Liens et nouvelle fenêtre

par Éric Daspet [Ganf]

Ce qui est gênant avec les mauvaises questions c'est que la réponse semble toujours hors-sujet à notre correspondant. En réalité c'est souvent la question qui est hors sujet par rapport à son problème. Si je vous parais hors sujet dans la première partie de cet article, dites vous bien que c'est parce que j'essaie de répondre au problème réel, pas à la question.

> *« En HTML strict on ne peut pas utiliser l'attribut target sur les liens, comment ouvre t-on une nouvelle fenêtre alors ? »*

Hier encore j'ai vu cette question. Je la vois trop souvent et malheureusement elle est suivie à chaque fois par une horde de solutions, toutes aussi mauvaises. Y répondre complètement est long, je vais m'y essayer ici. Tout ce que je vous demande c'est de ne pas sauter sur le code à la fin sans avoir lu le début. Les premiers textes sont les plus importants, de loin. C'est là que vous trouverez ma réponse à l'essentiel du problème.

Des liens ciblés

À titre de rappel, on signalise les liens HTML avec la balise <a>. Pour être un lien cette balise doit contenir au moins un attribut href (qui spécifie l'adresse de la page cible) : mon lien. Par défaut, quand on clique sur ce lien, la page cible remplace la page courante.

Optionnellement, on peut ajouter un attribut *target* qui permet de spécifier un nom de fenêtre. La page s'affichera alors dans la fenêtre portant ce nom au lieu de la fenêtre courante. Cet attribut a une valeur *magique* nommée *_BLANK* qui demande au navigateur d'ouvrir une nouvelle fenêtre pour afficher la page. Il se trouve que cet attribut target est autorisé dans les versions transitionnelles de HTML 4 et XHTML 1, mais est inexistant dans les versions strictes de ces mêmes formats.

Une fenêtre ouverte sur le monde extérieur

Ma première réaction à ceux qui posent la question de l'attribut target c'est leur demander pourquoi ils veulent absolument ouvrir une nouvelle fenêtre. En général, sur le Web, c'est une mauvaise idée.

Ouvrir une nouvelle fenêtre c'est perturber l'utilisateur débutant, qui n'en comprend pas toujours le principe. Les fenêtres s'empilent souvent et celles de dessous sont oubliées à jamais. Quant à celle ouverte sur le dessus, elle risque de dérouter l'utilisateur, elle lui fait perdre son principal outil de navigation et son meilleur point de repère : le bouton pour revenir en arrière. L'historique de navigation ne sera en effet remis à zéro dans cette nouvelle fenêtre. Qu'on soit débutant ou averti, c'est quelque chose qui est gênant, ou au moins agaçant.

On entend parfois que les nouvelles fenêtres permettent de ne pas perdre l'utilisateur en laissant l'ancienne page visible. En réalité c'est tout le contraire, et si vous voulez faciliter la vie de vos visiteurs, contentez vous de mettre un titre pertinent à vos pages pour qu'il puisse naviguer correctement dans son historique quand il revient en arrière.

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)

Pensez d'ailleurs qu'en plus de perturber les utilisateurs débutants, vous agacerez l'utilisateur averti. Ce dernier sait très bien ouvrir une fenêtre tout seul quand il le veut, mais n'appréciera pas qu'on le force à quoi que ce soit. Si en plus cette nouvelle fenêtre lui casse son historique et lui encombre sa barre des taches vous risquez de ne plus le voir revenir. Je ne parle même pas des logiciels anti-popup qu'il peut avoir lancé et qui, s'ils sont mal réglés, fermeront automatiquement toutes vos nouvelles fenêtres.

L'attribut target ne sert pas qu'à l'ouverture de nouvelles fenêtres, il sert aussi à envoyer un lien vers une fenêtre déjà ouverte et nommée explicitement. Mais là c'est encore pire, l'historique ne veut plus rien dire et l'utilisateur, même averti, ne comprendra pas forcément que le lien est en train de s'ouvrir dans une seconde fenêtre qu'il n'a pas sous les yeux. Globalement les fenêtres nommées ne servent que pour les frames, et comme vous ne devriez pas utiliser les frames ...

Oh, il existe des cas où cibler une fenêtre particulière est utile, mais c'est rare. Ouvrir une nouvelle fenêtre est pertinent presque uniquement pour les liens de contexte. Ce que j'appelle les liens de contexte ce sont ces petites popup qui donnent une information sur la page en cours, qu'on utilise et qu'on referme tout de suite : les liens d'aide, les confirmation, les outils de zoom, etc.

Vous ai-je convaincu ? Alors n'allez pas plus loin, le reste c'est pour ceux qui persistent. Si toutefois vous persistez à vouloir cibler autre chose que la fenêtre courante, alors mettez au moins en oeuvre le minimum indispensable : prévenir l'utilisateur que le lien ne s'ouvrira pas comme il peut s'y attendre. C'est généralement faisable par une petite icône et/ou une note manuscrite en haut de page.

Be cool, be strict

Il existe deux modèles en HTML, le modèle dit *transitionnel* et le modèle dit *strict*. Le premier est ce qu'on pourrait appeler « le bon vieux HTML », utilisé par la majorité des gens. Il permet de contrôler la mise en forme et la présentation du document : couleurs, alignements, tailles, positionnement, ouvertures de fenêtres, ...

Le modèle strict vise lui à séparer la description du contenu (sens et structure des différentes parties du document) de sa présentation (mise en forme, contrôle de l'affichage, gestion de l'interface avec les fenêtres, etc.). Il en résulte logiquement que certains éléments y sont inexistants, on peut citer par exemple les balises ou , ainsi que les attributs align et target.

La question posée au départ contient donc à la base une demi contradiction. Pourquoi vouloir en même temps faire une séparation stricte et pourtant vouloir contrôler la présentation à coup d'attributs HTML target ? C'est soit l'un soit l'autre.

Faire un javascript avec <a href="mapage.html" onclick="window.open(this.href)"> n'y changera rien. Le validateur HTML ne râlera plus, mais dans les faits vous ne respecterez pas plus le principe du modèle strict. Si vous voulez vraiment contrôler l'interface utilisateur à l'aide de code au milieu de votre document, c'est que vous ne faites pas une séparation stricte entre la description du contenu et sa présentation. Vouloir mettre une déclaration stricte pour votre HTML ne trompera que vous, ça ne changera rien pour vos clients ni pour la maintenance de vos pages.

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)

Utiliser le modèle transitionnel n'est pas mal en soi, c'est simplement une philosophie de conception de la page. Si c'est la vôtre, alors ne vous cachez pas : Utilisez les outils pour ça, utilisez une déclaration de HTML transitionnel et l'attribut target. Cela sera supporté partout et plus accessible que les onclick ou autres astuces javascript.

Si vous choisissez le modèle strict (et je vous le conseille), alors considérez qu'il n'est pas une bonne idée de spécifier « ce lien doit s'ouvrir dans une nouvelle fenêtre » au milieu du HTML. Le problème ne vient pas de l'attribut target en lui-même, mais simplement de votre but de contrôler l'interface utilisateur en modifiant le balisage du document.

La solution existe, je l'ai rencontrée

La suite vous donne une des possibilités pour gérer la problématique des nouvelles fenêtres en modèle strict, mais réfléchissez bien aux deux réflexions plus haut avant : êtes-vous sûrs qu'une nouvelle fenêtre est profitable à l'utilisateur ? êtes-vous sûr que ce que vous cherchez n'est pas simplement le modèle transitionnel ?

Si vous lisez encore c'est que vous devez être sûrs de vous. Nous allons donc respecter la philosophie du modèle strict et séparer la description du document de sa présentation (je classe les comportements comme celui ci dans la rubrique présentation). Je vous propose donc une procédure en trois étapes :

1. Catégorisation sémantique : Lors de la rédaction du contenu, on catégorise chaque lien suivant son rôle ou son sens. On spécifie s'il s'agit d'un lien qui sert pour appeler de l'aide, afficher un exemple, etc.

2. Définition du comportement : Dans un fichier javascript lié, on définit une fonction dont le seul rôle est de recevoir un événement quant quelqu'un clique sur un lien. Elle se contente alors d'ouvrir une popup vers la page souhaitée et demande au navigateur de ne pas exécuter son action par défaut (chargement de la page dans la fenêtre en cours).

3. Association sens-comportement : On créé un second code javascript qui va explorer la page HTML pour repérer tous les liens. Il filtre les liens suivant leur type, et, pour chaque lien, s'il faut l'ouvrir dans une nouvelle fenêtre, il demande au navigateur d'intercepter les clics avec la fonction précédente.

Cette procédure utilise du javascript dit « non intrusif ». Il a deux avantages : Tout d'abord vous n'avez pas à l'insérer partout au milieu de votre contenu. De plus, il se base sur une description correcte du HTML. Si ce javascript n'est pas compris par un navigateur, le visiteur restera avec une page HTML tout à fait classique totalement fonctionnelle. Javascript agit comme une surcouche pour paufiner le comportement de la page, et n'est en rien indispensable ou pertirbateur.

À titre d'information, il existe une méthode bien plus efficace et élégante pour résoudre notre problème. Elle exploite les XBL de Mozilla ou les HTC de Microsoft Internet Explorer. Le défaut de cette solution est qu'elle ne repose sur aucun standard commun et se révélera incompatible avec tous les autres navigateurs. Le code présenté ci-dessous est normalement bien plus compatible et repose sur des comportements standardisés.

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)

Show the source Luke

L'intégralité du code javascript est fait à partir de DOM, une interface normalisée par le W3C qu'on retrouve sur la plupart des navigateurs modernes. Il ne fonctionnera par contre pas sous Netscape 4 (il doit être possible de l'adapter mais on sort alors du code simple, standard et rapide à faire). Il n'y a pas à craindre d'incompatibilité puisque de toute façon la page restera totalement utilisable, sans dégradation réelle, même si le javascript est totalement désactivé.

Baliser le contenu

Avant toute chose, ce code se base sur une description complète du contenu. Il va nous falloir catégoriser les différents liens de notre page : déterminer quels sont les liens d'aide, quels sont les liens qui mènent à des exemples, à des pages d'articles, etc. Pour qualifier les liens nous avons deux attributs, l'attribut *rel* et l'attribut *class*. Le premier est spécifique aux liens et c'est celui qui devrait avoir notre préférence. Malheureusement, si on souhaite par la suite donner un style spécifique à chaque type de lien par CSS, pour compatibilité avec MSIE il vaut mieux utiliser l'attribut class.

Dans notre cas nous cherchons à ouvrir les pages d'aide en popup. On pourra les noter ainsi :

```
<a class="help" href="aide.html#sujet">aide</a>
```

Gérer les navigateurs non standards

On utilisera de DOM quelques fonctions pour naviguer à travers la page HTML (pour repérer les liens) et quelques fonctions pour gérer les événements (clic sur les liens). Malheureusement, au niveau des événements, Microsoft joue solo et a une syntaxe non standard. Il va donc nous falloir créer deux méthodes d'abstraction pour pouvoir rester compatibles.

```
function addEvent(source, type, callback) {
  // fonction d'abstraction pour enregistrer un gestionnaire d'evenement
  // comprend le DOM standard, la syntaxe prorietaire MSIE, l'ancien modele HTML
  // source : objet sur lequel ajouter le gestionnaire d'evenement
  // type : type d'evenement
  // callback : fonction qui traitera l'evenement
  if (source.addEventListener){                // code standard DOM
    source.addEventListener(type, callback, false);
    return true;
  } else if (source.attachEvent){    // code propriétaire MSIE
    var r = source.attachEvent("on"+type, callback);
    return r;
  } else {              // code navigateur sans support DOM-event
    eval('source.on' + type + '= callback') ;
  }
}

function getStandardEvent(e) {
 // abstraction pour recuperer un objet standard pour l'evenement en cours
 // comprend le modele DOM standard et le modele proprietaire de MSIE
 // e : parametre recu lors de l'appel du gestionnaire d'evenement
```

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)

```
 // retour : objet d'evenement standard
 if (e == null && window.event) {
   // cas particulier de MSIE pour recuperer l'evenement en cours
   e = window.event ;
 }
 if (e.target == null && e.srcElement) {
   // cas particulier de MSIE pour recuperer la balise DOM cible
   e.target = e.srcElement ;
 }
 if (! e.preventDefault ){
   // cas particulier de MSIE pour empecher l'action par defaut du navigateur
   e.preventDefault = function () { this.returnValue = false ; } ;
 }
 return e ;
}
```

Demander l'interception des clics

Le coeur de notre application se compose de deux autres fonctions : une qui sait recevoir un événement de clic et demander l'ouverture d'une popup à la place, une qui sait explorer la page HTML pour demander l'interception des clics sur les liens d'aide.

```
function openLinkInPopupWhenClick(e) {
  // gestionnaire d'evenement actif lors d'un clic sur les liens
  // ouvre le lien dans une popup et pas dans une page normale
  // e : evenement de clic
  e = getStandardEvent(e)  ;
  var link =  e.target  ;
  var addr = link.getAttribute('href') ;
  window.open(addr, '_blank', 'resizable=yes,width=200,height=300')  ;
  e.preventDefault()  ;
  return false ;
}

function prepareHelpLinks() {
 // explore le document pour rechercher les liens d'aide
 // à chaque lien, on verifie s'il a "help" dans la liste de ses classes
 // si oui, on enregistre un gestionnaire d'evenement pour le clic de ce lien
 var link, list, i ;
 list = document.getElementsByTagName('a') ;
 for(i=0; i<list.length; i++) {
   link = list.item(i) ;
   if (link.getAttribute('href') && link.getAttribute('class')) {
     if ((' '+link.className+' ').indexOf(' help ') != -1) {
       addEvent(link, 'click', openLinkInPopupWhenClick) ;
     }
   }
 }
}
```

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)

Il ne nous reste qu'à initialiser notre page en exécutant la fonction prepareHelpLinks(). Cette exécution doit être différée jusqu'à que la page soit complètement chargée, reçue et interprétée (sinon on ne trouvera pas tous les liens). On utilise donc ici aussi un événement DOM :

```
if (document.getElementById) {
  addEvent(window, 'load', prepareHelpLinks) ;
}
```

Pour faire fonctionner notre code il suffit de le regrouper dans un unique fichier javascript, et d'inclure ce fichier à l'aide d'un balise <script> dans l'entête de la page. Nous avons au final un composant qu'on peut ajouter ou retirer à volonté pour modifier le comportement de notre page, sans rien toucher au contenu lui-même, il suffit d'une ligne :

```
<script type="text/javascript" src="helplinks.js"></script>
```

Le mot de la fin

Si j'avais un mot de la fin il serait « n'interférez pas avec l'interface de vos utilisateurs, ne créez pas de fenêtre ». Maintenant, comme je sais que vous n'allez pas m'écouter, je dirai « ne cédez pas à la versionite aiguë ». Si vous utilisez un modèle où vous mixez contenu et présentation/comportement, assumez le et arrêtez de vouloir marquer « strict » en haut de votre document uniquement pour être à la mode.

Le code développé plus haut ne devrait finalement presque jamais être utilisé. Il peut par contre vous donner un exemple concret de javascript non intrusif. Les excès passés avec du javascript partout n'importe comment ont conduit les concepteurs modernes à fuir ce langage. Et si on se remettait plutôt à faire les choses correctement, avec une bonne séparation contenu/comportement/présentation et des composants qui s'ajoutent au HTML au lieu de le remplacer ? Le javascript est utile, il a une réelle valeur ajoutée, il suffit de bien s'en servir. Maintenant que vous avez une solution, la balle est dans votre camp.

Article publié sur CYBERcodeur.net le 30 octobre 2004 :

http://www.cybercodeur.net/weblog/articles/art_20041030.php
http://www.cybercodeur.net/weblog/commentaires/detailsCarnet.php?idmessage=1088 (billet)