

# Le XML-Script

---

## XML-Script : la mort du JavaScript ?

Introduit avec le Framework Atlas, le XML-Script est une nouvelle forme de programmation Web. Il se présente sous la forme d'un langage déclaratif, comme son nom nous le laisse penser, celui-ci est construit autour de XML.

## HelloWorld en XML-Script

Dans cet article, nous ne dérogerons pas à la règle de l'exemple d'introduction d'une nouvelle technologie, nous allons réaliser une application simple de type « Hello World ». Celui-ci sera constitué d'une TextBox, d'un label et d'un bouton, ce dernier permettant de spécifier la valeur du TextBox.

   
Bonjour Cyril

## Le HTML

Commençons tout d'abord par définir le code HTML :

```
<input type="Text" id="TextBoxName" value="Cyril" />
<button id="btn1" runat="server">Click me</button>
<br />
<span id="result"></span>
```

Nous voyons qu'il s'agit d'un document HTML tout ce qu'il y a de plus classique, l'utilisation du XML-Script ne modifie en rien nos habitudes.

## Le XML-SCRIPT

Rajoutons maintenant du code XML-Script grâce à une balise *script* de type « *text/xml-script* », cette balise contiendra notre code xml-script et sera parsé par le moteur d'Atlas. Il faut savoir que l'on peut mettre autant de balises de ce type que l'on veut, elles seront toutes prise en compte par le Framework.

Tout document XML-Script doit commencer avec l'élément *page* qui ne possède que 2 éléments fils : *references* qui permet de charger des fichiers JavaScript externe et *components* qui nous permet d'instancier et manipuler des objets JavaScript.

```
<script type="text/xml-script">
  <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
    <references>
    </references>
    <components>
    </components>
  </page>
```

```
</script>
```

Rajoutons alors notre TextBox, button et label dans le xml-script.

```
<components>
  <textBox id="TextBoxName" />
  <button id="btn1" />
  <label id="result" />
</components>
```

A ce stade les différents contrôles sont maintenant enregistrés dans Atlas, si l'on veut récupérer itérativement ces objets, c'est-à-dire via du code JavaScript classique il faut utiliser la fonction `$object(id)`.

Abonnons nous à l'événement *click* du bouton, pour cela on rajoute un objet `invokeMethod` à la collection d'handler « *click* » de l'objet `button`. L'objet `invokeMethod` prend 2 arguments : le nom de la méthode qui sera appelé et l'id de l'objet qui contient la méthode.

```
<button id="btn1">
  <click>
    <invokeMethod method="evaluateIn" target="resultBindingID" />
  </click>
</button>
```

Dans notre cas, on va appeler la méthode `evaluateIn` de l'objet d'ID « *resultBindingID* » qui est un objet de type *binding*. Pour bien comprendre le fonctionnement du binding commençons d'abord par un exemple d'utilisation :

```
<label id="result">
  <bindings>
    <binding id="resultBindingID" dataContext="TextBoxName"
      dataPath="text" property="text" automatic="false"
      transform="ToString"
      transformerArgument="Bonjour {0}" />
  </bindings>
</label>
```

L'objet `binding` est obligatoirement contenu dans une collection de `bindings` d'un autre objet, dans notre cas il est contenu dans les `bindings` du `label`. Un `binding` est associé à une propriété d'un objet, pour savoir avec quel objet et quelle propriété on lie notre `binding` on utilise respectivement les propriétés `dataContext` et `dataPath`. Dans l'exemple le `binding` est associé avec la propriété `text` de l'objet `TextBoxName`. L'objet `Binding` possède la méthode `evaluateIn` qui appelle un `transformer` renseigné par la propriété `transform` du `binding` c'est-à-dire le `transformer` « *ToString* ». Pour une introduction au XML-Script nous considérons un `transformer` comme équivalent à une méthode.

Une fois appelé ce `Transformer` prendra en paramètre la propriété associée au `binding` (`dataContext.dataPath` ie `TextBoxName.text`) ainsi que la valeur de la propriété `transformerArgument` : « *Bonjour {0}* ». Le résultat de ce `transformer` définit la propriété `text` du contrôle qui contient le `binding`, cette propriété est connue de l'objet `binding` par sa propriété « *property* ».

On note également que l'on a défini la propriété `automatic` de notre binding à « `false` » ceci pour dire que l'on n'appelle pas la méthode `evaluateIn` à chaque fois que la propriété associée au binding change : le label ne changera pas de valeur suite à chaque changement de valeur de la `TextBox`.

Pour résumer : lorsque l'on clique sur le bouton on appelle la méthode `evaluateIn` du binding qui va modifier le texte du label grâce au Transformer `ToString` qui prend « `Bonjour {0}` » et la valeur du `textBox` en paramètres.

## Avantages du XML-Script par rapport au JavaScript

Aux premiers abords le XML-Script semble déroutant, avec un peu d'habitude celui-ci devient simple et intuitif. Nous avons vu que nous pouvons lier plusieurs objets entre eux, cela montre la puissance et la flexibilité du XML-Script très puissant cependant cela le rend plus compliqué à comprendre.

Tout comme XAML pour .NET, le XML-Script ne fait que créer et manipuler des instances d'objet JavaScript, le JavaScript itératif le fait déjà très bien, alors pourquoi avoir inventé un nouveau langage ? Étonnement la programmation déclarative nécessite beaucoup moins de lignes de code que la façon itérative. De plus, avec l'habitude il est souvent plus simple et rapide d'écrire et de comprendre du XML-Script que du code JavaScript. Cependant il est important de savoir que l'on ne peut pas tout faire avec cette technologie. Puisque le XML-Script sert uniquement à créer et manipuler des instances d'objet JavaScript, nous ne pouvons pas déclarer de nouveaux types, pour cela il faudra utiliser du code JavaScript itératif classique.

**Cyril DURAND**

Membre de l'équipe de développement *CodeS-SourceS*

<http://blogs.developpeur.org/cyril/>

