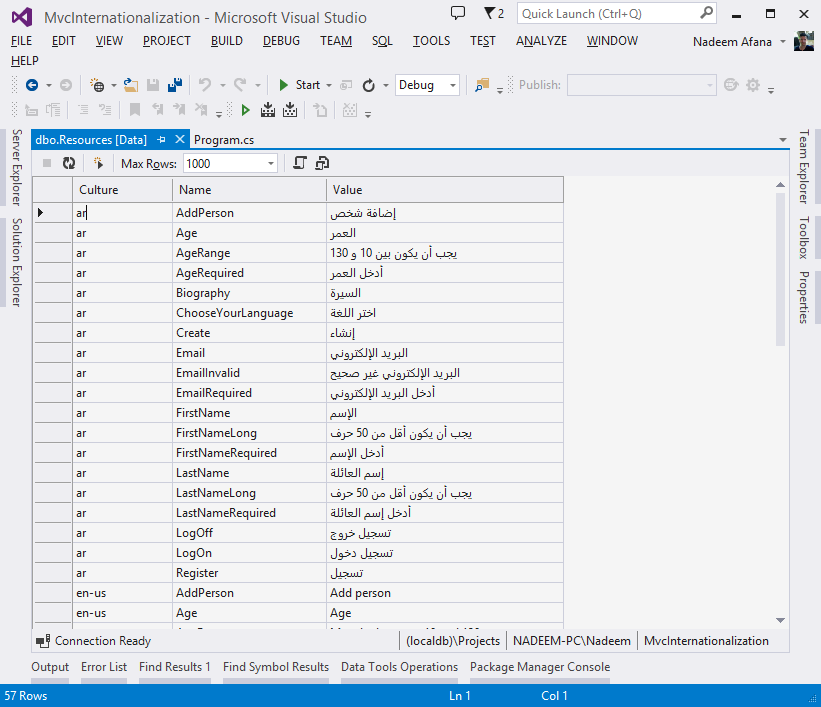


he primary key is a combination of two columns: culture and name. The data type of Value is nvarchar, which allows to store any language text. In this example, I stored all cultures in the same table, but you are not limited to this physical structure. You can change the table layout as you want by removing or adding more fields. You can even have a separate table per culture. It is flexible!

If you change the structure, you will need to update the provider DbResourceProvider.

Populate the table with all the resource strings (You can find them in the [**attachment**](http://afana.me/attachments/posts/archived/aspnet-mvc-internationalization-store-strings-in-database-or-xml.zip) in an SQL file)





Now delete the .resx resource files as they are not needed anymore.

The database provider code looks like the following:

using Resources.Abstract;

using Resources.Entities;

namespace Resources.Concrete

{

public class DbResourceProvider : BaseResourceProvider

{

// Database connection string

private static string connectionString = null;

public DbResourceProvider(){

connectionString = ConfigurationManager.ConnectionStrings["MvcInternationalization"].ConnectionString;

}

public DbResourceProvider(string connection)

{

connectionString = connection;

}

protected override IList<resourceentry> ReadResources()

{

var resources = new List<resourceentry>();

const string sql = "select Culture, Name, Value from dbo.Resources;";

using (var con = new SqlConnection(connectionString)) {

var cmd = new SqlCommand(sql, con);

con.Open();

using (var reader = cmd.ExecuteReader()) {

while (reader.Read()) {

resources.Add(new ResourceEntry {

Name = reader["Name"].ToString(),

Value = reader["Value"].ToString(),

Culture = reader["Culture"].ToString()

});

}

if (!reader.HasRows) throw new Exception("No resources were found");

}

}

return resources;

}

protected override ResourceEntry ReadResource(string name, string culture)

{

ResourceEntry resource = null;

const string sql = "select Culture, Name, Value from dbo.Resources where culture = @culture and name = @name;";

using (var con = new SqlConnection(connectionString)) {

var cmd = new SqlCommand(sql, con);

cmd.Parameters.AddWithValue("@culture", culture);

cmd.Parameters.AddWithValue("@name", name);

con.Open();

using (var reader = cmd.ExecuteReader()) {

if (reader.Read()) {

resource = new ResourceEntry {

Name = reader["Name"].ToString(),

Value = reader["Value"].ToString(),

Culture = reader["Culture"].ToString()

};

}

if (!reader.HasRows) throw new Exception(string.Format("Resource {0} for culture {1} was not found", name, culture));

}

}

return resource;

}

}

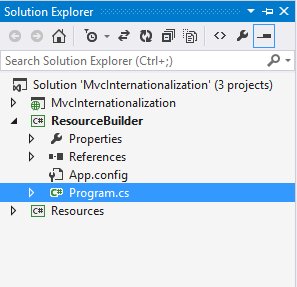
}

I used plain ADO.NET to access the resources, but you can use Entity Framework or any other tool you want.

Nothing needs to be changed in the Model class Person unless you have changed the namespace or the resources assembly. However, you will need to add or change the database connection string in the main Web.config file.

Now the final step is to generate a strongly typed resource class that will expose each resource name (ie key) as a static property. Since the resources are dynamic, I decided to write a helper tool (ResourceBuilder) to generate a C# class based on the resource provider you are implementing.

I prefer to create a new Console Application project and add it to that solution. This way it is separate from your main project, and every time you remove or add new resources in the future, you can just run this project and get a fresh copy.



using Resources.Utility;

using Resources.Concrete;

namespace ResourceBuilder

{

class Program

{

static void Main(string[] args)

{

var builder = new Resources.Utility.ResourceBuilder();

string filePath = builder.Create(new DbResourceProvider(@"Data Source=(localdb)\Projects;Initial Catalog=MvcInternationalization;Integrated Security=True;Pooling=False"),

summaryCulture: "en-us");

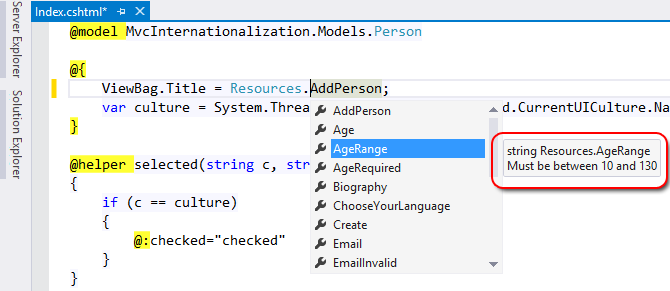
Console.WriteLine("Created file {0}", filePath);

}

}

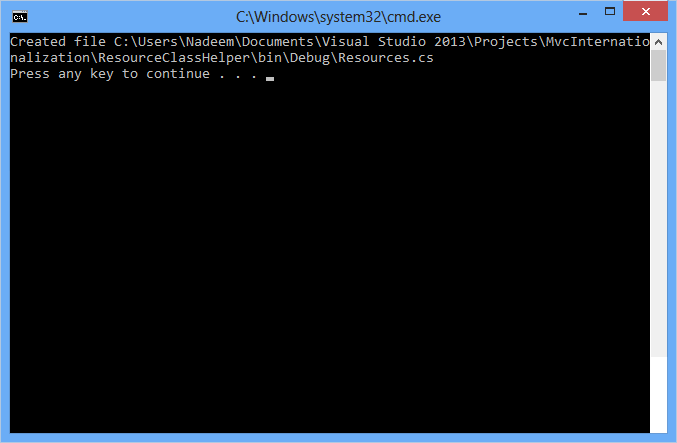
}

The ResourceBuilder.Create method accepts several parameters. The only required one is the resource provider instance from which the resources will be pulled. The summaryCulture parameter is optional and is intended to help you see the resource value summary while you type them (without going to the database to check the value):



summaryCulture accepts one of the cultures that you already implemented, and it displays the value for each resource in a summary tooltip. The other method parameters are documented in the source file.

Running the program generates a C# source file:



Copy the generated file into the Resource project. The file looks like this:

namespace Resources {

public class Resources {

private static IResourceProvider resourceProvider = new DbResourceProvider();

/// <summary>Add person</summary>

public static string AddPerson

{

get

{

return (string) resourceProvider.GetResource("AddPerson", CultureInfo.CurrentUICulture.Name);

}

}

/// <summary>Age</summary>

public static string Age

{

get

{

return (string) resourceProvider.GetResource("Age", CultureInfo.CurrentUICulture.Name);

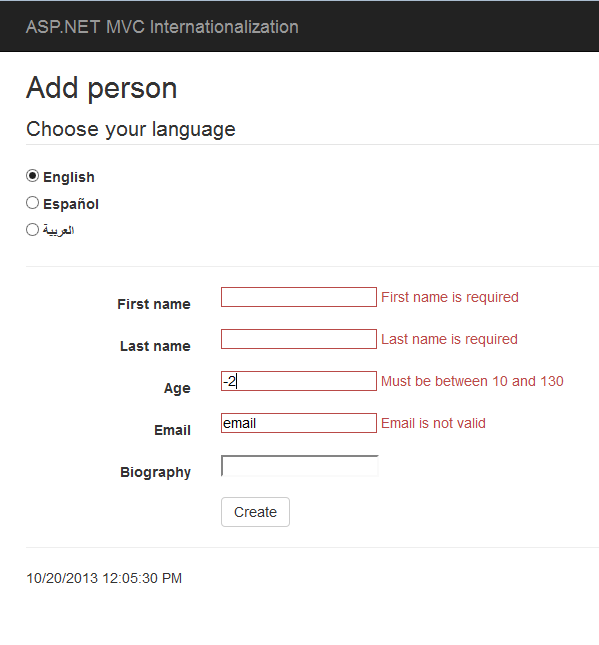
}

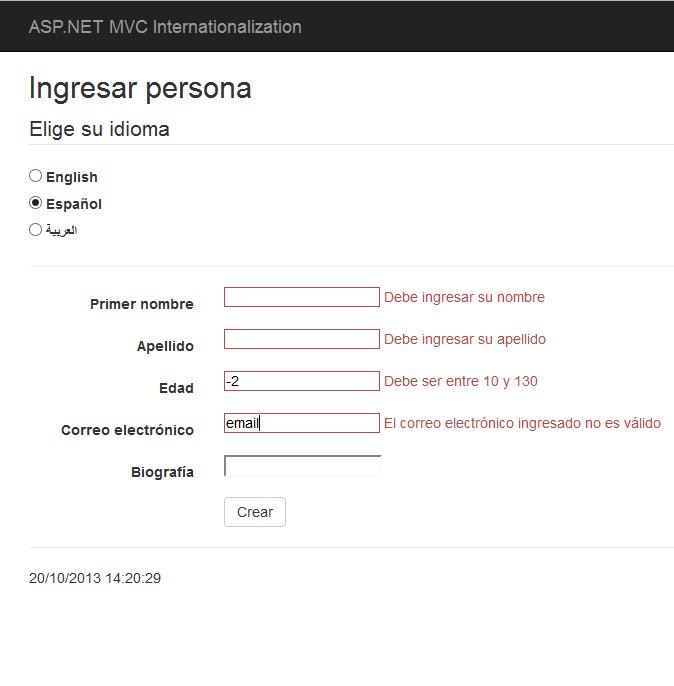
}

// and so on

}

}

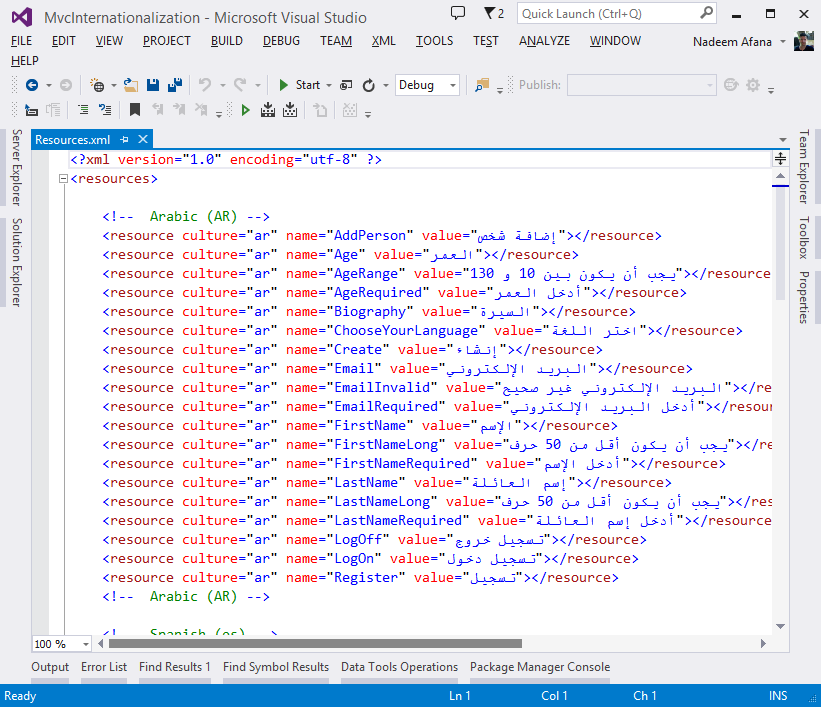






**How to store resources in Xml**

Create a new XML file called Resoucres.xml under the Resources project. Again, the file structure can be different from the one used in this example. However, if you change the XML file layout, you will need to update XmlResourceProvider.



Now you can either re-generate the resource class or simply update the provider property like below:

private static IResourceProvider resourceProvider = new XmlResourceProvider(

Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"bin\Resources.xml")

); // assume Resources.xml is in the bin folder

The Xml provider looks like this:

using Resources.Abstract;

using Resources.Entities;

namespace Resources.Concrete

{

public class XmlResourceProvider: BaseResourceProvider

{

// File path

private static string filePath = null;

public XmlResourceProvider(){}

public XmlResourceProvider(string filePath)

{

XmlResourceProvider.filePath = filePath;

if (!File.Exists(filePath)) throw new FileNotFoundException(

string.Format("XML Resource file {0} was not found", filePath)

);

}

protected override IList<ResourceEntry> ReadResources()

{

// Parse the XML file

return XDocument.Parse(File.ReadAllText(filePath))

.Element("resources")

.Elements("resource")

.Select(e => new ResourceEntry {

Name = e.Attribute("name").Value,

Value = e.Attribute("value").Value,

Culture = e.Attribute("culture").Value

}).ToList();

}

protected override ResourceEntry ReadResource(string name, string culture)

{

// Parse the XML file

return XDocument.Parse(File.ReadAllText(filePath))

.Element("resources")

.Elements("resource")

.Where(e => e.Attribute("name").Value == name && e.Attribute("culture").Value == culture)

.Select(e => new ResourceEntry {

Name = e.Attribute("name").Value,

Value = e.Attribute("value").Value,

Culture = e.Attribute("culture").Value

}).FirstOrDefault();

}

}

}

I used LINQ to XML to access the resources, but you can use any method you like!

I hope this helps. Any questions or comments are welcome.

**Related Posts**

* [**How MIT Keytab Files Store Passwords 29 Dec 2016**](http://afana.me/archive/2016/12/29/how-mit-keytab-files-store-passwords.aspx/)
* [**.NET Cryptographic Functions 27 Dec 2016**](http://afana.me/archive/2016/12/27/dot-net-cryptographic-functions.aspx/)
* [**.NET AES CTS Implementation 15 Nov 2016**](http://afana.me/archive/2016/11/15/dot-net-aes-cts.aspx/)