



**Formation C# – Delphi.NET – Delphi Win32
Développement & Sous-traitance**

© Copyright 2006 Olivier DAHAN

**Reproduction, utilisation et diffusion interdites sans l'autorisation de
l'auteur. Pour plus d'information contacter odahan@e-naxos.com**

La gestion des dialogues modaux (ou non) sous ASP.NET

ASP.NET est une révolution, et ceux qui pratiquent cet environnement sont d'accord sur une chose : c'est un ensemble bien pensé et bien conçu.

... Hélas rien n'est parfait...

Alors que les dialogues sont indispensables à toute programmation un peu sérieuse (confirmations diverses, saisies d'information modales, etc), cet aspect des choses a été totalement zappé par Microsoft, même dans la seconde version de ASP.NET !

Incroyable ? oui !

Il faut donc s'en remettre à des « trucs et astuces » plus ou moins fiables, plus ou moins bien expliqués, à des exemples écrits dans des langages exotiques que certains ont parfois du mal à traduire.

Bref c'est la galère et tout le monde est d'accord aussi sur ce point...

Comme tout le monde, donc, l'auteur de ces lignes a « galéré » pour régler ce fichu problème des dialogues sous ASP.NET. A force de tests et de glanage d'informations de qualité diverse sur le Web il est arrivé à des solutions qui fonctionnent correctement et qui restent simples à mettre en

œuvre (même si l'exemple final devient un peu plus complexe il faut l'avouer).

L'auteur ne clame pas la paternité du code qui sera démontré ici, tout juste sa recherche, sa collecte, sa traduction en C#, son adaptation, sa correction, le mariage de solutions éparées, ses tests, et l'écriture de cet article. Ce n'est déjà finalement pas si mal !

Etape 1 : afficher un message simple

C'est la chose la plus simple à réaliser mais elle met en œuvre une astuce que nous allons réutiliser tout au long de cet article.

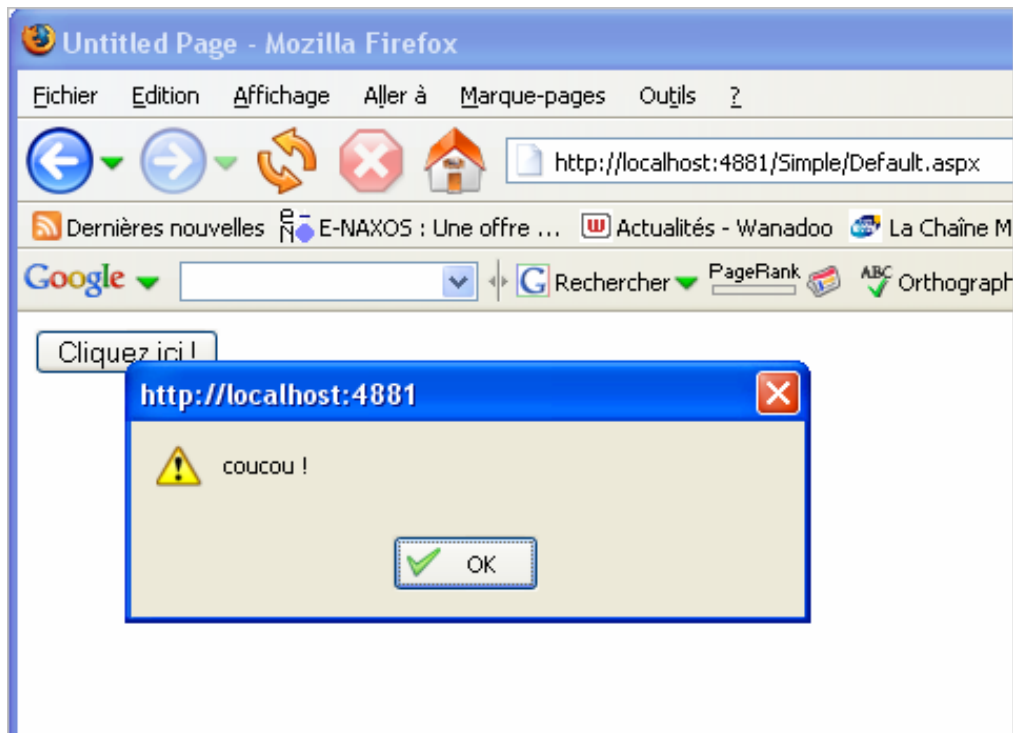
La solution consiste à enregistrer du JavaScript directement dans le contrôle de la façon suivante :

```
protected void Page_Load(object sender, EventArgs e)
{
    if(!Page.IsPostBack)
        btnDialog.Attributes.Add("onclick", "alert('coucou !');");
}
```

Vous noterez ici plusieurs choses :

- Le bouton que nous « modifions » est btnDialog
- L'initialisation de l'attribut s'effectue dans le Page_Load de la page
- Nous testons IsPostBack pour éviter de refaire l'initialisation à chaque aller-retour entre le client et le serveur
- Le code JS est ajouté en utilisant les attributs du bouton.

La capture écran suivante montre l'effet de ce code.



Etape 2 : Demander une confirmation

On procède de la même façon mais on exploite le résultat de la fonction « confirm » pour stopper la gestion de l'évènement ou non. En effet, le « onclick » JavaScript ajouté dans le Page_Load sera exécuté avant le OnClick programmé dans ASP.NET sans remplacer ce dernier. On cumule les deux événements en utilisant le premier pour, en quelque sorte, couper l'herbe sous les pieds du second en cas de réponse négative de l'utilisateur.

```
protected void Page_Load(object sender, EventArgs e)
{
    // bouton exemple 2
    if (!Page.IsPostBack)
    {
        btnSuppress.Attributes.Add("onclick",
            "if(confirm('Voulez-vous réellement supprimer cet enregistrement ?')){}else{return false}");
    }
}
```

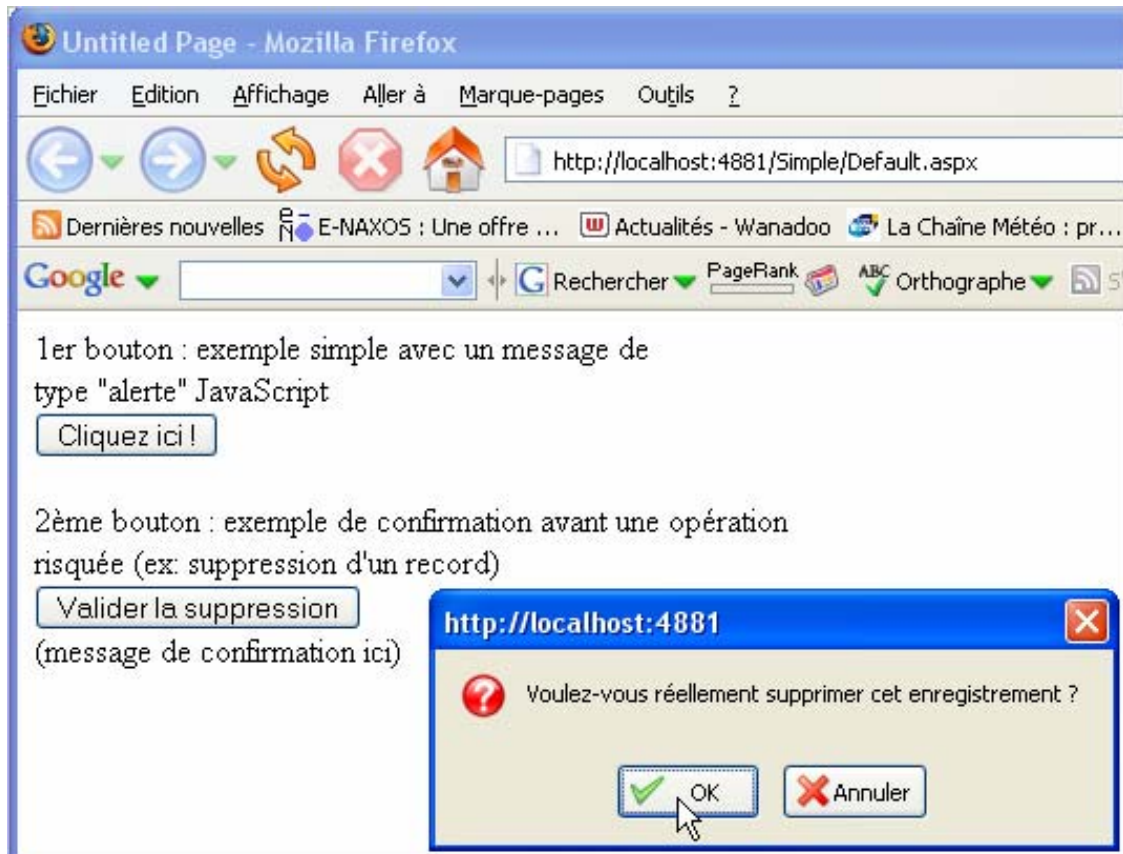
Le code ASP.NET du bouton « btnSuppress » est le suivant :

```
protected void btnSuppress_Click(object sender, EventArgs e)
{
    // ici code applicatif du bouton "suppress"
}
```

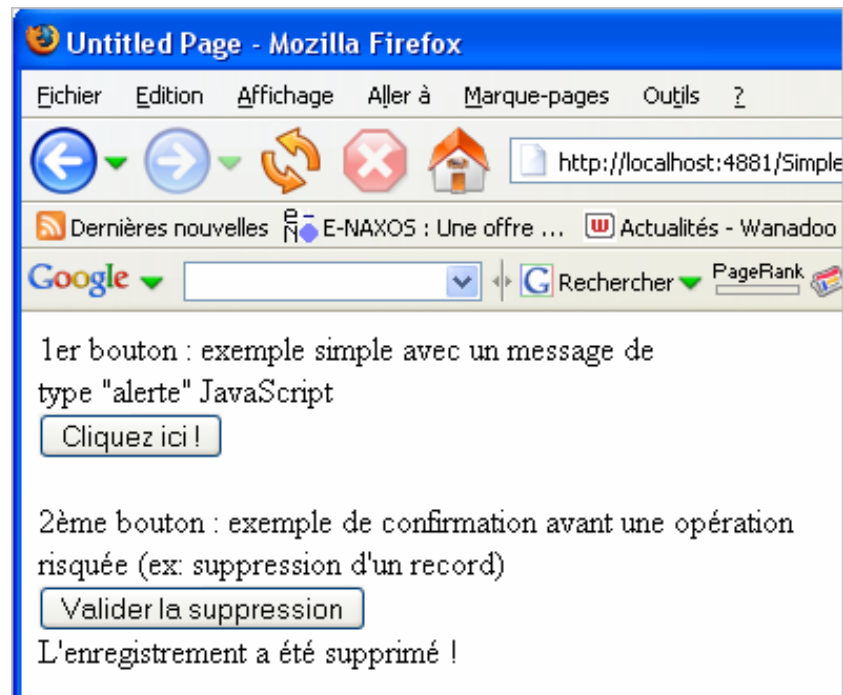
```
lConfirmation.Text = "Enregistrement supprimé !";
}
```

On notera que c'est dans ce code que le travail effectif du bouton sera réalisé (ou appelé). Dans cet exemple nous avons matérialisé le travail effectué par le changement du contenu d'un label.

Le dialogue avec confirmation :



L'effet de la validation est visible dans cette autre capture écran :



Le label a bien été modifié, ce qui prouve à la fois que le procédé fonctionne (ce qui est rassurant !) mais surtout qu'on peut bien cumuler le « onclick » JavaScript avec le « OnClick » ASP.NET et qu'ils sont traités dans l'ordre JavaScript / ASP.NET ce qui autorise l'effet démontré ici.

Etape 3.a : utiliser une « vraie » page ASP.NET comme dialogue

Les alertes et les confirmations de JavaScript sont très pratiques mais il faut bien l'avouer c'est un peu ... léger.

En effet, une application a généralement besoin d'un peu plus complexe que ça pour fonctionner. Il est alors nécessaire de pouvoir concevoir le dialogue de bout en bout pour qu'il contienne plus qu'un texte agrémenté d'un ou deux boutons de validation / annulation.

Tout naturellement on désire ainsi pouvoir utiliser une page ASP.NET comme page de dialogue et pouvoir l'appeler de la même façon qu'on a pu le faire pour les alertes ou les confirmations.

Pour notre exemple nous allons concevoir une page appelée « Dialogue.aspx » qui contiendra le dialogue. Il s'agira d'une fiche ASP.NET classique contenant du code et gérant des événements. Classique, oui, mais avec une différence : tel que nous allons l'appeler, cette fiche ne saura pas très bien retrouver ces petits lors de la gestion des

postbacks. Pour rétablir un fonctionnement normal il faut impérativement ajouter une balise particulière dans la partie <head> de la page (dans le code de la page aspx donc, et non pas dans le code behind) :

```
<head runat="server">
    <title>Simulation d'un dialogue</title>
    <base target="_self">
</head>
```

C'est bien entendu la seconde balise <base> à l'intérieur de <head> qui nous intéresse ici.

Depuis la page « mère » il est très simple d'appeler une autre page aspx, il suffit d'utiliser un `Server.Transfer` ou un `Response.Redirect` et l'affaire est jouée. Mais ce n'est bien entendu pas ce que nous souhaitons obtenir. Nous voulons ouvrir le dialogue dans une autre page sans perdre celle en cours...

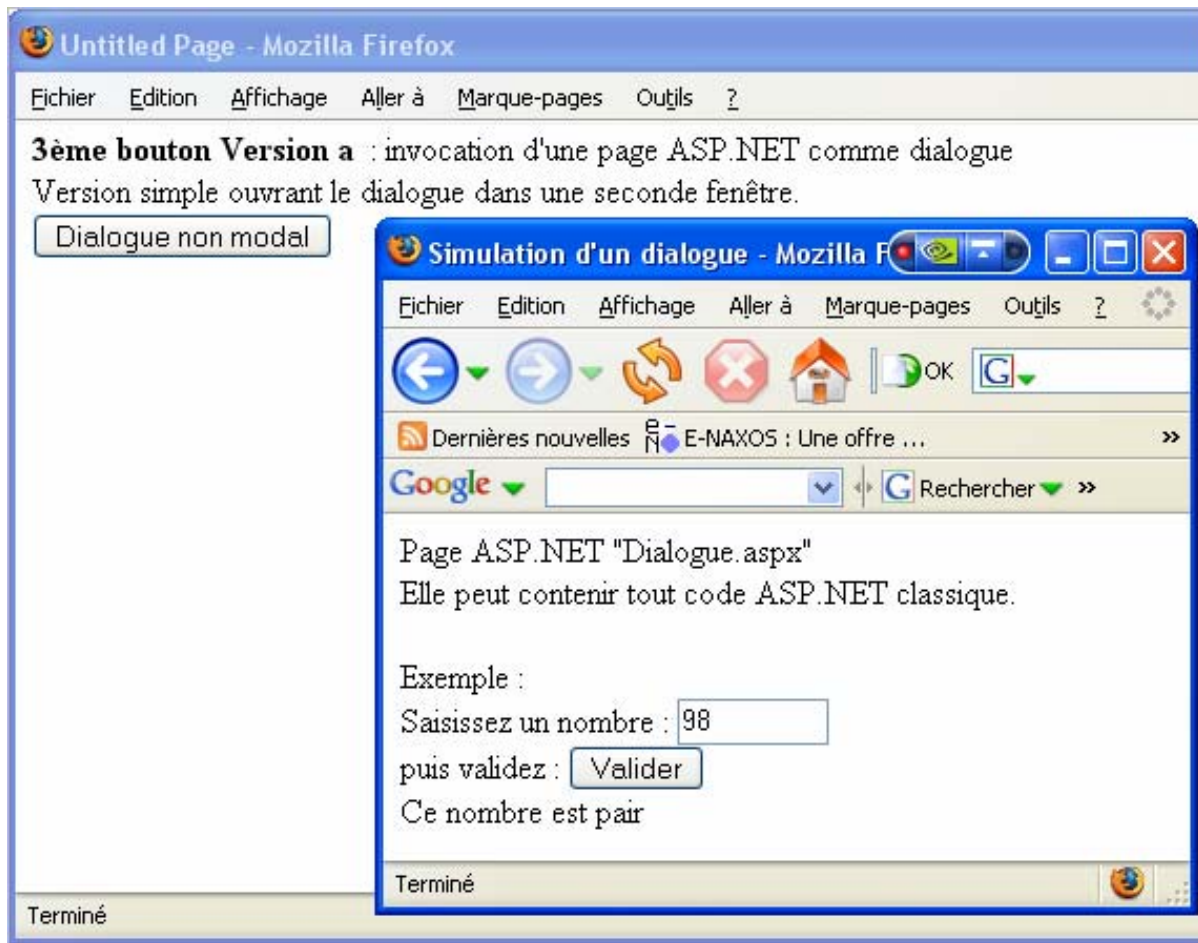
La façon la plus simple qui vient à l'esprit une fois connue les astuces exposées précédemment c'est d'ajouter au bouton de la page mère un code ressemblant à cela :

```
// bouton exemple 3.a
if (!Page.IsPostBack) btnNotModal.Attributes.Add("onclick",
    "window.open('Dialogue.aspx');");
```

Cela fonctionnera mais le résultat n'est pas terrible il faut l'avouer.

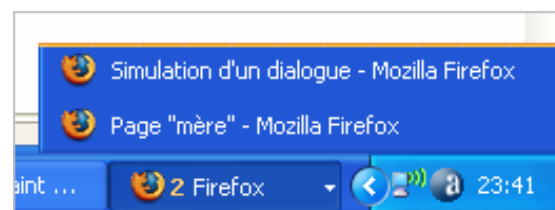
En effet cette commande va se contenter d'ouvrir une nouvelle instance de l'explorer et d'y afficher la page dialogue. La barre des tâches de Windows contiendra aussi un nouveau bouton (pour l'instance du dialogue), ce qui n'est pas très beau, mais le pire c'est que l'utilisateur pourra ignorer ce dialogue puisqu'il n'est pas bloquant, c'est-à-dire qu'il n'est pas « modal » dans le jargon du petit monde des dialogues.

Bien entendu, en paramétrant l'instruction `window.open` de JavaScript on peut fixer la taille de la fenêtre de dialogue, supprimer les barres de l'explorer etc. Dans certains cas il est utile d'ouvrir des dialogues non modaux, c'est pour cela que nous montrons cette technique même si elle reste d'un intérêt limité.



La capture écran ci-dessus montre la page de dialogue qui s'est affichée au dessus de la page « mère » et après que nous l'ayons redimensionnée « manuellement » (pour que cela soit fait automatiquement il faudrait modifier l'ordre `window.open`).

On peut voir que deux instances de l'explorer sont ouvertes dans la barre des tâches de Windows :



Etape 3.b – Ouvrir le dialogue en mode modal

Ouvrir une page ASP.NET dans une nouvelle instance du browser est une chose qui peut s'avérer utile, mais très souvent il est nécessaire de rendre le dialogue modal, c'est-à-dire bloquant.

Hélas, si `window.open` est une commande de script normalisée ECMA, les browsers ne sont pas encore mis d'accord pour gérer les dialogues modaux.

On pourra dire tout le mal qu'on veut d'Internet Explorer, il faut malgré tout se rendre à l'évidence que c'est un outil complet et pensé en accord avec les évolutions de l'informatique. Ainsi, le JavaScript de IE intègre une commande `window.showModalDialog` qui fait exactement ce que nous souhaitons, de façon simple, efficace et en mimant le comportement d'un dialogue modal tel qu'on le connaît en programmation sous Windows.

Bien entendu les browsers dérivés de Netscape, dont Firefox, même dans les versions les plus récentes, n'intègrent pas du tout ce type de comportement.

Et c'est là que les choses se compliquent... Car il ne nous est plus possible de programmer l'attribut « `onclick` » d'un bouton aussi facilement que nous l'avons fait dans les exemples précédents. Non pas que la technique que nous allons utiliser maintenant soit différente, c'est exactement la même, mais parce que nous allons être obligés d'écrire plus de code JavaScript pour détecter le browser et adapter la commande...

L'auteur a testé bon nombre de solutions trouvées ici et là, et presque aucune ne fonctionne correctement. Il a donc fallu adapter ce qui existe jusqu'à ce que cela fonctionne à la fois sous IE et Firefox.

La stratégie reste donc identique : ajouter un attribut « `onclick` » au bouton chargé d'ouvrir la fiche modale. Mais au lieu de coder directement le `window.open` à cet endroit, nous devons enregistrer dans la page ASP.NET un script déclarant une fonction ouvrant la fenêtre et appeler cette fonction depuis l'ajout de l'attribut « `onclick` ».

Il existe certainement d'autres façons de faire que celle qui va être présentée, mais celle-ci fonctionne sur les principaux browsers et c'est ce que nous désirons.

Le script peut être placé « manuellement » directement dans la section `<head>` de la page aspx, nous choisirons plutôt de l'enregistrer depuis le code behind ce qui nous semble plus souple. Mais chacun pourra faire comme il l'entend cela ne change rien au résultat.

Voici le code JavaScript utilisé tel que vous le verriez s'il était intégré dans le source de la page aspx :

```
<script>
    var winModalWindow
```



```
function IgnoreEvents(e)
{
    return false
}

function ShowWindow()
{
    if (window.showModalDialog)
    {
        window.showModalDialog("Dialogue.aspx",null,
            "dialogWidth=200px;dialogHeight=100px")
    }
    else
    {
        window.top.captureEvents
(Event.CLICK|Event.FOCUS)
        window.top.onclick=IgnoreEvents
        window.top.onfocus=HandleFocus
        winModalWindow =
        window.open ("Dialogue.aspx", "ModalChild",
            "dependent=yes,width=200,height=100")
        winModalWindow.focus()
    }
}

function HandleFocus()
{
    if (winModalWindow)
    {
        if (!winModalWindow.closed)
        {
            winModalWindow.focus()
        }
    }
    else
    {

```

```

        window.top.releaseEvents
(Event.CLICK|Event.FOCUS)
        window.top.onclick = ""
    }
}
return false
}
</script>

```

La technique, intelligente, utilisée par l'auteur de ce code consiste non pas à détecter le navigateur (ce qui ne marche pas toujours, Opera se faisant passer pour IE par exemple), mais à détecter le support de la méthode `showModalDialog`. Ce travail est effectué dans `ShowWindow()`.

Selon si la fonction est supportée ou non elle est appelée ou simulée.

La simulation est un petit peu complexe car il faut jongler avec les caprices et les différences entre les navigateurs. L'auteur a choisi de simuler le modal en mettant la page en avant plan et en détournant tous les événements de la page principale pour éviter que l'utilisateur ne puisse cliquer dessus. En réalité il peut le faire, mais cela ne fonctionne plus. C'est le rôle des gestionnaires d'événements `HandleFocus()` et `IgnoreEvent(e)` que de redonner sans cesse le focus à la page modale et de couper court à tous les événements de la page principale tant que la page modale est affichée. Le gestionnaire `HandleFocus()` s'occupe aussi rétablir le fonctionnement normal de la fiche principale dès lors que la page modale n'est plus active (ce qui est géré par le biais de la variable `winModalWindow`).

Comme indiqué plus haut nous n'allons pas placer directement le code JavaScript dans la page aspx mais nous allons l'enregistrer comme script au niveau du code behind.

Le code dans le `Page_Load` devient donc le suivant (notez l'utilisation du caractère *verbatim* @ simplifiant la frappe de la constante chaîne contenant le code JS) :

```

string script =
@"var winModalWindow

function IgnoreEvents(e)
{
    return false
}

```

```
function ShowWindow()  
{  
    if (window.showModalDialog)  
    {  
  
window.showModalDialog("Dialogue.aspx",null,"dialogWidth=300px;dialogHeight=240px;center=yes")  
return true  
    }  
    else  
    {  
        window.top.captureEvents (Event.CLICK|Event.FOCUS)  
        window.top.onclick=IgnoreEvents  
        window.top.onfocus=HandleFocus  
        winModalWindow =  
        window.open  
        ("Dialogue.aspx","ModalChild","dependent=yes,width=300,height=240")  
        winModalWindow.focus()  
    }  
}
```

```
function HandleFocus()  
{  
    if (winModalWindow)  
    {  
        if (!winModalWindow.closed)  
        {  
            winModalWindow.focus()  
        }  
        else  
        {  
            window.top.releaseEvents (Event.CLICK|Event.FOCUS)  
            window.top.onclick = ""  
        }  
    }  
    return false  
} ";
```

```
if (!Page.IsPostBack)  
{  
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(),  
        "OpenModal", script, true);  
}
```

```

    btnModal.Attributes.Add("onclick", "ShowWindow();");
}

```

Vous noterez que nous enregistrons le code JavaScript en utilisant une nouvelle fonction de ASP.NET 2.0 (en passant par `ClientScript`), les anciennes méthodes étant *deprecated* (à ne plus utiliser).

Le bouton est programmé exactement de la même façon que lors des exemples précédents, par ajout d'un attribut « `onclick` ». Sauf qu'ici au lieu que l'attribut contiennent directement le code à exécuter il contient le nom de la fonction JavaScript que nous avons enregistrée la ligne au-dessus. Le principe reste donc rigoureusement le même, en un peu plus complexe pour gérer les différences entre navigateurs, c'est tout.

Le code JavaScript utilisé ici provient de la page Web suivante :

<http://www.devguru.com/features/kb/kb100203.asp>

La documentation MS de la fonction `showModalDialog` se trouve ici :

<http://msdn.microsoft.com/workshop/author/dhtml/reference/methods/showModalDialog.asp>

La documentation de la fonction standard `window.open` se trouve ici :

<http://www.devguru.com/technologies/javascript/10894.asp>

Pour parfaire l'exemple nous avons ajouté un bouton « fermer » à la page de dialogue. En utilisant la même technique d'ajout d'attribut :

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        btnClose.Attributes.Add("onclick", "self.close();return true;");
    }
}

```

Etape 4 – Retourner des valeurs de la fiche modale vers la fiche mère

Tout cela est intéressant, mais comment récupérer des valeurs depuis la fiche modale ?

En effet, il est très fréquent, suite à l'ouverture d'une fiche modale, d'avoir à traiter les choix qui y ont été faits ou les valeurs qui y ont été saisies.

En JavaScript l'objet `window` permet de fixer la valeur de retour par le biais de sa propriété `returnValue`. Ce ne peut être qu'une seule valeur de tout type scalaire de JavaScript. C'est assez pratique mais très limité.

De plus, l'obligation de gérer un script assez complexe pour tenir compte des différences entre navigateurs rend l'ajout du support d'une valeur de retour encore plus compliqué, tout cela pour un résultat médiocre (une seule valeur retournée) ce qui nous obligerait de toute façon à implémenter une autre solution dans le cas de valeurs multiples...

L'auteur préfère ainsi vous conseiller de n'utiliser qu'une seule méthode, valable dans tous les cas : exploiter la variable `Session` pour faire transiter les informations entre la fiche fille et la fiche mère. C'est simple et puissant et autant de valeurs que nécessaire peuvent être passées dans les deux sens (de la mère vers la fille avant l'appel, de la fille vers la mère en fin de dialogue modal).

C'est cette méthode que nous allons voir maintenant.

Malheureusement nous allons rencontrer un nouveau problème qui va nous obliger à de nouvelles contorsions...

En effet, sous IE tout est parfait puisque `showModalDialog` existe et fait exactement ce pour quoi elle est faite : un dialogue modal. De fait, il est très simple d'ajouter un gestionnaire de `OnClick` dans notre page mère. Il sera exécuté après le code JavaScript (attribut « `onclick` » du bouton) et nous pourrons y faire tout ce qui doit être réalisé séquentiellement après la fermeture de la fiche modale. Cela est en tout point identique à la logique qui prévaut lors de l'écriture d'application Windows.

Mais vous devez vous douter qu'avec la simulation du mode modal que nous avons mise en place pour Firefox les choses ne sont pas aussi simples...

L'appel à `showModalDialog` sous IE est bloquant, en retournant la valeur `true` juste après dans notre fonction JavaScript `ShowWindow()` l'événement continuera sa course et le `OnClick` standard dans notre code `behind` sera exécuté. Sous Firefox (et tout autre browser de ce type) le dialogue n'est pas réellement modal, il « fait semblant » de l'être et c'est un appel à `window.open` qui est utilisé. Cet appel n'est pas modal. Une fois ouverte, la fenêtre fille vit sa vie sans aucune liaison avec la fenêtre mère. Dans cette dernière c'est plutôt lorsque le code JavaScript détecte la fermeture de la fille qu'il faudrait retourner une valeur. Mais cela ne fonctionnera pas car la mère a déjà exécuté le code, non bloquant, du bouton et n'attend plus rien... Le traitement de l'événement `onclick` est terminé depuis longtemps et rien ne sert de retourner `true` comme nous le faisons pour IE, la mère n'est pas en train d'attendre une valeur de retour qui ne sera donc jamais lu. Il en résulte que le `OnClick` côté code `behind` ne sera en réalité jamais exécuté !

Cela est réellement fâcheux...

Nous savons que pour les navigateurs ne gérant pas `showModalDialog` la simulation que nous avons mise en place utilisera le code JavaScript de la fonction `HandleFocus()`. Si la fenêtre fille est active cette fonction force le focus sur la fille pour qu'elle reste en avant-plan. Si la fille n'est plus active la fonction annule le suivi des événements qui simule le mode modal.

C'est donc à la suite de cette séquence que nous aimerions déclencher le `OnClick` en code behind.

Comment réaliser ce tour de force au milieu de notre script ?

En surfant (encore et encore...) et en essayant pas mal de bouts de solutions l'auteur a finalement retenu une technique assez simple. Elle est décrite dans l'article suivant :

<http://weblogs.asp.net/mnolton/archive/2003/06/04/8260.aspx>

Dans cette approche nous utilisons la fonction `GetPostBackEventReference` du framework. L'auteur vous laisse le plaisir de vous plonger dans la documentation de cette fonction, sa raison d'être et ses multiples utilisations. Ici nous nous contenterons de l'utiliser pour détourner à notre profit le mécanisme interne de ASP.NET servant à la gestion des postbacks.

Ce qui est nécessaire de faire :

- a) il nous faut obtenir une référence par `GetPostBackEventReference` sur un événement que nous allons créer pour notre usage.
- b) Il faut insérer le bout de JavaScript retourné par cette fonction au bon emplacement dans notre script original avant de l'enregistrer.
- c) Dans le `Page_Load` de la page mère nous testerons le postback et détecterons notre événement personnel pour enfin invoquer le `OnClick` du bouton.

Les étapes (a) et (b) se réalisent comme suit :

```
string s = script + this.Page.GetPostBackEventReference(this,
"@@@@buttonPostBack") + script2;
```

Nous avons simplement découpé la chaîne « script » en deux bouts pour y insérer le code du postback, rien de compliqué (reportez vous au code fourni pour plus de détail).

Le script est ensuite enregistré comme nous le faisons avant :

```
Page.ClientScript.RegisterClientScriptBlock(this.  
GetType(), "OpenModal", s, true);
```

Pour la troisième étape (c) nous ajoutons le code suivant dans le Page_Load de la page mère :

```
if (IsPostBack)  
{  
    string eventArg = Request["__EVENTARGUMENT"];  
    if( eventArg != null )  
    {  
        int offset = eventArg.IndexOf("@@@@");  
        if (offset > -1)  
        {  
            btnModal_Click(this, null);  
        }  
    }  
}
```

Les arobases servent à repérer notre événement personnel. Pour plus d'explications lisez l'article cité en référence plus haut. Ce qu'il faut retenir ici c'est que si la variable `offset` est supérieure à -1 en fin de code c'est que l'événement ayant causé le postback est le nôtre. Il suffit alors dans notre cas d'appeler le gestionnaire du `OnClick` du bouton pour simuler le même fonctionnement que sous IE.

Conclusion

Laborieux et complexe ? oui, certainement.

A noter malgré tout que si vous êtes certain que les utilisateurs de votre site n'utilisent que IE (ce qui est facilement contrôlable dans un Intranet par exemple), alors la gestion des dialogues modaux avec retour de valeurs devient très simple : ajout de l'attribut `onclick` sur le bouton avec ouverture de la fille par `showModalDialog`. C'est tout... Tout le code ajouté qui rend au final la solution assez lourde n'est justifié non par les lacunes de Microsoft mais bien par celles de Firefox et autres navigateurs de type Netscape...

Internet Explorer a ses défauts, cela est indubitable, mais il suit avec cohérence les évolutions des modèles de programmation Web comme

ASP.NET, ce qui n'est pas le cas des concurrents, et on comprend aisément pourquoi.

Le principal défaut de IE est d'être considéré comme une « passoire ». Cela oblige la mise en place d'un firewall et d'un bon anti-virus. Mais finalement peut-on s'en passer avec les autres navigateurs ? ... non. Cela donne à réfléchir ne trouvez-vous pas ?

Code source fourni :

Le projet « simple » qui reprend tous les exemples sauf celui retournant une valeur.

Le projet « retVal » qui contient toutes les modifications pour faire fonctionner le dernier exemple avec gestion de valeurs de retour.