

# L'auto-complétion dans une application .Net

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

Date de publication : 30/10/2007

Dernière mise à jour :

Dans ce tutoriel, nous allons présenter la fonctionnalité d'auto-complétion et expliquer son implémentation au sein de vos applications .Net.

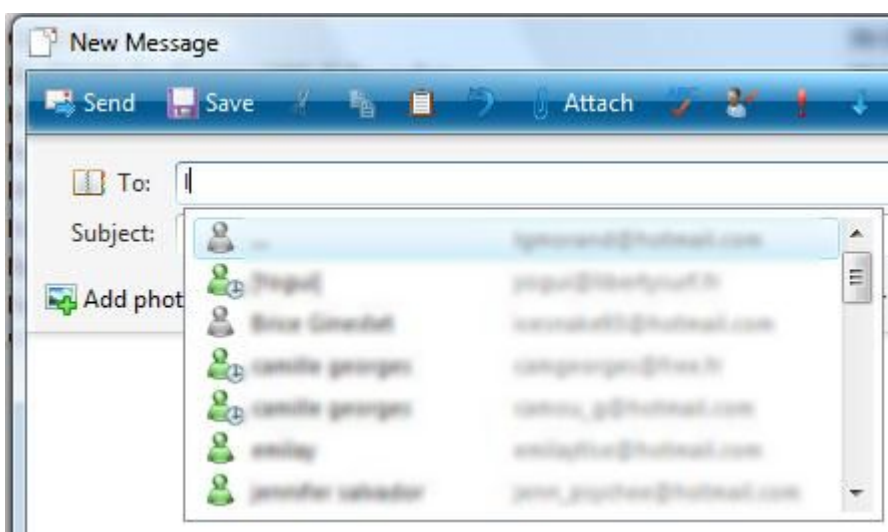
- 1 - Présentation
  - 1.1 - Principe
  - 1.2 - Types d'auto-complétion
  - 1.3 - Sources d'auto-complétion
- 2 - Implémentation
- Conclusion

## 1 - Présentation

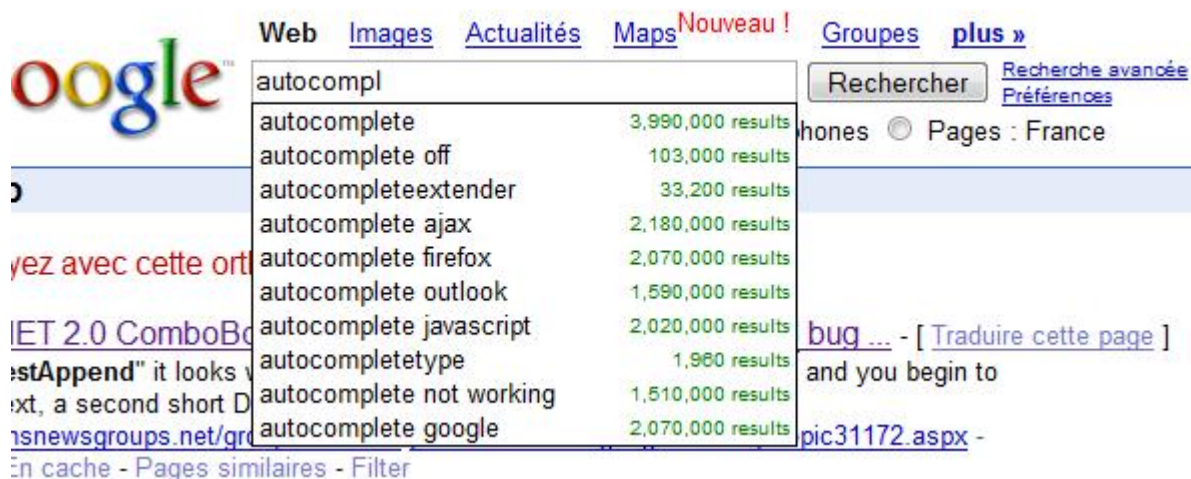
Dans cette première partie, nous verrons comment fonctionne l'auto-complétion dans une application .Net.

### 1.1 - Principe

L'auto-complétion est un mot étrange qui pourrait être également appelé "aide à la saisie". Sans le savoir, vous l'utilisez régulièrement dans vos applications de tous les jours, que ce soit votre client mail, votre navigateur Web ou même votre moteur de recherche habituel et pourtant, vous ne vous êtes peut-être jamais interrogé sur son fonctionnement ou son implémentation.



*Client Mail*



*Google Suggest*

### 1.2 - Types d'auto-complétion

L'auto-complétion comme nous l'avons vu consiste à proposer des saisies afin d'aider l'utilisateur dans ses choix, mais elle existe surtout sous différentes formes. Pour montrer ces différentes méthodes, nous prendrons un exemple un control TextBox (mais nous aurions pu aussi prendre une ComboBox).

La méthode de complétion est contrôlée par la propriété AutoCompleteMode et qui prend quatre valeurs possibles:

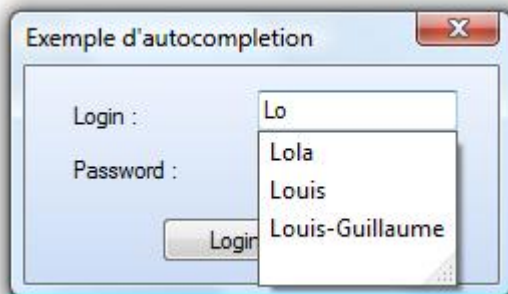
La valeur **None** qui désactive l'auto-complétion pour ce contrôle.



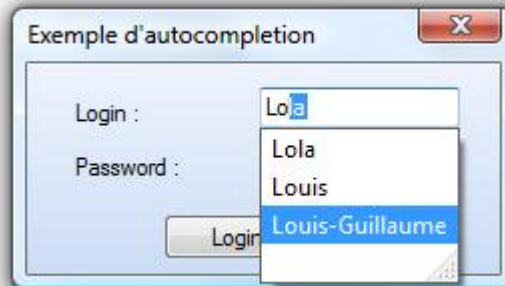
La valeur **Append** active la complétion qui écrit automatiquement la fin du texte avec la valeur la plus probable. Il est alors possible avec les flèches haut et bas de naviger parmi les saisies possibles.



La valeur **Suggest** active une liste déroulante n'affichant que les éléments corrects (dont le début correspond au texte actuellement saisi dans la TextBox).

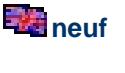


La valeur **SuggestAppend** qui combine l'effet des valeurs **Append** et **Suggest**.



### 1.3 - Sources d'auto-complétion

L'auto-complétion c'est utile mais concrètement, où le contrôle va-t-il chercher ces mots conseillés? Peut-on personnaliser cette liste? Peut-elle être modifiée au fur et à mesure? Des questions auxquelles je vais tâcher de répondre.

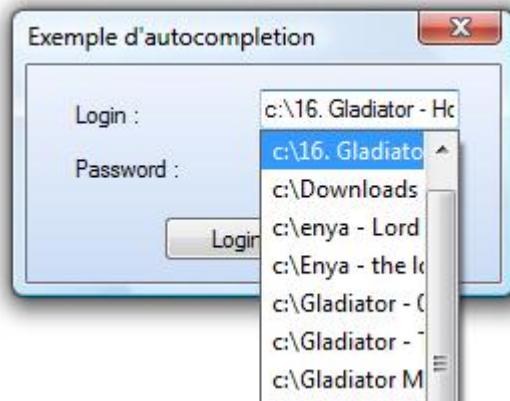
L'auto-complétion se fait à l'aide d'une liste d'éléments que l'on appelle la "source". Cette propriété peut avoir  neuf valeurs différentes.

La valeur **None** permet de désactiver l'auto-complétion. En effet, puisqu'aucune source n'est associée, il est impossible d'afficher des saisies conseillées.

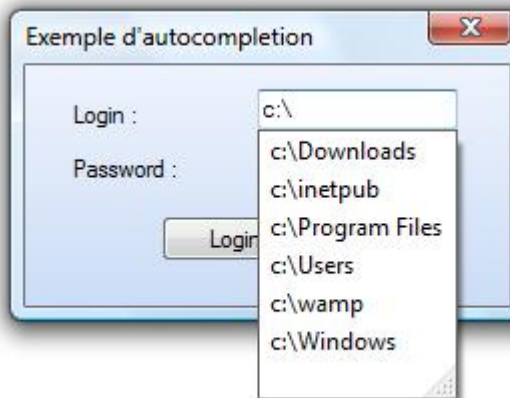
La valeur **CustomSource** permet comme son nom l'indique de créer soit même la source de données.



La valeur **FileSystem** utilise les chemins/noms des fichiers et dossiers pour proposer des saisies.



La valeur **FileSystemDirectories** fonctionne comme **FileSystem** mais n'affiche que les dossiers.

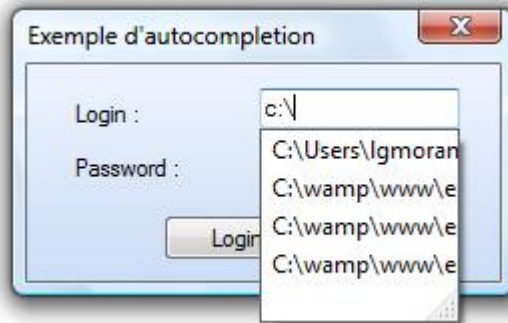


La valeur **HistoryList** affichera l'historique des éléments ouverts, que ce soit des fichiers ou des URLs.

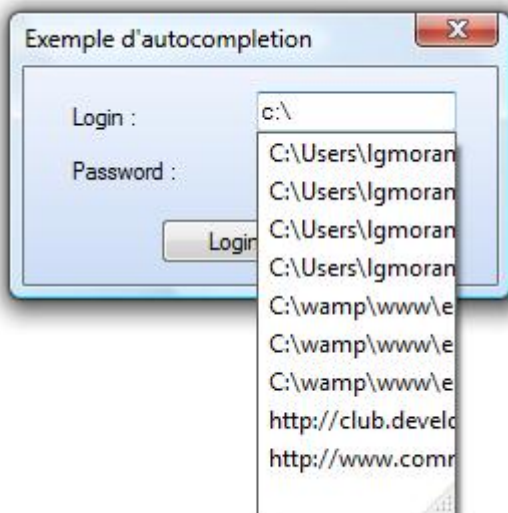


La valeur **ListItems** (réservée aux ComboBox) permet d'utiliser la collection d'éléments de la combobox en tant que source d'auto-complétion.

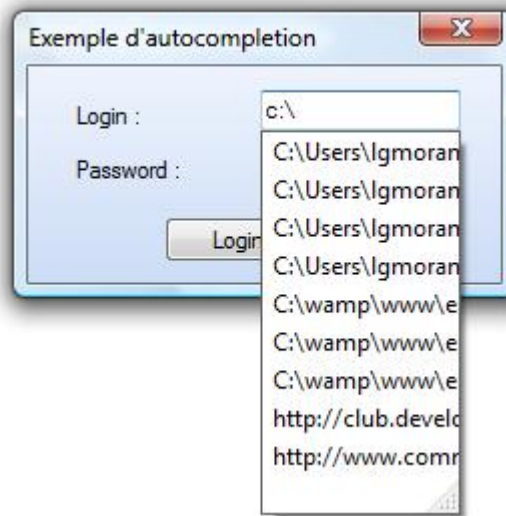
La valeur **RecentlyUsedList** affiche la liste des éléments récemment utilisés. Soit les documents ouverts, soit les commandes taper dans la fenêtre Exécuter.



La valeur **AllSystemSources** est la combinaison des propriétés **FileSystem** et **AllUrl**.



La valeur **AllUrl** est la combinaison des propriétés **HistoryList** et **RecentlyUsedList**.



## 2 - Implémentation

Pour l'implémentation, il n'y a rien de plus simple puisqu'il suffit de glisser-déposer votre contrôle puis dans l'éditeur de propriétés de votre IDE ou via le code, choisir une valeur parmi une énumération bien précise et tout se fait alors tout seul.

Maintenant, il est aussi possible de personnaliser soit même cette source en choisissant la valeur CustomSource pour la propriété AutoCompleteSource. Nous allons prendre comme exemple, le chargement d'un fichier XML intégré à l'application mais ce fichier XML pourrait aussi bien venir d'un Webservice ou alors nous pourrions aussi chercher les informations dans une base de données. La "difficulté" consiste à transformer notre XML en une collection utilisable comme source d'auto-complétion, représentée par la classe AutoCompleteStringCollection. Ainsi nous commençons par instancier notre classe.

```
AutoCompleteStringCollection collec = new AutoCompleteStringCollection();
```

Il nous faut dans un second temps récupérer notre fichier XML. Puisqu'il est intégré (embedded) dans notre application, nous devons utiliser la réflexion

```
System.IO.Stream file =  
    System.Reflection.Assembly.GetExecutingAssembly().GetManifestResourceStream("DVPAutocompletion.Resources.login");  
  
XmlDocument objXmlDoc = new XmlDocument();  
objXmlDoc.Load(file);
```

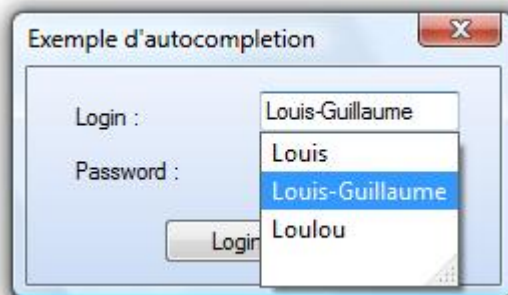
Nous remplissons alors notre collection

```
XmlNodeList list = objXmlDoc.GetElementsByTagName("login");  
foreach (XmlElement login in list)  
{  
    collec.Add(login.FirstChild.Value.ToString());  
}
```

Avant de la définir comme source d'auto-complétion de notre TextBox

```
textBox1.AutoCompleteSource = AutoCompleteSource.CustomSource;  
textBox1.AutoCompleteCustomSource = collec;
```

Et comme nous pouvons le vérifier, les logins renseignés dans le fichier XML sont alors utilisés pour l'auto-complétion.



## Conclusion

Bien que cette fonctionnalité ne révolutionnera pas vos interfaces graphiques, néanmoins elle améliorera ce qu'on nomme "l'expérience utilisateur" et qui fait que l'interface de votre application est parfaitement désignée pour répondre au mieux aux besoins de l'utilisateur tout en restant le plus simple possible. À l'avenir, lorsque vous aurez une saisie utilisateur, vous pourrez envisager d'aider sa saisie

