Créer une Feature d'administration sous SharePoint

Comment créer un composant d'administration

Dans les précédents articles, nous avons évoqué la création de pages personnalisées dans SharePoint, la création de master pages ou de modèles de site.

Une des grandes avancées de SharePoint 2007 (MOSS ou WSS) dans le développement est le déploiement et l'activation au travers de Features. Nous verrons donc dans cet article comment créer une Feature pour l'administration.

Introduction

On trouve de nombreux articles vous expliquant comment développer des modules sous SharePoint. Ces développements peuvent souvent paraître impossibles à attaquer pour tous les développeurs débutant.

Pourtant, il n'en est rien, même si les méthodes de déploiement peuvent paraître un peu rudes, les bons outils associés avec une méthodologie efficace peuvent vous permettre de créer vos propres pages.

Présentation

Nous prendrons un exemple de page d'administration, car on ne se trouve pas dans un cas nécessitant l'optimisation des performances (pas d'exigence de code pré-compilé par exemple). Notre exemple sera donc une page ASPX avec le code inline (script C#).

Nous prendrons le cas d'un composant SharePoint (Feature) que j'ai développé pour des besoins d'uniformisation de charte graphique par l'application d'un thème. Ce composant est disponible sur le site CodePlex :

- **CodePlex - SharePoint 2007 Features - Reset Theme**

Nous verrons rapidement les différentes parties de la page ASPX, puis la création des fichiers de définition du composant et enfin la création du package de solution.

Fichier ASPX d'administration

Notre composant est une page d'administration qui se trouvera au niveau de l'administration de la collection de site (site racine de chaque collection) et qui permettra d'appliquer à tous les sites et sous-sites de cette collection le thème choisi ainsi que le logo et le texte alternatif de ce logo.

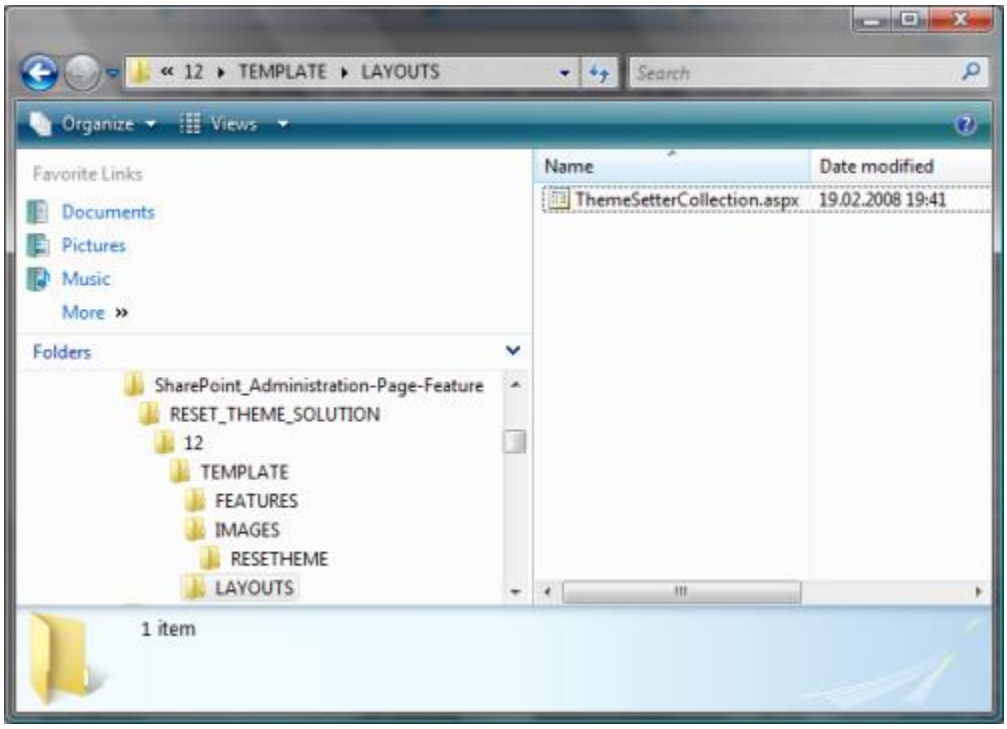
Cette page ne nécessitera pas de compilation et peut donc être édité directement dans Notepad, nous utiliserons Visual Studio afin de bénéficier de la coloration syntaxique.

Cette page pourra être utilisée par tous les responsables de collection de sites (Content Manager) et donc sera stockée dans le répertoire :

- [12]\TEMPLATE\LAYOUTS\ThemeSetterCollection.aspx

Préparation du projet

Nous devons donc commencer par reproduire la structure du 12 dans le répertoire de notre projet.



Pour des raisons de simplification, on ne créera pas de solution Visual Studio pour ne pas mélanger les deux approches.

- Une solution SharePoint n'exige aucunement une solution Visual Studio .NET si on n'utilise pas de code Behind

Déclaration des références

Notre page ASPX contient un ensemble de références nécessaires pour l'utilisation des composants SharePoint (Master Page, DLL, objets graphiques, ...)

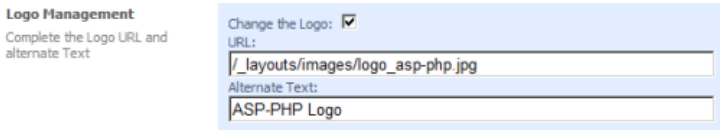
```
<%@ Assembly
   Name="Microsoft.SharePoint.ApplicationPages, Version=12.0.0.0,
     Culture=neutral, PublicKeyToken=71e9bce111e9429c"%>
<%@ Page
   Language="C#"
   MasterPageFile="~/_layouts/application.master"
   ValidateRequest="False" %>
<%@ Import
   Namespace="Microsoft.SharePoint.ApplicationPages" %>
<%@ Import
   Namespace="Microsoft.SharePoint" %>
<%@ Import
   Namespace="System.Xml" %>
<%@ Register
   Tagprefix="SharePoint"
   Namespace="Microsoft.SharePoint.WebControls"
   Assembly="Microsoft.SharePoint, Version=11.0.0.0,
    Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register
   Tagprefix="Utilities"
   Namespace="Microsoft.SharePoint.Utilities"
   Assembly="Microsoft.SharePoint, Version=11.0.0.0,
    Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register
   Tagprefix="SharePoint"
   Namespace="Microsoft.SharePoint.WebControls"
   Assembly="Microsoft.SharePoint, Version=12.0.0.0,
    Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register
   TagPrefix="wssuc"
   TagName="ButtonSection"
   src="~/_controltemplates/ButtonSection.ascx" %>
```

Ces définitions nous permettent ensuite d'utiliser les composants graphiques SharePoint comme les boutons de validation :

```
<wssuc:ButtonSection runat="server">
   <Template_Buttons>
      <input type="submit"
            class="ms-ButtonHeightWidth"
            value="Apply Theme"
            id="Submit1"
            name="Submit1"
            ACCESSKEY="S"
            OnServerClick="Submit1_Click"
            runat="server"/>
   </Template_Buttons>
</wssuc:ButtonSection>
```

Mais aussi les blocs de formulaires SharePoint :

```
<wssuc:InputFormSection
   Title="Result"
   Description="Result for the changes"
   runat="server">
      <Template_InputFormControls>
         Change the Logo:  <asp:CheckBox
               ID="CheckBoxChangeLogo"
               runat="server" /><br />
         URL: <asp:TextBox
               ID="TextBoxLogoUrl"
               runat="server"
               Width="400"></asp:TextBox><br /
>
         Alternate Text: <asp:TextBox
               ID="TextBoxLogoDescription"
               runat="server"
               Width="400"></asp:TextBox>
      </Template_InputFormControls>
</wssuc:InputFormSection>
```



Code applicatif

Notre page va donc charger dans une liste déroulante les différents thèmes disponibles pour la langue active dans le site racine de la collection, ainsi que l'URL pour l'image logo de ce site et le texte alterné.

Ce chargement se fait par la fonction **OnLoad** :

```
protected override void OnLoad(EventArgs e)
{
      if (!this.IsPostBack)
      {
         try
         {
            using (SPWeb myNewWeb = SPControl.GetContextWeb(Context))
            {
               this.TextBoxLogoDescription.Text = myNewWeb.SiteLogoDescription.ToString();
               this.TextBoxLogoUrl.Text = myNewWeb.SiteLogoUrl;
               LoadAllThemesInDDL((int)myNewWeb.Language, myNewWeb.Theme);

               this.HyperLinkSiteSettings.Text = "Return to the Site Settings";
               this.HyperLinkSiteSettings.NavigateUrl = myNewWeb.Url + "/_layouts/settings.aspx";
            }
         }
         catch (Exception Myexception)
         {
            this.LiteralResult.Text += "Error: " + Myexception.ToString();
         }
      }
}
```

Et la fonction **LoadAllThemesInDDL** :

```
private void LoadAllThemesInDDL(int WebLCID, string CurrentTheme)
{
   string content;
   try
   {
      this.DropDownListThemes.Items.Clear();
      string filePath = Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles)
          + @"\Microsoft Shared\web server extensions\12\TEMPLATE\LAYOUTS\"
          + WebLCID
          + @"\SPTHEMES.XML";
      if (!System.IO.File.Exists(filePath))
      {
         this.LiteralResult.Text += "Unable to find SPTHEMES.XML at " + filePath;
      }
      using (System.IO.TextReader tr = new System.IO.StreamReader(filePath))
      {
         content = tr.ReadToEnd();
      }
      XmlDocument xDoc = new XmlDocument();
      xDoc.LoadXml(content);
      int count = 0;
      foreach (XmlNode xNode in xDoc.DocumentElement.ChildNodes)
      {
         string themeID = xNode["TemplateID"].InnerXml;
         this.DropDownListThemes.Items.Add(new ListItem(themeID.ToUpper(), themeID));
      }
      this.DropDownListThemes.Items.FindByText(CurrentTheme.ToUpper()).Selected = true;
   }
   catch (Exception ex)
   {
      this.LiteralResult.Text += "<BR>Error: " + ex.Message;
   }
}
```

Lorsque le Content Manager a choisi son thème et renseigné les champs du formulaire rapide, la validation provoque l'application de ces modifications sur tous les sites et sous-sites.

Ceci s'effectue par la fonction **Submit1_Click** :

```
private void Submit1_Click(object sender, EventArgs e)
{
   string DefinedTheme = DropDownListThemes.SelectedValue;
   using (SPWeb myNewWeb = SPControl.GetContextWeb(Context))
  {
      this.LiteralResult.Text += "Selected Theme: " + DefinedTheme +"<BR>";
      updateThemeWeb(myNewWeb, DefinedTheme);
   }
}
```

Ainsi que la fonction **UpdateTheme** :

```
private void updateThemeWeb(SPWeb web, string ThemeName)
{
       this.LiteralResult.Text += "URL: " + web.Url + "<BR>";
       this.LiteralResult.Text += "Choosed theme: " + ThemeName + "<BR>";
       try
       {
          web.AllowUnsafeUpdates = true;
          if (this.CheckBoxChangeLogo.Checked == true)
          {
             web.SiteLogoUrl = this.TextBoxLogoUrl.Text.Trim();
             web.SiteLogoDescription = this.TextBoxLogoDescription.Text.Trim();
             this.LiteralResult.Text += "Logo: <B>OK</B><BR>";
          }
          web.ApplyTheme(ThemeName);
          web.Update();
          this.LiteralResult.Text += "Theme: <B>OK</B><BR>";
       }
       catch (Exception MyException)
       {
          this.LiteralResult.Text += "Error: " + MyException.ToString() + "<BR>";
       }
     foreach (SPWeb subweb in web.GetSubwebsForCurrentUser())
     {
       updateThemeWeb(subweb, ThemeName);
     }
     web.Dispose();
}
```

On voit donc bien que ce code reste vraiment très simple :

- Quatre fonctions ultra basiques en C#
- Pas de code behind
- Couche graphique gérée par SharePoint et sa Master Page
- Utilisation des outils graphiques de SharePoint
- ...

Nous pouvons donc maintenant regarder comment créer les fichiers de déclaration du composant au niveau de SharePoint.

Définition de la Feature

Une Feature (ou composant) SharePoint peut être quasiment n'importe quoi, on trouve classiquement :

- Des pages applicatives
- Les WebParts
- Les WorkFlows
- Des WebServices additionnels
- ...

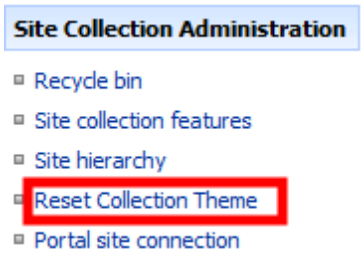
Nous sommes dans le premier cas pour cet article, nous allons donc créer des fichiers techniques rendant notre page d'administration disponible au responsable de contenu suivant son besoin.

Ainsi, un des principes classiques des Features reste la notion d'activation et de désactivation, ainsi l'activation va pouvoir dans certains cas copier des fichiers directement dans des répertoires voulus (suivant son besoin).

Une seconde notion primordiale est la notion des niveaux dans la disponibilité (ou de Scope) des composants. Ainsi certains composants vont pouvoir être activé/désactivé au niveau du site ou sous-site, d'autres vont être activable au niveau des collections ou directement au niveau de la ferme.

Notre exemple s'activera au niveau de la collection et ajoutera un lien dans le menu d'administration des collections.

Ceci parce que la page ASPX permet de modifier le thème de tous les sous-sites d'une collection, il est donc logique de donner cette option pour l'administrateur de la collection et le lien visible sur le site racine (comme dans la capture suivante).



Création du répertoire de la Feature

On trouve l'ensemble des features (ou composants) installés sur sa ferme SharePoint dans le répertoire :

- [12]\TEMPLATE\FEATURES\

Nous devons créer un répertoire dans ce répertoire afin que notre composant soit pris en compte. Ce répertoire aura le nom "ResetAllSiteCollectionTheme" dans notre exemple.

Nous pouvons dès lors ajouter les fichiers XML de définition dans ce répertoire.

Fichiers de déclaration de la Feature

La Feature sera dans notre cas composée de deux fichiers XML que nous allons décrire.

Fichier de définition du lien

Nous créons dans un premier temps le fichier XML pour définir ce lien dans la page d'administration de la collection de sites.

Ce fichier est bien sur un fichier XML que l'on peut nommer comme on le souhaite, dans notre cas ce sera "ResetCollectionTheme.xml". On doit alors y ajouter un ensemble de données permettant de déclarer le lien de notre composant et l'action qui y est associée.

Cela se transforme dans ce fichier XML par le bloc "CustomAction" qui est associé à un bloc "UrlAction" fournissant l'URL :

```
<?xml version="1.0" encoding="us-ascii"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
   <CustomAction Id="MPCHOOSEREXT"
      GroupId="SiteCollectionAdmin"
      Location="Microsoft.SharePoint.SiteSettings"
      RequireSiteAdministrator="TRUE"
      Title="Reset Collection Theme"
      Description="This page change all the sites for a collection to use selected theme"
      Sequence="50">
   <UrlAction Url="_layouts/ThemeSetterCollection.aspx"/>
   </CustomAction>
</Elements>
```

L'ensemble des paramètres possibles pour ce fichier de déclaration est fourni sur le site de la MSDN :

- **Custom Action definitions**

A ce stade, la déclaration de notre fichier ASPX est prête, mais il nous faut dire à SharePoint qu'un nouveau composant vient d'être ajouté à notre ferme SharePoint.

Fichier de définition du composant

Afin que SharePoint sache qu'un nouveau composant a été déployé sur le serveur, il faut que dans le répertoire de ce composant se trouve un fichier XML spécifique :

- **FEATURE.XML**

On trouve la documentation de ce type de fichier sur le site de la MSDN

- **Feature.xml Files**

Cela commence comme toujours dans le développement SharePoint par la définition du GUID (paramètre **Id**) pour ce composant afin que celui-ci soit bien unique sur la ferme. Il peut être pratique d'utiliser l'outil fourni dans Visual Studio .NET, vous trouverez d'ailleurs l'astuce pour récupérer l'outil dans Visual Studio 2008 (ou les solutions en commentaire) :

- **SharePoint : Comment retrouver le Create GUID dans Visual Studio 2008**

On trouve ensuite les paramètres standards dans toute définition de composant :

- Le nom : **Title**
- La description : **Description**
- La version : **Version**
- L'auteur : **Creator**
- L'URL de l'image icone dans le menu (relatif par rapport au répertoire images) : **ImageUrl**
- La visibilité du composant dans les composants activables : **Hidden**

On trouve ensuite le paramètre de définition du niveau de visibilité (**Scope**) qui permet de dire à SharePoint à quel niveau sera activé le composant :

- **Farm** : activation au niveau de la ferme par la centrale admin
- **Site** : activation au niveau de chaque collection de sites de façon indépendante
- **Web** : activation au niveau de chaque site ou sous-site de façon indépendante

On définit ensuite la visibilité du composant (la possibilité de fournir la gestion à l'utilisateur). Dans le cas d'un composant masqué (comme tous les composants natifs), l'activation ou la désactivation se fait avec STSADM.

On peut aussi fournir le fichier de resources (localisation du composant pour les textes et URL) par défaut.

Une fois les informations fournies pour le composant, on ajoute la liste des éléments de ce composant. Pour notre cas d'exemple, nous donnons le fichier XML du lien personnalisé "ResetCollectionTheme.xml".

Le résultat est le suivant :

```
<?xml version="1.0" encoding="us-ascii"?>
<Feature
  Id="B3A8CBB5-855E-40f7-8BC9-B0A09AF18D9B"
  Title="Reset theme on a Collection"
  Description="Reset the Site Collection with the selected theme"
  Version="1.0.0.0"
  Scope="Site"
  Creator="Fabrice Romelard [MVP]"
  Hidden="false"
  xmlns="http://schemas.microsoft.com/sharepoint/"
  ImageUrl="RESETHEME/myfeatureicon.gif"
  DefaultResourceFile="_Res">
    <ElementManifests>
      <ElementManifest Location="ResetCollectionTheme.xml" />
    </ElementManifests>
</Feature>
```

Notre composant SharePoint est maintenant prêt à être installé dans notre ferme SharePoint.

Installation de la Feature

Lorsque l'on copie les différents fichiers manuellement dans les répertoires adéquats, il faut encore signaler au moteur SharePoint qu'un nouveau composant est utilisable.

Ceci s'effectue avec la commande STSADM.EXE, qui se trouve dans le sous-répertoire :

- C:\Program Files\Common Files\microsoft shared\Web Server Extensions\12\BIN\

La commande adaptée pour notre besoin est "installfeature" à laquelle on ajoute le chemin d'accès au fichier "feature.xml" (en relatif depuis le sous-répertoire FEATURES), cela donne pour notre cas :

- **stsadm.exe -o installfeature -filename ResetAllSiteCollectionTheme\feature.xml**

Si le fichier de définition du composant est "feature.xml", on peut directement utiliser :

- **stsadm.exe -o installfeature -name ResetAllSiteCollectionTheme**

Enfin, on peut ajouter le "-force" à cette commande afin de forcer l'installation de ce composant (surtout si on a modifié quelque chose dans les fichiers XML).

Voyons maintenant comment activer et utiliser notre composant.

Activation ou désactivation du composant

L'activation ou la désactivation d'un composant peuvent se faire de trois manières :

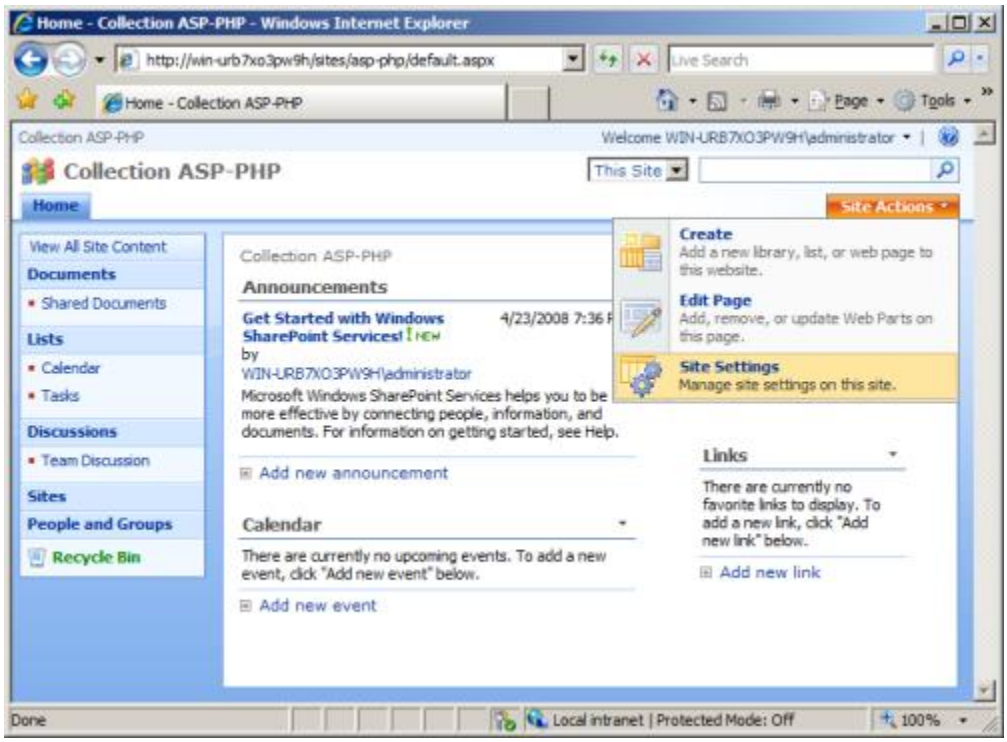
- **Par le site SharePoint**
- **Par la commande STSADM**
- **Par le modèle Objet (Code C#)**

Quoi qu'il en soit, l'idée est toujours la même, on doit activer le composant au niveau du site racine d'une collection de sites.

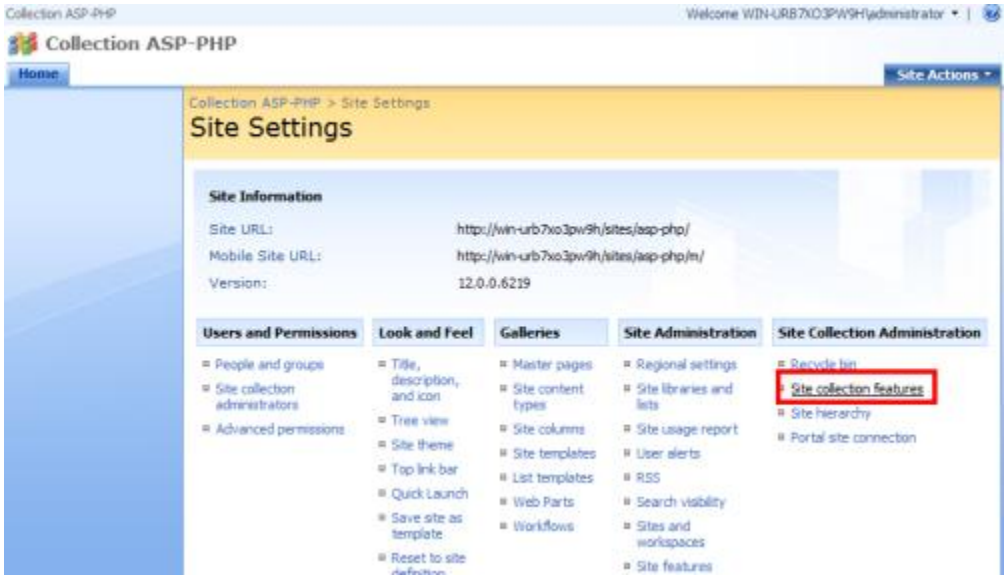
Voyons chacun de ces cas.

Activation et désactivation depuis le site SharePoint

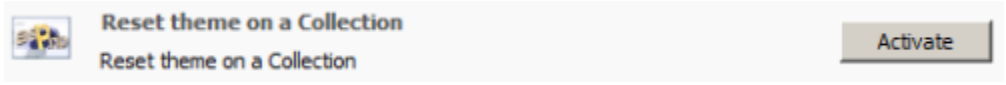
Pour utiliser le site SharePoint, il suffit de se placer en tant qu'administrateur (de la collection) sur le site racine de cette collection, puis d'aller dans les "Sites Settings".



Puis de cliquer sur "Site Collection features" dans la rubrique "Site Collection Administration"



On trouve alors un ensemble de composant dont le notre, il nous faut alors cliquer sur "Activate"



La procédure inverse permet de désactiver un composant, en cliquant sur "Deactivate".

Activation et désactivation depuis la commande STSADM

La commande est similaire à celle d'installation, mais on ajoute l'URL du site ou on souhaite activer le composant, ce qui donne :

- **stsadm.exe -o activatefeature -name ResetAllSiteCollectionTheme -url "http://SharePointFarm/SiteCollection/"**

On peut aussi utiliser l'option "-force" pour forcer la mise à jour du composant. Il est aussi possible de passer par le GUID (plutôt que le nom du composant) avec le paramètre "-id".
La désactivation s'effectue quasiment de la même façon :

- **stsadm.exe -o deactivatefeature -name ResetAllSiteCollectionTheme -url "http://SharePointFarm/SiteCollection/"**

Les options "-force" et "-id" sont aussi utilisables.

Activation et désactivation depuis le modèle Objet (Code C#)

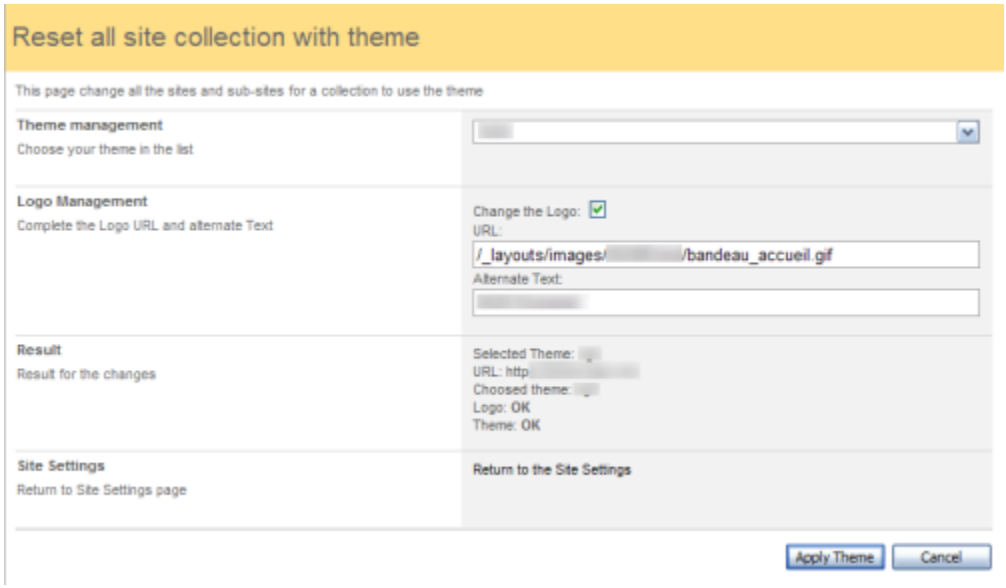
Dans un développement SharePoint, il faut connaître le GUID du composant, et charger ce GUID dans une collection qu'on aura chargé en mémoire.

```
// Pour activer le composant sur la collection http://SharePointFarm/SiteCollection/
using (SPWeb web = new SPSite("http://SharePointFarm/SiteCollection/").OpenWeb())
{
   web.Features.Add("B3A8CBB5-855E-40f7-8BC9-B0A09AF18D9B");
}

// ------------------------------------------------
// Pour désactiver le composant sur la collection http://SharePointFarm/SiteCollection/
using (SPWeb web = new SPSite("http://SharePointFarm/SiteCollection/").OpenWeb())
{
   web.Features.Remove("B3A8CBB5-855E-40f7-8BC9-B0A09AF18D9B");
}
```

Utilisation du composant

L'utilisation du composant est très simple puisqu'il suffit de cliquer sur le lien ajouté, puis de choisir son thème et l'url du logo (ainsi que le texte alternatif). On clique alors sur "Apply Theme" et la mise à jour s'effectue en affichant un retour de ces mises à jour.



Conclusion

Nous avons vu dans cet article comment créer une page ASPX plus avancée que **celle du premier article, en ajoutant du code .NET**. Nous avons ensuite ajouté cette page dans un composant (Feature) SharePoint avec la mise à disposition de l'activation et désactivation par le responsable de la collection de site.

Vous pouvez télécharger le ZIP contenant tous les fichiers présentés dans cet article :

- **Composant d'administration - Reset Theme Feature**

Nous verrons dans un prochain article comment créer cette solution SharePoint afin de le déployer sur une ferme SharePoint le plus simplement possible.

Voici quelques liens utiles si cet article vous a intéressé :

- **Les features avec Sharepoint (FR)**
- **CodePlex - SharePoint 2007 Features - Reset Theme**
- **Custom Action definitions**
- **Feature Schemas**
- **Activate features through code**
- **Activating and Deactivating Features Programmatically**
- **Sharepoint 2007 : Les Application Pages (FR)**
- **Créer une page ASPX personnalisée pour WSS (FR)**
- **Modifier la Master Page de SharePoint (FR)**
- **Créer un Site Template pour WSS V3 (FR)**
- **Installation de WSS V3 (FR)**
- **Office Online**
- **Club SPS MOSS FRANCE(FR)**

En vous souhaitant de bons projets de développement.

Romelard Fabrice (alias F___)
Intranet/Extranet CTO - **SGS**