

Gestion des utilisateurs et utilisation du contrôle PeopleEditor

par Philippe Vialatte ([ma page DVP](#))

Date de publication : 29/04/2008

Dernière mise à jour : 29/04/2008

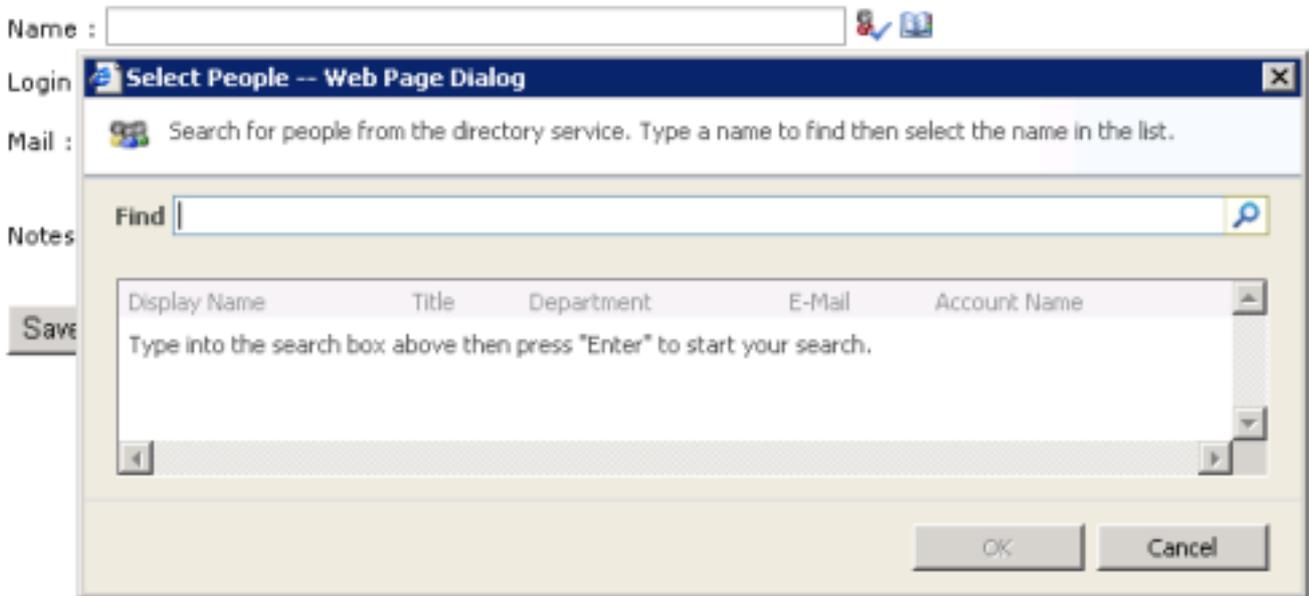
Dans cet article, on va voir comment lister les utilisateurs d'un site SharePoint, ainsi que comment utiliser le contrôle PeopleEditor pour ajouter ou modifier un utilisateur existant.

I - Introduction.....	3
II - Première version du contrôle.....	4
III - Introduction du PeopleEditor.....	6
Les propriétés importantes du PeopleEditor.....	6
Ajout d'un champ de type PeopleEditor.....	6
IV - Sauvegarde / modification d'un utilisateur existant.....	8
V - Fonctionalites "avancées" du PeopleEditor.....	9
Récupération de données supplémentaires.....	9
Appel de JavaScript après résolution AD.....	10
VI - Remerciements.....	12

I - Introduction

La version 3 de Windows SharePoint Services contient une mine de nouveaux contrôles très intéressants.

Au nombre de ces contrôles, le PeopleEditor est, à mon avis, un des plus sympathiques à avoir dans notre boîte à outils.



Pour les besoins de l'article, on va développer un petit contrôle permettant de lister les utilisateurs, d'éditer leurs informations, et d'ajouter des utilisateurs à SharePoint.

Le PeopleEditor permet à nos contrôles d'effectuer des recherches et/ou des sélections d'utilisateurs sur Active Directory, tout en conservant une interface graphique intégrée à SharePoint.

Classiquement, on l'utilisera dès que notre workflow, webpart ou page web nécessitera de sélectionner une ou plusieurs personnes de façon déterministe (comprendre, en ayant une garantie que la personne sélectionnée existe bel et bien).

Le développement est fait sur une machine avec WSS installé en local, et avec un utilisateur administrateur de liste.

Pour gagner un peu de temps, les développements sont faits sous forme de user control, et chargés dans SharePoint par le biais de la SmartPart (voir <http://lefortludovic.developpez.com/tutoriels/sharepoint/smartpart/> pour une explication sur la SmartPart).

II - Première version du contrôle

On va commencer par une interface graphique minimale#

Users	<input type="text"/>
Name	<input type="text"/>
Login	<input type="text"/>
Mail	<input type="text"/>
Notes	<input type="text"/>

Pour réaliser cette interface, on va créer un fichier ManageUsers.ascx dans un site web, ainsi que le fichier ManageUsers.ascx.cs correspondant pour le code-behind.

Une fois ce fichier créé, on va coder notre petite interface, ainsi que le code-behind permettant, lorsque l'on sélectionne un utilisateur, d'afficher, dans les textbox correspondantes, le nom, les informations de login, mail et les notes correspondant à l'utilisateur.

ManageUsers.ascx

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="ManageUsers.ascx.cs" Inherits="ManageUsers" %>
<table>
  <tr>
    <td>
      Users :
    </td>
    <td>
      <asp:DropDownList ID="dropUsers" AutoPostBack="true"
OnSelectedIndexChanged="dropUsers_SelectedIndexChanged" runat="server" /></td>
  </tr>
  <tr>
    <td>
      Name :
    </td>
    <td>
      <asp:TextBox ID="tbName" runat="server" /></td>
  </tr>
  <tr>
    <td>
      Login :
    </td>
    <td>
      <asp:TextBox ID="tbLogin" runat="server" /></td>
  </tr>
  <tr>
    <td>
      Mail :
    </td>
    <td>
      <asp:TextBox ID="tbMail" runat="server" /></td>
  </tr>
  <tr>
    <td>
      Notes :
    </td>
    <td>
      <asp:TextBox ID="tbNotes" TextMode="MultiLine" Rows="3" runat="server" /></td>
  </tr>
</table>
```

ManageUsers.ascx

```
</tr>
</table>
```

Maintenant, le code-behind :

ManageUsers.ascx.cs

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.SharePoint;

public partial class ManageUsers : UserControl {

    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) LoadUsersCombobox();
    }

    private void LoadUsersCombobox() {
        dropUsers.Items.Clear();
        dropUsers.Items.Add("");

        foreach (SPUser user in SPContext.Current.Web.SiteUsers) {
            dropUsers.Items.Add(new ListItem(user.Name, user.ID.ToString()));
        }
    }

    protected void dropUsers_SelectedIndexChanged(object sender, EventArgs e) {

        ClearUserPanel();
        if (!string.IsNullOrEmpty(dropUsers.SelectedValue)) {
            LoadUser(Convert.ToInt32(dropUsers.SelectedValue));
        }
    }

    private void ClearUserPanel() {
        tbName.Text = string.Empty;
        tbLogin.Text = string.Empty;
        tbMail.Text = string.Empty;
        tbNotes.Text = string.Empty;
    }

    private void LoadUser(int userId) {
        SPUser user = web.SiteUsers.GetByID(userId);
        tbName.Text = user.Name;
        tbLogin.Text = user.LoginName;
        tbMail.Text = user.Email;
        tbNotes.Text = user.Notes;
    }
}
```

Le code en lui-même ne pose pas de réelles difficultés, on va tout de même revenir sur la seule ligne concernant SharePoint pour le moment (ne vous inquiétez pas, ça ne va pas durer...), à savoir:

```
foreach (SPUser user in SPContext.Current.Web.SiteUsers)
```

Ici, il faut noter que l'on demande à SharePoint la liste des utilisateurs avec la propriété **SiteUsers**, et non pas **Users**. En effet, **SiteUsers** renvoie la liste des utilisateurs ayant des droits d'accès au site en question, tandis que **Users** ne renvoie que les utilisateurs que l'on a explicitement associés au site (en les ajoutant au groupe des visiteurs, membres ou propriétaires du site). De façon plus concrète, imaginons que l'utilisateur **domaine\pvialatte** existe dans SharePoint, et que le site **site1** contienne le groupe **Everyone** dans ses visiteurs.

(Vous me suivez toujours, hein ?)

SiteUsers contiendra à la fois **Everyone** et **domaine\pvialatte**, mais Users ne contiendra qu'Everyone.

C'est le genre de petite différence entre deux propriétés qui a tendance à envoyer les développeurs à l'asile...

III - Introduction du PeopleEditor

Les propriétés importantes du PeopleEditor

Normalement, pour avoir le maximum d'informations sur un contrôle, mon premier arrêt est la MSDN... Malheureusement, pour ce contrôle, la MSDN ne contient quasiment que les signatures des fonctions et propriétés, sans indication sur le comportement du contrôle en fonction de ses paramètres...

On va s'intéresser aux paramètres suivants :

- AfterCallbackClientScript : permet de spécifier le script à appeler après une tentative de résolution
- AllowEmpty : permet de spécifier si le champ peut être vide
- AllowTypeIn : si vrai, autorise l'utilisateur à taper du texte dans le champ
- MaximumEntities : nombre maximum d'entités résolues
- MultiSelect : définir de définir si le champ peut contenir une (false) ou plusieurs (true) personnes
- NoMatchesText : texte affiché si, après une tentative de résolution de valeur, le texte entré ne correspond à rien (ou plutôt, à personne)
- SelectionSet : définit le groupe dans lequel la recherche va s'effectuer. Peut prendre les valeurs **User**(Utilisateur), **DL**(liste de distribution), **SecGroup**(groupe AD), ou **SPGroup**(groupe Sharepoint)
- ValidatorEnabled : permet spécifier si un message d'erreur doit être affiché en cas de contenu invalide

Ajout d'un champ de type PeopleEditor

On va modifier notre page de façon à ajouter un champ utilisateur. En premier lieu, on va ajouter à notre contrôle utilisateur une référence à la bibliothèque de contrôles de SharePoint.

```
<%@ Register TagPrefix="wssawc" Namespace="Microsoft.SharePoint.WebControls"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
```

Puis, on va ajouter notre PeopleEditor, avec un jeu de propriétés minimal (une seule sélection, de type utilisateur).

```
<wssawc:PeopleEditor MultiSelect="false" ID="tbUser" SelectionSet="User" runat="server" />
```

Côté client, c'est tout ce qu'il y a à faire...Par contre, côté serveur, il va falloir utiliser le mécanisme de résolution de login du PeoplePicker pour pouvoir afficher notre utilisateur.

```
SPUser user = web.SiteUsers.GetByID(userId);
PickerEntity pe = new PickerEntity();
pe.Key = user.LoginName;

ArrayList entityArrayList = new ArrayList();
entityArrayList.Add(pe);

tbUser.ResolvedEntities.Clear();
tbUser.UpdateEntities(entityArrayList);
tbUser.Validate();
```

En effet, le seul moyen d'effectuer une résolution d'une entité est de la passer au PeoplePicker à l'intérieur d'une ArrayList d'entités. A noter, il n'est pas nécessaire de passer la propriété loginname, il est possible de passer un mail ou le nom et prénom de l'utilisateur.

Après ces modifications, notre contrôle ressemble à ça :

Users : PHILIPPE VIALATTE

User : PHILIPPE VIALATTE

Name : PHILIPPE VIALATTE

Login : \pvialatte

Mail :

Notes :

IV - Sauvegarde / modification d'un utilisateur existant

Maintenant que notre contrôle est en place, on va ajouter, côté ascx, un bouton **btnSave**, auquel on va affecter la fonction **btnSave_OnClick** (ici, je vous laisse imaginer). Pour pouvoir sauvegarder notre contrôle, on va devoir effectuer les actions suivantes :

- Tester si l'utilisateur existe déjà
- S'il existe déjà, récupérer l'utilisateur, et mettre à jour son nom, ses notes et son email. Son login ne peut pas être modifié. En effet, c'est la propriété qui permet de relier un compte Windows à un compte SharePoint. Cette propriété est donc en lecture seule.
- S'il n'existe pas, insérer le nouvel utilisateur dans la liste des utilisateurs de SharePoint.

Le code appelé par notre bouton Save sera le suivant :

```
protected void btnSave_Click(object sender, EventArgs e) {  
  
    bool unsafeUpdateAllowed = web.AllowUnsafeUpdates;  
    web.AllowUnsafeUpdates = true;  
  
    if (string.IsNullOrEmpty(dropUsers.SelectedValue)) {  
        PickerEntity objEntity = (PickerEntity)tbUser.ResolvedEntities[0];  
        String loginName = objEntity.Key;  
        String name = objEntity.DisplayText;  
        String notes = tbNotes.Text;  
        String email = tbMail.Text;  
        web.SiteUsers.Add(loginName, email, name, notes);  
    } else {  
        SPUser user = web.SiteUsers.GetByID(Convert.ToInt32(dropUsers.SelectedValue));  
        user.Email = tbMail.Text;  
        user.Notes = tbNotes.Text;  
        user.Update();  
    }  
  
    web.AllowUnsafeUpdates = unsafeUpdateAllowed;  
}
```

Maintenant, on va disséquer ces quelques lignes...pour voir ce qu'on a d'intéressant...

```
bool unsafeUpdateAllowed = web.AllowUnsafeUpdates;  
web.AllowUnsafeUpdates = true;  
...  
...  
web.AllowUnsafeUpdates = unsafeUpdateAllowed;
```

Ces lignes vont nous permettre de faire la mise à jour ou l'insertion. En effet, au niveau du modèle de sécurité de SharePoint, il n'est pas possible, par défaut, de faire une mise à jour pendant une requête GET, ce qui est notre cas actuellement.

```
PickerEntity objEntity = (PickerEntity)tbUser.ResolvedEntities[0];
```

Pour récupérer des données depuis notre PeopleEditor, il est important de se rappeler qu'un PeopleEditor affiche en fait une liste d'entités. On va, dans notre cas, récupérer uniquement la première de ces entités, mais dans le cas où on voudrait gérer plusieurs utilisateurs, on devrait traiter indépendamment chaque utilisateur.

V - Fonctionalites "avancées" du PeopleEditor

Les quelques lignes de code vues précédemment permettent assez facilement de gérer finement les utilisateurs de SharePoint... Maintenant qu'on a rempli notre contrat d'origine, on va pouvoir s'amuser un peu avec le contrôle.

Récupération de données supplémentaires

Après ces petits exemples, la question qui doit vous brûler les lèvres, c'est "mais comment SharePoint fait-il pour récupérer tout seul comme un grand le mail et le nom des utilisateurs que j'ajoute ?".

En fait, SharePoint utilise bien le même PeopleEditor pour gérer les utilisateurs. Lorsque l'on fait une résolution d'un utilisateur sur AD, SharePoint stocke dans des champs cachés le login de l'utilisateur (sous Firefox), voire toutes les informations disponibles sur l'utilisateur (pour IE).

Ces informations sont ensuite transmises dans le code-behind, dans le champ EntityData des entités résolues, et peuvent ensuite être manipulées directement.

Name	Value
tbUser.ResolvedEntities[0]	{Microsoft.SharePoint.WebControls.PickerEntity}
Description	
DisplayText	"PHILIPPE VIALATTE"
EntityData	Count = 5
["DisplayName"]	"PHILIPPE VIALATTE"
["Email"]	
["SPUserID"]	"29"
["Department"]	
["PrincipalType"]	"User"
Raw View	
IsResolved	true
Key	
MultipleMatches	Count = 0

Dans le cas où on voudrait entrer les données Active Directory de l'utilisateur sélectionné, on pourrait réécrire le code ci-dessus ainsi :

```
PickerEntity objEntity = (PickerEntity)tbUser.ResolvedEntities[0];
String loginName = objEntity.Key;
String name = objEntity.DisplayText;
String notes = string.Empty;
String email = string.Empty;

if (objEntity.EntityData.ContainsKey("Department")){
    notes = "Département : " + objEntity.EntityData["Department"];
}

if (objEntity.EntityData.ContainsKey("Email")){
    email = objEntity.EntityData["Email"];
}
```

 *Rien ne vous garantit que les données de mail ou de département, soient bien remplies côté Active Directory.*

Appel de JavaScript après résolution AD

Comme on l'a vu juste avant, SharePoint récupère par défaut les utilisateurs depuis Active Directory. Comment faire si notre référence principale n'est pas Active Directory, mais, par exemple, LDAP ?

Imaginons que ce soit le cas, et que l'on veuille pré-remplir notre zone Note avec certaines informations venant de LDAP, comment pourrait-on faire ?

En fait, on va utiliser une fonctionnalité déjà existante de notre PeopleEditor, la propriété AfterCallbackClientScript, pour récupérer ces données en Ajax.

Pour cela, dans un premier temps, on va ajouter un fichier GetUserNotes qui va nous servir de bouchon (le but n'étant pas de montrer comment faire une requête sur LDAP...).

GetUserNotes.aspx

```
<%@ Page Language="C#" %>
<% if (!string.IsNullOrEmpty(Request.Params["login"])) {

    /* Normalement, ici, on ferait appel à une fonction de résolution LDAP
    Mais ici, on va juste retourner notre login...*/
    Response.Write("Test Ajax...");
} else {
    Response.Write("Pas de login fourni !!!");
}%>
```

Et, dans notre page ascx, on va ajouter le code nécessaire à l'appel Ajax. Je ne vais pas détailler la logique derrière cet appel, c'est un appel assez standard à l'objet XMLHttpRequest, assez bien documenté sur Internet.

```
function AjaxCall(fichier){

    if(window.XMLHttpRequest) // FIREFOX
        xhr_object = new XMLHttpRequest();
    else if(window.ActiveXObject) // IE
        xhr_object = new ActiveXObject("Microsoft.XMLHTTP");
    else return(false);
    xhr_object.open("GET", fichier, false);
    xhr_object.send(null);
    if(xhr_object.readyState == 4)
        return(xhr_object.responseText);
    else
        return(false);
}
```

Finalement, il ne nous reste plus qu'à ajouter un appel à GetUserNotes. Ce n'est en fait pas si facile, car le PeoplePicker ne nous donne pas de moyen simple et rapide pour retrouver le contenu des entités résolues...

Après avoir fouillé à droite à gauche dans le DOM, on s'aperçoit que la donnée que l'on cherche se trouve dans un span, lui-même dans une div, dont l'id est l'id du contrôle PeopleEditor, suffixé par _upLevelDiv.

Un moyen commun à Firefox et Internet Explorer de récupérer le login de l'utilisateur résolu est donc:

```
document.getElementById('<%= tbUser.ClientID %>_upLevelDiv').childNodes[0].getAttribute('title');
```

On va donc créer (encore !) une nouvelle fonction dans notre fichier ascx

```
function GetUserNotes(){
```

```
var login = document.getElementById('<%= tbUser.ClientID %>_upLevelDiv').childNodes[0].getAttribute('title');
var fileName = "http://" + location.host + '/GetUserNotes.aspx?login=' + login;
var notesValue = AjaxCall(fileName);
document.getElementById('<%= tbNotes.ClientID %>').value = notesValue;
}
```

Et enfin, on va modifier le code de notre PeoplePicker ainsi :

```
<wssawc:PeopleEditor MultiSelect="false" ID="tbUser" SelectionSet="User" AfterCallbackClientScript="GetUserNotes(
```

Et voilà ! Une démonstration rapide nous donne l'écran suivant :

Users :

User :  

Name :

Login :

Mail :

Notes :

VI - Remerciements

Merci à **SaumonAgile** pour sa relecture.

