ASP.Net MVC

Outils d'Asp.Net MVC

Structuration des vues, Helpers, Annotations,
Validation, Envois de fichiers,
Il n'y a pas que HTML dans la vie

Structuration des vues

- Layout : imposer une disposition
  - Sections rendues par le Layout, définies par la vue
  - Configuration des défauts dans Views_ViewStart.cshtml
    - Disposition par défaut dans Views\Shared_Layout.cshtml

```
@{  Layout = "~/Views/Shared/_Layout.cshtml"; }
```

    - Principaux membres utilisables dans `_ViewStart` et `_Layout`

Méthode	Info	Exemple
Ajax	Fournit des méthodes utilitaires	<script type="text/javascript"> alert(@Ajax.JavaScriptStringEncode(Model.Message)) </script>
Html	Fournit des méthodes utilitaires	@using (Html.BeginForm()) { . . . }
Page	Comme ViewBag, partage avec les vues et vues partielles	Page.Truc = unTruc;
PageData	Comme ViewData	PageData["Truc"] = unTruc;
IsPost, IsAjax		

⚠ Chaque objet à ses propres dynamics `ViewBag` et `Page`

⚠ Il faut partager des références et pas des types simples ou utiliser `ViewContext.Controler.ViewBag`

Structuration des vues

- Enchainement et transmission de données
  - _ViewStart

```
@{Layout = "~/Views/Shared/_Layout.cshtml";
Page.Titre = new ClasseTitre() { Titre = "Titre ViewStart" };}     ← Pas de ViewBag => Page ou ViewContext.ViewBag
```

  - Code global de la vue

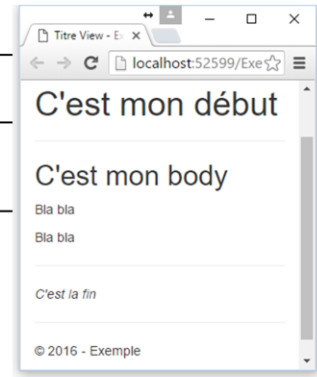
```
@{ Layout = "~/Views/Shared/_LayoutMaison.cshtml"; }
```

  - La disposition – qui exécute des `RenderSection`

```
<head>
  @{ Page.Titre.Titre = "Titre LayoutMaison"; } }          ← Une copie du Page de _ViewStart, pas grave...
  @RenderSection("Entete")
  <title>@Page.Titre.Titre - Exemple</title>
```

  - Les sections de la vue

```
@section Entete {
  @{ Page.Titre.Titre = "Titre View"; }
}
```



Structuration des vues

_ViewStart.cshtml

```
@{Layout = "~/Views/Shared/_Layout.cshtml";
Page.Titre = new ClasseTitre() { Titre = "Titre ViewStart" };}     ← Pas de ViewBag => Page ou ViewContext.ViewBag
```

_LayoutMaison.cshtml

```
<html><head> @{ Page.Titre.Titre = "Titre LayoutMaison"; }
  @RenderSection("Entete")
  <title>@Page.Titre.Titre - Exemple</title>
  @Styles.Render( "~/Content/css" ) @Scripts.Render( "~/bundles/modernizr" )                    ← Qu'est-ce donc ?
</head>
<body> <div class="container body-content">
    @RenderSection("Debut") <hr />
    @RenderBody() <hr />
    <footer> @RenderSection( "Fin" ) <hr /> <p>&copy; @DateTime.Now.Year - Exemple</p> </footer>
  </div>
  @Scripts.Render( "~/bundles/jquery" )  @Scripts.Render( "~/bundles/bootstrap" )               ← Qu'est-ce donc ?
  @RenderSection( "scripts", required: false )
</body>
</html>
```

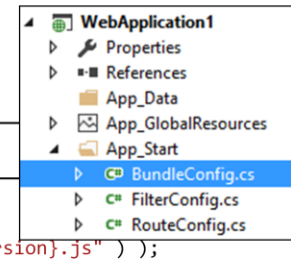
Index.cshtml

```
@{ Layout = "~/Views/Shared/_LayoutMaison.cshtml"; }
@section Entete {
  @{ Page.Titre.Titre = "Titre View"; }
}
@section Debut {
  <h1>C'est mon début</h1>
}
<h2>C'est mon "body"</h2> <p>Bla bla</p> <p>Bla bla</p>
@section Fin { <em>C'est la fin</em> }
```

Bundles

- Global.asax.cs

```
public class MvcApplication : System.Web.HttpApplication {
  protected void Application_Start() {
    BundleConfig.RegisterBundles( BundleTable.Bundles ); [ . . . ]
  } [ . . . ]
}
```



- BundleConfig.cs

```
public class BundleConfig {
  public static void RegisterBundles( BundleCollection bundles ) {
    bundles.Add( new ScriptBundle( "~/bundles/jquery" ).Include( "~/Scripts/jquery-{version}.js" ) );
    bundles.Add( new ScriptBundle( "~/bundles/jqueryval" ).Include( "~/Scripts/jquery.validate*" ) );
    bundles.Add( new ScriptBundle( "~/bundles/modernizr" ).Include( "~/Scripts/modernizr-*" ) );
    bundles.Add( new ScriptBundle( "~/bundles/bootstrap" ).Include( "~/Scripts/bootstrap.js" ) );
    bundles.Add( new StyleBundle( "~/Content/css" ).Include( "~/Content/bootstrap.css", "~/Content/site.css" ) );
  }
}
```

- Utilisation dans les Vues / Layouts

```
@Styles.Render("~/Content/css")                 ← Dans <head>
@Scripts.Render("~/bundles/modernizr")

@Scripts.Render("~/bundles/jquery")             ← En bas de page
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
```

Vues partielles

- Le contrôleur peut être utilisé
  - `return PartialView()`
  - La méthode peut être décorée `[ChildActionOnly]`
    - Sans cet attribut la vue est invocable directement (Ajax, par exemple)
- La vue partielle peut être rendue directement par la vue parent

Html. ...	**Info**
Partial(string partialViewName); Partial(string partialViewName, object model); Partial(string partialViewName, ViewDataDictionary viewData); Partial(string partialViewName, object model, ViewDataDictionary viewData);	Le rendu de la vue est retourné dans une MvcHtmlString, càd. Une chaîne qui est sensée être déjà encodée. Il est vaguement possible qu'on ait ponctuellement besoin d'en ré-encoder une (ToHtmlString) mais généralement on la ToString() simplement.
RenderPartial(. . .)	Plus performant, mais plus suant à écrire, RenderPartial écrit directement le rendu dans le flux http.
Action(string actionName, Object routeValues) Action(string actionName, string controllerName) Action(string actionName, string controllerName, Object routeValues)	Exécute une action et retourne sa vue. Le rendu de la vue est retourné dans une MvcHtmlString que le moteur ne ré-encode pas sans votre demande explicite par .ToHtmlString().
RenderAction(. . .)	Plus performant, mais plus suant à écrire, RenderAction écrit directement le rendu dans le flux http.

Il s'agit bien sûr d'invocations directes de méthodes de la vue parent vers le contrôleur ou le rendu de vue partielle, aucune nouvelle requête http n'est envoyée au serveur.

Vues partielles

- Modèle

```
public class NomFormatable {
  public string Nom { get; set; }
  public string Format { get; set; }
}
```

- Contrôleur

```
public class ExempleVuesPartiellesController : Controller {
  public ActionResult Index() {                    ← Il me faut tout de même une vue classique pour l'exemple !
    return View();
  }
  [ChildActionOnly]
  public ActionResult LireNom( string format ) {              ← Sera null si l'action est invoquée sans paramètre
    var nom = User.Identity.IsAuthenticated ? User.Identity.Name : "Inconnu";
    return PartialView( new NomFormatable { Nom = nom, Format = format != null ? format : "{0}" } );
  }
  public ActionResult LireNomPourAjax( string format ) {          ← Si ChildActionOnly, pas d'appel ajax possible
    var nom = User.Identity.IsAuthenticated ? User.Identity.Name : "Inconnu";
    return PartialView( "LireNom", new NomFormatable { Nom = nom, Format = format != null ? format : "{0}" } );
  }
}
```

Vues partielles

- Vue partielle : LireNom.cshtml

```
@using WebApplication1.Models
@model NomFormatable
@{
  NomFormatable nf;
  nf = Model != null ? Model : nf = new NomFormatable() {
    Nom = "Rien dans Model, car on n'a pas exécuté d'action et on n'a pas reçu de modèle",
    Format = "{0}"
  };
}
@String.Format( nf.Format, nf.Nom )
```

Vues partielles

- Index.cshtml : Utilisation de Html.Partial / RenderPartial

```
@using WebApplication1.Models;
@{ ViewBag.Title = "Exemple de vue partielle"; }

<h3>Via Html.RenderPartial / Html.Partial</h3>
<h4>Écriture dans le flux HTTP</h4>
@{Html.RenderPartial( "LireNom" );}                    ← Syntaxe plus lourde car écrit directement dans le flux

<h4>Le rendu est dans une MvcHtmlString</h4>
@Html.Partial( "LireNom" ).ToHtmlString().ToUpper()    ← Syntaxe plus légère mais moins performante
```

Vues partielles

- Index.cshtml : Utilisation de Html.Action / RenderAction
  - On passe par le contrôleur

```
<h3>Exemple de vue partielle via Html.RenderAction / Action</h3>

<h4>Écriture dans le flux HTTP</h4>
@{Html.RenderAction( "LireNom" );}

<h4>Le rendu est dans une MvcHtmlString</h4>
@Html.Action( "LireNom" ).ToHtmlString().ToUpper()
```

Vues partielles

- Index.cshtml : passage de paramètres à Partial
  - ⚠Un NomFormatable est nécessaire (pas de ModelBinder)

```
<h3>Passer un paramètre à Html.RenderPartial / Partial</h3>
<h4>Écriture dans le flux HTTP</h4>
@{Html.RenderPartial(
  "LireNom",
  new NomFormatable { Nom = "Inconnu", Format = "Le nom {0} est donné par la vue" } );}

<h4>Le rendu est dans une MvcHtmlString</h4>

@Html.Partial(
  "LireNom",
  new NomFormatable { Nom = "Inconnu", Format = "Le nom {0} est donné par la vue" }
  ).ToHtmlString().ToUpper()
```

Vues partielles

- Index.cshtml : passage de paramètres à Action
  - ☺Ça y ressemble… en plus simple

```
<h3>Passer un paramètre à Html.RenderAction / Action</h3>
<h4>Écriture dans le flux HTTP</h4>
@{Html.RenderAction( "LireNom", new { format = "Le nom {0} est donné par le ctrlr" } );}← Paramètre de la méthode

<h4>Le rendu est dans une MvcHtmlString</h4>
@Html.Action( "LireNom", new { format = "Le nom {0} est donné par le ctrlr" } ).ToHtmlString().ToUpper()
```

Passer un paramètre à Html.RenderAction / Action

Écriture dans le flux HTTP

Le nom Inconnu est donné par le ctrlr

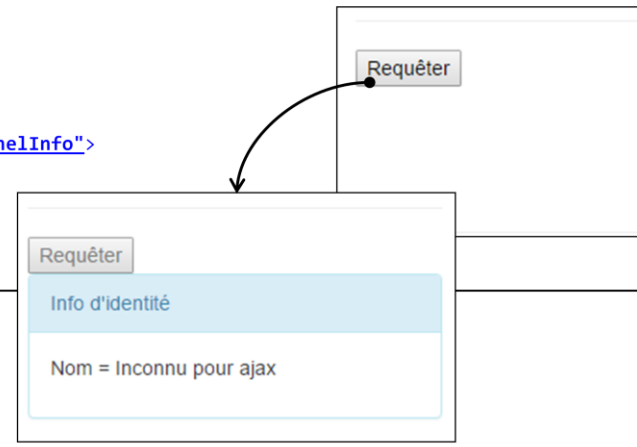
Le rendu est dans une MvcHtmlString

LE NOM INCONNU EST DONNé PAR LE CTRLR

Vues partielles

- Ajax
  - ☺Facile

```
<input type="button" value="Requêter" onclick="afficher(this)" />
<script type="text/javascript">
  function afficher(bt) {
    bt.disabled = true;
    $('#info').load(
      '/ExempleVuesPartielles/LireNomPourAjax?format=@Url.Encode("Nom = {0} pour ajax")',
      function ( texte ) {
        $("#panelInfo")
          .animate({ opacity: 1 }, "slow");
      })
  }
</script>
<div class="panel panel-info" style="opacity: 0" id="panelInfo">
  <div class="panel-heading">Info d'identité</div>
  <div class="panel-body">
    <p id="info"></p>
  </div>
</div>
```



Même vue pour plusieurs contrôleurs ?

- Pas commun… mais possible !
  - Permet d'éviter des RedirectToAction coûteux
  - Le chemin complet de la vue doit être utilisé au niveau de l'action

```
return View( "~/Views/UnDesDeuxContrôleurs/VuePartagée.cshtml", modèle );
```

  - Mais les vues partielles utilisées par la vue partagée doivent être complètement explicitées :

```
@Html.Partial( "~/Controllers/UnDesDeuxContrôleurs/UneActionQuiRetourneUneVuePartielle" )
```

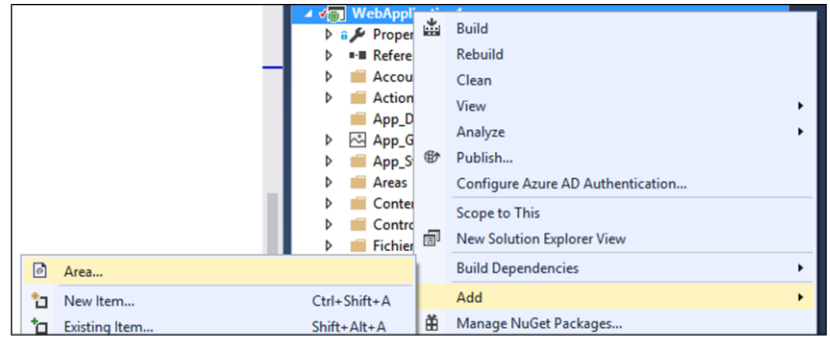
  - La vue peut également être placée dans ~/Views/Shared

Zones (Areas)

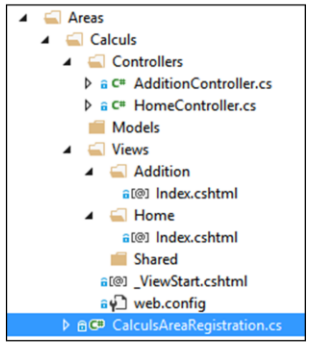
- Organiser un gros projet en petites sections
  - Dès Asp.Net MVC 2
  - Regroupe des contrôleurs et des vues traitant un même sujet
    - Backoffice / Frontoffice
    - Utilisateurs / Administrateurs / Managers
    - Gestion des clients / Gestion des produits / Comptabilité
    - …

Zones (Areas)

- Création



- Fichiers



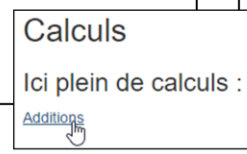
- Web.config, _ViewStart.cshtml et _Layout.cshtml
  - Possibilités de configuration spécifique
- Espace de nom `WebAplication1.Areas.Calculs`

Zones (Areas)

- Code : classique…
  - Home

```
namespace WebApplication1.Areas.Calculs.Controllers {
  public class HomeController : Controller {
    public ActionResult Index() {
      return View();
    }
  }
}
```

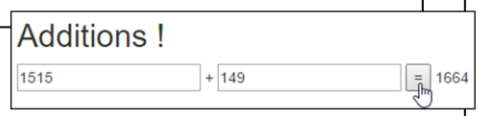
```
@{ ViewBag.Title = "Calculs"; }
<h2>@ViewBag.Title</h2>
<h3>Ici plein de calculs :</h3>
@Html.ActionLink( "Additions", "Index", "Addition" )
```



  - Addition

```
public class AdditionController : Controller {
  public ActionResult Index( int ?n1, int? n2 ) {
    return View( (n1 ?? 0) + (n2 ?? 0) );
  }
}
```

```
@model int
@{ ViewBag.Title = "Additions"; }
<h2>@ViewBag.Title !</h2>
@using( Html.BeginForm() ) {
  @Html.TextBox( "n1", 0, new { type = "number" } )
  <span>+</span>
  @Html.TextBox( "n2", 0, new { type = "number" } )
  <input type="submit" value="=" />
  <span>@Model</span>
}
```



  - Redirection interzones

```
return RedirectToAction( "LogIn", "Account", new { area = "Admin" } );
```

Zones (Areas)

- ∃ des contrôleurs de même nom ?
  - Configuration de la route

http://localhost:52599/Home

Multiple types were found that match the controller named 'Home'. This can happen if the route that services this request ('{controller}/{action}/{id}') does not specify namespaces to search for a controller that matches the request. If this is the case, register this route by calling an overload of the 'MapRoute' method that takes a 'namespaces' parameter.

The request for 'Home' has found the following matching controllers:
WebApplication1.Controllers.HomeController
WebApplication1.Areas.Calculs.Controllers.HomeController

- ~/areas/Calculs/CalculsAreaRegistration.cs

```
public override void RegisterArea( AreaRegistrationContext context ) {
  context.MapRoute(
      "Calculs_default",
      "Calculs/{controller}/{action}/{id}",
      new { action = "Index", id = UrlParameter.Optional },
      namespaces: new[] {                          ← À ajouter si ∃ des ctrlrs de même nom dans d'autres zones
        "WebApplication1.Areas.Calculs.Controllers"
      }
  );
}
```

- Idem dans ~/App_Start/RouteConfig.cs

```
 [ . . . ]      namespaces: new[] { "WebApplication1.Controllers" }
```

Liaison requêtes ⇔ actions personnalisée

- Exemple : Plusieurs actions par formulaire
  - Ça serait bien pratique
  - Évite de créer plusieurs formulaires, ce qui n'est d'ailleurs pas toujours possible
- Intervenir lors de la liaison Requête/Action
  - ☺ Facile : faire comme `ActionNameAttribute`…

```
public class EstCeLActionDemandeeAttribute : ActionNameSelectorAttribute {
  public override bool IsValidName( ControllerContext controllerContext, string actionName,
                                    MethodInfo methodInfo ) {
    if( actionName.Equals( methodInfo.Name, StringComparison.InvariantCultureIgnoreCase ) )
      return true;                                                    ← Simple : action == méthode
    var request = controllerContext.RequestContext.HttpContext.Request;
    return request[methodInfo.Name] != null;          ← ∃ entête ou QueryString portant le nom de la méthode
  }
}
```

  - ☺ Pas d'URL en dur dans du JavaScript
  - ☹ Contrat implicite entre la vue et le contrôleur

Liaison requêtes ⇔ actions personnalisée

- Vue

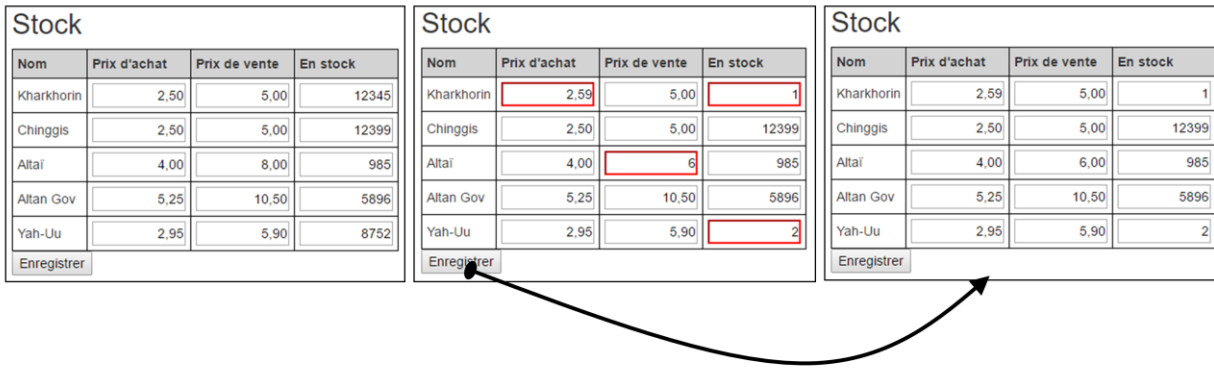
```
@{ ViewBag.Title = "Exemple plusieurs boutons d'action"; }
<h2>@ViewBag.Title</h2>
@using( Html.BeginForm() ) {
  @Html.Label("nom", "Nom :") @Html.TextBox( "nom", string.Empty, new { placeholder = "votre nom..." } )
  <br />
  <input type="submit" name="DireBonjour" value="Dire bonjour" />
  <input type="submit" name="DireAuRevoir" value="Dire au revoir" />
}
<hr />
@ViewBag.Message
```

- Contrôleur

```
public class ExemplePlusieursBoutonsController : Controller {
  [HttpGet] public ActionResult Index() {
    return View();
  }
  [EstCeLActionDemandee()] public ActionResult DireBonjour( string nom ) {
    ViewBag.Message = "Bonjour " + nom;
    return View( "Index" );
  }
  [EstCeLActionDemandee()] public ActionResult DireAuRevoir( string nom ) {
    ViewBag.Message = "Au revoir " + nom;
    return View( "Index" );
  }
}
```

Exercice

- Éditer le stock
  - Représenter chaque article en répétant une vue partielle
  - Les prix et quantités sont éditables
  - Toutes les lignes sont éditables simultanément

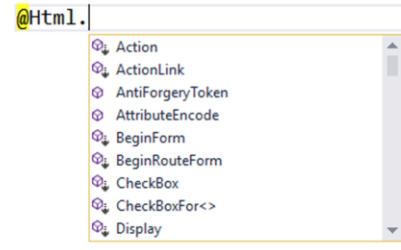


Helpers

- "Close to the metal"
  - Utiliser un anonyme pour préciser les attributs HTML

```
@using( Html.BeginForm("Action", "Ctrlr", FormMethod.Post, new { data_validatable="true", @class="beau" } ) ) { …
```

  - …Sans se noyer dans les détails
  - `Html.BeginForm`
    - Utilise `IDisposable`
    - Respecte les routes
- Extensions de `HtmlHelper<TModel>`
  - `using System.Web.Mvc.Hml` dans Views/Web.Config
  - Ajoutez vos helpers !



Helpers

- Html.TextBox / TextArea

```
@Html.TextBox("nomTB", "l'<b>contenu</b>", "formate \"{0}\"", new { maxlength = 100, @class = "std" } )
@Html.TextArea( "nomTA", "l'<b>contenu</b>", 3, 20, new Dictionary<string, object>() { {"class", "std"} } )
```

  - Rendu bien encodé

```
<input class="std" id="nomTB" maxlength="100" name="nomTB" type="text"
       value="formate &quot;l&#39;&lt;b>contenu&lt;/b>&quot;" />
<textarea class="std" cols="20" id="nomTA" name="nomTA"
  rows="3">l&#39;&lt;b&gt;contenu&lt;/b&gt;</textarea>
```

  - Binding bidirectionnel sur le ViewBag

```
@Html.TextBox( "Message.length" )                    ← Lit le ViewBag
@Html.TextBox( "Message" )                           ← Lit le ViewBag + récupérable dans un paramètre de l'action
```

  - Liaison au modèle simplifiée

```
@Html.TextBoxFor( monModele => monModele.MaPropriété )
```

    - Utilisera

```
ModelMetadata ModelMetadata.FromLambdaExpression<TParameter, TValue>(
  System.Linq.Expressions.Expression<Func<TParameter, TValue>> expression,
  ViewDataDictionary<TParameter> viewData
)
```

Helpers

- `Html.Checkbox`
  - Hack pour rouler ce $%*&#! de comportement par défaut

```
@Html.CheckBox( "gentil", false, new { @class = "std" } )
```

```
<input class="std" id="gentil" name="gentil" type="checkbox" value="true">
<input name="gentil" type="hidden" value="false">  ← L'entête 'gentil' sera présent, même si la case est décochée
```

Helpers

- Html.RadioButton

```
public enum Niveau { Secret, TresSecret, TresTresSecret }
public class AgentSecret {
  public Niveau Niveau { get; set; }
  [ . . . ]
}
```

```
<label>@Html.RadioButtonFor( m => m.Niveau, Niveau.Secret ) Secret</label><br />
<label>@Html.RadioButtonFor( m => m.Niveau, Niveau.TresSecret ) Très secret</label><br />
<label>@Html.RadioButtonFor( m => m.Niveau, Niveau.TresTresSecret ) Très trés secret</label><br />
```

  - Alternative

```
@foreach( Niveau n in Enum.GetValues( typeof( Niveau ) ) ) {
  <label>@Html.RadioButtonFor( m => m.Niveau, n ) @n.ToString()</label><br />
}
```

- Pas de LabelFor car plusieurs labels par propriété
- Liaison automatique au modèle

Helpers

- `Html.DropDownList` et ses variantes

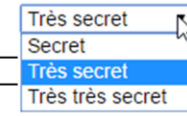
```
public static MvcHtmlString SelectExtensions.DropDownList(
  this HtmlHelper htmlHelper,
  string name,
  IEnumerable<SelectListItem> selectList,          ← SelectListItem = Disabled + Group + Selected + Text + Value
  string optionLabel,                                ← Le texte d'une éventuelle option de valeur vide
  object htmlAttributes )

public static MvcHtmlString DropDownListFor<TModel, TProperty>(
  this HtmlHelper<TModel> htmlHelper,
  Expression<Func<TModel, TProperty>> expression,
  IEnumerable<SelectListItem> selectList,
  string optionLabel,      ← Option supplémentaire Utile si le modèle n'est pas pré-renseigné (Ex. mode création)
  object htmlAttributes )

public static MvcHtmlString EnumDropDownListFor<TModel, TEnum>(
  this HtmlHelper<TModel> htmlHelper,
  Expression<Func<TModel, TEnum>> expression,
  string optionLabel,
  object htmlAttributes )
```

Helpers

- `Html.DropDownList` sur valeurs d'énumérés

```
public enum Niveau {
  Secret,
  [Display(Name = "Très secret")] TresSecret,
  [Display(Name = "Très très secret")] TresTresSecret
}
```

```
@Html.EnumDropDownListFor( m => m.Niveau, null, new {  @class = "std" } )
```



Helpers

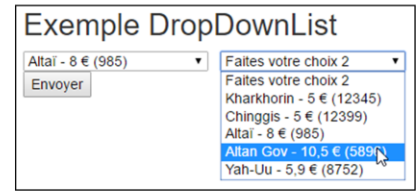
- `DropDownList` sur valeurs de modèle - Contrôleur

```
public class ExempleDropDownListController : Controller {
  public ActionResult Index() {
    lock ( Article.Stock ) {

      ViewBag.OptionsDuSelect = Article.Stock.Values.Select( a =>
        new SelectListItem() {
          Text = $"{a.Nom} - {a.PrixVente} € ({a.Quantite})",                    ← Est-ce du contrôle ?
          Value = a.Id.ToString()
        } ).ToList();

      ViewBag.Articles = Article.Stock.Values.Select( a => new ArticleCommerce( a ) ).ToList();      ← Alternative

    }
    return View();
  }

  [HttpPost]
  public string Index( Guid? idArticleParCtrlr, Guid? idArticleParVue ) {      ← Annulables car ∃ une option vide
    Article articleParCtrlr;
    Article.Stock.TryGetValue( idArticleParCtrlr.GetValueOrDefault(), out articleParCtrlr );
    Article articleParVue;
    Article.Stock.TryGetValue( idArticleParVue.GetValueOrDefault(), out articleParVue );
    return $"1er article : {articleParCtrlr?.Nom}<br />2nd article : {articleParVue?.Nom}";       ← Le fainéant !
  }
}
```

Helpers

- DropDownList sur valeurs de modèle - Vue

```
@using WebApplication1.Models
@using WebApplication1.Repositories.ArticleCommerceRepository;
@{ ViewBag.Title = "Exemple DropDownList"; }
<h2>@ViewBag.Title</h2>

@using( Html.BeginForm() ) {

  @Html.DropDownList( "IdArticleParCtrlr", (List<SelectListItem>)ViewBag.OptionsDuSelect, "Faites votre choix 1",
                      new { @class = "std" } );
  <span> </span>
  @Html.DropDownList( "IdArticleParVue", OptionsBienPrésentées, "Faites votre choix 2", new { @class = "std" } )

  @functions {
    private List<SelectListItem> OptionsBienPrésentées {
      get {
        IEnumerable<ArticleCommerce> articles = ViewBag.Articles;
        return articles.Select( a =>
          new SelectListItem() {
            Text = $"{a.Nom} - {a.PrixVente} € ({a.Quantite})",
            Value = a.Id.ToString()
          } ).ToList();
      }
    }
  }
<br /><input type="submit" value="Envoyer" />
}
```



1er article : Altaï
2nd article : Altan Gov

1er article :
2nd article : Altan Gov

Data annotations

- `System.ComponentModel.DataAnnotations`
  - Validation client par helper + jquery.validate.js[1]
  - Validation serveur par model binding

DRY !

```
public class Article {
  [System.ComponentModel.DataAnnotations.Key]
  public Guid Id { get; set; }

  [Required( ErrorMessage = "Champ nom requis !" )]                    ← On peut également utiliser une ressource
  [StringLength( 50, MinimumLength = 5, ErrorMessage = "50 caractères maxi, 5 mini !" )]
  public string Nom { get; set; }

  [Required( ErrorMessage = "Champ '{0}' requis !" )]
  [Display( Name = "Date de péremption" )]
  [DisplayFormat( DataFormatString = "{0:d}", ApplyFormatInEditMode = true )]
  public DateTime Ddp { get; set; }

  [Range( 0.0, double.MaxValue, ErrorMessage = "Le champ {0} doit être positif ou nul" )]
  [Required( ErrorMessage = "Champ {0} requis !" )]
  public decimal Prix { get; set; }
}
```

  - + `RegularExpression`, `Compare` (pour mots de passe), `DataType` (MultilineText, DateTime, Password, …), UIHint, …

[1] /!\ Un bundle nommé jqueryval est bien enregistré mais il n'est pas inclus par _Layout.cshtml

Data annotations

- `System.Web.Mvc.RemoteAttribute`
  - Validation client avec callback Ajax
  - Modèle + annotations

```
public class AgentSecret {

  [Required( ErrorMessage = "T'es timide ? Donne ton nom !" )]
  public string Nom { get; set; }


  [Remote( "ValiderCodeSecret", "ExempleValidationAvecRemote", AdditionalFields = "Nom",
                                                               ErrorMessage = "Code erroné" )]
  [Display( Name = "Code secret" )]
  [DataType( DataType.Password )]
  public string CodeSecret { get; set; }
}
```

Data annotations

- Prises en compte par des helpers
  - Variantes des suffixes "For"

```
public static MvcHtmlString DropDownListFor<TModel, TProperty>( this HtmlHelper<TModel> htmlHelper,
  Expression<Func<TModel, TProperty>> expression,                          ← leModele => leModele.UnePropriété
  IEnumerable<SelectListItem> selectList,
  string optionLabel,                                                        ← "Choisissez une option valide"
  object htmlAttributes                                                ← new { disabled = "disabled", @class = "std" }
)

public static MvcHtmlString LabelFor<TModel, TValue>( this HtmlHelper<TModel> html,
  Expression<Func<TModel, TValue>> expression,
  string labelText,                      ← Optionnel, pour contredire le nom de la propriété ou les annotations
  object htmlAttributes                                                ← new { disabled = "disabled", @class = "std" }
)

public static MvcHtmlString EditorFor<TModel, TValue>( this HtmlHelper<TModel> html,
  Expression<Func<TModel, TValue>> expression,
  string templateName,                                              ← Pour template personnalisé (voir + loin)
  string htmlFieldName,                                                 ← Si plusieurs propriétés de même nom
  Object additionalViewData                                             ← Pour paramétrer le template perso
)                                              ← ou souvent : new { htmlAttributes = new { @class = "std" } })

public static MvcHtmlString DisplayNameFor<TModel, TValue>( ← Juste la chaine Html-encodée pour plus de souplesse
  this HtmlHelper<IEnumerable<TModel>> html,
  Expression<Func<TModel, TValue>> expression
)
```

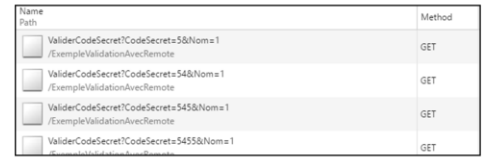
Data annotations

- Contrôleur

```
public class ExempleValidationAvecRemoteController : Controller {

  public ActionResult Index() {
    ViewBag.Message = "Créez votre premier agent secret";
    return View( new AgentSecret() );
  }

  [ActionName( "Index" )]
  [HttpPost]
  public ActionResult Creer( AgentSecret nvAgent ) {                    ← Model binding
    ViewBag.Message = "OK. Agent créé. Créez autre agent.";             ← Et si le navigateur interdisait JS ? ? ?
    return View( new AgentSecret() );
  }

  public JsonResult ValiderCodeSecret( string codeSecret, string nom ) {
    if( !Regex.IsMatch( codeSecret, @"^\d+$" ) )
      return Json( false, JsonRequestBehavior.AllowGet );
    for( int i = 0; i < codeSecret.Length / 2; i++ ) {
      if( codeSecret[i] != codeSecret[codeSecret.Length - i - 1] )
        return Json( false, JsonRequestBehavior.AllowGet );
    }
    return Json( true, JsonRequestBehavior.AllowGet );
  }
}
```

Data annotations

- Vue




```
@using WebApplication1.Models;
@model AgentSecret
@{ ViewBag.Title = "L'attribut de validation Remote"; }

<h2>L'attribut de validation Remote</h2>

@using( Html.BeginForm() ) {
  <fieldset>
    <legend>@ViewBag.Message</legend>
    @Html.LabelFor( agent => agent.Nom )<br />
    @Html.EditorFor( agent => agent.Nom ) @Html.ValidationMessageFor( agent => agent.Nom ) <br />

    @Html.LabelFor( agent => agent.CodeSecret )<br />
    @Html.EditorFor( agent => agent.CodeSecret ) @Html.ValidationMessageFor( agent => agent.CodeSecret )<br />
    <br />
    <input type="submit" value="Créer" />
  </fieldset>
}
@section Scripts {
  @Scripts.Render( "~/bundles/jqueryval" )          ← /!\ Pas présent par défaut
}
```

Model binding et validation serveur

- `ModelState` = dictionnaire regroupant les erreurs de model binding
  - `UpdateModel` / `TryUpdateModel` : Model binding manuel

Controler.ModelState	Info
ModelState.Keys	Liste des noms de propriétés liées
ModelState.Values	Liste de ModelState, un par propriété liée
ModelState.Values.First() ["Propriété"]	Objet ModelState pour la 1ère clé
ModelState["Propriété"].Errors[0].ErrorMessage	Message de la 1ère erreur sur la propriété "Propriété"
ModelState.IsValid	Teste s'il existe au moins une erreur
ModelState["Propriété"].Value.AttemptedValue	Valeur que le model binding a essayé d'affecter à Propriété
ModelState.AddModelError("Propriété", "Msg d'erreur")	Ajoute une entrée au dictionnaire Controller.ModelState

- Affichage des erreurs
  - `Html.ValidationMessageFor` : 1ère erreur de la propriété
  - `Html.ValidationSummary` : toutes les erreurs du modèle
  - Parcours manuel de `ViewData.ModelState` ;)

UpdateModel renseigne Controller.ModelState et soulève une InvalidOperationException.

TryUpdateModel renseigne Controller.ModelState mais ne soulève pas d'exception.

Un ValidationFor permet de fournir les erreurs en javascript, mais sans les afficher (contrairement au ValidationMessageFor.) Un ValidationSummary peut alors centraliser leur affichage.

Model binding et validation serveur

- `CustomValidationAttribute`
  - Validation serveur personnalisée par data annotation
  - Référencer la méthode de classe à utiliser
  - Ajouts à la classe AgentSecret :

```
public class AgentSecret {
  // [ . . . ]
  [CustomValidation( typeof( MissionValidator ), "ValiderMission", ErrorMessage = "Mission non approuvée" )]
  public string Mission { get; set; }
}
```

  - Classe de validation

```
public static class MissionValidator {
  public static ValidationResult ValiderMission( string mission ) {
    if( !string.IsNullOrWhiteSpace(mission) && mission.ToUpper() == mission )
      return ValidationResult.Success;       ← Ou null, ça marche aussi...
    else
      return new ValidationResult( null );  ← Pour utiliser le message d'erreur par défaut
  }
}
```

Model binding et validation serveur

- Contrôleur

```
[ActionName( "Index" ), HttpPost]
public ActionResult Creer( AgentSecret nvAgent ) {     ← ModelBinding automatique
  if( !ModelState.IsValid ) {                          ← Rempli this.ModelState
    ViewBag.Message = "Revoyez votre copie.";
    ModelState.AddModelError( "Mission", "J'ai dû faire une validation côté serveur" );  ← /!\ 2nde erreur mission
    return View( nvAgent ); ← Utilisation auto du ModelState mais Sans nvAgent, on perdrait le CodeSecret
  }                         ←   car (type="password")
  ViewBag.Message = "OK. Agent créé. Créez autre agent.";
  ModelState.Clear();                                  ← Pour ne pas garder les valeurs saisies
  return View();
}
```

- Alternative

```
[ActionName( "Index" ), HttpPost]
public ActionResult Creer2() {                         ← Pas de ModelBinding automatique
  var nvAgent = new AgentSecret();
  if( !TryUpdateModel<AgentSecret>( nvAgent ) ) {      ← ModelBinding explicitement demandé
    ViewBag.Message = "Revoyez votre copie.";
    ModelState.AddModelError( "Mission", "J'ai dû faire une validation côté serveur" );    ← "" si erreur globale
    return View( nvAgent );
  }
  ViewBag.Message = "OK. Agent créé. Créez autre agent.";
  ModelState.Clear();
  return View();
}
```


AddModelError permet d'ajouter une erreur globale (de niveau "modèle" et non au niveau d'une propriété particulière) en fournissant une chaîne vide en guise de nom de propriété.

La méthode helper ValidationSummary offre le paramètre `excludePropertyErrors` pour n'afficher que les erreurs de niveau modèle.

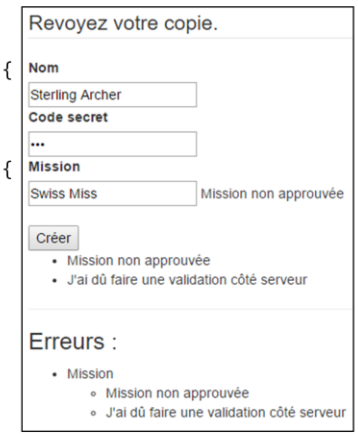
Bien pratique lorsque les erreurs de niveau propriété sont déjà affichées à côté des zones de saisie concernées.

Dans la vue, les EditorFor, TextBoxFor, etc. travailleront sur le modèle envoyé à la vue, ou, à défaut, sur les données qui ont été placées dans le ModelState lors de l'invocation de l'action.

Model binding et validation serveur

- Vue

```
@using( Html.BeginForm() ) {
  <fieldset> <legend>@ViewBag.Message</legend>
    [ . . . ]
    @Html.LabelFor( agent => agent.Mission )<br />
    @Html.EditorFor( agent => agent.Mission ) @Html.ValidationMessageFor( agent => agent.Mission )<br /><br />
    <input type="submit" value="Créer" />
    @Html.ValidationSummary() <hr />     ← ∃ param excludePropertyErrors si on veut seulement les erreurs globales
    @if( !ViewData.ModelState.IsValid ) {
      <h3>Erreurs :</h3>
      <ul>
        @{ var modelState = ViewData.ModelState; }
        @foreach( var propEnErreur in modelState.Keys
                                .Where( prop => modelState[prop].Errors.Count > 0 ) ) {
          <li> @propEnErreur
            <ul>
              @foreach( var erreur in modelState[propEnErreur].Errors
                                              .Select( err => err.ErrorMessage ) ) {
                <li>@erreur</li>
              }
            </ul>
          </li>
        }
      </ul>
    }
  </fieldset>
}
@section Scripts { @Scripts.Render( "~/bundles/jqueryval" ) }
```



Source des données du model binding

- TextBoxFor, EditorFor, … prennent les données de préférence dans :
  1. Le ModelState
     - C'est pour cette raison qu'on doit souvent faire

```
ModelState.Clear();
```

     - Ou

```
ModelState.SetModelValue(nameof(Classe.Propriété), new ValueProviderResult("", "", CultureInfo.InvariantCulture));
```

  2. Si non, dans le ViewBag
     - On peut également s'en servir pour préinitialiser un contrôle

```
ViewBag.Propriété = "Valeur";
```

  3. Si non, dans le Model passé à la vue

```
return View( new Classe() { Propriété = "Valeur" } );
```

Helper maison

- Afficher tous les messages d'erreur d'une propriété

```
namespace WebApplication1.Helpers {
  public static class HelpersExtension {
    public static MvcHtmlString AllValidationMessagesFor<TModel, TProperty>( this HtmlHelper<TModel> hh,
                                  Expression<Func<TModel, TProperty>> expression, object attributs = null ) {
      var prop = ModelMetadata.FromLambdaExpression( expression, hh.ViewData ).PropertyName;
      var ms = hh.ViewData.ModelState;
      var dico = attributs as IDictionary<string, object>;                    ← Dictionnary ou anonyme ? ? ?
      if( ms.ContainsKey( prop ) && ms[prop].Errors.Count > 1 ) {            ← Plus d'une erreur => concaténer
        var msgs = new StringBuilder();
        foreach( ModelError error in ms[prop].Errors )
          msgs.AppendLine( error.ErrorMessage );
        if( dico != null )
          return hh.ValidationMessageFor( expression,                        ← Réutiliser le helper classique
            msgs.ToString(),                                    ← Mais en lui faisant afficher la concaténation
            dico );              ← Passer le dico dont les clés/valeurs seront utilisées pour créer des attributs
        else
          return hh.ValidationMessageFor( expression,                        ← Réutiliser le helper classique
            msgs.ToString(),                                    ← Mais en lui faisant afficher la concaténation
            attributs );    ← Passer l'objet dont les propriétés/valeurs seront utilisées pour créer des attributs
      }
      return dico != null ? hh.ValidationMessageFor( expression, null, dico ) ← - de 2 => le helper classique
                          : hh.ValidationMessageFor( expression, null, attributs );
    }
  }
}
```


Les helpers sont régulièrement surchargés pour permettre de fournir les attributs HTML sous forme d'anonyme ou de dictionnaire.

Elles peuvent également supporter que l'objet passé soit un dictionnaire et non un anonyme (un dictionnaire, un anonyme… ce sont des "Object"s.)

La méthode utilitaire `HtmlHelper.ObjectToDictionary( object )` (qui fait ce qu'elle dit…) peut vous faciliter la vie pour mettre en œuvre vos propres helpers.

HtmlHelper offre d'autres membres de classe utiles.

Helper maison

- Utilisation dans la vue

```
Toutes les erreurs de Mission :
<div style="white-space: pre">@Html.AllValidationMessagesFor( m => m.Mission, new { @class = "btn-danger" })</div>
```

Toutes les erreurs de Mission :
Mission non approuvée
J'ai dû faire une validation côté serveur

Custom templates

- Vue pour édition spéciale
  - .cshtml dans View ou Shared/DisplayTemplates ou /EditorTemplates
  - Choisi en décorant la propriété du modèle `[UIHint]`
  - Ou avec `Html.EditorFor / DisplayFor`

```
public static MvcHtmlString EditorExtensions.EditorFor<TModel, TValue>(
  this HtmlHelper<TModel> html,
  Expression<Func<TModel, TValue>> expression,
  string templateName,                                    ← Nom du template à utiliser
  Object additionalViewData                               ← En général un anonyme
)
public static MvcHtmlString DisplayExtensions.Display( this HtmlHelper html, string expression,   ← Idem
  string templateName, Object additionalViewData
)
```

  - Contexte du template dans `ViewData.TemplateInfo`
    - `FormattedModelValue`, `GetFullHtmlFieldId`, `GetFullHtmlFieldName`, `HtmlFieldPrefix`
- Sur NuGet :

```
http://www.nuget.org/packages?q=EditorTemplates
```


DisplayExtensions.DisplayFor est pratique lorsque le type du modèle est connu, on utilise alors une lambda de sélection de propriété à la place du nom (string) de la propriété dont le label doit être affiché.

DisplayExtensions.Display est plus souple, elle permet d'aller chercher le label correspondant à une propriété à partir de son nom, et ce, quelque soit le type du modèle.

Les mêmes remarques s'appliquent aux méthodes de EditorExtensions.

Custom templates – prénom composé

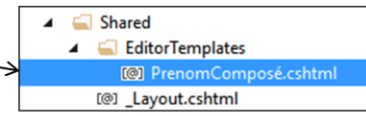
- Contrôleur

```
public class ExempleCustomTemplateController : Controller {
  public ActionResult Index() {
    ViewBag.Message = "Créez votre premier individu :";
    return View( new Individu() { Prenom = "Prénom-Composé", Nom = "Nom" } );
  }
  [HttpPost]
  public ActionResult Index( Individu qq1 ) {
    ViewBag.Message = $"OK, {qq1.Prenom} {qq1.Nom} est créé. Saisissez un nouvel individu :";
    ModelState.Clear();
    return View( new Individu() { Prenom = "Prénom-Composé", Nom = "Nom" } );
  }
}
```

Custom templates – prénom composé

- Vue

```
@model WebApplication1.Models.Individu

@{ ViewBag.Title = "Exemple de custom template"; }

<h2>@ViewBag.Title</h2>
<h3>@ViewBag.Message</h3>
@using( Html.BeginForm() ) {

  @Html.LabelFor( model => model.Prenom )
  <span>
    (Vue du modèle en lecture seule :
    <strong>@Html.DisplayFor( m => m.Prenom ))</strong>                    ← Pas de custom template pour affichage
  </span> <br />
  @Html.EditorFor( model => model.Prenom, "PrenomComposé", new { creation = true } ) ← ∃ custom template édition

  <br /><br />
  @Html.LabelFor( model => model.Nom ) <br />
  @Html.EditorFor( model => model.Nom ) <br />

  <br />
  <input type="submit" value="Créer" class="btn btn-default" />
}
```



Custom templates – prénom composé

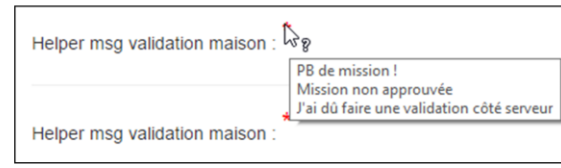
- Template (bien tassé…)

```
<input id="Prenom_debut" placeholder="Pr&#233;nom" /> -
<input id="Prenom_fin" placeholder="Compos&#233;" />
<input type="hidden" name="Prenom" id="Prenom" />
```

```
@model String
@{ string valueOuPlaceHolder = ViewBag.creation == null || !(bool)ViewBag.creation ? "value" : "placeholder";}
<input id="@ViewData.TemplateInfo.GetFullHtmlFieldId( "debut" )" @valueOuPlaceHolder="@Debut" /> -
<input id="@ViewData.TemplateInfo.GetFullHtmlFieldId( "fin" )" @valueOuPlaceHolder="@Fin" />
<input type="hidden" name="@ViewData.TemplateInfo.GetFullHtmlFieldName(string.Empty)"
       id="@ViewData.TemplateInfo.GetFullHtmlFieldId(string.Empty)" />
@functions {
  private string Debut {
    get { if( Model == null ) return string.Empty;
      int i = Model.IndexOf( '-' ); return i == -1 ? Model : Model.Substring( 0, i ); }
  }
  private string Fin { get { if( Model == null ) return string.Empty;
      int i = Model.IndexOf( '-' ); return i == -1 ? string.Empty : Model.Substring( i + 1 ); }
  }
}
<script type="text/javascript">
  setTimeout(                                          ← S'assurer que les scripts soient chargés...
    function() {
      $(function () {
        var baseId = "@ViewData.TemplateInfo.GetFullHtmlFieldId( "" )";
        var $hiddenConcat = $("#" + baseId);
        var $inputDebut =  $("#" + baseId + "_debut"); var $inputFin =  $("#" + baseId + "_fin");
        $hiddenConcat.parents("form").submit(
          function () { $hiddenConcat.val( ($inputDebut.val() + "-" + $inputFin.val()).replace(/-$/, "")); });
      });
    });
</script>
```

Exercice

- Création d'employé :
  - Id doit être nouveau : vérification serveur (simulée) lors de "l'enregistrement"
  - Nom 2-25 caractères, pas de caractères spéciaux
  - Poste : un énum + "choisir un poste", saisie obligatoire
- Un helper : les messages d'erreur dans le tooltip de la remarque




```
<hr /> Helper msg validation maison :
@Html.AllValidationMessagesToolTipFor(
  m => m.Mission,
  new Dictionary<string, object>() { { "title", "PB de mission !" }, { "class", "erreurAvecToolTip" } } ← Dico OK
)

<hr /> Helper msg validation maison :
@Html.AllValidationMessagesToolTipFor(
  m => m.Mission,
  new { title = "PB de mission !", @class = "erreurAvecToolTip" }
)
```

```
.erreurAvecToolTip {
  position: relative; top: -0.25em;
  font-size: 1.6em;
  color: red;
  cursor: help;
}
```

← Anonyme OK

Uploads

- Reçus dans la requête HTTP
  - Traités par une action classique
  - Lecture du flux => toutes opérations envisageables
    - Sauvegarde fichier serveur / BD / interprétation directe / ...
- Données "hors-ligne"
  - Fomulaire `enctype='multipart/form-data'`
  - L'action reçoit un `HttpPostedFileBase`
    - `Request.Files` est néanmoins utilisable

Membre de `HttpPostedFileBase`	**Info**
`ContentLength`	En octet
`ContentType`	Ex: "application/pdf"
`FileName`	Pour raison de sécurité/confidentialité, PAS DE CHEMIN
`InputStream`	Flux binaire, à lire / sauver directement ou à habiller d'un `StreamReader`
`public virtual void SaveAs( string filename )`	

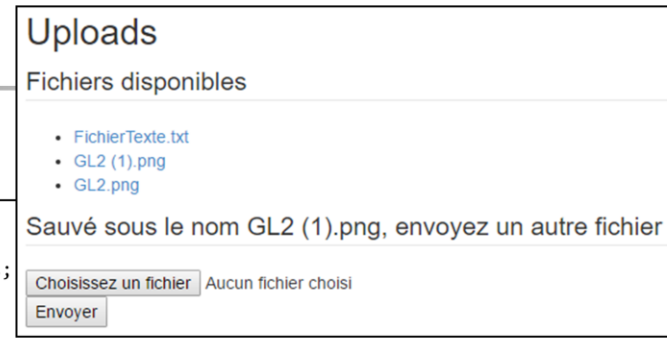
Uploads

- Contrôleur

```
public class UploadController : Controller {
  public ActionResult Index() {
    if( ViewBag.Message == null ) ViewBag.Message = "Envoyez un fichier";
    ViewBag.Fichiers = Directory.GetFiles( Server.MapPath( Settings.Default.DossierFichiersUtilisateur ) )
                                .Select( f => Path.GetFileName( f ) );
    return View("Index");
  }
  [HttpPost, ActionName( "Index" )]
  public ActionResult Enregistrer( HttpPostedFileBase fichier ) {
    string f;
    {
      ControllerContext.HttpContext.Application.Lock();
      for( int i = 0; System.IO.File.Exists( f = FormaterFichierIndex( fichier.FileName, i ) ); i++ ) ;
      fichier.SaveAs( f );
      ControllerContext.HttpContext.Application.UnLock();
    }
    ViewBag.Message = "Sauvé sous le nom " + Path.GetFileName( f ) + ", envoyez un autre fichier";
    return Index();
  }
  private string FormaterFichierIndex( string f, int i ) {
    return string.Format( "{0}/{1}{2}{3}",
      Server.MapPath( Settings.Default.DossierFichiersUtilisateur ),
      Path.GetFileNameWithoutExtension( f ),
      i > 0 ? " (" + i + ")" : string.Empty,
      Path.GetExtension( f ) );
  }
}
```

Uploads

- Vue




```
@{
  ViewBag.Title = "Uploads";
  var fichiers = (IEnumerable<string>)ViewBag.Fichiers;
}
<h2>@ViewBag.Title</h2>
<fieldset>
  <legend>Fichiers disponibles</legend>
  <ul>
    @foreach( var f in fichiers ) {
      <li><a href='@($"../FichiersUtilisateur/{f}")' target="=_blank">@f</a></li>
    }
  </ul>
</fieldset>
<fieldset>
  <legend>@ViewBag.Message</legend>
  @using( Html.BeginForm( "Index", "Upload", FormMethod.Post, new { enctype = "multipart/form-data" } ) ) {
    <input type="file" name="Fichier" />
    <input type="submit" value="Envoyer" />
  }
</fieldset>
```

Résultats non-HTML

- Une action peut retourner autre chose qu'une vue

Retour	Détails

string	La chaine brute sera envoyée au client. Vous pouvez *jouer* avec `Response.ContentType`.
	`Response.ContentType = "application/json";`
	`return "{chaine:'toto',nombre:1}";`

JavaScript()	Retourne un format JavaScript, permet de générer dynamiquement les scripts à inclure.
Json(objet)	Retourne la version sérialisée en JSON de l'objet. Permet de faire de l'AJAX. Nous en reparlerons plus tard.
File(octets, contentType, nom)	Permet au client de télécharger un fichier binaire.

  - Alternative à l'écriture directe dans le flux `Response.OutputStream` / `Response.ContentType`

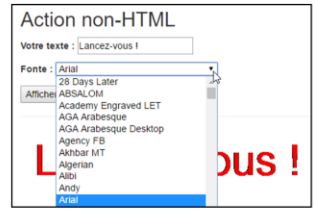
Résultats non-HTML

- Génération d'images avec GDI+

```
public class ExempleNonHtmlController : Controller {
  static readonly object VerrouContextePourMesurer = new object();
  static readonly Graphics ContextePourMesurer = Graphics.FromImage( new Bitmap( 1, 1 ) );
  public ActionResult Index() {
    ViewBag.Texte = "Lancez-vous !"; ViewBag.Fonte = "Arial";
    return View( FontFamily.Families.Select( f => f.Name ) );
  }
  private MemoryStream FluxASortir;
  public ActionResult Tracer( string fonte, string texte ) {
    Font f = new Font( FontFamily.Families.First( ff => ff.Name == fonte ), 48.0f, FontStyle.Regular );
    Size taille;
    lock ( ExempleNonHtmlController.VerrouContextePourMesurer ) {
      var tailleF = ContextePourMesurer.MeasureString( texte, f );
      taille = new Size( (int)tailleF.Width + 20, (int)tailleF.Height + 20 );
    }
    using( Bitmap image = new Bitmap( taille.Width, taille.Height ) ) {
      using( var dc = Graphics.FromImage( image ) ) {
        dc.FillRectangle( Brushes.Transparent, 0, 0, image.Width, image.Height );
        dc.DrawString( texte, f, Brushes.Red, 10, 10 );
      }
      FluxASortir = new MemoryStream();
      image.Save( FluxASortir, ImageFormat.Png );
      FluxASortir.Seek( 0, SeekOrigin.Begin );
      return File( FluxASortir, "image/png" );
    }
  }
  protected override void Dispose( bool disposing ) {
    if( disposing && FluxASortir != null ) FluxASortir.Dispose();
    base.Dispose( disposing );
  }
}
```

Résultats non-HTML

- Vue : presque que du JavaScript




```
@model IEnumerable<string>
@{ ViewBag.Title = "Action non-HTML"; }
<h2>@ViewBag.Title</h2>
<p>
  @Html.Label( "texte", "Votre texte : " )
  @Html.TextBox( "texte" )
</p>
<p>
  @Html.Label( "fonte", "Fonte : " )
  @Html.DropDownList( "fonte", Model.Select( f => new SelectListItem() { Text = f } ) )
</p>
<input type="button" value="Afficher" onclick="afficher()" />
<hr />
<img id="image" />
@section scripts {
  <script type="text/javascript">
    var urlDeBaseImage = '@Html.Raw( Url.Action( "Tracer", new { Fonte= "__LA_FONTE__", Texte= "__LE_TEXTE__"}))';
    $(afficher);
    function afficher() {
      var f = encodeURIComponent($("#fonte").val());
      var t = encodeURIComponent($("#texte").val());
      $("#image").attr("src", urlDeBaseImage.replace("__LA_FONTE__", f).replace("__LE_TEXTE__", t));
    }
  </script>
}
```