# Les nouveautés d'ASP.Net 4.0

par Louis-Guillaume MORAND (Page perso)

Date de publication : 07 November 2009

Dernière mise à jour :

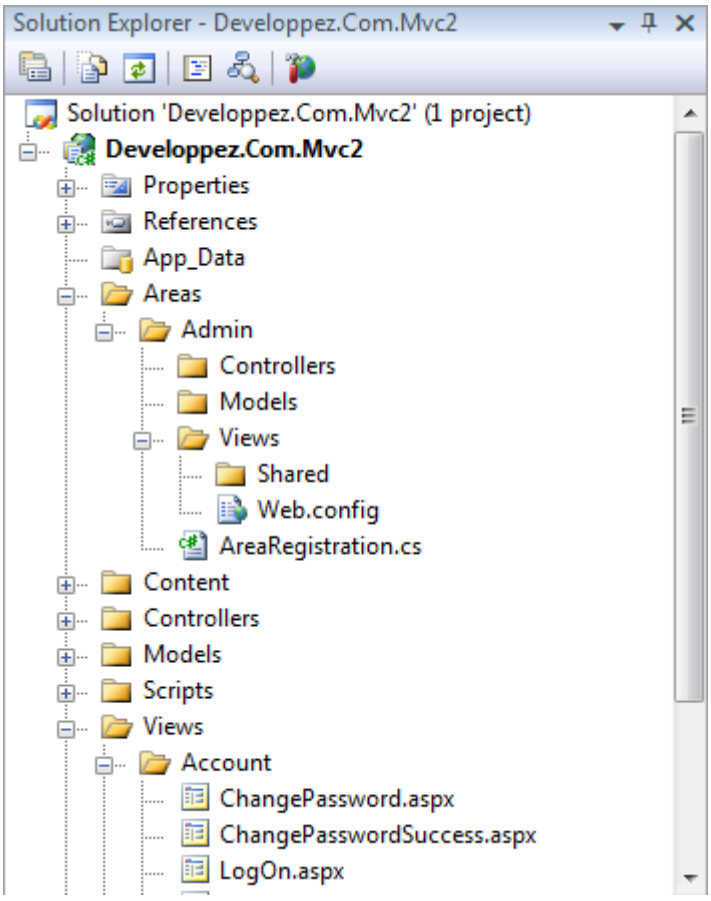
## 0 - Introduction

Après une toute première version sortie en 2009, Phil Haack et son équipe sont de nouveau au travail pour sortir la version d'ASP .Net MVC 2, qui sera livrée directement au sein de Visual Studio 2010 dont la sortie est prévue fin mars 2010. Cette nouvelle version apporte son lot de nouveautés tire parti du framework 4. Néanmoins, ASP .Net sera compilée à l'aide du framework 3.5 SP1 afin de rendre 100% de ses fonctionnalités accessibles, y compris pour ceux qui restent en ASP .Net 3.5 SP1.

## 1 - Support des Areas

La première chose que l'on apprécie dans un projet ASP .Net MVC, c'est la structure organisée, découpée et claire du projet de notre site Web. Bien que non obligatoire, il est courant de voir un répertoire pour les vues, un répertoire pour les contrôleurs et un autre pour le modèle.
Malheureusement, plus notre site Web évolue et gagne en fonctionnalités et pages, plus notre projet se compléxifie et plus il est difficile de s'y retrouver. Imaginez un site où le nombre de contrôleurs atteindrait aisément la cinquantaine, le nombre de vues peut alors atteindre la centaine voire bien plus. Comment s'y retrouver rapidement parmi cette multitude de répertoire?
La solution passe alors par l'utilisation des **Areas**.

Une Area est un découpage de votre site afin de regrouper ensemble certains modèles, certaines vues et certains contrôleurs. Une image étant parfois plus parlante qu'un long discours, observez la capture suivante qui montre que notre projet MVC standard contient une Area nommée Admin dans laquelle nous placerons tout ce qui à rapport avec la partie administrative de notre portail.




*A noter que les Areas seront toutes fusionnées au moment de la compilation de l'application. L'utilisation des Areas n'entrain aucune modification de votre code, que cela soit pour les liens ou pour les règles de routage. Il s'agit purement et simplement d'une*

*fonctionnalité qui aide le développeur à mieux s'y retrouver au sein de ses solutions Visual Studio.*

## 2 - Support des annotations

Les annotations, ou plus précisément les DataAnnotations, sont une nouveauté qui va permettre de déclaration explicitement les règles de validation des modèles, et d'avoir une validation automatique au niveau de la vue, via le ModelState.
Il existe quatre attributs de validation :
- [Required] : pour rendre une propriété obligatoire
- [StringLength] : pour définir la longueur maximale d'un champ
- [Range] : pour définir une plage de valeurs possibles pour un champ
- [RegularExpression] : pour valider le format d'une propriété

Au niveau utilisation, rien de bien compliqué. Vous placez les attributs au niveau de vos classes modèles, et vous laissez la génération automatique des vues fortement typées, faire le reste. Prenons par exemple, notre classe User :

Classe User

```
public class User
    {
        [Required(ErrorMessage = "Le nom est requis")]
        [StringLength(75, ErrorMessage = "Le nom ne peut pas faire plus de 75 caractÃ¨res")]
        public string Name
        {
            get; set;
        }

        [Required(ErrorMessage = "L'adresse est requise")]
        public string Address
        {
            get; set;
        }

        [Range(1, 95, ErrorMessage = "Le numÃ©ro de dÃ©partement doit Ãªtre compris entre 1 et 95")]
        public string DepartmentNumber
        {
            get; set;
        }

        [RegularExpression(@"^([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-
Z0-9\-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$",ErrorMessage = "Le format de l'email est incorrect")]
        public string Mail
        {
            get; set;
        }
    }
```

Puis utilisons notre classe au niveau de notre contrôleur :

Constrôleur UserController

```
public ActionResult Create()
        {
            User usr = new User();
            return View(usr);
        }

        [HttpPost]
        public ActionResult Create(User usr)
        {
            try
            {
                if (!ModelState.IsValid)
                    return View(usr);
```

Constrôleur UserController

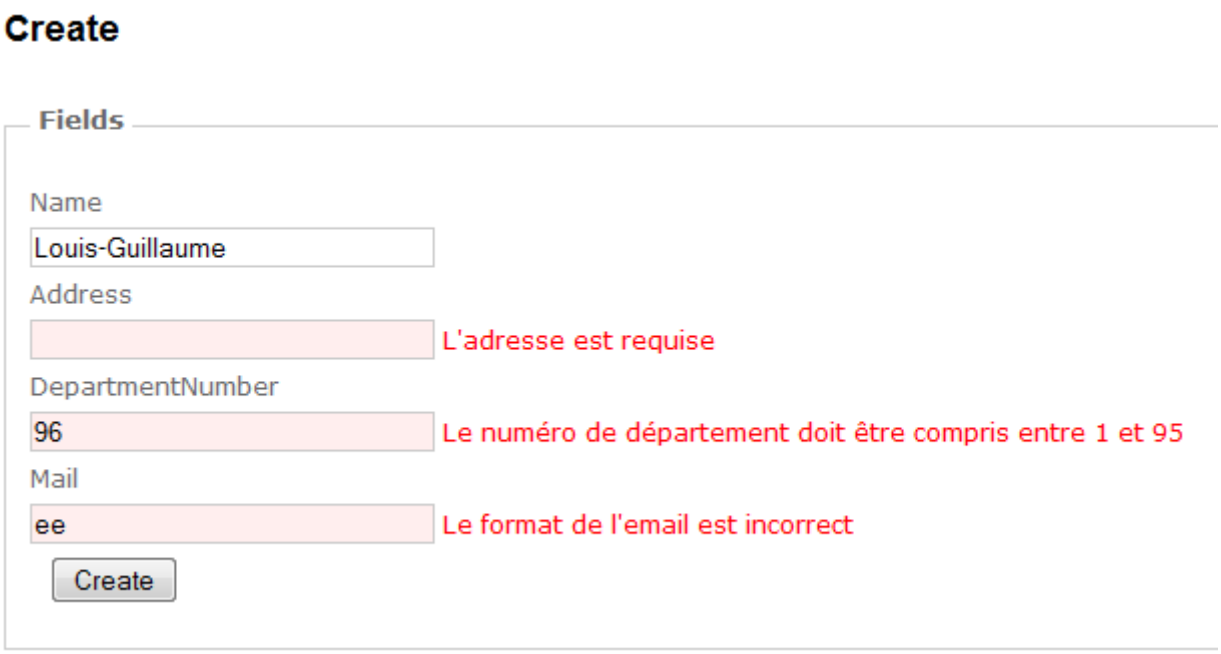
```
                // Insertion de l'utilisateur en base de donnÃ©es

                return RedirectToAction("Index");
            }
            catch
            {
                return View(usr);
            }
        }
```

Enfin, vous utilisez les validateurs au sein de la vue

Vue /User/Create

```
<% using (Html.BeginForm()) {%>

        <fieldset>
            <legend>Fields</legend>

            <div class="editor-label">
                <%= Html.LabelFor(model => model.Name) %>
            </div>
            <div class="editor-field">
                <%= Html.TextBoxFor(model => model.Name) %>
                <%= Html.ValidationMessageFor(model => model.Name) %>
            </div>

            <div class="editor-label">
                <%= Html.LabelFor(model => model.Address) %>
            </div>
            <div class="editor-field">
                <%= Html.TextBoxFor(model => model.Address) %>
                <%= Html.ValidationMessageFor(model => model.Address) %>
            </div>

            <div class="editor-label">
                <%= Html.LabelFor(model => model.DepartmentNumber) %>
            </div>
            <div class="editor-field">
                <%= Html.TextBoxFor(model => model.DepartmentNumber) %>
                <%= Html.ValidationMessageFor(model => model.DepartmentNumber) %>
            </div>

            <div class="editor-label">
                <%= Html.LabelFor(model => model.Mail) %>
            </div>
            <div class="editor-field">
                <%= Html.TextBoxFor(model => model.Mail) %>
                <%= Html.ValidationMessageFor(model => model.Mail) %>
            </div>

            <p>
                <input type="submit" value="Create" />
            </p>
        </fieldset>
```

Au moment de la validation, rien à faire. Le moteur de validation du ModelState se charge de valider pour vous et de retourner les informations à la vue, qui s'adapte pour afficher les messages d'erreur :




## 3 - Helpers fortement typés

### 3.1 - Link Helpers

### 3.2 - Input Helpers

## 4 - Modèles de mise en forme des Helper (templating helpers ?)

http://msdn.microsoft.com/en-us/library/ee308450%28VS.100%29.aspx

http://bradwilson.typepad.com/blog/2009/10/aspnet-mvc-2-templates-part-1-introduction.html

## 5 - Actions asynchrones

http://geekswithblogs.net/rajeshpillai/archive/2009/12/01/asynccontroller.aspx

## 6 - Ameliorations du cache

## 7 - Autres ameliorations

### 7.1 - Les valeurs par défaut

Avec ASP .Net MVC 1, il était possible de définir des paramètres optionnels, soit en rendant un paramètre d'une action Nullable, soit en modifiant la route dans global.asax pour inclure des valeurs par défaut.

Avec MVC 2, il est possible possible d'ajouter à vos méthodes, des attributs **DefaultValueAttribute** afin de s'assurer de la présence d'une valeur pour un paramètre donné. Ainsi, en ajoutant une valeur par défaut sur notre paramètre de filtrage, les URLs /ListUsers et /ListUsers/Administrateurs permettent d'obtenir le même résultat.

```
public ActionResult ListUsers([DefaultValue("administrateurs")]string groups)
```

```
{
}
```

*A noter que Visual Studio 2010 et le C# 4.0 vous permet de définir des valeurs par défaut, directement dans la signature de la méthode, ce qui permet de transformer le code précédent en:*

```
public ActionResult ListUsers(string groups = "administrateurs")
{
}
```

## 7.2 - L'attribute HttpPost

Ici, aucune grande nouveauté, simplement la possibilité de simplifier le code en transformant vos anciens [AcceptVerbs(HttpVerbs.Post)] en un simple [HttpPost]. Ainsi, les deux codes suivants, tous deux valables en MVC 2, font exactement la même chose :

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult CreateUser(User usr)
{
}
...
[HttpPost]
public ActionResult CreateUser(User usr)
{
}
```

### 7.2.1 - Le type MvcHtmlString

Dorénavant, les helpers retourneront un objet typé MvcHmtlString en lieu et place de l'objet String. Ce nouvel objet permet de tirer profit d'un format de chaine qui n'a pas besoin d'être réencodé pour l'affichage. Ce format qui apparait notamment au sein d'ASP .Net 4.0 a été intégré grâce à un petit peu bout de magie comme l'explique Phill Haack **sur son blog**.
Ainsi, plus besoin de faire sans cesse un Html.Encode(XXX) de chacun de vos helpers.

## 8 - Conclusion