

Microsoft Volta - Partie 1 : Installation et découverte



par Louis-Guillaume MORAND ([Page perso de Louis-Guillaume MORAND](#))

Date de publication :


Dernière mise à jour :

Une des toutes dernières technologies de Microsoft nommée Volta fait de plus en plus parler d'elle. Nous allons, au cours de cet article, essayer de savoir un peu plus ce qui se cache derrière ce nom étrange.

Cet article se base sur la toute première CTP disponible au public.


- 0 - Introduction
- 1 - Présentation
 - 1.1 - Fonctionnement
 - 1.2 - Tiers-splitting
 - 1.3 - Support des navigateurs
- 2 - Limitations
 - 2.1 - Librairies
 - 2.2 - Support du langage Visual Basic
 - 2.3 - Temps de téléchargement
 - 2.4 - Compilateur Volta
 - 2.5 - Découpage et architecture
 - 2.6 - Imports et autres limitations
- 3 - Utilisation
 - 3.1 - Pré-requis
 - 3.2 - Hello World
 - 3.3 - Release
 - 3.4 - Cas réel
- 5 - Conclusion
- 6 - Liens et téléchargement
- 7 - Remerciements

0 - Introduction

Depuis quelques temps, un nouveau projet de chez Microsoft fait de plus en plus parler de lui. Ce projet, dont la preview fut officiellement présentée sur le  **blog dédié** le 5 décembre, se nomme Volta. Il s'agit d'une nouvelle technologie qui se base sur un compilateur bien spécifique puisqu'il permet de créer des applications Web à partir de code MSIL (.Net). La différence par rapport à l'ASP.Net réside dans le fait que le code fonctionnel (code behind) est transformé en JavaScript et l'application devient alors portable sur tout type de plateforme. Volta permet également mais surtout de faire du "lean programming" et de modifier l'architecture de l'application à n'importe quel moment de son développement.

*"Web application development
using only the materials in the room..."*



 *Il est très important de noter qu'il s'agit de la toute première CTP de test. Nous sommes encore loin d'une version pré-alpha. L'équipe de développement travaille pour le moment à tester la faisabilité des différentes fonctionnalités qu'ils souhaitent intégrer.*

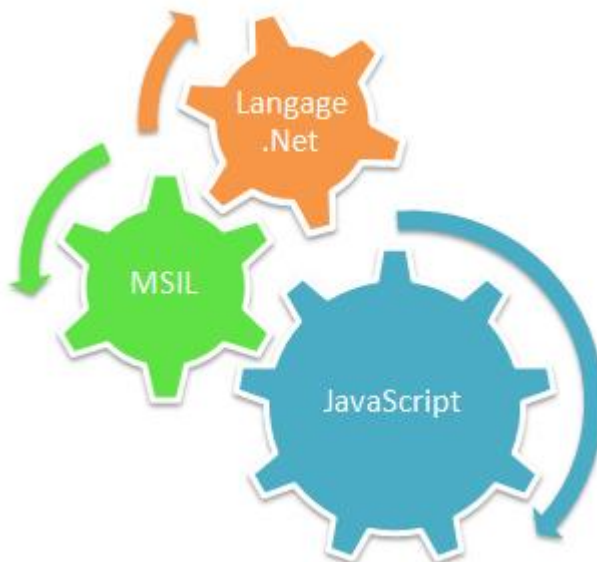
1 - Présentation

Volta, qui n'est pour le moment qu'expérimental, a pour objectif de proposer une boîte à outils complète afin de permettre au développeur de créer rapidement des applications Web. A première vue (mais à première vue uniquement), Volta pourrait ressembler fortement à d'autres boîtes à outils tel GWT (Google Web Toolkit) et c'est justement cette ressemblance qui fit (et fera encore) parler les mauvaises langues, considérant Volta comme un simple clone de GWT.

Nous allons justement voir au sein de cet article que Volta est bien plus que cela.

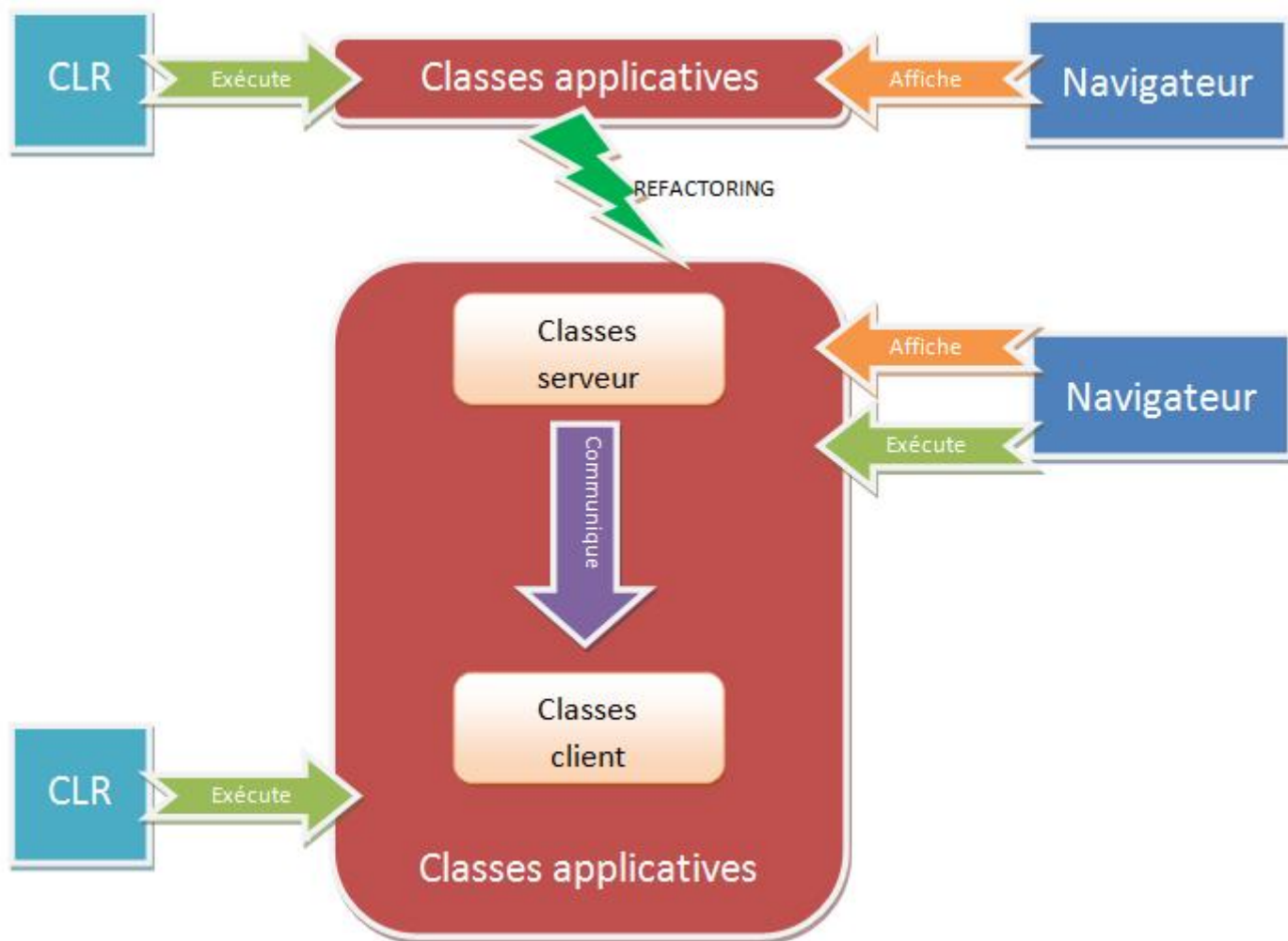
1.1 - Fonctionnement

Volta est ce qu'on pourrait appeler un "recompilateur". En effet, il se base du code MSIL (qui comme vous le savez est obtenu après la compilation d'un langage .Net), pour générer du JavaScript tel que le montre le schéma suivant:



Cela paraît "incroyable" (ou un peu "gros") dans le sens où beaucoup de gens trouvaient le JavaScript trop limité, mais l'équipe de Volta a effectivement transformé une grande partie de la plateforme .Net en équivalent JavaScript!

Ce qui paraît simple sur le schéma est en fait un peu plus complexe que cela, surtout que Volta peut faire plus que de la transformation. Volta se base sur le refactoring du code, particulièrement les attributs (nous en parlerons dans la seconde partie), pour générer l'application Web souhaitée.



Comme le montre le schéma précédent, nous obtenons après compilation des classes serveurs (en .Net) et des classes clientes (celles qui ont été transformées) et déjà, par ce simple découpage, l'application est passée en application 2-tiers.

Comme nous le verrons juste après, ce découpage mais surtout cette "transformation" en JavaScript ne se contente pas de simplement essayer de reproduire le même comportement, il crée des classes entières en JavaScript capables d'agir et de réaliser leur code métier propre, tout en étant capable d'interagir avec la partie restée en .Net (si on a choisi cette solution).

1.2 - Tiers-splitting

Un des avantages de Volta (mais que vous ne verrez que dans un second article à venir) est de pouvoir choisir comment sera découpée l'application au tout dernier moment. Cela s'appelle du "lean programming". Pour faire court et ne pas gâcher la surprise du second article, Volta permet de développer son application bien avant de se demander quelle partie sera cliente et quelle partie sera serveur.

Ainsi, on peut très rapidement lister un certain nombre d'avantages de Volta:

Indépendance du langage : On peut écrire du code dans n'importe quel langage qui est compilé en MSIL (C#, VB, IronPython, etc.)

Utilisation de toute la chaîne .Net : Utilisation des bibliothèques, de l'IDE, des snippets, des profilers, des outils comme FxCop ou ILDASM, etc

Splitting : Possibilité de découper l'application en autant de tiers que nécessaire

1.3 - Support des navigateurs

Le but de Volta étant de générer des applications Web, il devient important que ces applications soient supportées par tout type de navigateur. Les développeurs ASP.Net 1.1 se rappellent des problèmes de normes W3C non suivies. Ici, le code est du JavaScript et n'a donc pas forcément besoin de répondre à des spécifications strictes (si ce n'est de s'approcher des normes de DHTML) mais se doit surtout de fonctionner directement sur les navigateurs courants. Ainsi donc, sans avoir rien à changer, le code compilé est garanti de fonctionner de la même façon, tant sur Internet Explorer que sur Mozilla Firefox. Volta génère effectivement des blocs de code s'adaptant au navigateur utilisé.

Nous avons donc:

Un support multi-navigateur : Même code pour Internet Explorer et Firefox.

Transparence de débogage : Debug du code avec un navigateur spécifique

Fonctionnalités spécifiques à un navigateur : Quand nécessaires, les fonctionnalités spécifiques d'un navigateur sont utilisées

Intégration dans Visual Studio : Utilisation facile via Visual Studio

2 - Limitations

Comme toutes les bonnes choses, il y a des limitations. Si ces limitations peuvent déjà paraître nombreuses (et méritent pour cela un chapitre entier), elles s'expliquent naturellement pour la plupart d'entre elles.

En voici la liste, telle que décrite sur le site officiel du projet.

2.1 - Librairies

Les bibliothèques sont limitées de plusieurs façons:

- le multithreading n'est pas supporté
- La réflexion est quasi inexistante
- La BCL (Base Class Library) n'est qu'en partie supportée. Elle contient la plupart des méthodes communes aux différents langages du Framework .Net.

2.2 - Support du langage Visual Basic

Certaines méthodes qui n'étaient alors que présentes dans Visual Basic .Net ne sont pas supportées.

Ainsi, le mot clé My et tous les services qui en découlent ne sont pas supportés

Le late-binding (au moment de l'exécution) n'est pas supporté

Les méthodes de conversion comme Cint ne marchent que si le type visé supporte la conversion

Certains éléments hérités du VB6 comme la méthode Beep ne sont pas non plus supportés.



Le non support d'éléments du VB.Net a un rapport avec le support de la BCL et les développeurs de Volta disent déjà comment savoir corriger cela pour la suite, mais ceci demandant énormément de travail, ils n'ont pas tenu à l'intégrer dans la première CTP. Cela devrait donc s'améliorer dans les futures versions de Volta.

2.3 - Temps de téléchargement

Durant l'exécution, une application Volta télécharge plusieurs fichiers depuis le serveur, un pour chaque classe utilisée. Les classes ne sont téléchargées que lorsqu'elles sont requises, ainsi, le temps de premier lancement n'est pas affecté par le téléchargement de classes utilisées plus tard dans l'exécution. Par contre, à cause du nombre de fichiers souvent requis pour supporter l'application, les temps de chargement peuvent être affectés. Dans des versions futures, il est prévu d'ajouter une logique dans la façon de compiler et de charger les assemblées afin de répondre à cela.



Si vous vous amusez à analyser ce qui est téléchargé en lançant les samples, vous vous rendrez compte de la taille conséquente qui est téléchargée. C'est plus que conséquent pour le moment mais l'équipe de développement précise que les optimisations seront faites dans le futur.

2.4 - Compilateur Volta

Le compilateur Volta est capable de générer du JavaScript en sortie mais ce JavaScript n'est pas entièrement optimisé. Il répondra clairement au besoin exprimé mais ne sera optimisé pour tel ou tel navigateur, ni ne fera d'optimisation en nombre de lignes de code, ceci alourdissant le poids des pages

2.5 - Découpage et architecture

Volta qui présente comme fonctionnalité de pouvoir rapidement faire du découpage multi-tiers nécessite pour cela certains contraintes:

- Seules les sous-classes dérivant de `Objet` peuvent être découpées
- La sérialisation entre deux tiers n'est pas rapide
- L'URI du serveur peut seulement être spécifié en tant qu'argument d'un attribut (dans le code source). Ainsi donc, pour changer l'adresse du serveur, il faudra modifier l'argument puis recompiler le projet.
- Pour le moment, il n'est possible de ne découper l'application que sur un seul serveur

2.6 - Imports et autres limitations

Le débogage multiple de plusieurs application Volta n'est pas possible et le binding grâce à l'attribut `Import` ne permet pas l'utilisation des tableaux (`Arrays`)


3 - Utilisation

3.1 - Pré-requis

Il vous faut: d'abord un environnement de développement et actuellement, c'est

- Visual Studio 2008 (beta 2 ou finale) (**Version d'essai 90 jours ici**)
- Microsoft Volta (**dernière CTP**)

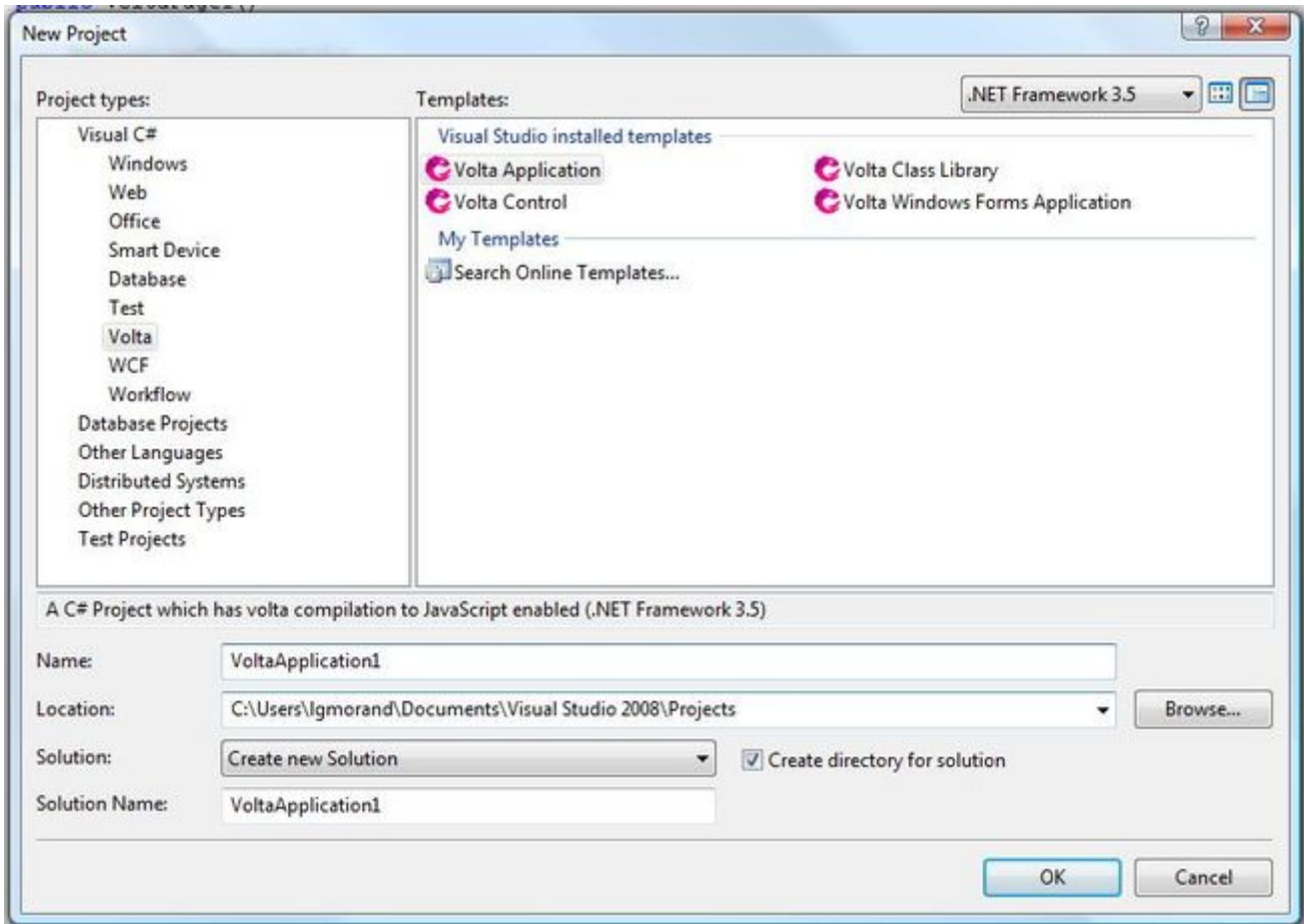
Vous n'avez qu'à lancer le fichier d'installation et relance Visual Studio. Créez un nouveau projet et choisissez le framework 3.5 pour voir apparaître les projets de type volta.

 *Visual Studio 2008 Express ne permettant pas de charger les extensions nécessaires à Volta, vous ne pourrez pas faire marcher Volta avec celui-ci.*

3.2 - Hello World

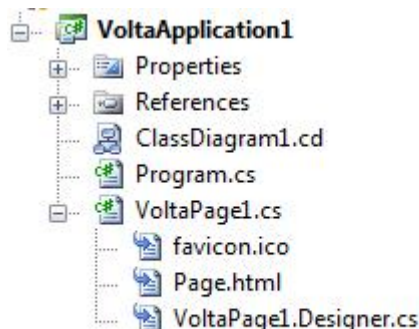
Commençons par l'application la plus simple qui soit, le mythique "Hello World!". Qui n'a jamais commencé par cela?

Ouvrez donc Visual Studio puis choisissez de faire un nouveau projet (Fichier > Nouveau > Projet) puis choisissez un projet de type Volta Application (le filtre en haut à droite doit être placé sur Framework 3.5)



Vous remarquez alors en regardant l'explorateur de solution qu'il contient un élément `VoltaPage1`, fichier class C#, lui même relié à un designer de cette même classe mais également à un fichier `Page.html` qui sera la partie visuelle de votre application.

Pour ce premier exemple, nous ne nous occuperons pas de cet élément bien qu'il soit utilisé une fois la compilation accomplie.



Dans le projet vidé crée, vous constaterez que votre classe hérite de **Page**, mais pas l'objet **Page** que vous utiliseriez en ASP.Net, non je parle de l'objet `Microsoft.LiveLabs.Volta.Html.Page`. Celui-ci ne marche pas exactement de la même façon mais vous en rendrez compte au fur et à mesure que vous découvrirez Volta.

Continuons donc et remarquons notre constructeur. Même pas de `Page_Load`, juste un constructeur.

Vous remarquerez aussi les deux using qui nous intéressent:

Imports principaux

```
using Microsoft.LiveLabs.Volta.Html;  
using Microsoft.LiveLabs.Volta.Xml;
```

Modifions maintenant notre constructeur dans lequel nous allons rajouter quelques lignes:

Hello World

```
Button b = new Button();  
b.Value = "Hello?";  
b.Click += delegate { Window.Alert("Hello Developpez.com!"); };  
Document.Body.AppendChild(b);
```

Nous créons un bouton, lui spécifions son texte via la propriété **Value** (comme en HTML), puis nous lui assignons un évènement (comme en C#) mais dans lequel nous tapons directement notre code JavaScript. Enfin nous ajoutons notre composant bouton à notre page. Compilons et exécutons.

Une page html est alors lancée et...magie, nous y retrouvons notre bouton. Cliquons alors dessus pour voir un message "Hello Developpez!" s'afficher. Notre Hello World fonctionne!

Analysons ce qui s'est réellement passé. Nous avons du code C# qui au final nous génère plusieurs fichiers. Nous avons notre fichier Page.html avec un squelette de page vide mais respectant les standards Web!, des fichiers JavaScript génériques VoltaManagedInteropHelpers.js et compact.js (pour la compatibilité avec d'autres navigateurs. Mais sont également générés des fichiers pour représenter l'assembly de l'application (assembly.js) et des objets en JavaScript représentant notre couche métier. Ainsi donc, notre bouton et sa méthode, que l'on a écrit en 4 lignes se retrouve transcrit en 13 grosses lignes de **code**

C'est à ce moment que l'on se rend compte d'où vient le poids des pages Volta. Je vous laisse déjà imaginer le poids avec une application plus complète.

3.3 - Release

"Release", pourquoi nous parle-t-il de release. Et bien essayez vous même et compilez votre projet en mode Release puis analysez le code généré.

- La page.html a été modifiée et contient directement le code qui appelle les objets Javascript
- Un fichier jsfile.aspx est crée et qui contient les entêtes normaux d'une page aspx
- Un fichier loader.js est crée, qui lui, va se charger de charger tous les éléments nécessaires et se servir de la page aspx comme noyau central de chargement, en lui passant des paramètres.

C'est, je trouve, uniquement en mode release que l'on peut réellement se rendre compte du fonctionnement de Volta même s'il faut pour cela s'amuser à lire et analyser les centaines de lignes de code JavaScript générées.

3.4 - Cas réel

Un Hello World c'est bien joli, mais ce n'est pas avec cela qu'on fait une application. D'ailleurs ce n'est pas avec l'exemple suivant non plus que vous ferez une application néanmoins, je pense qu'il est intéressant pour commencer à voir comment il est possible de coder comme nous le ferions en ASP.Net (mais aussi en JavaScript+DOM) pour arriver à créer simplement l'application qui sera portable sur n'importe quelle plateforme.

Nous allons implémenter un formulaire de login tout ce qu'il y a de plus simple, qui affiche un message en cas d'erreur et qui redirige vers la partie privée (ici [Developpez.com](#), lorsque les credentials sont identiques)

Recréons un nouveau projet de type Volta Application, que nous nommerons VoltaApplication2 (10 minutes pour trouver ce nom très recherché! :))

Intéressons nous cette fois au fichier page.html dans lequel nous allons dessiner notre code xHTML contenant un bouton et deux textbox mais aucun code, pas même une balise <form>

```
<body>
  <table id="frmLogin">
    <tr>
      <td>
        Login :</td>
      <td>
        <input id="txtLogin" type="text" /></td>
    </tr>
    <tr>
      <td>
        Mot de passe :</td>
      <td>
        <input id="txtPwd" type="password" /></td>
    </tr>
    <tr>
      <td colspan="2">
        <input id="btnLogin" type="button" value="Login" /></td>
    </tr>
  </table>
  <div id="msgBox"></div>
</body>
```

Vous avez remarqué les ID placés sur les contrôles. Qu'importe le langage, c'est le seul moyen de viser directement tel ou tel contrôle. Vous aurez aussi remarqué qu'il y a pas de tag `runat="server"` comme vous auriez en ASP.Net.

Mais alors, comme cela va-t-il marcher? Et bien, nous allons utiliser DOM à travers du code C# tout simplement:

Dans notre constructeur, nous n'allons plus créer un bouton mais récupérer celui que nous avons placé dans le code HTML, en utilisant la méthode bien connue `GetById`. Ici, nous n'utilisons pas directement la méthode JavaScript qui fait cela, mais une classe wrapper qui le fait pour nous et qui a été créée par l'équipe de Volta.

Puis nous ajoutons un événement mais cette fois, nous appelons une autre méthode C#, et pas une fonction JavaScript comme dans le Hello World.

Constructeur

```
Input btn = Document.GetById<Input>( "btnLogin" );
btn.Click += btnLogin_Click;
```

Regardons maintenant ce que nous allons mettre dans notre méthode `btnLogin_Click`. Voici son code:

Méthode `btnLogin_Click`

```
void btnLogin_Click()
{
```

Méthode btnLogin_Click

```
Input txtLogin = Document.GetById<Input>("txtLogin");
Input txtPwd = Document.GetById<Input>("txtPwd");
if ( txtLogin.Value != String.Empty && txtLogin.Value == txtPwd.Value)
{
    Document.GetById<HtmlElement>("msgBox").InnerHTML = "<font
color='green'>Bienvenue!</font>";
    Document.GetById<HtmlElement>("frmLogin").Style.Display = "none" ;
    this.Window.Alert("Maintenant la redirection");
    this.Window.Navigate("http://www.developpez.com");
}
else Document.GetById<HtmlElement>("msgBox").InnerHTML = "<font color='red'>Try again!</font>";
}
```

Sur les deux premières lignes, nous récupérons une référence vers nos objets HTML. Puis nous testons si les valeurs sont identiques et agissons en fonction du test.

Pour les développeurs ASP.Net, cela ne les change pas de leurs habitudes même si je vous rappelle que nos contrôles n'ont pas été mis en tant que contrôles serveurs pour pouvoir y accéder.

Sur les lignes suivantes, j'affiche un message en vert et je cache le formulaire de login lorsque le login est équivalent au mot de passe.

Notez bien par contre, les deux lignes suivantes:

```
this.Window.Alert("Maintenant la redirection");
this.Window.Navigate("http://www.developpez.com");
```

Vous vous doutez de ce que cela fait mais néanmoins, cela ne fait pas partie du framework .Net. Là encore, il s'agit de méthodes contenues dans une classe wrapper qui vous permet très simplement d'agir comme du JavaScript. Fini les ClientRegisterStartupScript, grâce aux très nombreuses classes wrapper, vous pouvez écrire du code côté client très simplement.


5 - Conclusion

C'en est fini pour cette présentation mais vous aurez rapidement un second article qui vous montrera plus en détails les avantages principaux de Volta. En attendant, nous avons déjà pu nous rendre compte d'un certain nombre de choses concernant Volta.

Dans un article à venir, nous verrons le découpage multi-tiers et le fonctionnement des attributs mais nous verrons également comment utiliser AJAX au sein de nos applications Volta.

6 - Liens et téléchargement

Forum d'entraide officiel de Volta:  [ici](#)

Interview d'Erik Meijer, chef de projet de Volta:  [ici](#)

Téléchargement des sources de cet article : [ici](#)

7 - Remerciements

Je tiens à remercier ChristopheJ (de l'équipe JAVA!) pour m'avoir fait découvrir Volta, l'équipe .Net pour leurs commentaires avisés pour ses conseils pour l'amélioration de cet article et Baptiste Wicht pour ses corrections.

