

# **Tutoriel XUL – [xulfr.org/xulplanet.com](http://xulfr.org/xulplanet.com)**

Neil Deakin, traduit par les contributeurs au projet de traduction de Xulfr.org

# Sommaire du tutoriel XUL

<b>Préface.....</b>	<b>1</b>
<u>XUL Tutorial de XulPlanet.com.....</u>	1
<b>1. Introduction.....</b>	<b>2</b>
1.1 Introduction.....	2
1.2 La structure XUL.....	4
1.3 L'URI Chrome.....	10
1.4 Les fichiers contents.rdf.....	12
<b>2. Éléments simples.....</b>	<b>18</b>
2.1 Créer une fenêtre.....	18
2.2 Ajouter des boutons.....	20
2.3 Ajouter des libellés et des images.....	22
2.4 Les champs de saisie.....	23
2.5 Les contrôles de listes.....	26
2.6 Indicateurs de progression.....	29
2.7 Ajout d'éléments HTML.....	31
2.8 Utilisation des spacers.....	34
2.9 Plus de caractéristiques sur les boutons.....	37
<b>3. Le modèle de boîte.....</b>	<b>40</b>
3.1 Le modèle de boîte.....	40
3.2 Positionnement d'éléments de fenêtres.....	43
3.3 Détails sur le modèle de boîte.....	49
3.4 Les boîtes de groupe.....	51
3.5 Ajouter plus d'éléments.....	53
<b>4. Éléments communs.....</b>	<b>56</b>
4.1 Piles et Paquets.....	56
4.2 Positionnement dans une pile.....	58
4.3 Onglets.....	59
4.4 Grilles.....	62
4.5 Cadres de contenu.....	66
4.6 Séparateurs.....	68
4.7 Barres de défilement.....	72
<b>5. Menus et barres d'outils.....</b>	<b>74</b>
5.1 Barres d'outils.....	74
5.2 Barres de menu simples.....	76
5.3 Plus de fonctionnalités de menu.....	78
5.4 Menus surgissants.....	81
5.5 Menus défilants.....	85
<b>6. Évènements et scripts.....</b>	<b>87</b>
6.1 Ajout de gestionnaires d'évènements.....	87
6.2 Plus sur les gestionnaires d'évènements.....	90
6.3 Raccourcis clavier.....	93
6.4 Focus et Selection.....	98
6.5 Commandes.....	102
6.6 Mise à jour de commandes.....	105
6.7 Broadcasters et Observateurs.....	108

# Sommaire du tutoriel XUL

<b>7. Modèle Objet de Document (DOM).....</b>	<b>112</b>
7.1 Document Object Model.....	112
7.2 Modification d'une interface XUL.....	117
7.3 Manipulation de listes.....	120
7.4 Les objets boîtes.....	123
7.5 Interfaces XPCOM.....	127
7.6 Exemples XPCOM.....	132
7.7 Utilisation du presse-papiers.....	134
7.8 Glisser-Déposer.....	138
7.9 Conteneur JavaScript pour le Glisser-Déposer.....	140
7.10 Exemple Drag and Drop.....	144
<b>8. Arbres.....</b>	<b>148</b>
8.1 Arbres.....	148
8.2 Autres caractéristiques des arbres.....	152
8.3 Selection dans les arbres.....	155
8.4 Vues d'arbre personnalisées.....	157
8.5 Détails sur les vues d'arbres.....	160
8.6 Les objets boîtes des arbres.....	167
<b>9. RDF et templates.....</b>	<b>172</b>
9.1 Introduction à RDF.....	172
9.2 Gabarits.....	174
9.2bis Exemples de syntaxe de gabarits.....	179
9.3 Arbres et Gabarits.....	181
9.4 Sources de données RDF.....	184
9.5 Règles avancées.....	188
9.6 Données persistantes.....	192
<b>10. Thèmes et localisation.....</b>	<b>195</b>
10.1 Ajouter des feuilles de styles.....	195
10.2 Styler un arbre.....	198
10.3 Modification du thème par défaut.....	201
10.4 Créer un thème.....	203
10.5 Localisation.....	207
10.6 Les fichiers de propriétés.....	213
<b>11. Bindings.....</b>	<b>215</b>
11.1 Introduction à XBL.....	215
11.2 Contenu anonyme.....	217
11.3 Héritage d'attributs XBL.....	221
11.4 Ajout de propriétés.....	223
11.5 Ajout de méthodes.....	226
11.6 Ajout de gestionnaire d'évènements.....	230
11.7 Héritage XBL.....	233
11.8 Exemple XBL.....	234
<b>12. Fenêtres spécialisées.....</b>	<b>239</b>
12.1 Caractéristiques d'une fenêtre.....	239
12.2 Créer des boîtes de dialogues.....	241
12.3 Boîte de dialogue de fichiers.....	243
12.4 Creation d'un assistant.....	245

# Sommaire du tutoriel XUL

<b><u>12. Fenêtres spécialisées</u></b>	
<u>12.5 Assistant avancé</u>	247
<u>12.6 Overlays</u>	249
<u>12.7 Overlays inter-paquetage</u>	254
<b><u>13. Installation</u></b>	<b>256</b>
<u>13.1 Création d'un programme d'installation</u>	256
<u>13.2 Les scripts d'installation</u>	258
<u>13.3 Fonctions additionnelles d'installation</u>	261

# Préface

## XUL Tutorial de XulPlanet.com

Écrit par Neil Deakin. Traduit par différents contributeurs de xulfr.org.

Original : <http://www.xulplanet.com/tutorials/xultu/>.

· Bienvenue sur le *XUL tutorial*. Ce tutoriel décrit XUL, le langage XML pour interfaces utilisateurs. Ce langage a été créé pour l'application Mozilla et est utilisé pour définir son interface graphique.

Ce tutoriel décrit XUL tel qu'il est défini dans Mozilla 1.4.

Dernière mise à jour : 28/08/2005

## Contributeurs

Ce tutoriel a été traduit par des contributeurs de xulfr.org.

### *Coordinateurs de la traduction*

Laurent Jouanneau et Alain Boquet.

### *Traducteurs*

Par ordre alphabétique : Julien Appert, Alain Boquet, Adrien Bustany, BrainBooster, Caffeine, Cyril Cheneson, Sylvain Costard, Romain D., Cyril Delalande, Dkoo, Durandal, Julien Etaix, Chaddaï Fouché, Gnunix, Damien Hardy, Nadine Henry, Laurent Jouanneau, Gérard L., Maximilien, Medspx, Jean Pascal Milcent, Adrien Montoille, Gabriel de Perthuis, Tristan Rivoallan, Vincent S., Benoit Salandre, Cyril Trumpler.

### *Principaux relecteurs et correcteurs*

Alain Boquet, Laurent Jouanneau. Merci aux autres personnes qui nous ont signalé ça et là des erreurs.

## Éditions successives

*28 novembre 2004*

Première publication du tutoriel

*30 novembre 2004*

Corrections de fautes de français sur les sections 1.2, 1.3, 1.4, 2.1

Corrections de tous les exemples RDF, en rajoutant *RDF*: devant les attributs RDF (la déclaration de l'espace de nom RDF pour les attributs rdf est obligatoire d'après les spécifications du format RDF).

*Avril 2005*

Début de la mise à jour du tutoriel.

*Août 2005*

Publication de la mise à jour complète du tutoriel.

# 1. Introduction

## 1.1 Introduction

Écrit par Neil Deakin. Traduit par *Tristan Rivoallan* (25/02/2004), mise à jour par Laurent Jouanneau (17/11/2004) .

Page originale : <http://www.xulplanet.com/tutorials/xultu/intro.html>

### Introduction

Ce tutoriel est destiné à vous guider dans l'apprentissage de XUL (XML User–interface Language) qui est un langage multi plates–formes permettant de décrire les interfaces utilisateurs des applications.

Ce tutoriel vous montrera la création d'une simple interface utilisateur de recherche de fichiers, comme celle fournie par l'interface Sherlock du Macintosh, ou la boîte de dialogue de recherche de fichier de Windows. Notez que seule l'interface utilisateur sera créée avec quelques fonctionnalités limitées. La recherche de fichier proprement dite ne sera pas implémentée. Une ligne bleue apparaîtra sur la gauche des paragraphes où la boîte de dialogue de recherche de fichier sera modifiée. Vous pourrez suivre cela à travers les sections.

### Qu'est ce que XUL et pourquoi a–t–il été créé ?

XUL (prononcez *zool*, cela rime avec cool) a été créé pour avoir un développement du navigateur Mozilla plus facile et plus rapide. C'est un langage XML donc toutes les caractéristiques disponibles dans XML sont aussi disponibles dans XUL.

La plupart des applications ont besoin d'être développées en utilisant les caractéristiques d'une plate–forme spécifique, rendant l'adaptation multi plate–forme consommatrice en temps et coûteuse. Cela peut sembler peu important pour certains, mais des utilisateurs peuvent vouloir utiliser une application sur d'autres périphériques comme les assistants personnels ou les portables.

Un certain nombre de solutions multi plates–formes a été développé dans le passé. Java, par exemple, a comme principal argument de vente la portabilité. XUL est l'un de ces langages conçus spécialement pour créer des interfaces utilisateurs portables.

Cela prend beaucoup de temps pour bâtir une application, même pour une seule plate–forme. Le temps requis pour compiler et débbugger peut être long. Avec XUL, une interface peut être implémentée et modifiée rapidement et facilement.

XUL a tous les avantages des autres langages XML. Par exemple, XHTML ou d'autres langages XML comme Math–ML ou SVG peuvent y être insérés. De plus, les textes affichés avec XUL sont aisément localisables, ce qui signifie qu'ils peuvent être traduits dans d'autres langues avec peu d'effort. Les feuilles de styles peuvent être appliquées pour modifier l'apparence de l'interface utilisateur (beaucoup mieux que les systèmes de skins ou thèmes de WinAmp ou de certains gestionnaires de fenêtre).

### Quels types d'interface utilisateurs peuvent être réalisés avec XUL ?

XUL fournit la possibilité de créer la plupart des éléments que l'on trouve dans les interfaces graphiques modernes. Il est suffisamment générique pour être utilisé pour les besoins particuliers de certains périphériques, et suffisamment puissant pour permettre aux développeurs de créer des interfaces sophistiquées.

Quelques éléments pouvant être créés :

- Champs de saisie tels que des boîtes de textes et des cases à cocher
- Barres d'outils avec boutons et autres contenus
- Menus dans des barres de menus ou des menus surgissants
- Boîtes de dialogues à onglets
- Arbres pour informations hiérarchiques ou tabulaires
- Raccourcis claviers

Le contenu affiché peut être créé à partir du contenu d'un fichier XUL ou à partir d'une source de données. Dans Mozilla, de telles sources de données sont utilisées pour les messages des boîtes aux lettres, les Marque-pages, et les résultats de recherche. Les contenus des menus, arbres, et autres éléments peuvent être remplis avec ces données, ou avec vos propres données fournies dans des fichiers RDF.

Le contenu XUL est habituellement chargé à partir d'un paquetage installé dans Mozilla. Ces paquetages permettent à une application d'avoir des privilèges, comme lire des fichiers locaux ou modifier les préférences de l'utilisateur. Les fichiers XUL, les scripts associés et les images d'une application sont empaquetés dans un seul fichier, et ensuite téléchargés et installés par l'utilisateur. Mozilla fournit une méthode d'installation et d'enregistrement de ces paquetages sans avoir à écrire des lignes de code complexes. De plus, ces paquetages peuvent modifier le navigateur ou d'autres applications pour ajouter des fonctionnalités. C'est de cette façon que fonctionnent les extensions Firefox.

Il est possible d'ouvrir des fichiers XUL directement depuis le système de fichiers ou à partir d'un site Web distant. Cependant, ils seront restreints dans les types d'opérations qu'ils peuvent faire, et certains aspects de XUL ne fonctionneront pas. Toutefois, si vous voulez charger du contenu XUL à partir d'un site distant, le serveur Web doit être configuré pour envoyer les fichiers XUL avec le type de contenu *application/vnd.mozilla.xul+xml*. XUL est habituellement stocké dans des fichiers avec l'extension ".xul". Vous pouvez ouvrir un fichier XUL avec Mozilla comme vous le feriez avec d'autres fichiers, en utilisant la commande "ouvrir un fichier" du menu "fichier", ou en tapant l'URL dans la barre d'adresse.

## Que dois-je savoir pour comprendre le tutoriel ?

Vous devez connaître HTML et avoir au moins des connaissances de base sur XML et CSS. Voici des indications à garder à l'esprit :

- Les éléments XUL et attributs doivent être en minuscule car XML est sensible à la casse (contrairement à HTML).
- Les valeurs d'attributs doivent être placés entre guillemets, même si ce sont des nombres.
- Les fichiers XUL sont habituellement divisés en quatre fichiers : un pour la mise en page et les éléments, un pour les styles, un pour les déclarations d'entités (utilisées pour la localisation) et un pour les scripts. De plus, vous pouvez avoir des fichiers supplémentaires pour les images ou les données spécifiques à la plate-forme.

XUL est supporté dans Mozilla et les navigateurs basés sur le moteur Gecko, comme Netscape 6 ou plus, et Mozilla Firefox. À cause des différents changements dans la syntaxe XUL au fil du temps, vous devriez avoir la dernière version pour que les exemples fonctionnent correctement. La plupart des exemples devraient fonctionner dans Mozilla 1.0 et plus. XUL est pratiquement similaire dans Firefox et les autres navigateurs, bien qu'il y ait des différences spécifiques, comme le support des barres de boutons personnalisables.

Ce tutoriel tente de couvrir la plupart des fonctionnalités de XUL. Cependant, toutes les spécificités ne seront pas examinées. Une fois que vous vous serez familiarisés avec XUL, vous pourrez utiliser [la référence des éléments XUL](#) pour trouver les autres fonctionnalités supportées par d'autres éléments spécifiques.

---

Pour commencer, regardons comment les fichiers XUL sont organisés.

## 1.2 La structure XUL

Écrit par Neil Deakin. Traduit par **Gérard L.** (22/11/2003), mise à jour par Laurent Jouanneau (20/11/2004) .  
Page originale : <http://www.xulplanet.com/tutorials/xultu/xulfile.html>

Nous commencerons par regarder comment le système de fichiers de XUL est organisé sous Mozilla.

### Comment XUL est géré

Dans Mozilla, XUL est géré pratiquement de la même manière que HTML. Quand vous tapez l'URL d'une page HTML dans le champs de saisie d'adresse du navigateur, ce dernier localise le site Web et télécharge le contenu. Le moteur d'affichage de Mozilla prend le contenu du code source HTML, et le transforme en un arbre de document. L'arbre est alors converti en un ensemble d'objets qui peuvent être affichés sur l'écran. CSS, images et autres technologies sont utilisés pour contrôler la présentation. XUL fonctionne de la même manière.

En fait, sous Mozilla, tous les types de documents, que ce soit HTML ou XUL, ou même SVG, sont tous traités par le même code sous-jacent. Cela signifie que les mêmes propriétés CSS peuvent être utilisées pour décorer à la fois XUL et HTML, et beaucoup de spécificités peuvent être partagées entre les deux. Cependant, il y a des fonctionnalités qui sont spécifiques au HTML, comme les formulaires, et d'autres spécifiques à XUL comme les overlays.

Puisque les fichiers XUL et HTML sont gérés de la même manière, vous pouvez charger les deux à partir du système de fichier local ou d'une page Web. De plus, Mozilla fournit un moyen spécial pour l'installation et la déclaration des fichiers dans son système chrome. Il nécessite la création d'une archive des fichiers que l'utilisateur pourra télécharger et installer. Une fois déclarés, ces paquetages ont des privilèges avancés comme la possibilité de lire des fichiers, examiner les préférences et les marque-pages de l'utilisateur, et accéder à d'autres fonctionnalités du système. Bien sûr, les pages Web n'ont pas ces privilèges, tant qu'elles ne sont pas signées avec un certificat numérique et que l'utilisateur les y a autorisé.

La déclaration des paquetages est le moyen pour les extensions de Firefox de pouvoir ajouter des fonctionnalités au navigateur. Les extensions sont de petits paquetages de fichiers XUL, Javascript, feuilles de styles et images, rassemblés en un seul fichier. Ce fichier peut être créé en utilisant un outil ZIP. Quand l'utilisateur le télécharge, il est installé sur sa machine. Il sera intégré dans le navigateur en utilisant une fonctionnalité spécifique appelée *overlay*, qui permet au XUL de l'extension et au XUL du navigateur de se combiner ensemble. Pour l'utilisateur, c'est comme si l'extension avait modifié le navigateur, mais en réalité, le code est séparé et l'extension peut être désinstallée facilement.

Les paquetages déclarés ne sont pas requis pour utiliser les overlays, bien sûr. Si ils ne le sont pas, vous ne pourrez pas y accéder via l'interface principale du navigateur, mais vous pouvez toujours y accéder via un type spécial d'URL, désigné spécifiquement pour accéder aux paquetages installés. Ce type d'URL est appelé une URL chrome, et commence toujours par *chrome://*. De la même manière que les URLs *http://* réfèrent toujours aux sites Web distants, et que les URL *file://* réfèrent toujours aux fichiers locaux, les URL *chrome://* réfèrent toujours aux paquetages installés et aux extensions. Nous en verrons plus sur la syntaxe des URL chrome dans la prochaine section.

Il est important de noter que lorsque l'on accède au contenu avec une URL chrome, il obtient les privilèges avancés décrit plus haut contrairement aux autres types d'URL. Par exemple, une URL HTTP n'a pas de privilèges particuliers et une erreur apparaîtra si une page Web essaye, par exemple, de lire un fichier local. Alors qu'un fichier chargé via une URL chrome permettra de lire les fichiers sans restrictions.

La distinction est importante. Elle signifie qu'il y a certaines choses que le contenu des pages Web ne peut pas faire, comme lire les marque-pages de l'utilisateur. Cette distinction n'est pas basée sur le type du contenu affiché ; seul le type de l'URL est important. HTML et XUL placés sur un site Web n'ont pas de



permissions supplémentaires. Mais HTML et XUL chargés avec une URL chrome ont des permissions avancées.

Il faut noter également que le navigateur Mozilla lui-même est juste un ensemble de paquetages contenant des fichiers XUL, Javascript et CSS. Ces fichiers sont accédés via une URL chrome et ont alors des privilèges avancés. Ils fonctionnent alors comme n'importe quel autre paquetage. Bien sûr, le navigateur est bien plus conséquent et plus compliqué que la plupart des extensions. Le client de messagerie de Mozilla, l'éditeur Composer, Firefox et Thunderbird comme un certain nombre d'autres composants, sont tous écrits en XUL et sont accessibles via des URLs chrome.

Si vous désirez utiliser XUL sur un site Web, vous pouvez juste mettre les fichiers XUL sur le site comme vous le feriez avec un fichier HTML, et indiquer leur URL dans un navigateur. Assurez vous que votre serveur Web est configuré pour envoyer les fichiers XUL avec le type de contenu `application/vnd.mozilla.xul+xml`. Ce type de contenu permet à Mozilla de faire la différence entre HTML et XUL. Mozilla n'utilise pas l'extension du fichier sauf pour lire les fichiers locaux, mais vous devrez alors utiliser l'extension ".xul" pour tous les fichiers XUL. Vous pouvez ouvrir les fichiers XUL à partir de votre propre machine en les ouvrant dans le navigateur, ou en double cliquant sur le fichier dans votre gestionnaire de fichier. Souvenez vous que les fichiers XUL distants ont des restrictions significatives sur ce qu'ils peuvent faire.

En ce moment, un travail est en cours pour permettre de créer des applications XUL autonomes, avec leurs propres programmes d'installation et leurs exécutables. Puisqu'elles partagent les mêmes bibliothèques que Mozilla, vous n'aurez pas à installer le navigateur pour utiliser XUL. Il est possible de le faire aujourd'hui, mais le processus est compliqué et rarement fait. Le but est de rationaliser le processus.

Tandis que la plupart des fonctionnalités sont partagées entre HTML et XUL, Mozilla utilise différents types de document pour chacun d'eux. Il y a trois principaux types de document dans Mozilla : HTML, XML et XUL. Bien entendu, le type HTML est utilisé pour les documents HTML, le type XUL est utilisé pour les documents XUL, et le type XML est utilisé pour les autres documents en XML. Puisque XUL est aussi du XML, le type de document XUL est un sous-type du type XML générique. Il y a de subtiles différences dans les fonctionnalités. Par exemple, tandis que les contrôles de formulaire dans les pages HTML sont accessibles via la propriété `document.forms`, cette propriété n'est pas disponible pour les documents XUL puisque XUL n'a pas de formulaire dans le sens HTML du terme. D'un autre côté, les fonctionnalités spécifiques de XUL comme les overlays ou les gabarits ne sont utilisable que dans les documents XUL.

La distinction entre les documents est importante. Il est possible d'utiliser plusieurs fonctionnalités XUL en HTML ou en XML si elles ne sont pas spécifiques à des types de documents. Cependant, d'autres fonctionnalités nécessitent les bons types de documents. Ainsi, vous pouvez utiliser les types de mise en page XUL dans d'autres documents étant donné qu'ils ne reposent pas sur le type de document XUL pour fonctionner.

Pour résumé les points précédents :

- Mozilla affiche HTML et XUL en utilisant le même moteur sous-jacent et utilise CSS pour spécifier leur présentation.
- XUL peut être chargé à partir d'un site distant, du système de fichier local, ou installé comme un paquetage et être accédé en utilisant une URL chrome. C'est ce que font les extensions du navigateur.
- Les URLs chrome peuvent être utilisées pour accéder aux paquetages installés, et les ouvrent avec des privilèges avancés.
- HTML, XML et XUL ont chacun un type de document différent. Certaines fonctionnalités peuvent être utilisées dans n'importe quel type de document tandis que d'autres sont spécifiques à un type de document.

Les prochaines sections décrivent la structure de base d'un paquetage chrome qui peut être installé dans Mozilla. Cependant, si vous voulez juste commencer à créer une simple application, vous pouvez aller directement à la section 2 et revenir à cette section plus tard.

## Organisation d'un paquetage

Mozilla est conçu de telle manière que vous puissiez avoir autant de modules que vous désirez installer. Une installation typique (NdT : de la suite Mozilla) devrait se composer du navigateur, du client de messagerie et d'un éditeur. Elle aura également un module pour chaque thème (NdT : skin) et localisation (NdT : locale, paquetage de langue) installés. Chacun de ces modules, ou de ces paquetages, se compose d'un ensemble de dossiers qui décrivent son interface utilisateur. Par exemple, le module de messagerie comportera des descriptions pour la fenêtre de liste des messages de courriers, pour la fenêtre de rédaction et pour les boîtes de dialogues du carnet d'adresses.

Les paquetages qui sont inclus dans Mozilla sont situés dans le dossier chrome, que vous trouverez dans le répertoire où vous avez installé Mozilla. Le dossier chrome est l'endroit où sont situés tous les fichiers qui décrivent l'interface utilisateur employé par Mozilla, le client de messagerie et d'autres applications. Plutôt déroutant, le répertoire est appelé "chrome" mais il n'est que très légèrement lié à l'URL chrome. Une simple copie d'un fichier dans le répertoire "chrome" ne lui donne pas des permissions supplémentaire, et ne lui permet pas d'être accessible via une URL chrome. Le seul moyen de créer du contenu accessible à travers une URL chrome, est la création d'un paquetage comme décrit dans les prochaines sections. Ce répertoire chrome est appelé "chrome" parce qu'il semble être un nom commode pour utiliser ce répertoire où les paquetages chrome inclus avec Mozilla sont rangés.

Pour augmenter la confusion, il y a deux autres endroits où le mot "chrome" peut apparaître. Le premier est l'argument en ligne de commande `-chrome`, et le modificateur "chrome" pour la fonction `window.open`. Aucune de ces fonctionnalités autorise plus de privilèges. Elles sont plutôt utilisées pour ouvrir une nouvelle fenêtre principale sans l'interface utilisateur du navigateur comme les menus ou la barre d'outils. Vous utiliserez en général ces caractéristiques dans des applications XUL plus complexes lorsque vous ne voulez pas de fenêtre de navigation autour de vos boîtes de dialogue.

Les fichiers d'un paquetage sont généralement combinés dans un simple fichier JAR. Un fichier JAR peut être créé et examiné en utilisant un utilitaire ZIP. Par exemple, ouvrez quelques uns des fichiers JAR du répertoire chrome de Mozilla, pour voir la structure de ces paquetages. Bien qu'il est normal de combiner les fichiers dans un fichier JAR, les paquetages peuvent être aussi accédés dans une forme décompressée dans un ensemble de répertoire. Bien que vous ne distribuerez pas généralement un paquetage par ce moyen, c'est pratique durant le développement puisque vous pouvez éditer les fichier du répertoire et ensuite recharger le fichier XUL sans avoir à ré-empaqueter ou réinstaller.

Il y a habituellement trois parties différentes dans un paquetage chrome, bien qu'elles soient optionnelles. Chaque partie est stockée dans un répertoire différent. Ces trois ensembles décrits en dessous sont le contenu, le thème graphique et la localisation. Un paquetage particulier pourrait fournir un ou plusieurs thèmes et localisations, mais un utilisateur peut les remplacer par les siens. De plus, un paquetage peut inclure plusieurs applications différentes chacune accessible via des URLs chrome différentes. Le système de paquetage est suffisamment souple pour n'inclure que les parties dont vous avez besoin, et permettre le téléchargement séparé d'autres parties, comme le texte pour les différentes langues.

### **Content** – Fenêtres et scripts

Contient les déclarations des fenêtres et des éléments d'interface utilisateur. Ceux-ci sont stockés dans les fichiers XUL, qui ont l'extension xul. Il peut y avoir plusieurs fichiers XUL, mais la fenêtre principale devrait toujours avoir un nom de fichier identique au nom du paquetage. Par exemple, le paquetage editor contiendra un fichier appelé editor.xul. Les scripts (NdT : javascript) sont placés dans des fichiers séparés à côté des fichiers XUL.

### **Skin** – feuilles de style, images et autres fichiers de thèmes

Les feuilles de style décrivent des détails de l'aspect d'une fenêtre. Elles sont stockées séparément des fichiers XUL pour faciliter la modification du thème d'une application. Toutes les images utilisées sont également stockées ici.

**Locale** – *fichiers spécifiques de langues*

Tous les textes qui sont affichés dans une fenêtre sont stockés séparément. De cette façon, un utilisateur peut avoir une configuration pour sa propre langue.

Jetez un coup d'oeil au dossier chrome de Mozilla. Vous devriez voir un groupe de fichiers JAR, un pour chaque paquetage installé. Par exemple, le fichier messenger.jar décrit l'interface utilisateur pour le module de messagerie. Le fichier modern.jar décrit le thème moderne.

## Analyse d'un paquetage type

Le nom du fichier JAR devrait décrire ce qu'il contient, mais vous pouvez vous en assurer en regardant son contenu. Utilisons le paquetage de la messagerie comme exemple. Si vous extrayez les fichiers de messenger.jar, vous constaterez que cette archive contient une structure arborescente comme ce qui suit :

```
content
  messenger
    contents.rdf
    messenger.xul
    -- les autres fichiers XUL et scripts pour les mails sont ici --
    addressbook
      -- les fichiers du carnet d'adresses sont ici --
    messengercompose
      -- les fichiers de composition des messages vont ici --
  .
  .
  .
```

On peut facilement identifier cela comme le contenu d'un paquetage, car le dossier supérieur s'appelle 'content'. Pour les thèmes, le dossier aurait été appelé 'skin' et pour les localisations, il aurait été appelé 'locale'. En fait, ce n'est pas une obligation mais vous devriez suivre cette convention pour rendre votre paquetage plus clair. Certains paquetages peuvent inclure une section *content*, *skin* et *locale*. Dans ces paquetages, vous trouverez un sous-répertoire pour chaque section. Par exemple, Chatzilla est distribué de cette manière.

Le dossier content/messenger contient un certain nombre de fichiers avec les extensions xul et js. Les fichiers XUL sont ceux qui ont une extension xul. Les fichiers avec l'extension js sont des fichiers de Javascript qui contiennent les scripts qui exécutent les fonctionnalités d'une fenêtre. Beaucoup de fichiers XUL ont un fichier de script qui leur est associé, et certains en ont plus d'un.

Dans la liste ci-dessus, deux fichiers ont été montrés. Il y en a évidemment d'autres, mais pour des raisons de simplicité, ils ne sont pas montrés. Le fichier messenger.xul est le fichier XUL qui décrit la fenêtre principale affichant la liste des courriers de la messagerie. La fenêtre des courriers est assez complexe car elle est faite de plusieurs fichiers combinés entre eux en utilisant des overlays. La fenêtre principale pour le contenu d'un paquetage doit avoir le même nom que le paquetage avec l'extension xul. Dans notre cas, le nom du paquetage est *messenger*, aussi nous nous attendons à trouver *messenger.xul*. D'autres fichiers XUL décrivent des fenêtres séparées. Par exemple, le fichier subscribe.xul décrit la boîte de dialogue pour souscrire aux newsgroups.

Le fichier contents.rdf est trouvé dans chaque paquetage, un pour chaque contenu, thème et localisation dans le paquetage. C'est un fichier important puisqu'il spécifie le nom du paquetage, son auteur et sa version. Mozilla lira ce fichier et utilisera son contenu pour déclarer le paquetage et lui assigner une URL chrome, ainsi les fichiers pourront être accessibles à travers une URL chrome. Sans ce fichier, une URL chrome ne peut être assignée au paquetage. Les détails de ce fichier seront décrits dans une section ultérieure.

Deux sous-répertoires, *addressbook* et *messengercompose*, décrivent les sections additionnelles du composant de messagerie. Ils sont placés dans différents dossiers simplement pour les séparer. Ils n'ont pas besoin de fichiers *contents.rdf* parce qu'ils peuvent être accédés à partir des mêmes URL chrome.

## Thèmes graphiques

Le code sous-jacent de Mozilla les appellent *skins*, bien que l'interface utilisateur les appellent *thèmes*, mais ils désignent la même chose. Les fichiers *modern.jar* et *classic.jar* décrivent les thèmes inclus dans Mozilla. Leur structure est semblable aux paquetages habituels. Par exemple, *modern.jar* :

```
skin
  modern
    navigator
      contents.rdf
      -- les fichiers thèmes du navigateur sont ici --
    messenger
      contents.rdf
      -- les fichiers thèmes de la messagerie sont ici --
    editor
      contents.rdf
      -- les fichiers thèmes de l'éditeur sont ici --
    communicator
      contents.rdf
      -- les fichiers thèmes du communicator sont ici --
  global
    contents.rdf
    -- les fichiers thèmes globaux sont ici --
.
.
.
```

C'est un peu plus compliqué, bien que la structure est similaire à la partie contenu. À la place de 'content', le nom 'skin' est utilisé. Souvenez-vous que c'est purement conventionnel, puisque vous pouvez en fait mettre tous les fichiers dans un seul répertoire au niveau supérieur, et ne pas utiliser de sous-répertoires.

Cependant, pour de plus grosses applications comme Mozilla, les sous-répertoires sont utilisés pour séparer les différents composants. Dans l'exemple ci-dessus, cinq répertoires existent, un pour chaque paquetage sur lesquels le thème est appliqué. Le répertoire "global" contient les fichiers du thème qui sont généraux à tous les paquetages. Ces fichiers seront appliqués sur tous les composants et normalement vous utiliserez celui-ci. La partie "global" définit l'apparence de tous les composants graphiques XUL communs, alors que les autres répertoires ont des fichiers spécifiques aux autres applications. Par exemple, le répertoire "editor" décrit le thème pour le composant "editor" et contient, entre autre, les images pour les boutons de la barre d'outils de l'éditeur.

Vous avez pu noter qu'il y a cinq fichiers *contents.rdf*. De cette façon, les thèmes sont appliquées séparément à chaque module. Vous pourriez théoriquement avoir un thème différent pour le navigateur et pour la messagerie, mais la plus grande partie de l'aspect étant déterminé par la partie globale, vous ne verrez donc pas beaucoup de différences. D'ailleurs, Mozilla ne fournit pas la possibilité de sélectionner un thème différent pour chaque application. Ils sont également séparés en fichiers pour que de nouveaux modules puissent être ajoutés et que des modules existants puissent être enlevés facilement. Par exemple, vous pouvez créer un nouveau thème pour la messagerie et les utilisateurs peuvent le télécharger séparément. En empaquetant les parties séparément, les bonnes pièces peuvent être sélectionnées pour être utilisées.

Un thème se compose de fichiers CSS et d'un certain nombre d'images utilisés pour définir l'aspect d'une interface. Le fichier *messenger.css* est utilisé par *messenger.xul* et contient les styles pour définir l'apparence des différentes parties de l'interface de messagerie. À nouveau, notez le nom du fichier *messenger.css*, qui a le même nom que le paquetage. En changeant de fichier CSS, vous pouvez modifier l'apparence d'une fenêtre sans changer ses fonctions. C'est ainsi que vous pouvez créer un nouveau thème. La partie XUL est toujours la même, mais la partie thème change indépendamment.

## Localisation

Le fichier en-US.jar décrit l'information de langage pour chaque module, dans le cas présent pour l'anglais des États-Unis. Comme les thèmes, chaque langue contiendra les fichiers qui indiquent le texte utilisé par le paquetage mais pour une langue spécifique. Comme d'habitude, un fichier contents.rdf est inclus, listant les paquetages pour lesquels la partie locale fournit le texte. Les sous-répertoires fournissent le texte pour chaque paquetage. La structure est similaire à la partie thème, aussi nous ne la listerons pas ici.

Le texte localisé est stocké dans deux types de fichiers, des fichiers DTD, et des fichiers de propriétés. Les fichiers DTD ont une extension dtd et contiennent les déclarations d'entités, une pour chaque chaîne de caractères qui est utilisée dans une fenêtre. Par exemple, le fichier messenger.dtd contient des déclarations d'entités pour chaque commande du menu. En outre, des raccourcis clavier pour chaque commandes sont également définis, parce qu'ils peuvent être différents selon la langue. Les fichiers DTD sont utilisés par des fichiers XUL, et donc en général, vous en aurez un par fichier XUL.

La partie "locale" contient également des fichiers de propriétés, qui sont similaires, mais sont utilisés par les fichiers de script. Le fichier messenger.properties contient quelques chaînes de caractères de ce type.

Cette structure vous permet de traduire Mozilla ou un module dans une langue différente en ajoutant juste une nouvelle partie "locale" pour cette langue. Vous n'avez pas à changer la partie XUL. De plus, d'autres personnes peuvent fournir des paquetages séparés qui appliquent des thèmes ou des localisations à votre partie contenu, apportant ainsi un nouveau thème ou une nouvelle langue sans avoir à changer le paquetage original.

## Autres Paquetages

Plusieurs paquetages dans Mozilla sont des sous-paquetages de communicator.jar. Par exemple, vous trouverez la fenêtre des marque-pages, le visualiseur d'historique et les boîtes de dialogue des préférences dans le paquetage communicator. Ils sont mis là parce qu'ils sont communs à un certain nombre de paquetages. Il n'y a rien de spécial à leur sujet.

Il y a un paquetage spécial appelé toolkit (ou global). Nous avons déjà vu le dossier global pour des thèmes. Le toolkit.jar contient la partie *content* lui correspondant. Il contient quelques boîtes de dialogues et définitions globales. Il définit également l'aspect par défaut et les fonctionnalités des divers éléments graphiques tels que des boîtes de textes et des boutons. Les fichiers situés dans la partie globale de l'archive du thème contiennent l'apparence par défaut de tous les éléments d'interface de XUL. La plupart des changements de thème entraîneront des variations de ces fichiers.

## Ajouter un paquetage

Mozilla place les paquetages qui sont inclus dans l'installation, dans le répertoire chrome. Cependant, ils n'ont pas besoin d'être placés ici. Si vous avez un autre paquetage installé, il peut être placé n'importe où sur le disque. Le fichier chrome.rdf dans le dossier chrome stocke la liste des paquetages, thèmes et locales installés et l'endroit où ils sont situés. Il est commun d'installer les nouveaux paquetages dans le répertoire chrome simplement parce que c'est commode. cependant ils fonctionneront aussi bien à partir d'un autre répertoire, ou de quelque part sur votre réseau local. Vous ne pouvez pas les placer sur un site distant, à moins que le site distant soit monté sur le système de fichier local.

Un utilisateur peut avoir plusieurs thèmes et localisations installés modifiant le même paquetage. Un seul thème et une seule langue d'un paquetage ne sont actifs à un moment donné. Le fichier chrome/chrome.rdf dans le profil utilisateur spécifie quels thèmes et localisations sont actifs. Ce fichier peut aussi spécifier les paquetages "content". Il fonctionne de manière similaire au fichier chrome.rdf de l'installation principale de Mozilla. Cependant les paquetages spécifiés dans le profil ne sont installés que pour cet utilisateur, alors que le fichier chrome.rdf du répertoire de Mozilla est utilisé pour les paquetages disponibles pour tous les

utilisateurs.

---

Dans la prochaine section, nous regarderons comment se référer aux paquetages en utilisant les URL chrome.

## 1.3 L'URI Chrome

Écrit par Neil Deakin. Traduit par **Romain D.** (11/02/2004), mise à jour par Laurent Jouanneau (22/11/2004)

Page originale : <http://www.xulplanet.com/tutorials/xultu/chromeurl.html>

La présente section décrit comment faire référence à des documents XUL et d'autres fichiers Chrome.

### L'URL Chrome

Les fichiers XUL peuvent être référencés par une URL HTTP habituelle tout comme les fichiers HTML (ou tout autre type d'URL). Cependant, les paquetages qui sont installés sur le système chrome de Mozilla peuvent être référencés avec des URLs spéciales "chrome". Les paquetages livrés avec Mozilla seront déjà installés mais vous pouvez déclarer les vôtres.

Les paquetages installés ont l'avantage de ne pas avoir de restrictions de sécurité, lesquelles sont nécessaires pour de nombreuses applications. Un autre avantage comparé aux autres types d'URLs est qu'ils manipulent automatiquement les thèmes (NdT : skins) et les localisations (NdT : locales). Par exemple, une URL chrome vous permet de vous référer à un fichier dans le thème, comme une image, sans avoir besoin de savoir quel est le thème utilisé par l'utilisateur. Tant que les noms de fichiers sont les mêmes dans chaque thème, vous pouvez vous référer à un fichier en utilisant une URL chrome. Mozilla s'attachera à déterminer où le fichier est situé et renverra la bonne donnée. Cela signifie aussi que l'endroit où le paquetage est installé n'est pas important pour être capable d'y accéder. Les URLs chrome sont indépendantes du lieu où les fichiers sont stockés physiquement. Il est donc plus simple d'écrire des applications contenant beaucoup de fichiers sans vous soucier des détails sur les chemins des fichiers.

La syntaxe de base d'une URL chrome est la suivante :

```
chrome://<paquetage>/<partie>/<fichier.xul>
```

Le texte <paquetage> est le nom du paquetage, tel que messenger ou editor. Le texte <partie> vaut soit *content*, soit *skin* ou soit *locale*, selon la partie que vous voulez. *fichier.xul* est simplement le nom du fichier.

**Exemple :** `chrome://messenger/content/messenger.xul`

Ici, l'exemple se réfère à la fenêtre messenger. Vous pouvez pointer vers un fichier qui fait parti d'un thème en remplaçant *content* par *skin* et en changeant le nom du fichier. De même, vous pouvez pointer vers un fichier qui fait parti de la localisation en utilisant *locale* au lieu de *content*.

Quand vous ouvrez une URL chrome, Mozilla regarde à travers sa liste de paquetages installés et essaie de trouver le fichier JAR qui correspond au nom et à la partie recherchés. Une fois trouvé, il regardera dans ce fichier JAR s'il y a *fichier.xul*. Mozilla commencera toujours par regarder dans le même dossier du fichier JAR où est situé le fichier *contents.rdf* associé, comme décrit dans la section précédente. Cela signifie que si plusieurs paquetages ou parties sont tous situés dans le même fichier JAR, les fichiers seront localisés à la bonne place. Par exemple, le fichier *contents.rdf* pour l'URL chrome de messenger est situé dans le fichier *messenger.jar* et à l'intérieur de l'archive, dans le répertoire 'content/messenger'. Le fichier *messenger.xul* sera chargé à partir de cet endroit, et si vous ouvrez *messenger.jar*, vous trouverez ce fichier dans ce répertoire. Si vous utilisez une forme décompressée plutôt qu'une archive JAR, la même chose

s'applique excepté que Mozilla va directement dans le répertoire sans regarder dans un fichier JAR.

Si vous déplacez le fichier messenger.jar et mettez à jour son chemin dans la liste des paquetages chrome déclarés de Mozilla, la messagerie fonctionnera toujours puisqu'elle ne dépend pas de son emplacement d'installation. En utilisant les URLs chrome, nous pouvons laisser ce genre de détails à Mozilla. De même, si l'utilisateur change son thème, la partie 'skin' de l'URL chrome se traduit vers un ensemble de fichiers différent, et pourtant les fichiers XUL et les scripts n'ont pas besoin d'être changés.

Mozilla est capable de comprendre quel langage et thème sont actuellement utilisés et applique les répertoires appropriés sur les URLs chrome. Le fichier chrome.rdf dans le répertoire chrome et les fichiers contents.rdf sont là pour dire à Mozilla comment procéder. De cette façon, l'utilisateur peut utiliser n'importe quel thème ou langage mais les URLs qui référencent les fichiers chrome n'ont pas à être changés. Par exemple, le fichier navigator.css par défaut peut être référencé par :

```
chrome://navigator/skin/navigator.css
```

Si vous changez le thème du navigateur, l'URL chrome ne change pas, même si l'emplacement réel des fichiers utilisés par le thème change.

Le système chrome prend les sections du navigateur du contenu, du thème courant et du langage courant, et les groupe ensemble pour former une interface utilisateur. Voici d'autres exemples, cette fois pour messenger. Notez qu'aucune des URLs ne spécifie un thème, une langue ou un repertoire spécifique.

```
chrome://messenger/content/messenger.xul
chrome://messenger/content/mime.js
chrome://messenger/skin/icons/images/folder-inbox.jpg
chrome://messenger/locale/messenger.dtd
```

Pour des sous-paquetages, la même structure peut être employée. Les URLs suivantes se rapporteront à la fenêtre des marque-pages, l'une pour la suite Mozilla et l'autre pour Firefox car leur nom de paquetage est différent :

```
chrome://communicator/content/bookmarks/bookmarksManager.xul (Mozilla)
chrome://browser/content/bookmarks/bookmarksManager.xul (Firefox)
```

Vous pouvez entrer les URLs chromes n'importe où les URLs normales peuvent être utilisées. Vous pouvez même les entrer directement dans la barre d'adresse d'une fenêtre du navigateur Mozilla. Si vous entrez une des URLs mentionnées ci-dessus dans la barre d'adresse du navigateur, vous devriez voir cette fenêtre apparaître comme une page Web le ferait et dans la plupart des cas, dans une fenêtre séparée. Cependant quelques boîtes de dialogue pourraient ne pas bien fonctionner, si elles attendent des arguments fournis par la fenêtre qui les a ouvertes.

Vous pourriez voir également des URLs chrome sans noms de fichiers spécifiés, tel que :

```
chrome://navigator/content/
```

Dans le cas présent, seul le nom du paquetage et la partie sont spécifiés. Ce type de référence sélectionnera automatiquement un fichier approprié depuis le bon répertoire. Pour le contenu, un fichier avec le nom du paquet et une extension XUL est choisi. Dans l'exemple ci-dessus, le fichier navigator.xul est choisi. Pour messenger, messenger.xul serait sélectionné. Lorsque vous créez vos propres applications, vous pourriez créer un fichier pour votre fenêtre principale avec le même nom que le paquetage. Ainsi il pourrait être appelé en utilisant une forme plus courte. C'est commode car ainsi tout ce qu'à besoin de savoir un utilisateur pour pouvoir ouvrir l'application, est le nom du paquetage. Bien sûr, pour les extensions qui modifient l'interface du navigateur, l'utilisateur n'aura pas besoin de connaître l'URL car l'extension sera présente elle-même dans l'interface utilisateur.

Pour un thème, le fichier *paquetage.css* est choisi ; pour une localisation, le fichier *paquetage.dtd* est choisi.

Souvenez-vous, l'URL chrome n'est pas relative à un emplacement sur le disque. Les deux premières pièces sont le nom du paquetage et la partie (*content*, *skin*, ou *locale*). Bien qu'il soit commun de mettre les fichiers de contenu dans un répertoire appelé *content*, c'est purement une convention, et ces fichiers peuvent être placés dans une structure totalement différente. La seule règle est que la partie nom du fichier de l'URL chrome se réfère aux fichiers situés dans le même répertoire où le fichier associé *contents.rdf* est stocké.

Dans la section suivante, nous verrons comment créer des fichiers *contents.rdf* et des paquetages.

## 1.4 Les fichiers *contents.rdf*

Écrit par Neil Deakin. Traduit par **Benoit Salandre** (08/02/2004), mise à jour par Laurent Jouanneau (23/11/2004) .

Page originale : <http://www.xulplanet.com/tutorials/xultu/packaging.html>

Dans cette section, nous allons voir comment mettre les fichiers XUL et chrome dans un paquetage et créer les fichiers *contents.rdf* associés.

### Paquetages

Un paquetage est un ensemble de fichiers XUL et de scripts qui définissent la fonctionnalité d'une interface utilisateur. Les paquetages peuvent être installés dans Mozilla et référencés avec des URLs chrome. Un paquetage peut contenir tous les fichiers que l'on veut et peut-être découpé en sous-répertoires pour les différentes parties du paquetage. Par exemple, les marque-pages et l'historique font partie du paquetage *communicator*, mais sont stockés dans des sous-répertoires différents.

Un paquetage peut-être stocké comme un répertoire ou comme une archive JAR. Chaque paquetage aura un fichier *manifest*, *contents.rdf*, qui décrit le paquetage. Ce fichier sera placé dans le fichier JAR au côté des fichiers qu'il décrit. Le fichier doit s'appeler *contents.rdf* et être un fichier au format RDF (Resource Description Framework). Nous en apprendrons plus à propos de RDF ultérieurement.

### Les fichiers *contents.rdf*

Le fichier *contents.rdf* décrit les contenus d'un paquetage. Il peut aussi être utilisé pour décrire un thème ou une localisation. Ces fichiers *manifest* sont assez faciles à créer une fois que vous savez comment faire. Le modèle ci-dessous peut-être utilisé comme point de départ.

```
<?xml version="1.0"?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <RDF:Seq RDF:about="urn:mozilla:package:root">
    <RDF:li RDF:resource="urn:mozilla:package:monapplication"/>
  </RDF:Seq>

  <RDF:Description RDF:about="urn:mozilla:package:monapplication"
    chrome:displayName="Mon Application"
    chrome:author="nom"
    chrome:name="monapplication">
  </RDF:Description>

</RDF:RDF>
```

Vous pouvez utiliser ce modèle et effectuer quelques petites modifications spécifiques pour votre paquetage.



Décomposons celui-ci pour comprendre la fonction de chaque bloc.

```
<?xml version="1.0"?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
```

Ces trois lignes doivent être placées au tout début du fichier contents.rdf. Parce que RDF est un format XML, il contient les mêmes premières lignes standards que tout fichier XML doit posséder. Ensuite, nous déclarons les espaces de nom qui vont être utilisés, un pour RDF et un autre pour le système chrome. Si vous ne comprenez pas ce que cela signifie, ne vous inquiétez pas. Ajoutez les simplement au début du fichier.

```
<RDF:Seq RDF:about="urn:mozilla:package:root">
  <RDF:li RDF:resource="urn:mozilla:package:monapplication"/>
</RDF:Seq>
```

Ces lignes sont utilisées pour déclarer quels paquetages, thèmes et localisations sont décrits par le fichier manifest. Dans le cas présent, c'est un paquetage qui va être décrit (comme indiqué par le mot *package* dans le texte). Si vous créez un thème, vous mettriez *skin* à la place de *package*, et si vous créez une localisation, vous utiliseriez *locale*. Le nom du paquetage ici est *monapplication*. Bien entendu, il faudra le remplacer par le nom du paquetage que vous allez créer. Par exemple, l'application de courrier électronique de Mozilla se nomme 'messenger'. Le nom doit être court et significatif. Ce nom sera utilisé dans les URLs chrome pour le paquetage.

La balise RDF:li ci-dessus est similaire à la balise li du HTML, elle déclare un élément de liste. Ainsi, vous pouvez déclarer plusieurs paquetages par l'utilisation de plusieurs balises RDF:li.

Pour les thèmes, remplacez les deux occurrences de *package* par *skin* ; pour les localisations, remplacez les deux occurrences de *package* par *locale*. Par exemple, ce qui suit spécifie un thème :

```
<RDF:Seq RDF:about="urn:mozilla:skin:root">
  <RDF:li RDF:resource="urn:mozilla:skin:blueswayedshoes"/>
</RDF:Seq>
```

La partie suivante spécifie le nom et l'auteur de l'application :

```
<RDF:Description RDF:about="urn:mozilla:package:monapplication"
  chrome:displayName="Mon Application"
  chrome:author="nom"
  chrome:name="monapplication">
</RDF:Description>
```

Ce bloc est utilisé pour fournir plus de détails sur le paquetage, le thème ou la localisation. Vous aurez besoin d'une description pour chaque balise li. La valeur de l'attribut about devrait être identique à l'attribut resource de la balise li.

Les trois attributs supplémentaires donnent des informations sur le paquetage.

*displayName*

Le titre du paquetage tel qu'il devra être affiché à l'utilisateur. Par exemple, 'Messenger'.

*author*

Le nom de l'auteur du paquetage.

*name*

Le nom du paquetage, du thème ou de la localisation. Il devrait avoir la même valeur que ce qui vient après *urn:mozilla:package:* qui était spécifiée précédemment. Ce nom est utilisé comme première partie d'une URL chrome.

Vous pouvez aussi utiliser une variété d'autres valeurs. Quand Mozilla déclare votre paquetage, ces valeurs sont ajoutées dans le registre chrome.

Écrivons un fichier contents.rdf pour la boîte de dialogue de recherche de fichiers que nous voulons créer. Il sera nécessaire de décrire le paquetage. Comme il n'y a pas de sous-paquetages, de thèmes ou de localisations, il est assez semblable au modèle vu plus haut.

```
<?xml version="1.0"?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <RDF:Seq RDF:about="urn:mozilla:package:root">
    <RDF:li RDF:resource="urn:mozilla:package:findfile"/>
  </RDF:Seq>

  <RDF:Description RDF:about="urn:mozilla:package:findfile"
    chrome:displayName="Recherche de fichiers"
    chrome:author="N'importe qui"
    chrome:name="findfile">
  </RDF:Description>

</RDF:RDF>
```

Ici, le composant est appelé 'findfile' ce qui signifie que nous serons capables de le référencer en utilisant l'URL chrome suivante :

```
chrome://findfile/content/findfile.xul
```

## Installer un Paquetage

La liste des paquetages installés est enregistrée dans le fichier chrome.rdf du répertoire chrome. Vous ne devriez pas modifier ce fichier directement. Il est modifié automatiquement quand vous installez un nouveau paquetage. Comme pour les fichiers contents.rdf, c'est un fichier au format RDF. À première vue, il a l'air tout à fait différent des fichiers contents.rdf, mais si vous êtes familier de RDF, vous noterez que c'est en fait très proche.

Chaque fois que Mozilla démarre, il parcourt le répertoire chrome à la recherche d'un fichier nommé 'installed-chrome.txt'. Ce fichier contient une liste, dans un format très simple, de tous les paquetages, thèmes et localisations qui sont installés. Si ce fichier est modifié, Mozilla parcourt chaque entrée dans la liste et les enregistre ou les met à jour si nécessaire de sorte qu'elles puissent être utilisées.

Donc, pour enregistrer un nouveau paquetage, tout ce que vous avez à faire c'est de rajouter une entrée au fichier 'installed-chrome.txt' et de redémarrer Mozilla. Le nouveau paquetage sera enregistré et le fichier chrome.rdf sera régénéré puisqu'il doit nécessairement contenir le paquetage nouvellement installé. Mozilla possède également un système d'installation appelé XPInstall qui permet à des scripts d'installer automatiquement des paquetages via javascript sans avoir à modifier manuellement ce fichier. XPInstall sera décrit vers la fin de ce tutoriel. Cependant, pendant le développement, nous pouvons modifier directement le fichier.

Le fichier 'installed-chrome.txt' se trouve dans le répertoire chrome. Le fichier contient une liste d'entrée à installer, une par ligne. Par exemple :

```
content,install,url,resource:/chrome/findfile/content/findfile/
skin,install,url,resource:/chrome/findfile/skin/findfile/
```

Ce qui précède sera utilisé pour installer le paquetage, ainsi qu'un thème, pour la recherche de fichier. Le format de chaque ligne est assez simple. Il s'agit juste de quatre valeurs séparées par des virgules :

#### Type

Mettez *content* pour les paquetages chrome, *skin* pour les thèmes ou *locale* pour les localisations.

#### Install

Mettez le texte *install* pour que cette entrée soit installée. Pour les thèmes et les localisations, vous pouvez aussi mettre *profile* à la place pour installer cette entrée dans le répertoire profil de l'utilisateur. Elle ne sera installée et déclarée que pour un utilisateur en particulier.

#### Type d'URL

Mettez le texte *url* pour spécifier une URL indiquant où le nouveau paquetage, thème ou localisation est stocké. Si vous utilisez *path* à la place, vous pouvez utiliser un chemin de fichier (qui devra être au format de votre système d'exploitation).

#### URL

Mettez l'URL ou le chemin de l'emplacement du paquetage. Ce devrait être le répertoire contenant le fichier contents.rdf ou le chemin et le nom du fichier JAR. Comme on se réfère à un répertoire, assurez vous qu'il y'a une barre oblique de division (*slash*) à la fin sinon le paquetage ne sera pas trouvé.

Notez que les URLs utilisées sont des URLs *resource:*. Vous pouvez tout aussi bien utiliser des URLs de fichiers. L'URL de ressource est similaire aux URLs de fichiers, excepté qu'elle commence par *resource:* au lieu de *file:* et que son répertoire de base est le répertoire où Mozilla est installé plutôt que la racine du système de fichier. Cela signifie qu'elle peut être utilisée pour faire référence à des fichiers dans le répertoire de Mozilla ou dans ses sous-répertoires, indépendamment de l'emplacement où est installé Mozilla. L'URL de ressource doit avoir une seule barre de division après les deux points puisqu'il s'agit toujours d'un chemin relatif.

Ainsi, la ligne supplémentaire doit pointer vers le dossier où se trouve le fichier contents.rdf. Si vous avez plus d'un paquetage, ajoutez une ligne pour chacun d'eux.

Bien que Mozilla suive une convention de nommage des répertoires, vous pouvez mettre les fichiers où vous voulez. Par exemple, la ligne suivante installera un nouveau paquetage qui se trouve dans le répertoire `/main/calculator/`.

```
content,install,path,file:///main/calculator/
```

Vous pourrez noter également que des lignes du fichier installed-chrome.txt ont un autre type d'URL, le type *jar:*. Si vous compressez vos fichiers dans un fichier JAR, vous pouvez utiliser une URL JAR pour le référencer. Elle possède deux parties, séparées par un point d'exclamation (!). La première partie est l'URL du fichier JAR et la seconde partie est le répertoire ou le fichier dans l'archive. L'exemple ci-dessous pourrait référencer la boîte de dialogue de recherche de fichiers :

```
jar:resource:/chrome/findfile.jar!/content/findfile/
```

Cependant, vous n'aurez pas besoin normalement de faire attention aux URLs JAR quand vous créez vos propres paquetages. Vous garderez plutôt les paquetages décompressés et vous vous y référerez en utilisant les types d'URLs fichier ou ressource.

## Résumé

Les informations présentées ci-dessus peuvent avoir été confuses. Voici donc un guide rapide pour créer un simple paquetage. Vous pourrez suivre simplement les étapes du dessous et essayer de regarder les détails sur le fonctionnement de l'installation d'un paquetage une fois que vous serez plus familier avec XUL.

1. Créez un répertoire quelque part sur votre disque. Beaucoup de gens le mettent comme un sous-répertoire à l'intérieur du repertoire chrome de Mozilla, mais ce n'est pas nécessaire. Le répertoire peut être n'importe où sur le disque. Mettez vos fichiers XUL dans ce répertoire.
2. Créez un fichier appelé *contents.rdf* et placez le dans ce répertoire. Copiez le texte de la boîte du dessous dans ce nouveau fichier. Il est utilisé pour identifier l'identifiant (id) de votre application, son nom, son auteur, la version etc.

```
<?xml version="1.0"?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <RDF:Seq RDF:about="urn:mozilla:package:root">
    <RDF:li RDF:resource="urn:mozilla:package:myapplication"/>
  </RDF:Seq>

  <RDF:Description RDF:about="urn:mozilla:package:myapplication"
    chrome:displayName="Application Title"
    chrome:author="Author Name"
    chrome:name="myapplication"
    chrome:extension="true"/>

</RDF:RDF>
```

3. Changez les parties colorées du fichier du dessus avec vos propres informations. Le texte rouge *myapplication* doit être l'ID de votre application. Vous l'inventerez, mais typiquement l'ID est similaire au nom de votre application. Remplacez le texte coloré en bleu avec le titre et l'auteur de l'application.
4. Si le champ `chrome:extension` est à *true*, l'application est alors une extension Firefox et apparaîtra dans la fenêtre des extensions du navigateur. Si c'est *false*, elle n'apparaîtra pas.
5. Sauvez le fichier *contents.rdf* et soyez sûr qu'il est dans le répertoire créé lors de l'étape 1.
6. Ouvrez le fichier `<mozilla-directory>/chrome/installed-chrome.txt`, où `<mozilla-directory>` est le répertoire d'installation de Mozilla. Quittez Mozilla au préalable.
7. Ensuite nous allons déclarer la nouvelle application dans Mozilla ainsi il saura où la trouver. Ajoutez une ligne à la fin du fichier `installed-chrome.txt` pointant vers le nouveau répertoire que vous avez créé.

```
content,install,url,file:///main/app/.
```

Changez le texte coloré par l'URL de fichier du répertoire. Soyez sûr que l'URL se termine bien avec un slash et que vous avez pressé **enter** à la fin de la ligne. Si vous n'êtes pas sûr de l'URL, ouvrez le répertoire créé à l'étape 1 dans le navigateur et copiez l'URL à partir de la barre d'adresse. Notez que la référence doit être un répertoire, pas un fichier.

8. Effacez le fichier `<mozilla-directory>/chrome/chrome.rdf`.
9. Démarrer Mozilla. Vous devriez pouvoir voir n'importe quel fichier que vous mettez dans le répertoire en utilisant une URL de la forme `chrome://applicationid/content/fichier.xul`, où `fichier.xul` est le nom du fichier. Votre fichier principal xul doit être `applicationid.xul` ce qui vous permet de le charger en utilisant le raccourci `chrome://applicationid/content/`.

Si vous avez créé des thèmes ou des localisations, répétez les étapes précédentes, excepté que le format du fichier *contents.rdf* est légèrement différent. Regardez les fichiers *contents.rdf* d'autres applications pour plus de détails.

## Résolution de problèmes

La création d'un paquetage chrome peut souvent être délicat et il est difficile de diagnostiquer les problèmes. Voici quelques astuces si vous n'y arrivez pas.

- Ouvrez le fichier <mozilla-directory>/chrome/chrome.rdf. Vous devriez trouver des références à l'ID de votre application. Si ce n'est pas le cas, quelque chose s'est mal passé avec la déclaration. Si elles y sont, vous avez probablement utilisé une mauvaise URL chrome quand vous chargez le fichier.
- Essayez d'effacer le fichier <mozilla-directory>/chrome/chrome.rdf. Il sera régénéré. Effacez aussi tout le répertoire <mozilla-directory>/chrome/overlayinfo/ si vous utilisez les overlays.
- Soyez sûr que l'URL de la ligne que vous avez ajouté au fichier installed-chrome.txt se termine bien par un slash, et que le fichier se termine par une ligne vide.
- Sous windows, les URLs de fichier sont de la forme file:///C:/files/app/ où C est la lettre du lecteur
- Soyez sûr que le fichier contents.rdf est dans le bon répertoire et est bien formé. Ouvrez le fichier contents.rdf dans Mozilla pour voir si il l'analyse bien comme étant du XML sans erreur. Si ce n'est pas le cas, vous verrez une erreur sur un fond jaune.
- Si vous utilisez une version debug de Mozilla, certaines informations seront affichées dans le terminal au démarrage, indiquant quelles applications chrome ont été vérifiées. Vérifiez si votre application est listée.

---

Dans le prochain chapitre, nous commencerons l'examen du langage XUL.

## 2. Éléments simples

### 2.1 Créer une fenêtre

Écrit par Neil Deakin. Traduit par *Gnunix* (14/11/2003).

Page originale : <http://www.xulplanet.com/tutorials/xultu/window.html>

Nous allons créer un simple outil de recherche de fichiers tout au long de ce tutoriel. Avant tout, cependant, nous devons étudier la syntaxe de base d'un fichier XUL.

#### Création d'un fichier XUL

Un fichier XUL peut avoir n'importe quel nom mais il doit avoir l'extension `.xul`. Le fichier XUL le plus simple a la structure suivante :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window
  id="findfile-window"
  title="Recherche de fichiers"
  orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  ...
</window>
```

Cette fenêtre ne sert à rien tant qu'elle ne comporte pas d'éléments d'interface utilisateur. Ceux ci seront ajoutés dans la section suivante. Voici l'analyse ligne par ligne du code ci-dessus :

1. `<?xml version="1.0"?>`  
Cette ligne déclare simplement qu'il s'agit d'un fichier XML. Vous devrez normalement ajouter cette ligne à l'identique au début de chaque fichier xul, tout comme vous mettriez la balise HTML au début d'un fichier HTML.
2. `<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>`  
Cette ligne est utilisée pour spécifier la feuille de style à utiliser pour le fichier. C'est la syntaxe que les fichiers XML emploient pour importer des feuilles de style. Dans ce cas, nous importons les styles trouvés dans le répertoire chrome global/skin. Nous n'avons pas indiqué de fichier spécifique ainsi Mozilla déterminera quels fichiers du dossier il emploiera. Ici, le fichier le plus important, `global.css`, est sélectionné. Ce fichier contient toutes les déclarations par défaut pour tous les éléments XUL. Puisque XML ne permet pas de savoir la manière dont les éléments doivent être affichés, ce fichier l'indique. De façon générale, vous mettez cette ligne au début de chaque fichier XUL. Vous pouvez également importer d'autres feuilles de style en utilisant la même syntaxe. Notez que vous devrez normalement importer la feuille de style globale à l'intérieur du fichier de votre propre feuille de style.
3. `<window`  
Cette ligne déclare que vous allez décrire une fenêtre window. Chaque fenêtre d'interface utilisateur est décrite dans un fichier séparé. Cette balise ressemble à la balise BODY en HTML qui entoure la totalité du contenu. Plusieurs attributs peuvent être placés dans l'élément window (ici il y en a quatre). Dans cet exemple, chaque attribut est placé sur une ligne séparée bien que ce ne soit pas une obligation.
4. `id="findfile-window"`  
L'attribut `id` est utilisé comme un identifiant, de sorte que des scripts puissent y référer. Vous mettez normalement un attribut `id` sur tous les éléments. Vous êtes libre de choisir n'importe quel nom, toutefois, il est préférable qu'il soit pertinent.

5. `title="Recherche de fichiers"`

L'attribut `title` décrit le texte qui apparaîtra dans la barre de titre de la fenêtre quand elle sera affichée. Dans le cas présent, le texte `Recherche de fichiers` sera affiché.

6. `orient="horizontal"`

L'attribut `orient` spécifie l'arrangement des éléments de la fenêtre. La valeur *horizontal* indique que les éléments doivent être placés horizontalement dans la fenêtre. Vous pouvez également utiliser la valeur *vertical* signifiant que les items seront affichés en colonne. Comme c'est la valeur par défaut, vous pouvez omettre cet attribut si vous souhaitez avoir l'orientation verticale.

7. `xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">`

Cette ligne déclare l'espace de nommage pour XUL que vous devrez mettre dans l'élément `window` pour indiquer que tous ses enfants sont XUL. Notez que cette URL n'est jamais téléchargée réellement. Mozilla reconnaîtra cette URL en interne.

8. ...

C'est ici que les éléments (les boutons, les menus et les autres composants de l'interface utilisateur) sont déclarés. Nous en ajouterons quelques uns dans les prochaines sections.

9. `</window>`

Et enfin, nous devons fermer la balise `window` à la fin du fichier.

## Ouverture d'une fenêtre

Afin d'ouvrir une fenêtre XUL, il y a plusieurs méthodes qui peuvent être employées. Si vous n'êtes qu'à l'étape de développement, vous pouvez juste taper l'URL (commençant par `chrome:`, `file:` ou d'autre type d'URL) dans la barre d'adresses du navigateur Mozilla. Vous devriez également pouvoir double-cliquer sur le fichier dans votre gestionnaire de fichiers, en supposant que les fichiers XUL soient associés à Mozilla. La fenêtre XUL apparaîtra cependant dans la fenêtre de navigateur et non dans une nouvelle, mais c'est souvent suffisant durant les premières étapes de développement.

La manière correcte, naturellement, est d'ouvrir la fenêtre en utilisant JavaScript. Aucune nouvelle syntaxe n'est nécessaire, vous pouvez employer la fonction `window.open()` comme pour tout document HTML. Cependant, une option additionnelle, appelée *chrome* est nécessaire pour indiquer au navigateur que le document à ouvrir est un chrome. Celle-ci s'ouvrira sans barre d'outils, sans menu et sans aucun élément qu'une fenêtre de navigateur dispose normalement. La syntaxe est décrite ci-dessous :

```
window.open(url,windowname,flags);
```

où l'argument `flags` contient `"chrome"`.

Exemple :

```
window.open("chrome://navigator/content/navigator.xul", "bmarks",
"chrome,width=600,height=300");
```

## Votre premier fichier XUL

Commençons par créer le fichier de base pour la boîte de dialogue de recherche de fichiers. Créez un fichier appelé `findfile.xul` et mettez le dans un nouveau répertoire quelque part. L'emplacement importe peu, mais le répertoire `chrome/findfile/content` est un endroit adéquat. Ajoutez au fichier le squelette XUL montré au début de cette page et sauvegardez le.

Vous pouvez utiliser le paramètre en ligne de commande *-chrome* pour indiquer le fichier XUL à ouvrir au démarrage de mozilla. S'il n'est pas spécifié, la fenêtre par défaut s'ouvrira (Habituellement la fenêtre de navigateur). Par exemple, nous pourrions ouvrir la boîte de dialogue de recherche de fichiers avec l'une des commandes suivantes :

```
mozilla -chrome chrome://findfile/content/findfile.xul
mozilla -chrome resource:/chrome/findfile/content/findfile/findfile.xul
```

Si vous utilisez cette commande en ligne de commande (dans l'hypothèse que cela soit possible sur votre plate-forme), la boîte de dialogue de recherche de fichiers s'ouvrira par défaut au lieu de la fenêtre de navigateur de Mozilla. Naturellement, puisque nous n'avons pas mis d'éléments graphiques dans la fenêtre, vous ne verrez rien. Dans la section suivante, quelques éléments y seront ajoutés.

Pour voir l'effet, la commande suivante ouvre la fenêtre de signets :

```
mozilla -chrome chrome://communicator/content/bookmarks/bookmarksManager.xul
```

---

Dans la section suivante, nous ajouterons des boutons dans la fenêtre.

## 2.2 Ajouter des boutons

Écrit par Neil Deakin. Traduit par *Gnunix* (14/11/2003).

Page originale : <http://www.xulplanet.com/tutorials/xultu/buttons.html>

Dans cette section, nous regarderons comment ajouter quelques boutons simples dans une fenêtre.

### Ajouter des boutons dans une fenêtre

La fenêtre que nous avons créée jusqu'ici ne comprenait rien, aussi elle n'est pas encore très intéressante. Dans cette section, nous ajouterons deux boutons, un bouton de recherche et un bouton d'annulation. Nous apprendrons également une manière simple de les positionner dans la fenêtre.

Comme pour HTML, XUL dispose d'un certain nombre de balises qui peut être utilisé pour créer des éléments d'interface utilisateur. La plus basique de toute est la balise button. Cet élément est employé pour créer de simples boutons.

L'élément bouton a deux propriétés principales : un libellé et une image. Vous avez besoin soit de l'un, soit de l'autre soit des deux en même temps. Ainsi, un bouton peut avoir seulement un libellé, seulement une image ou un libellé et une image à la fois. Les boutons sont souvent utilisés pour valider ou annuler dans une boîte de dialogue, par exemple.

La balise button a la syntaxe suivante :

```
<button
  id="identifiant"
  class="dialog"
  label="OK"
  image="images/image.jpg"
  disabled="true"
  accesskey="t" />
```

Les attributs sont définis comme suit (ils sont tous optionnels) :

*id*

Identifiant unique vous permettant d'identifier le bouton. Vous verrez cet attribut sur tous les éléments. Vous pouvez l'utiliser si vous voulez vous référer au bouton dans une feuille de style ou dans un script. Cependant, vous devriez ajouter cet attribut à presque tous les éléments. Il ne l'est pas toujours tout le long de ce tutoriel, pour des raisons de simplicité.

*class*

La classe de style du bouton. Elle fonctionne de la même manière qu'en HTML. Elle est employée pour indiquer le style avec lequel le bouton apparaît. Dans le cas présent, la valeur *dialog* est utilisée. Dans la plupart des cas, vous n'emploieriez pas de classe pour un bouton.

*label*



Le libellé qui apparaîtra sur le bouton. Par exemple : *OK* ou *Annuler*. S'il est omis, aucun texte n'apparaît.

*image*

L'URL de l'image qui apparaît sur le bouton. Si cet attribut est omis, aucune image n'apparaîtra. Vous pouvez également indiquer l'image dans une feuille de style en utilisant la propriété `list-style-image`.

*disabled*

Si cet attribut prend la valeur *true*, le bouton est désactivé. Celui-ci est habituellement dessiné avec le texte écrit en gris. Si le bouton est désactivé, la fonction du bouton ne peut pas être exécutée. Si cet attribut est omis, le bouton est activé. Vous pouvez commuter l'état d'activation du bouton en utilisant Javascript.

*accesskey*

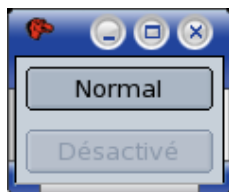
Cet attribut doit spécifier une lettre qui est employée comme raccourci clavier. Cette lettre doit apparaître dans le libellé et sera typiquement soulignée. Quand l'utilisateur appuie sur la touche **alt** (ou une touche semblable, qui varie en fonction de la plate-forme) et la touche de raccourci, à partir de n'importe où dans la fenêtre, le focus sera mis sur le bouton.

Notez qu'un bouton supporte plus d'attributs que ceux énumérés ci-dessus. Les autres seront vus plus tard.

Quelques exemples de boutons :

Exemple 2.2.1 :

```
<button label="Normal"/>
<button label="Désactivé" disabled="true"/>
```



Les exemples ci-dessus produiront des boutons comme dans la capture. Le premier

bouton est un bouton normal. Le deuxième bouton est désactivé aussi il apparaît grisé. L'image apparaît ici avec le thème modern. D'autres thèmes peuvent donner un rendu différent de ces boutons.

## Bouton dans la boîte recherche

Nous commencerons par créer un bouton simple de recherche qui permettra de lancer la recherche de fichiers. Le code de l'exemple ci-dessous nous montre comment faire :

```
<button id="find-button" label="Rechercher"/>
```

Si vous utilisez Firefox 1.0, notez qu'il ne vous est pas permis d'ouvrir des fenêtres chrome à partir de pages Web. Aussi le lien "voir" dans le tutoriel n'ouvrira pas une fenêtre de navigateur normale. À cause de cela, les boutons apparaîtront agrandis le long de la fenêtre. Vous pouvez ajouter `align="start"` sur la balise `window` pour éviter cette déformation.

Ajoutons ce code au fichier `findfile.xul` que nous avons créé dans la section précédente. Le code doit être inséré entre les balises `window`. Le code à ajouter est mis en rouge ci-dessous :

```
<?xml version="1.0" encoding="ISO-8859-1"?>

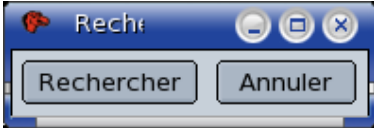
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
  id="findfile-window"
  title="Recherche de fichiers"
  orient="horizontal"
```

```
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<button id="find-button" label="Rechercher"/>
<button id="cancel-button" label="Annuler"/>

</window>
```

Vous noterez que le bouton *Annuler* a été également ajouté. La fenêtre a une orientation horizontale de sorte que les deux boutons apparaissent l'un à côté de l'autre. Si vous ouvrez le fichier dans Mozilla, vous devriez obtenir quelque chose comme l'image montrée ici.



Notez que nous ne devrions pas mettre le texte des libellés directement dans le fichier XUL. Nous devrions plutôt employer des entités de sorte que le texte puisse être facilement traduit.

L'exemple de la boîte de recherche :

---

Dans la section suivante, nous découvrirons comment ajouter des étiquettes et des images dans une fenêtre XUL.

## 2.3 Ajouter des libellés et des images

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (10/03/2004).  
 Page originale : <http://www.xulplanet.com/tutorials/xultu/textimage.html>

Cette section décrit la façon d'ajouter des libellés et des images dans une fenêtre.

### Éléments de texte

Vous ne pouvez pas écrire du texte directement dans un fichier XUL sans la placer entre des balises, et espérer qu'il s'affiche.

La façon la plus basique pour inclure du texte dans une fenêtre est d'utiliser l'élément description. Voici un exemple :

Exemple 2.3.1 :

```
<label value="ceci est du texte"/>
```

L'attribut *value* peut être utilisé pour spécifier le texte que vous voulez afficher. Le texte ne sera pas coupé, et donc apparaîtra sur une seule ligne. C'est adapté pour de courtes sections de textes.

Pour les textes plus long, vous pouvez placer le contenu à l'intérieur de balises description ouvrantes et fermantes. Contrairement au texte spécifié avec l'élément label et l'attribut *value*, le texte sera coupé en plusieurs lignes si nécessaire. Comme en HTML, les sauts de lignes et les espaces en trop seront regroupés en un seul espace. Plus tard nous allons voir comment définir la taille des éléments, et donc nous pourrons voir le découpage.

Exemple 2.3.2 :

```
<description>
  Cette longue section de texte est affichée.
```

```
</description>
```

Intrinsèquement, les deux éléments `label` et `description` sont les mêmes, ce qui signifie que les éléments `label` peuvent avoir du texte tronqué si celui-ci est placé à l'intérieur de balises, et que les éléments `description` peuvent avoir un attribut `value`. Les éléments `label` sont généralement utilisés pour des libellés, tels que ceux des champs de saisie. L'élément `description` est généralement employé pour d'autres textes descriptifs tels qu'une information placée en haut d'une boîte de dialogue. Par convention, vous devriez suivre ces règles.

Vous pouvez utiliser l'attribut `control` pour indiquer à quel contrôle il est associé. Quand l'utilisateur cliquera sur le libellé, le contrôle aura le focus. Spécifiez la valeur de l'attribut `control` avec l' `id` de l'élément qui aura le focus.

Exemple 2.3.3 :

```
<label value="Cliquez ici" control="open-button" />
<button id="open-button" label="Ouvrir"/>
```

Dans l'exemple du dessus, en cliquant sur le libellé, le focus sera mis sur le bouton.

## Les images

Comme en HTML, XUL a un élément pour afficher des images à l'intérieur d'une fenêtre. Cet élément est nommé naturellement `image`. Notez que le nom de la balise est différent de celle en HTML (`image` au lieu de `img`). Vous pouvez utiliser l'attribut `src` pour indiquer l'URL de l'image. L'exemple suivant le montre :

```
<image src="images/banner.jpg" />
```

Bien que vous puissiez utiliser cette syntaxe, il est préférable d'utiliser une feuille de style pour spécifier une URL. Une section ultérieure vous décrira comment utiliser des feuilles de styles, mais l'exemple ci-dessous vous permettra d'appréhender le sujet. Vous pouvez utiliser le style CSS `list-style-image` pour définir l'URL de l'image. Voici des exemples :

XUL :

```
<image id="image1"/>
<image id="search"/>
```

Feuille CSS :

```
#image1 {
    list-style-image : url(chrome://findfile/skin/banner.jpg);
}
#search {
    list-style-image: url(chrome://findfile/skin/images/search.jpg);
}
```

Ces images proviennent du répertoire `chrome`, dans le thème du paquetage `findfile`. Parce que les images varient selon les thèmes, vous devriez normalement mettre les images dans le répertoire `skin`.

---

Dans la section suivante, nous apprendrons comment ajouter des contrôles de saisie à une fenêtre.

## 2.4 Les champs de saisie

Écrit par Neil Deakin. Traduit par *Benoit Salandre* (14/01/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/inputs.html>

XUL possède des éléments qui sont similaires aux champs de formulaires du HTML.

## Les champs de saisie de texte

Le HTML possède une balise `input` pouvant être utilisée comme champ de saisie de texte. XUL possède un élément similaire, `textbox`, utilisé comme champ de saisie de texte. Sans aucun attribut, l'élément `textbox` crée une boîte dans laquelle l'utilisateur peut entrer du texte. Les boîtes de texte acceptent plusieurs attributs similaires à la balise `input` de l'HTML. En voici quelques uns :

*id*

Un identifiant unique permettant d'identifier la boîte de texte.

*class*

La classe du style de la boîte de texte.

*value*

Si vous voulez donner une valeur par défaut à la boîte de texte, renseignez l'attribut `value`.

*disabled*

Mettez la valeur `true` pour empêcher l'insertion de texte.

*type*

Vous pouvez renseigner cet attribut avec la valeur spéciale `password` pour créer une boîte de texte dans laquelle tout ce qui est saisi est caché. Il est utilisé habituellement pour les champs de saisie de mot de passe.

*maxlength*

Le nombre maximum de caractères que l'on peut saisir dans la boîte de texte.

Notez que, tandis qu'en HTML différentes sortes de champs peuvent être créées avec l'élément `input`, il y a un élément différent pour chaque type de champs en XUL.

L'exemple suivant montre quelques boîtes de texte :

Exemple 2.4.1 :

```
<label control="some-text" value="Entrez du texte"/>
<textbox id="some-text"/>
<label control="some-password" value="Entrez un mot de passe"/>
<textbox id="some-password" type="password" maxlength="8"/>
```

Les exemples de boîtes de texte ci-dessus créent des champs de saisie qui ne peuvent être utilisés que pour la saisie d'une seule ligne de texte. Le HTML possède aussi un élément `textarea` pour créer une zone de saisie plus grande. Dans XUL, vous pouvez utiliser l'élément `textbox` de la même façon — deux éléments distincts ne sont pas nécessaires. Si vous mettez l'attribut `multiline` à la valeur `true`, le champ de saisie de texte affichera plusieurs lignes.

Par exemple :

Exemple 2.4.2 :

```
<textbox multiline="true"
  value="Voici du texte qui pourrait s'étaler sur plusieurs lignes."/>
```

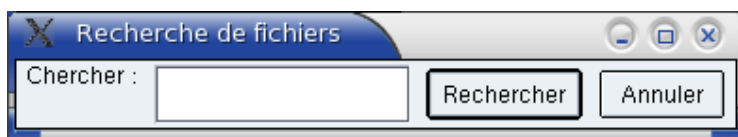
Comme avec la balise `textarea` du HTML, vous pouvez utiliser les attributs `rows` et `cols` pour définir la taille. Ils vous permettent d'ajuster le nombre de lignes et de colonnes de caractères à afficher.

Maintenant, ajoutons un champ de saisie de recherche à la boîte de dialogue de recherche de fichiers. Nous allons utiliser l'élément `textbox`.

```
<label value="Chercher :" control="find-text"/>
<textbox id="find-text"/>
```

```
<button id="find-button" label="Rechercher" />
```

Ajoutez ces lignes avant les boutons que nous avons créés dans la section précédente. Si vous ouvrez cette fenêtre, vous verrez quelque chose ressemblant à l'image ci-dessous.



Notez que l'étiquette et le champ de saisie de

texte sont maintenant apparus dans la fenêtre. La boîte de texte est complètement fonctionnelle et vous pouvez taper et sélectionner du texte dedans. Notez aussi que l'attribut `control` a été utilisé. Ainsi, le champ de saisie est sélectionnée lorsque l'on clique sur son libellé.

## Les cases à cocher et les boutons radio

Deux éléments supplémentaires sont utilisés pour la création de cases à cocher et de boutons radio. Ce sont des variantes de boutons simples. La case à cocher est utilisée pour les options qui peuvent être activées ou désactivées. Les boutons radio peuvent être utilisés dans un but similaire quand il y a un ensemble de boutons où un seul d'entre eux peut-être sélectionné à la fois.

Vous pouvez employer la plupart des attributs des boutons avec les cases à cocher et les boutons radio. L'exemple ci-dessous montre des cases à cocher et boutons radio simples.

```
<checkbox id="case-sensitive" checked="true" label="Sensible à la casse" />
<radio id="orange" label="Orange" />
<radio id="violet" selected="true" label="Violet" />
<radio id="yellow" label="Jaune" />
```

La première ligne crée une simple case à cocher. Quand l'utilisateur clique sur la case à cocher, son état coché et décoché s'inverse. L'attribut `checked` peut être utilisé pour indiquer l'état par défaut. Il vous faudra l'initialiser soit à *true*, soit à *false*. L'attribut `label` peut être utilisé pour ajouter un libellé qui apparaîtra à côté de la boîte à cocher. Pour les boutons radio, vous utiliserez l'attribut `selected` à la place de l'attribut `checked`. Initialisez le à *true* pour avoir un bouton radio sélectionné par défaut, ou ne le mettez pas pour les autres boutons radio.

Pour grouper des boutons radio, vous avez besoin d'utiliser l'élément `radiogroup`. Un seul bouton radio peut-être sélectionné dans un `radiogroup`. Cliquer sur l'un deux désélectionnera tous les autres du même groupe. C'est ce que démontre l'exemple suivant.

Exemple 2.4.3 :

```
<radiogroup>
  <radio id="orange" label="Orange" />
  <radio id="violet" selected="true" label="Violet" />
  <radio id="Jaune" label="Jaune" />
</radiogroup>
```

Comme pour les boutons, les cases à cocher et les boutons radio sont constitués d'un libellé et d'une image, où l'image change en fonction de l'état coché ou décoché. Les cases à cocher ont certains attributs communs aux boutons :

*label*

Le libellé de la case à cocher ou du bouton radio.

*disabled*

Mettez la valeur à *true* ou à *false* pour désactiver ou activer la case à cocher ou le bouton radio.

*accesskey*

La touche clavier pouvant être utilisée comme raccourci pour sélectionner l'élément. La lettre spécifiée est habituellement soulignée dans le libellé.

Dans la prochaine section, nous allons voir quelques éléments pour la création de boîtes de listes.

## 2.5 Les contrôles de listes

Écrit par Neil Deakin. Traduit par *Alain B.* (06/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/lists.html>

XUL a de nombreux types d'éléments pour créer des boîtes de listes.

### Zones de listes

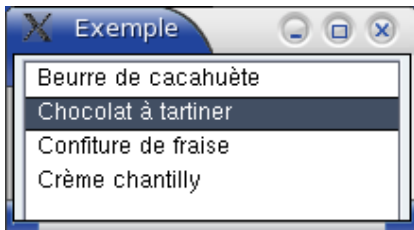
Une zone de liste est utilisée pour afficher un nombre d'items dans une liste. L'utilisateur doit pouvoir choisir un item dans la liste.

XUL propose deux types d'éléments pour créer des listes, un élément listbox pour créer des zones de liste multi-lignes, et un élément menulist pour créer des zones de listes déroulantes. Ils fonctionnent de la même manière que l'élément `select` du langage HTML, qui réalise ces deux fonctions, mais les éléments XUL ont plus de fonctionnalités.

La zone de liste la plus simple utilise l'élément listbox pour créer la boîte, et l'élément listitem pour chacun des items. Par exemple, cette zone de liste a quatre lignes, une pour chaque item.

Exemple 2.5.1 :

```
<listbox>
  <listitem label="Beurre de cacahuète"/>
  <listitem label="Chocolat à tartiner"/>
  <listitem label="Confiture de fraise"/>
  <listitem label="Crème chantilly"/>
</listbox>
```



À l'instar de la balise `option` du langage HTML, vous pouvez assigner

une valeur à chaque item en utilisant l'attribut `value`. Vous pouvez ensuite utiliser cette valeur dans un script. Par défaut, la zone de liste ajustera sa taille à son contenu, mais vous pouvez contrôler sa taille avec l'attribut `rows`. Affectez lui le nombre de lignes que vous souhaitez voir apparaître dans la zone de liste. Un ascenseur apparaîtra pour permettre à l'utilisateur d'afficher les lignes supplémentaires.

L'exemple suivant montre ces fonctionnalités supplémentaires :

Exemple 2.5.2 :

```
<listbox rows="3">
  <listitem label="Beurre de cacahuète" value="becacah"/>
  <listitem label="Chocolat à tartiner" value="chotart"/>
  <listitem label="Confiture de fraise" value="confraise"/>
  <listitem label="Creme chantilly" value="crchant"/>
</listbox>
```

L'exemple a été modifié pour n'afficher que 3 lignes à la fois. Des valeurs ont également été ajoutées à chaque item de la liste. Les zones de liste ont beaucoup d'autres fonctionnalités qui seront décrites plus tard.

## Boîte de liste multi-colonnes

Les boîtes de liste supportent également les colonnes multiples. Chaque cellule peut avoir un contenu arbitraire bien qu'il s'agisse souvent seulement de texte. Quand un utilisateur sélectionne un item dans la liste, la ligne entière est sélectionnée. Vous ne pouvez pas avoir une seule cellule sélectionnée.

Deux balises sont utilisées pour spécifier les colonnes dans une boîte de liste. L'élément `listcols` est utilisé pour contenir l'information sur les colonnes, chacune d'elles est spécifiée en utilisant un élément `listcol`. Vous aurez besoin d'un élément `listcol` pour chaque colonne de la liste.

L'élément `listcell` doit être utilisé pour chaque cellule dans une ligne. Si vous voulez avoir trois colonnes, vous aurez besoin d'ajouter trois éléments `listcell` à l'intérieur de chaque `listitem`. Pour spécifier le contenu d'un texte de la cellule, placez un attribut `label` sur cet élément `listcell`. Dans le cas simple où il n'y a qu'une seule colonne, vous pouvez aussi placer l'attribut `label` directement sur l'élément `listitem` et omettre complètement les éléments `listcell`, comme il est montré dans les exemples de listes précédents.

L'exemple suivant est une boîte de liste de deux colonnes et trois lignes.

Exemple 2.5.3 :

```
<listbox>
  <listcols>
    <listcol/>
    <listcol/>
  </listcols>
  <listitem>
    <listcell label="George" />
    <listcell label="Peintre en bâtiment" />
  </listitem>
  <listitem>
    <listcell label="Mary Ellen" />
    <listcell label="Fabriquant de bougies" />
  </listitem>
  <listitem>
    <listcell label="Roger" />
    <listcell label="Bravache" />
  </listitem>
</listbox>
```

## Lignes d'en-tête

Les boîtes de liste permettent également l'utilisation d'une ligne d'en-tête spéciale. Elle n'est rien d'autre qu'une ligne normale, sauf qu'elle est affichée différemment. Elle servira à créer des en-têtes de colonnes grâce à deux nouveaux éléments.

L'élément `listhead` est utilisé pour définir la ligne d'en-tête, comme l'élément `listitem` est utilisé pour définir une ligne normale. La ligne d'en-tête n'est toutefois pas une ligne normale, ainsi un script permettant de lire la première ligne d'une boîte de liste ne tiendra pas compte de cette ligne d'en-tête.

L'élément `listheader` est utilisé pour chaque cellule de l'en-tête. Utilisez l'attribut `label` pour définir le libellé de la cellule d'en-tête.

Voici l'exemple précédent avec une ligne d'en-tête :

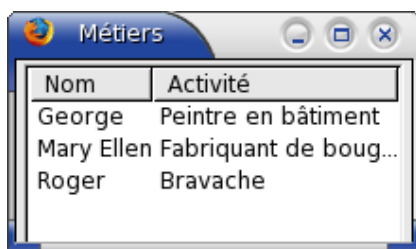
## Exemple 2.5.4 :

```

<listbox>
  <listhead>
    <listheader label="Nom" />
    <listheader label="Activité" />
  </listhead>

  <listcols>
    <listcol />
    <listcol flex="1" />
  </listcols>
  <listitem>
    <listcell label="George" />
    <listcell label="Peintre en bâtiment" />
  </listitem>
  <listitem>
    <listcell label="Mary Ellen" />
    <listcell label="Fabriquant de bougies" />
  </listitem>
  <listitem>
    <listcell label="Roger" />
    <listcell label="Bravache" />
  </listitem>
</listbox>

```



Dans cet exemple, l'attribut `flex` est utilisé pour rendre la colonne

flexible. Cet attribut sera décrit dans une section ultérieure, mais ici, cela permet à la colonne de remplir tout l'espace disponible horizontalement. Vous pouvez retailer la fenêtre pour voir que la colonne s'adapte en même temps que la fenêtre. Si vous diminuez la taille horizontalement, les libellés sur les cellules seront raccourcis automatiquement, en affichant des points de suite. Vous pouvez utiliser l'attribut `crop` sur les cellules ou sur les items, avec une valeur *none* pour désactiver cette coupure.

## Zones de listes déroulantes

Des listes déroulantes peuvent être créées en HTML avec la balise `select`. L'utilisateur ne voit qu'un unique choix dans une boîte de texte et doit cliquer sur une flèche ou un bouton similaire à côté de la boîte de texte pour changer la sélection. Les autres choix apparaîtront alors dans une fenêtre surgissante. XUL dispose de l'élément `menulist` qui peut être utilisé pour cet usage. Il est formé d'une boîte de texte avec un bouton à son côté. Son nom a été choisi parce qu'il propose un menu surgissant pour le choix.

Trois éléments sont nécessaires pour former une liste déroulante. Le premier est l'élément `menulist` qui crée la boîte de texte avec son bouton à côté. Le second, `menupopup`, crée la fenêtre surgissante qui apparaît lorsque le bouton est cliqué. Le troisième, `menuitem`, crée les choix individuels.

Cette syntaxe est mieux décrite dans l'exemple ci-dessous :

## Exemple 2.5.5 :

```

<menulist label="Bus">
  <menupopup>
    <menuitem label="Voiture" />
    <menuitem label="Taxi" />
  </menupopup>
</menulist>

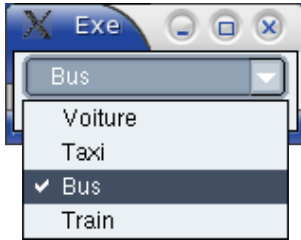
```



```

    <menuitem label="Bus" selected="true" />
    <menuitem label="Train" />
</menupopup>
</menulist>

```



Ce menu contient quatre choix, un pour chaque élément menuitem. Pour voir les

choix possibles, cliquez sur le bouton flèche du menu. Lorsqu'un item est sélectionné, il apparaît comme le choix fait dans le texte du menu. L'attribut `selected` est utilisé pour spécifier la valeur sélectionnée par défaut.

Par défaut, vous ne pouvez sélectionner qu'un seul choix proposé dans la liste. Vous ne pouvez pas entrer votre propre sélection en la tapant au clavier, comme vous pouvez le faire dans certaines listes déroulantes. Une variante de menulist le permet. Par exemple, le champ URL du navigateur a une liste déroulante pour afficher les adresses précédemment utilisées, mais vous pouvez aussi la saisir vous même.

Pour créer une liste déroulante éditable, ajoutez l'attribut `editable` comme ceci :

Exemple 2.5.6 :

```

<menulist editable="true">
  <menupopup>
    <menuitem label="www.mozilla.org" />
    <menuitem label="www.xulplanet.com" />
    <menuitem label="www.dmoz.org" />
  </menupopup>
</menulist>

```

Le champ URL créé ici a trois adresses pré-saisies que l'utilisateur peut sélectionner ou alors il peut taper la sienne dans le champ. Le texte entré par l'utilisateur ne sera pas ajouté comme un futur choix à cette liste. Comme l'attribut `label` n'a pas été utilisé dans cet exemple, sa valeur par défaut sera vide.

---

Dans la section suivante, nous apprendrons comment ajouter des indicateurs de progression.

## 2.6 Indicateurs de progression

Écrit par Neil Deakin. Traduit par **Alain B.** (08/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/progress.html>



Dans cette section, nous allons regarder la création d'indicateurs de progression.

### Ajout d'un indicateur de progression

Un indicateur de progression est une barre qui indique l'état d'avancement d'une tâche. Typiquement, vous la voyez lors du téléchargement de fichiers ou quand une longue opération s'exécute. XUL a un élément qui peut être utilisé pour la création d'un indicateur de progression (NdT : progressmeter). Il y a deux types d'état pour un indicateur : déterminé et indéterminé.

Les indicateurs de progression déterminés sont utilisés quand vous connaissez la durée d'une opération. L'indicateur de progression va se remplir, et une fois plein, l'opération doit être terminée. Il peut être utilisé pour une boîte de dialogue de téléchargement lorsque la taille des fichiers est connue.

Les indicateurs de progression indéterminés sont utilisés quand vous ne connaissez pas la durée d'une opération. Une animation visuelle représentera cet indicateur, telle qu'un rectangle zébré ou un cylindre hachuré tournant, et sera dépendante de votre système d'exploitation et du thème graphique utilisé.

Indicateur de progression déterminé :   
Indicateur de progression indéterminé : 

Un indicateur de progression a la syntaxe suivante :

```
<progressmeter
  id="identifiant"
  mode="determined"
  value="0%" />
```

Ses attributs sont les suivants :

*id*

L'identifiant unique de l'indicateur de progression.

*mode*

Le type d'indicateur de progression. Si sa valeur est *determined*, l'indicateur de progression est déterminé et se remplit au fur et à mesure de la tâche en cours. Si sa valeur est *undetermined*, l'indicateur de progression est indéterminé et vous ne connaissez pas la durée de la tâche en cours. La valeur par défaut est *determined* si vous n'avez pas spécifié cet attribut.

*value*

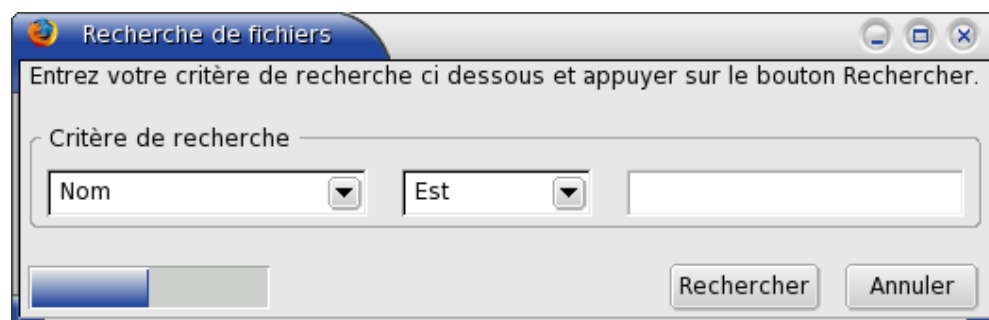
La valeur courante de l'indicateur de progression. Vous ne devez utiliser cet attribut que pour un indicateur de progression déterminé. La valeur doit être un pourcentage compris entre *0%* et *100%*. La valeur sera changée par un script selon l'avancement de la tâche.

Ajoutons maintenant un indicateur de progression à notre boîte de dialogue 'Recherche de fichiers'. Nous devrions normalement mettre un indicateur de progression indéterminé car nous ne connaissons pas le nombre de fichiers recherchés ou combien de temps prendra la recherche. Toutefois, nous ajouterons un indicateur normal pour l'instant car un indicateur animé risque d'être perturbant lors du développement. L'indicateur de progression devrait normalement n'apparaître que lorsque la recherche est lancée. Nous ajouterons plus tard un script pour l'afficher ou non.

```
<hbox>
```

```
  <progressmeter value="50%" style="margin: 4px;"/>
```

```
  <spacer flex="1"/>
```



La valeur a été mise à *50%* afin que vous puissiez voir la barre de progression dans la fenêtre. Une marge de 4 pixels a été définie pour séparer l'indicateur du bord de la fenêtre. Comme nous l'avons mentionné précédemment, nous voulons simplement que la barre de progression soit affichée pendant la recherche. Un script l'affichera et la masquera si nécessaire.

Exemple 'Recherche de fichiers' :

---

Nous allons voir dans la prochaine section comment ajouter des éléments additionnels à une fenêtre en utilisant HTML.

## 2.7 Ajout d'éléments HTML

Écrit par Neil Deakin. Traduit par *Alain B.* (07/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/htmllem.html>

Maintenant que nous avons ajouté quelques boutons, ajoutons d'autres éléments.

### Ajout d'éléments HTML dans une fenêtre

En plus de tous les éléments XUL déjà disponibles, vous pouvez également ajouter des éléments HTML directement dans un fichier XUL. En réalité, vous pouvez ajouter n'importe quels éléments HTML dans des fichiers XUL, ce qui signifie que des applets Java ou des tableaux peuvent être placés directement dans une fenêtre. Vous devriez proscrire l'emploi de ces éléments HTML dans vos fichiers XUL si vous le pouvez. Toutefois, cette section va néanmoins décrire comment les employer. Souvenez vous que le langage XML est sensible à la casse, et que de ce fait vous devrez taper les balises et les attributs en minuscules.

Afin de pouvoir utiliser des éléments HTML dans un fichier XUL, vous devez déclarer l'espace de nom XHTML correspondant (NdT : namespace). De cette façon, Mozilla peut faire la distinction entre les balises de HTML de celles de XUL. L'attribut ci-dessous doit être ajouté à la balise window du fichier XUL ou sinon à tous les éléments HTML.

```
xmlns:html="http://www.w3.org/1999/xhtml"
```

Il s'agit d'une déclaration HTML comme celle que nous avons utilisé pour déclarer XUL. Elle doit être saisie exactement comme indiquée pour fonctionner.

Notez qu'en réalité, Mozilla ne télécharge pas cette URL, mais il la reconnaît comme étant une déclaration HTML.

Voici un exemple qui peut être ajouté pour notre fenêtre de *Recherche de fichiers* :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
  id="findfile-window"
  title="Recherche de fichiers"
  orient="horizontal"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Ensuite, vous pouvez employer des balises HTML comme vous le feriez normalement, en gardant à l'esprit ceci :

- Vous devez ajouter un préfixe "html:" au début de chaque balise, en supposant que vous ayez déclaré l'espace de nom HTML comme ci-dessus.
- Vous devez taper les balises en minuscules.
- Des guillemets doivent être placés autour de toutes les valeurs d'attributs.
- XML impose qu'une barre oblique (slash) termine chaque balise qui n'ont aucun contenu. Les exemples ci-dessous vont vous éclairer la dessus.

Vous pouvez utiliser n'importe quelles balises HTML bien que head et body ne soient d'aucune utilité. Des exemples d'emploi des éléments HTML sont montrés ci-dessous :

```
<html:img src="banner.jpg" />

<html:input type="checkbox" value="true" />

<html:table>
  <html:tr>
    <html:td>
      Un tableau simple
    </html:td>
  </html:tr>
</html:table>
```

Ces exemples vont créer une image à partir du fichier banner.jpg, une case à cocher et un tableau avec une unique cellule. Vous devriez toujours employer des éléments XUL s'ils sont disponibles et vous ne devriez pas employer des tableaux pour la mise en page sous XUL. (il y a des éléments XUL pour faire la mise en page).

Notez que le préfixe html: a été ajouté au début de chaque balise, ceci pour que Mozilla sache qu'il s'agit d'une balise HTML et non XUL. Si vous oubliez un préfixe html:, le navigateur croira qu'il s'agit d'un élément XUL et il n'affichera probablement rien car les balises img, input, table, etc. ne sont pas des balises XUL valides.

Sous XUL, vous pouvez ajouter des libellés avec l'élément description ou label. Vous devriez utiliser ces éléments lorsque vous le pouvez. Vous pouvez également ajouter des libellés à des contrôles soit en utilisant la balise HTML label, soit en mettant simplement le texte dans un autre élément bloc HTML comme dans l'exemple ci-dessous.

Exemple 2.7.1 :

```
<html:p>
  Rechercher :
  <html:input id="find-text" />
  <button id="okbutton" label="OK" />
</html:p>
```

Ce code va afficher le texte *Rechercher* :, suivi d'un champ de saisie de texte et d'un bouton OK. Notez que le bouton XUL peut apparaître dans un bloc de balise HTML, comme c'est le cas ici. Du texte ne sera affiché que s'il est placé à l'intérieur de balises HTML qui sont normalement employées pour afficher du texte (comme une balise p). Tout texte placé en dehors de balises ne sera pas affiché, sauf si l'élément XUL contenant le texte le permet (l'élément description par exemple).

Les exemples suivant vont vous aider.

## Exemple d'éléments HTML

Ce qui suit sont quelques exemples d'ajout de balises HTML dans une fenêtre. À chaque fois, l'élément window et d'autres informations basiques ont été retirés pour plus de simplicité.

### 1. Une boîte de dialogue avec une case à cocher

Exemple 2.7.2 :

```
<html:p>
  Cliquez sur la boîte ci-dessous pour mémoriser cette décision.
```

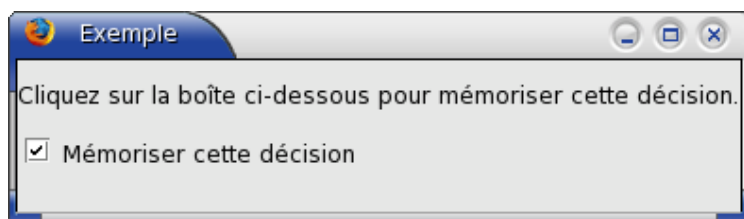
### 2. Éléments simples

```

<html:p>
  <html:input id="rtd" type="checkbox"/>
  <html:label for="rtd">Mémoriser cette décision</html:label>
</html:p>
</html:p>

```

Dans ce cas, la balise `p` a été utilisée pour placer du texte et une autre imbriquée a été utilisée pour séparer le texte en plusieurs lignes. (NdT : selon la norme XHTML, une balise `p` ne peut pas inclure une autre balise `p`, l'exemple est donc en principe erroné.)



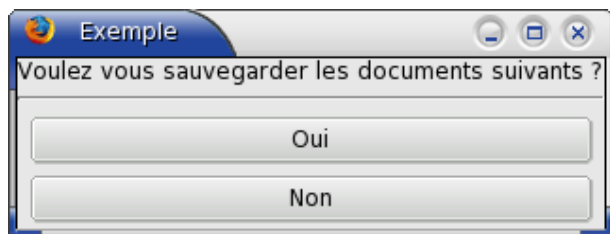
## 2. Texte en dehors de blocs HTML

Exemple 2.7.3 :

```

<html:div>
  Voulez vous sauvegarder les documents suivants ?
  <html:hr/>
</html:div>
Rapport de dépense 1
que j'ai fait l'été dernier
<button id="yes" label="Oui"/>
<button id="no" label="Non"/>

```



Comme vous pouvez le voir sur cette image, le texte placé à l'intérieur des balises `div` a été affiché mais l'autre texte (*Rapport de dépense 1* et *que j'ai fait l'été dernier*) ne l'a pas été. Ceci est dû au fait qu'il n'y a pas d'éléments HTML ou XUL entourant le texte à afficher et capables de le faire. Pour faire apparaître ce texte, vous devez le placer à l'intérieur de balises `div`, ou l'inclure dans une balise description.

## 3. Les éléments HTML invalides

```

<html:po>Cas 1</html:po>
<div>Cas 2</div>
<html:description value="Cas 3"/>

```

Aucun des trois cas ci-dessus ne s'affichera pour les raisons suivantes :

*Cas 1 :*

`po` n'est pas une balise HTML valide et Mozilla n'a aucune idée de ce qu'il faut faire avec.

*Cas 2 :*

`div` est une balise valide mais seulement en HTML. Pour qu'elle fonctionne, vous devez ajouter le qualificatif `html:`.

*Cas 3 :*

un élément description est seulement valide en XUL et pas en HTML. Il ne devrait pas avoir de qualificateur html:.

---

Dans la page suivante, nous apprendrons comment ajouter de l'espace entre les éléments.

## 2.8 Utilisation des spacers

Écrit par Neil Deakin. Traduit par *Alain B.* (07/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/springs.html>

Dans cette section, vous trouverez comment ajouter des espacements entre les différents éléments que vous avez créés.

### Ajout de spacers

Un des problèmes avec le développement des interfaces utilisateur est que chaque utilisateur a son propre affichage. Certains utilisateurs peuvent avoir de grands écrans avec une haute résolution et d'autres de faibles résolutions. De plus, chaque plate-forme peut avoir des spécificités pour l'interface utilisateur. En ajoutant le support multi-langue, un texte dans une langue peut avoir besoin de plus de place que dans une autre.

Les applications qui doivent être compatibles avec différentes plates-formes ou langages ont souvent des fenêtres créées avec beaucoup de place pour permettre leur exécution. Certaines plates-formes et outils de développement d'interfaces utilisateur proposent des composants qui sont suffisamment intelligents pour se redimensionner et se repositionner eux-mêmes selon les besoins de l'utilisateur (Java utilise des gestionnaires de mise en page par exemple).

XUL fournit la possibilité aux éléments de se positionner et se redimensionner automatiquement. Comme nous l'avons vu, la fenêtre de "recherche de fichiers" est apparue avec une taille qui correspond aux éléments qui y sont inclus. Chaque fois que nous ajoutons quelque chose, la fenêtre s'agrandit.

XUL utilise un système de mise en page appelé le " Box Model ". Nous en parlerons dans la prochaine section mais il vous permet essentiellement de diviser une fenêtre en une série de boîtes contenant vos éléments. Ces boîtes se positionnent et se redimensionnent en fonction des spécifications que vous avez définies. Pour l'instant, sachez simplement que l'élément window est un type de boîte.

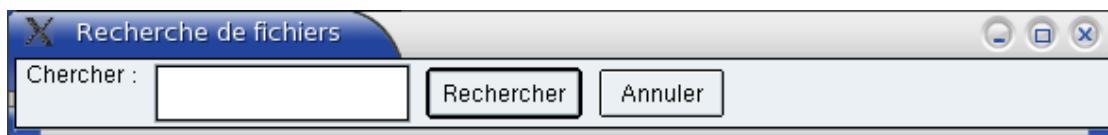
Avant d'entrer plus dans le détail avec les boîtes, nous allons présenter un autre élément XUL qui se montre très utile pour la mise en page, le spacer. Un spacer est très simple et ne requiert qu'un seul attribut qui sera expliqué dans un moment. Le spacer le plus simple ressemble à ceci :

```
<spacer flex="1" />
```

Un spacer est utilisé pour placer des espaces dans une fenêtre. Sa principale caractéristique est de pouvoir s'agrandir ou se rétrécir lorsque l'utilisateur redimensionne la fenêtre. Il permet à quelqu'un de placer des boutons à droite ou en bas d'une fenêtre et de vouloir les fixer sur le côté droit ou en bas, quelle que soit la taille de la fenêtre. Comme vous le verrez, vous pouvez utiliser des séries de spacer pour créer quelques effets de mise en page.

Dans la syntaxe ci-dessus, spacer a un seul attribut appelé flex. Il est utilisé pour définir la "flexibilité" de l'espacement. Dans le cas ci-dessus, le spacer a un flex de 1. Il en devient un élément d'espacement flexible. Si vous placez un spacer directement dans une fenêtre, il s'adaptera à la taille de la fenêtre lorsque celle-ci est modifiée.

Nous allons bientôt ajouter un élément spacer dans notre boîte de dialogue "Recherche de Fichiers". Tout d'abord, voilà ce qu'il se passe si la fenêtre est agrandie :



Si vous changez la taille de la fenêtre, vous pouvez voir que les éléments ont conservé leur position. Aucun d'eux n'a bougé ou s'est redimensionné bien que la fenêtre dispose de plus de place. Regardons maintenant ce qu'il se passe si un élément `spacer` est ajouté entre la boîte de texte et le bouton *Rechercher* :



En ajoutant un `spacer` et en agrandissant la fenêtre, vous voyez que le `spacer` s'est agrandi de façon à remplir l'espace libre. Les boutons ont été repoussés. Le code pour ajouter un `spacer` est indiqué juste après. Insérez le juste avant le bouton *Rechercher* :

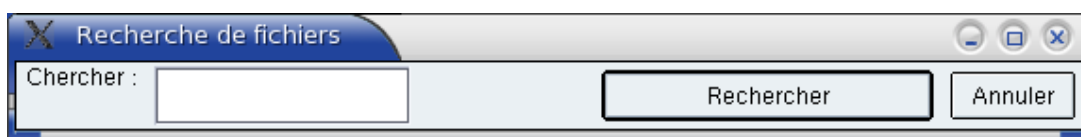
```
<spacer flex="1"/>
```

```
<button id="find-button" label="Rechercher"/>
```

## Plus d'informations sur la flexibilité

XUL place les éléments d'une fenêtre en calculant les largeurs et les hauteurs adéquates des éléments et ajoute ensuite des espacements là où ils sont flexibles. À moins que vous ne spécifiez la largeur et la hauteur d'un élément, la dimension par défaut de cet élément sera déterminée par son contenu. Vous noterez que le bouton *Annuler* des boîtes de dialogue a toujours adapté sa largeur pour contenir son texte. Si vous créez un bouton avec un très long libellé, la taille par défaut de ce bouton sera assez large pour contenir le libellé en entier. Les autres éléments, tels que la boîte de texte, se sont adaptés de façon adéquate.

L'attribut `flex` est utilisé pour spécifier si un élément peut changer sa dimension pour s'ajuster à sa boîte le contenant (dans notre cas, la fenêtre `window`). Nous avons montré l'application de l'attribut `flex` sur les `spacer`, mais il peut s'appliquer à n'importe quel élément. Par exemple, vous pouvez souhaiter avoir plutôt un redimensionnement du bouton *Rechercher* :



Comme vous pouvez voir sur l'image, en plaçant un attribut `flex` sur le bouton *Rechercher*, celui-ci s'est agrandi en même temps que la fenêtre. Un `spacer` n'a rien de spécial. Il peut être considéré comme un bouton invisible. Il fonctionne de la même manière qu'un bouton, excepté qu'il ne se dessine pas à l'écran.

Vous aurez remarqué quelque chose sur l'image ci-dessus. Il n'y a pas que le bouton *Rechercher* qui s'agrandit mais un espacement est également apparu entre le texte à gauche et le bouton. Bien entendu, il s'agit du `spacer` que nous avons placé tout à l'heure. Il s'est redimensionné de lui-même également. Si vous regardez suffisamment de près, vous devriez remarquer que ce changement de taille a été partagé en part égale entre le `spacer` et le bouton. Le `spacer` a reçu la moitié de l'espace libre et le bouton a reçu l'autre moitié.

La raison de ce comportement est que le `spacer` et le bouton ont chacun un attribut `flex`. Parce qu'ils sont flexibles, le `spacer` et le bouton se redimensionnent équitablement.

Comment faire si vous voulez qu'un élément s'agrandisse deux fois plus qu'un autre ? Vous pouvez choisir un numéro plus grand pour la valeur de l'attribut `flex`. Les valeurs de l'attribut `flex` sont des ratios. Si un élément a un `flex` de 1 et un second un `flex` de 2, le second s'agrandira du double par rapport au premier. En effet, un `flex` de 2 signifie que cet élément a une flexibilité de deux fois celle d'un élément de `flex` de 1.

L'attribut `flex` ne sert pas à définir une taille. Il spécifie au contraire comment se répartissent les espaces vides entre les différents éléments fils d'une boîte. Nous aborderons les boîtes dans le prochain chapitre. Dès lors que les dimensions par défaut des éléments fils d'une boîte sont déterminées, les valeurs de flexibilité sont appliquées pour diviser l'espace restant dans la boîte. Par exemple, si la boîte fait 200 pixels de large, qu'elle contient deux boutons flexibles respectivement de 50 pixels et 90 pixels, il restera un espacement de 60 pixels. Si ces deux boutons ont une valeur de `flex` de 1, cet espacement sera divisé en deux moitiés égales de 30 pixels affectées à chacun d'eux. Si le second bouton voit sa valeur de flexibilité augmentée à 2, le premier bouton recevra 20 pixels d'espacement supplémentaire et le second en recevra 40 pixels.

L'attribut `flex` peut être placé sur n'importe quel élément, toutefois il n'a de sens que si cet élément est directement inclus dans une boîte. Donc, même si vous placez un attribut `flex` sur un élément HTML, il restera sans effet cet élément est situé dans un élément n'étant pas une boîte.

Regardons quelques exemples :

Exemple 1 :

```
<button label="Chercher" flex="1"/>
<button label="Annuler" flex="1"/>
```

Exemple 2 :

```
<button label="Chercher" flex="1"/>
<button label="Annuler" flex="10"/>
```

Exemple 3 :

```
<button label="Chercher" flex="2"/>
<button label="Remplacer"/>
<button label="Annuler" flex="4"/>
```

Exemple 4 :

```
<button label="Chercher" flex="2"/>
<button label="Remplacer" flex="2"/>
<button label="Annuler" flex="3"/>
```

Exemple 5 :

```
<html:div>
<button label="Chercher" flex="2"/>
<button label="Remplacer" flex="2"/>
</html:div>
```

Exemple 6 :

```
<button label="Chercher" flex="145"/>
<button label="Remplacer" flex="145"/>
```

*Exemple 1 :*

dans ce cas, la flexibilité est divisée en part égale pour les deux boutons. Leurs tailles changeront en proportion égale.

*Exemple 2 :*

ici, les deux boutons sont flexibles, mais le bouton Chercher sera dix fois plus petit que le bouton Annuler qui a une valeur `flex` dix fois plus importante que celle du bouton Chercher. L'espace restant est divisé en une part pour le bouton Chercher et 10 parts pour le bouton Annuler.

*Exemple 3 :*

seuls deux boutons sont flexibles ici. Le bouton Remplacer ne changera jamais sa taille mais les deux autres le pourront. Le bouton Annuler aura toujours une taille du double du bouton Chercher parce qu'il a une valeur de `flex` du double.



*Exemple 4 :*

dans ce cas, les trois boutons sont flexibles. Les boutons Chercher et Remplacer auront exactement la même taille mais le bouton Annuler sera un peu plus large (50% plus large pour être exact).

*Exemple 5 :*

Ici, les deux boutons sont placés dans une balise `div`. La flexibilité perd toute signification puisque les boutons ne sont pas directement dans une boîte. L'effet serait le même si les attributs `flex` étaient enlevés.

*Exemple 6 :*

Comme les valeurs de `flex` sont identiques, les deux boutons auront la même taille. Cela fonctionne aussi bien que si vous aviez mis une valeur de `1` au lieu de `145`. Il est recommandé de mettre les valeurs les plus basses pour une meilleure lecture.

Notez que d'autres facteurs tels que les libellés des boutons et les tailles minimales des boutons peuvent affecter les dimensions réelles des boutons. Par exemple, un bouton ne peut pas être réduit au delà d'une taille nécessaire pour y placer son libellé.

Spécifier une valeur de `0` a le même effet que si vous enleviez l'attribut `flex` en entier. L'élément perd toute flexibilité. Vous verrez parfois des valeurs de `flex` avec un pourcentage. Il n'y a aucune signification spéciale et le signe pourcentage est traité comme s'il n'existait pas.

Vous avez dû remarquer que si vous redimensionnez verticalement la boîte de dialogue « Recherche de fichiers », les boutons se redimensionnent également en hauteur pour s'ajuster à la hauteur de la fenêtre. Ceci s'explique par le fait que les boutons ont une valeur de `flex` verticale implicite donnée par la fenêtre. Dans une prochaine section, nous expliquerons comment changer ce comportement.

Nous allons maintenant apprendre des nouvelles caractéristiques sur les boutons.

## 2.9 Plus de caractéristiques sur les boutons

Écrit par Neil Deakin. Traduit par *Alain B.* (08/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/advbtns.html>

Dans ce chapitre, nous allons voir quelques fonctionnalités supplémentaires des boutons.

### Ajouter une image

Vous pouvez ajouter une image dans un bouton en spécifiant une adresse URL dans l'attribut `image`. L'image chargée à partir de l'URL, qui peut être relative ou absolue, sera affichée sur le bouton.

Le bouton sur l'exemple ci-dessous aura en même temps un libellé et une image 'happy.png'. L'image apparaîtra à gauche du libellé. Vous pouvez changer cette position en utilisant deux autres attributs. Ce comportement sera expliqué dans un moment.

*Exemple 2.9.1 :*

```
<button label="Aide" image="happy.png"/>
```

Une autre façon est de spécifier une image en utilisant la propriété de style CSS `list-style-image` sur le bouton. Cette méthode permet de changer l'apparence (dans ce cas, l'apparence de l'image) sans modifier le fichier XUL. Un exemple vous est montré ci-dessous.

*Exemple 2.9.2 :*

```
<button id="find-button" label="Chercher"
```

```
style="list-style-image: url('happy.png') "/>
```

Dans ce cas, l'image 'happy.png' est affichée sur le bouton. L'attribut `style` fonctionne de façon similaire à son homologue HTML. En général, il peut être utilisé sur tous les éléments XUL. Notez que vous devriez placer toutes les déclarations de styles dans une feuille de style séparée.

## Le positionnement des images

Par défaut, l'image sur le bouton apparaît à gauche du libellé. Il y a deux attributs permettant de contrôler sa position.

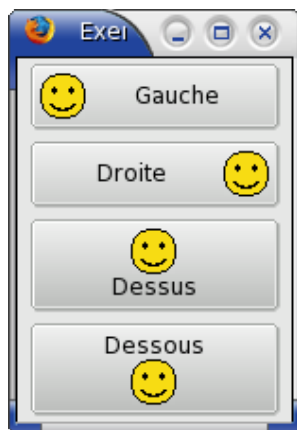
L'attribut `dir` contrôle la direction de l'image et du texte. En définissant cet attribut avec la valeur *reverse*, l'image sera placée sur le côté droit du texte. En utilisant la valeur *normal*, ou en omettant cet attribut, l'image sera placée sur le côté gauche du texte.

NdT : Sur des vieilles versions de Mozilla, les valeurs à utiliser sont respectivement *rtl* (right-to-left) et *ltr* (left-to-right).

L'attribut `orient` peut être utilisé pour placer l'image au dessus ou en dessous du texte. Sa valeur par défaut est *horizontal* et sert à placer l'image à gauche ou à droite. Vous pouvez aussi utiliser la valeur *vertical* pour placer l'image au dessus ou en dessous. Dans ce cas, l'attribut `dir` contrôle le placement vertical. Les mêmes valeurs sont utilisées, où *normal* signifie le placement de l'image au dessus du texte, et *reverse* signifie le placement de l'image en dessous du texte.

Exemple 2.9.3 :

```
<button label="Gauche" image="happy.png" />
<button label="Droite" image="happy.png" dir="reverse" />
<button label="Dessus" image="happy.png" orient="vertical" />
<button label="Dessous" image="happy.png" orient="vertical" dir="reverse" />
```



Cet exemple vous montre ici les quatre types d'alignement des boutons. Notez que les deux attributs ne sont pas spécifiés quand il s'agit d'utiliser leur valeur par défaut.

## Des boutons avec des contenus spéciaux

Les boutons peuvent contenir un balisage arbitraire, et qui sera représenté dans le bouton. Vous ne l'utiliserez probablement pas très souvent, mais vous pourrez l'utiliser pour créer des éléments personnalisés.

Cet exemple va créer un bouton dans lequel deux mots seront en magenta :

Exemple 2.9.4 :

```
<button>
  <description value="Ceci est" />
  <description value="un étrange" style="color: purple;" />
  <description value="bouton" />
</button>
```

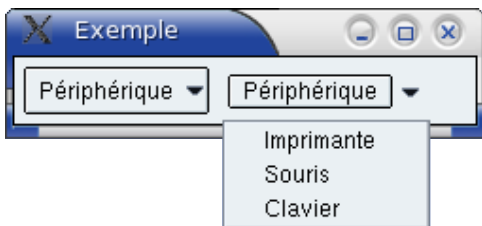
N'importe quels éléments XUL peut être placé à l'intérieur d'un bouton. Les éléments HTML seront ignorés, donc vous devez les intégrer à l'intérieur d'un élément description. Si vous spécifiez un attribut `label` sur un bouton, il supplantera n'importe quel autre contenu placé dans la définition du bouton.

Vous pouvez mettre un menupopup dans un bouton pour générer une liste déroulante lorsque le bouton est enfoncé, à l'instar de l'élément menulist. Toutefois, dans ce cas, vous devez indiquer l'attribut `type` avec la valeur *menu*.

Exemple 2.9.5 :

```
<button type="menu" label="Périphérique">
  <menupopup>
    <menuitem label="Imprimante" />
    <menuitem label="Souris" />
    <menuitem label="Clavier" />
  </menupopup>
</button>
```

Dans cet exemple, l'utilisateur doit cliquer sur le bouton pour faire apparaître un menu contenant trois items. Notez que la sélection d'un des items ne change pas le libellé du bouton, contrairement à l'élément menulist. Ce type de bouton est destiné à être utilisé comme un menu, avec des scripts pour chaque item exécutant des tâches. Nous en verrons plus sur les menu plus tard.



Vous pouvez également affecter la valeur *menu-button* à l'attribut

`type` (NdT : ). Cette valeur crée aussi un bouton avec un menu, mais son apparence est différente. L'image ci contre montre cette différence. Le bouton de gauche est un *menu* et celui de droite est un *menu-button*. Ils ont chacun une flèche pour indiquer la présence d'un menu déroulant. Pour le *menu*, l'utilisateur doit cliquer n'importe où sur le bouton pour ouvrir le menu. Pour le *menu-button*, l'utilisateur doit cliquer sur la flèche pour faire apparaître le menu.

---

Dans la prochaine section, nous en apprendrons plus sur le positionnement des éléments XUL dans une fenêtre.

## 3. Le modèle de boîte

### 3.1 Le modèle de boîte

Écrit par Neil Deakin. Traduit par *Benoît Salandre* (04/04/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/boxes.html>

Dans cette section, nous allons voir comment XUL gère la mise en page.

#### Introduction aux boîtes

La forme principale de mise en page dans XUL s'appelle le 'Modèle de Boîte'. Ce modèle vous permet de diviser une fenêtre en une série de boîtes. Les éléments à l'intérieur d'une boîte se placeront horizontalement ou verticalement. En combinant une série de boîtes, d'éléments spacer et d'éléments avec des attributs `flex`, vous pouvez contrôler la mise en page d'une fenêtre.

Bien qu'une boîte soit la partie fondamentale de la disposition d'éléments dans XUL, elle suit quelques règles très simples. Une boîte peut présenter ses enfants dans une des deux orientations, soit horizontalement, soit verticalement. Une boîte horizontale aligne ses éléments horizontalement et une boîte verticale place ses éléments verticalement. Vous pouvez vous représenter une boîte comme étant une rangée ou une colonne d'un tableau HTML. Divers attributs placés sur les éléments enfants, en plus de certaines propriétés de style CSS, contrôlent la position et la taille exactes des enfants.

La syntaxe basique d'une boîte est la suivante :

```
<hbox>
  ...
</hbox>

<vbox>
  ...
</vbox>
```

L'élément hbox est utilisé pour créer une boîte orientée horizontalement. Chaque élément placé dans la hbox sera placé horizontalement sur une rangée. L'élément vbox est utilisé pour créer une boîte orientée verticalement. Les éléments ajoutés seront placés les uns en dessous des autres dans une colonne.

Il y a aussi un élément box générique qui s'oriente horizontalement par défaut, ce qui veut dire qu'il est similaire à l'élément hbox. Cependant, vous pouvez utiliser l'attribut `orient` pour contrôler l'orientation de la boîte. Vous pouvez positionner cet attribut à la valeur *horizontal* pour créer une boîte horizontale et à la valeur *vertical* pour créer une boîte verticale.

Ainsi, les deux lignes ci-dessous sont équivalentes :

```
<vbox>

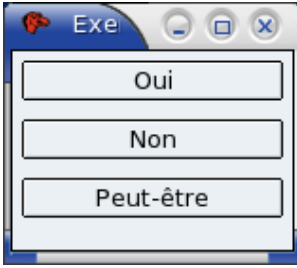
<box orient="vertical">
```

L'exemple suivant montre comment placer trois boutons verticalement.

Exemple 3.1.1 :

```
<vbox>
  <button id="yes" label="Oui" />
  <button id="no" label="Non" />
  <button id="maybe" label="Peut-être" />
```

```
</vbox>
```



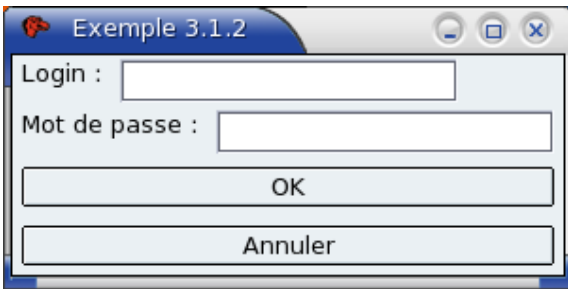
Ces trois boutons sont orientés verticalement comme le spécifie la boîte. Pour faire

en sorte que les boutons soient orientés horizontalement, tout ce que vous avez à faire est de remplacer l'élément `vbox` par l'élément `hbox`.

Vous pouvez ajouter autant d'élément que vous le souhaitez à l'intérieur de la boîte, y compris d'autres boîtes. Dans le cas d'une boîte horizontale, chaque élément additionnel sera placé à la droite du précédent. Les éléments ne se chevauchent pas, donc plus vous ajoutez d'éléments, plus large est la fenêtre. De même, chaque élément ajouté dans une boîte verticale sera placé sous le précédent. L'exemple ci-dessous montre une simple demande de login :

Exemple 3.1.2 :

```
<vbox>
  <hbox>
    <label control="login" value="Login :"/>
    <textbox id="login"/>
  </hbox>
  <hbox>
    <label control="pass" value="Mot de passe :"/>
    <textbox id="pass"/>
  </hbox>
  <button id="ok" label="OK"/>
  <button id="cancel" label="Annuler"/>
</vbox>
```



Ici quatre éléments ont été orientés verticalement, deux

balises `hbox` internes et deux éléments `button`. Notez que seuls les éléments qui sont des descendants directs de la boîte sont orientés verticalement. Les étiquettes et les boîtes de texte sont à l'intérieur d'éléments `hbox`, donc ils sont orientés suivant ces boîtes, lesquelles sont horizontales. Vous pouvez voir sur l'image que chaque libellé et leur champ de saisie est orienté horizontalement.

Si vous regardez attentivement l'image de la boîte de dialogue de login, vous verrez que les deux champs de saisie ne sont pas alignées horizontalement. Ce serait certainement mieux si elles l'étaient. Pour y parvenir, nous devons ajouter quelques boîtes supplémentaires.

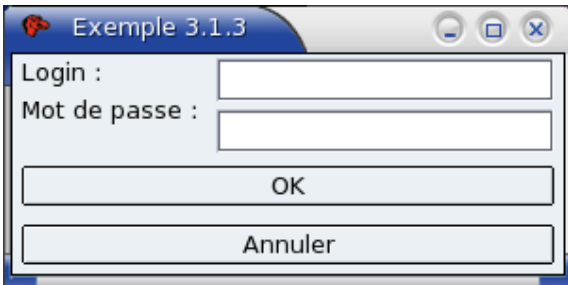
Exemple 3.1.3 :

```
<vbox>
  <hbox>
    <vbox>
      <label control="login" value="Login :"/>
      <label control="pass" value="Mot de passe :"/>
    </vbox>
  </hbox>
  <button id="ok" label="OK"/>
  <button id="cancel" label="Annuler"/>
</vbox>
```

```

</vbox>
<vbox>
  <textbox id="login"/>
  <textbox id="pass"/>
</vbox>
</hbox>
<button id="ok" label="OK"/>
<button id="cancel" label="Annuler"/>
</vbox>

```



Notez que les champs de saisie sont maintenant alignés.

Pour cela, il nous a fallu ajouter des boîtes à l'intérieur de la boîte principale. Les deux libellés et les champs de saisie sont placés à l'intérieur d'une boîte horizontale. Ensuite, les libellés sont placés à l'intérieur d'une autre boîte, une verticale cette fois, de même que les champs de saisie. C'est cette boîte interne qui fait s'orienter verticalement les éléments. La boîte horizontale est nécessaire puisque nous voulons que la vbox des libellés et que la vbox des champs de saisie soient placées horizontalement l'une de l'autre. Si cette boîte est enlevée, les champs de saisie apparaîtraient sous les libellés.

L'ennui est que le libellé 'Mot de passe' est désormais trop haut. Il nous faudra utiliser l'élément grid pour résoudre ce phénomène, ce que nous verrons dans une prochaine section.

## Des boîtes dans la boîte de dialogue de recherche de fichiers

Ajoutons quelques boîtes à la boîte de dialogue de recherche de fichiers. Une boîte verticale sera ajoutée autour de tous les éléments, et une boîte horizontale sera ajoutée autour du champ de saisie et des boutons. Le résultat sera que les boutons apparaîtront sous le champ de saisie.

```

<vbox flex="1">
  <description>
    Entrez votre critère de recherche ci-dessous et sélectionnez le bouton Rechercher
    pour démarrer la recherche.
  </description>

  <hbox>
    <label value="Rechercher :" control="find-text"/>
    <textbox id="find-text"/>
  </hbox>

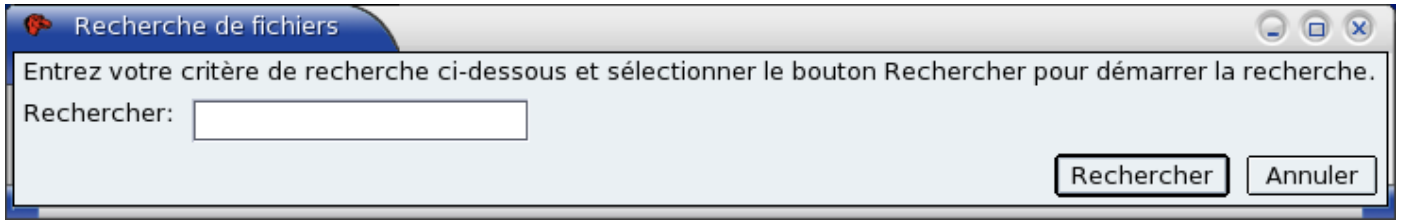
  <hbox>
    <spacer flex="1"/>

    <button id="find-button" label="Rechercher"/>
    <button id="cancel-button" label="Annuler"/>
  </hbox>
</vbox>

```

La boîte verticale entraîne l'orientation verticale du texte principal, de la boîte contenant le champ de saisie ainsi que de la boîte avec les deux boutons. Les boîtes internes orientent leurs éléments horizontalement. Comme vous pouvez le voir sur l'image ci-dessous, le libellé et le champ de saisie de texte sont placés côte à côte. L'élément spacer et les deux boutons sont aussi placés horizontalement dans leur boîte.

Notez que l'élément `spacer`, parce qu'il est flexible, pousse les boutons à apparaître sur le côté droit.



Dans la prochaine section, nous verrons comment spécifier les tailles des éléments individuellement et comment contraindre leurs tailles.

## 3.2 Positionnement d'éléments de fenêtres

Écrit par Neil Deakin. Traduit par *Jean Pascal Milcent* (25/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/boxstyle.html>

Ici nous apprendrons à contrôler la position et la taille d'un élément.

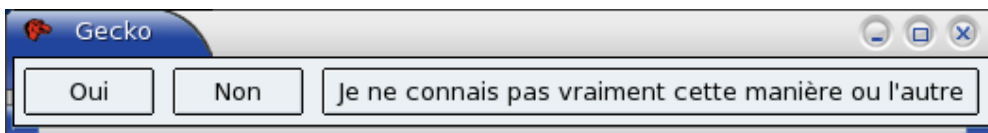
### Positionnement des éléments de boîte

Jusqu'ici, nous avons appris à placer des éléments horizontalement ou verticalement à l'intérieur d'une boîte. Nous aurons souvent besoin d'un meilleur contrôle sur la position et la taille des éléments à l'intérieur d'une boîte. Pour cela, nous devons d'abord comprendre le fonctionnement d'une boîte.

La position d'un élément est déterminée par le style de mise en page de son conteneur. Par exemple, la position d'un bouton dans une boîte horizontale est à droite du bouton précédent, s'il y en a plusieurs. La taille d'un élément est déterminée par deux facteurs, la taille que cet élément cherche à posséder et la taille que vous avez spécifié. La taille qu'un élément cherche à avoir est déterminée par son contenu. Par exemple, la largeur d'un bouton est déterminée par la quantité de texte à l'intérieur du bouton.

Un élément sera généralement juste assez large pour les besoins de l'affichage de son contenu, et pas plus large. Quelques éléments, tels que les boîtes de textes ont une taille par défaut, qui sera utilisée. Une boîte aura une largeur nécessaire à l'affichage des éléments qu'elle contient. Une boîte horizontale contenant trois boutons aura la largeur de ces trois boutons plus la marge.

Dans l'image ci-dessous, les deux premiers boutons ont la taille nécessaire à l'affichage de leur texte. Le troisième bouton est plus large parce que son contenu est plus grand. La largeur de la boîte contenant les boutons correspond au total des largeurs de ces boutons plus la marge entre eux. La hauteur des boutons est une taille adaptée à l'affichage du texte.



Vous pourriez avoir besoin de plus de contrôle sur la taille d'un élément d'une fenêtre. Il y a plusieurs dispositifs qui vous permettent de contrôler la taille d'un élément. La façon la plus rapide est d'ajouter simplement des attributs `width` et `height` sur un élément, un peu comme vous feriez avec la balise HTML `img`. Voir l'exemple ci-dessous :

Exemple 3.2.1 :

```
<button label="OK" width="100" height="40"/>
```

Cependant, il n'est pas recommandé de faire ainsi. Cette méthode n'est pas vraiment portable et peut ne pas être adaptable avec certains thèmes. Une meilleure solution est d'utiliser des propriétés de style, dont le fonctionnement est similaire aux feuilles de style du HTML. Les propriétés CSS suivantes peuvent être utilisées.

*width*

Ceci indique la largeur d'un élément.

*height*

Ceci indique la hauteur d'un élément.

En plaçant l'une ou l'autre de ces deux propriétés, l'élément sera créé avec cette largeur et cette hauteur. Si vous spécifiez seulement une de ces deux propriétés, l'autre est calculée en fonction de ses propres besoins. La taille de ces propriétés de style doivent être définie par un nombre suivi d'une unité de mesure.

Les tailles sont assez faciles à calculer pour les éléments non flexibles. Ils répondent simplement à leur spécificité de largeur et de hauteur, et si leurs dimensions ne sont pas définies, leurs tailles sont ajustées par défaut au contenu. Pour les éléments flexibles, le calcul est légèrement plus savant.

Les éléments flexibles sont ceux qui ont un attribut `flex` dont la valeur est supérieure à 0. Rappelez vous que les éléments flexible s'étirent et s'élargissent jusqu'à remplacer l'espace vide. Leurs tailles par défaut restent calculées de la même façon que les éléments non flexibles. L'exemple suivant démontre cela :

Exemple 3.2.2 :

```
<window orient="horizontal"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<box>
<button label="Oui" flex="1"/>
<button label="Non"/>
<button label="Je ne connais pas vraiment cette manière ou l'autre"/>
</box>
```

La fenêtre apparaîtra initialement comme sur l'image précédente. Les deux premiers boutons seront dimensionnés à une largeur par défaut convenable et le troisième bouton sera plus large parce que son libellé est plus long. Le premier bouton est flexible et les trois éléments ont été placés à l'intérieur d'une boîte. La largeur de la boîte correspondra à la largeur totale des trois boutons (environ 515 pixels dans l'image).

Si vous augmentez la largeur de la fenêtre, une vérification de la flexibilité des éléments est effectué afin d'assurer le remplissage de l'espace libre à afficher. Le bouton est le seul élément flexible, mais il ne s'élargira pas plus, parce que la boîte contenant le bouton n'est pas flexible. Un élément inflexible ne change jamais de taille même lorsque de l'espace est disponible, c'est pourquoi le bouton ne s'agrandit pas non plus. Ainsi, le bouton ne pourra pas s'élargir.

La solution est de rendre la boîte flexible également. De cette façon, quand vous élargirez la fenêtre l'espace supplémentaire sera disponible, et alors la boîte s'élargira pour remplir l'espace supplémentaire. Puisque la boîte est plus grande, une plus grande quantité d'espace libre est créée à l'intérieur de celle-ci, et le bouton flexible qu'elle contient s'élargira pour occuper l'espace disponible. Ce processus est répété pour toutes les boîtes présentes autant de fois que nécessaire.

## Réglage des tailles minimum et maximum

Vous pouvez vouloir autoriser un élément à être flexible tout en contraignant sa taille de sorte qu'elle ne puisse pas être plus grande qu'une certaine valeur. Ou, vous voulez peut-être définir une taille minimum. Vous pouvez régler ce comportement en employant quatre attributs :



*minwidth*

Ceci indique la largeur minimum que l'élément peut posséder.

*minheight*

Ceci indique la hauteur minimum que l'élément peut posséder.

*maxwidth*

Ceci indique la largeur maximum que l'élément peut posséder.

*maxheight*

Ceci indique la hauteur maximum que l'élément peut posséder.

Les valeurs sont toujours mesurées en pixels. Vous pouvez également employer les propriétés CSS correspondantes, `min-width`, `min-height`, `max-width` et `max-height`.

Ces propriétés sont seulement utiles pour les éléments flexibles. En plaçant une hauteur maximum, par exemple, un bouton extensible s'étirera seulement jusqu'à une certaine hauteur maximum. Vous pourrez toujours agrandir la fenêtre au delà de ce point mais la taille du bouton cessera de s'accroître. La boîte, dans laquelle le bouton se trouve, continuera également à s'agrandir, à moins que vous ne placiez aussi une hauteur maximum sur la boîte.

Si deux boutons sont également flexibles, normalement ils se partageront tous les deux la quantité d'espace supplémentaire. Si un bouton a une largeur maximum, le second devrait continuer de s'agrandir en prenant tout l'espace restant.

Si une boîte a une largeur ou une hauteur maximum, les enfants ne peuvent pas se développer au delà de cette taille maximum. Si une boîte a une largeur ou une taille minimum, les enfants ne peuvent pas rétrécir au delà de cette taille minimum. Voici quelques exemples de réglage des largeurs et des hauteurs :

```
<button label="1" style="width: 100px;"/>
<button label="2" style="width: 100em; height: 10px;"/>
<button label="3" flex="1" style="min-width: 50px;"/>
<button label="4" flex="1" style="min-height: 2ex; max-width: 100px;"/>
<textbox flex="1" style="max-width: 10em;"/>
<description style="max-width: 50px">Ceci est quelques peu ennuyant mais c'est un texte s'étalant
```

**Exemple 1 :** le premier bouton sera affiché avec une largeur de 100 pixels (le px signifie pixels). Vous devez ajouter l'unité ou la largeur définie sera ignorée.

**Exemple 2 :** le deuxième bouton sera affiché avec une hauteur de dix pixels et une largeur de 100 em (un em correspond à la taille d'un caractère dans la police courante).

**Exemple 3 :** le troisième bouton flexible s'agrandira en se basant sur la taille de la boîte qui le contient. Cependant, le bouton ne rétrécira jamais au-dessous de 50 pixels. D'autres composants flexibles tels que des éléments spacers absorberont l'espace restant, brisant ainsi le rapport de flexibilité.

**Exemple 4 :** le quatrième bouton est flexible et n'aura jamais une hauteur inférieure à 2 ex (un ex correspond habituellement la taille de la lettre X dans la police courante) ou une largeur supérieure à 100 pixels.

**Exemple 5 :** le champ de saisie de texte est flexible mais sa largeur ne sera jamais supérieure à 10 em. Vous voudrez souvent employer les ems quand vous précisez les tailles des champs de saisie de texte en fonction du texte qu'ils contiennent. Cette unité est utile pour des champs de saisie de texte qui peuvent ainsi s'adapter à la police de caractères utilisée, même si la police est très grande.

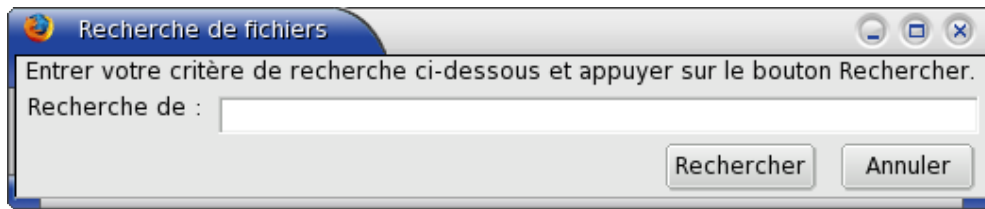
**Exemple 6 :** l'élément description est contraint d'avoir une largeur maximum de 50 pixels. À l'intérieur, une césure du texte sur la ligne suivante interviendra après cinquante pixels.

Ajoutons quelques uns de ces styles à notre exemple de fenêtre de recherche de fichiers. Nous le ferons de telle manière que le champ de saisie de texte s'étire afin de remplir entièrement la fenêtre.

```
<textbox id="find-text" flex="1" style="min-width: 15em;"/>
```

Ici, le champ de saisie de texte a été rendu flexible. De cette façon, il s'agrandira si l'utilisateur change la taille de la fenêtre. C'est utile si l'utilisateur veut saisir une longue chaîne de caractères. En outre, une largeur minimum de 15 em a été définie de sorte que le champ de saisie de texte affiche toujours au moins 15 caractères. Si l'utilisateur redimensionne la fenêtre en diminuant sa taille, le champ de saisie de texte ne se rétrécira pas en-dessous de 15 em. Il sera dessiné comme s'il se prolongait au-delà du bord de la fenêtre.

Notez dans l'image ci-dessous que le champ de saisie de texte s'est étiré de façon à occuper entièrement la largeur de la fenêtre.



## Empaquetage de boîte

Nous avons une boîte avec deux éléments enfants, qui ne sont pas flexibles, mais la boîte, elle, est flexible. Par exemple :

Exemple 3.2.3 :

```
<box flex="1">
<button label="Heureux"/>
<button label="Triste"/>
</box>
```

Si vous redimensionnez la fenêtre, la boîte s'étirera pour s'ajuster à la taille de la fenêtre. Les boutons n'étant pas flexibles, leur largeur ne changera pas. Le résultat est qu'un espace supplémentaire apparaîtra sur le côté droit de la fenêtre, à l'intérieur de la boîte. Cependant, vous pouvez désirer que l'espace supplémentaire apparaisse plutôt sur le côté gauche, pendant que les boutons restent alignés sur la droite de la fenêtre.

Vous pouvez réaliser ce comportement en plaçant un élément spacer à l'intérieur de la boîte, mais cela peut rendre le code confus quand vous devez le faire plusieurs fois. Une meilleure solution est d'utiliser l'attribut supplémentaire `pack` sur la boîte. Cet attribut indique comment empaqueter les éléments enfants à l'intérieur d'une boîte. Pour des boîtes orientées horizontalement, il contrôle le positionnement horizontal des enfants. Pour les boîtes orientées verticalement, il contrôle le positionnement vertical des enfants. Vous pouvez utiliser les valeurs suivantes :

*start*

Ceci positionne les éléments sur le bord gauche pour les boîtes horizontales et sur le bord haut pour les boîtes verticales. Il s'agit de la valeur par défaut.

*center*

Ceci centre les éléments enfants dans la boîte.

*end*

Ceci positionne les éléments sur le bord droit pour les boîtes horizontales et sur le bord bas pour les boîtes verticales.

L'attribut `pack` s'applique à la boîte contenant les éléments à empaqueter et non aux éléments eux-mêmes.

Nous pouvons modifier l'exemple précédent pour centrer les éléments de cette façon :

Exemple 3.2.4 :

```
<box flex="1" pack="center">
<button label="Heureux"/>
<button label="Triste"/>
</box>
```

Maintenant, quand la fenêtre est redimensionnée, les boutons sont centrés horizontalement. Comparez ce comportement à celui de l'exemple précédent.

## Alignement de boîte

Si vous redimensionnez horizontalement la fenêtre dans l'exemple "Heureux–Triste", la boîte devrait s'élargir. Si vous redimensionnez verticalement la fenêtre, vous noterez que les boutons sont étirés. Ceci est dû à la flexibilité qui est affectée par défaut à l'autre direction.

Vous pouvez contrôler ce comportement avec l'attribut `align`. Pour les boîtes horizontales, il contrôle le positionnement vertical des enfants. Pour les boîtes verticales, il contrôle le positionnement horizontal des enfants. Les valeurs possibles sont similaires à celles de l'attribut `pack`.

### *start*

Ceci aligne les éléments le long du bord haut pour les boîtes horizontales et le long du bord gauche pour les boîtes verticales.

### *center*

Ceci centre les éléments enfants dans la boîte.

### *end*

Ceci aligne les éléments le long du bord bas pour les boîtes horizontales et le long du bord droit pour les boîtes verticales.

### *baseline*

Ceci aligne les éléments et le texte. Il est utilisable seulement avec des boîtes horizontales.

### *stretch*

Cette valeur, affectée par défaut, provoque l'agrandissement des éléments proportionnellement à la taille de la boîte, un peu à la façon d'un élément flexible, mais dans la direction opposée.

Comme avec l'attribut `pack`, l'attribut `align` s'applique à la boîte contenant les éléments à aligner et non aux éléments eux-mêmes.

Dans l'exemple ci-dessous, la première boîte aura ses enfants étirés, car c'est le comportement par défaut. La seconde boîte a un attribut `align`, c'est pourquoi ses enfants sont placés au centre.

Exemple 3.2.5 :

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

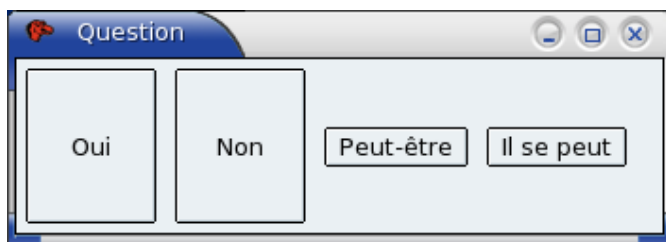
<window id="yesno" title="Question" orient="horizontal"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<hbox>
<button label="Oui"/>
<button label="Non"/>

</hbox>
<hbox align="center">
<button label="Peut-être"/>
```

```
<button label="Il se peut"/>
</hbox>

</window>
```



Vous pouvez aussi utiliser les propriétés de style `-moz-box-pack` et `-moz-box-align` à la place des attributs indiqués.

Vous pouvez trouver l' pratique pour tester les différentes propriétés de boîte (NdT : vous bénéficiez aussi une ).

## Coupage du texte et des boutons

Vous pourriez potentiellement créer un élément bouton qui contient un libellé plus large que la largeur maximum du bouton. Bien sûr, une solution serait d'augmenter la taille du bouton. Cependant, les boutons (et les autres éléments avec un libellé) ont un attribut spécial appelé `crop` qui vous permet d'indiquer de quelle manière le texte doit être coupé s'il est trop grand.

Si le texte est coupé, une ellipse (...) apparaît sur le bouton où le texte a été enlevé. Quatre valeurs possibles sont valides :

*left*

Le texte est coupé sur son côté gauche.

*right*

Le texte est coupé sur son côté droit.

*center*

Le texte est coupé au milieu.

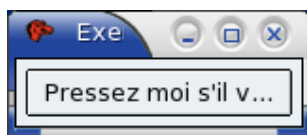
*none*

Le texte n'est pas coupé. C'est la valeur par défaut.

Cet attribut est vraiment utile quand une boîte de dialogue a été conçue pour être utilisable à n'importe quelle taille. L'attribut `crop` peut aussi être utilisé avec les autres éléments qui utilisent un attribut `label` en tant que libellé. L'exemple suivant montre l'utilisation de cet attribut :

Exemple 3.2.6 :

```
<button label="Pressez moi, s'il vous plait !" crop="right" flex="1"/>
```



Notez comment le texte sur le bouton voit son côté droit coupé après que la

fenêtre ait été réduite.

---

Dans la prochaine section, nous ferons un petit résumé et décrirons quelques détails supplémentaires sur le modèle de boîte.

### 3.3 Détails sur le modèle de boîte

Écrit par Neil Deakin. Traduit par *Damien Hardy* (27/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/boxdet.html>

Nous avons déjà vu beaucoup de fonctionnalités sur les modèles de boîtes. Dans cette section, nous verrons quelques détails supplémentaires en s'appuyant sur des exemples.

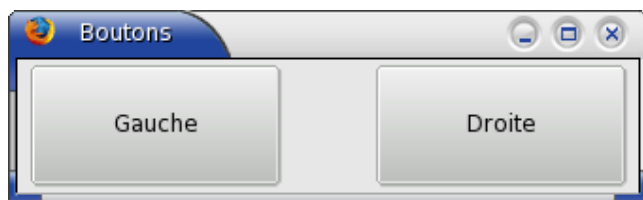
#### Plus de détails sur l'affichage

Les propriétés de style comme `min-width` et `max-height` peuvent être appliquées à tous les éléments. Nous les avons ajoutées aux boutons ou aux champs de saisie, mais nous pouvons aussi les ajouter aux éléments d'espacement ou aux boîtes. De plus, l'attribut `flex` peut être appliqué à n'importe quel élément.

Exemple 3.3.1 :

```
<hbox flex="1">
  <button label="Gauche" style="min-width: 100px;" flex="1"/>
  <spacer flex="1"/>
  <button label="Droite" style="min-width: 100px;" flex="1"/>
</hbox>
```

Dans l'exemple ci-dessus, les trois éléments se redimensionnent eux même car il sont tous flexibles. Les deux boutons indiquent une largeur minimum de 100 pixels. Les deux boutons ne seront jamais plus petits que cette taille mais ils pourront toujours s'élargir. Ici la fenêtre doit apparaître avec une largeur d'un peu plus de 200 pixels. C'est suffisant pour les deux boutons. Car si les trois éléments sont flexibles, ils ne nécessitent pas pour autant plus de place, la flexibilité n'ajoute pas d'espace supplémentaire.



Comme il est montré dans l'image ci-dessus, il y a deux boutons qui s'agrandissent verticalement pour remplir leur élément conteneur, qui dans ce cas est le `hbox`. L'attribut `align` contrôle ce comportement sur une boîte horizontale. Vous pouvez aussi empêcher cet étirement en précisant une hauteur maximale sur l'élément ou mieux, sur la boîte elle même. Si une boîte a une hauteur maximale, les éléments qu'elle contient sont contraints en hauteur par celle-ci. Cependant, le problème est que vous devez connaître au préalable la taille d'un élément pour pouvoir le spécifier à la la boîte. L'exemple suivant montre l'utilisation d'un d'attribut `align`.

Exemple 3.3.2 :

```
<hbox flex="1" align="top">
  <button label="Gauche" style="min-width: 100px;" flex="1"/>
  <spacer flex="1"/>
  <button label="Droite" style="min-width: 100px;" flex="1"/>
</hbox>
```

Pour parfaire une mise en page complexe, vous devrez généralement ajouter des boîtes imbriquées, spécifier une taille minimum et maximum sur quelques éléments, et rendre certains éléments flexibles. La meilleure interface est celle qui peut être affichée sans problème dans différentes tailles. Le modèle de boîte peut être difficile à appréhender si vous n'essayez pas différentes choses par vous même.

Voici une courte description des deux types de boîtes :

#### *Boîtes horizontales*

1. Affichent chacun de leurs éléments les uns à côté des autres horizontalement.
2. Les éléments flexibles le sont horizontalement.
3. Le groupement (NdT : attribut `pack`) contrôle le placement horizontal des éléments fils.
4. L'alignement contrôle la manière dont les lignes d'éléments seront alignées verticalement.

#### *Boîtes verticales*

1. Affichent chacun de leurs éléments verticalement en colonnes.
2. Les éléments flexibles le sont verticalement.
3. Le groupement contrôle le placement vertical des éléments fils.
4. L'alignement contrôle la manière dont les colonnes d'éléments seront alignées horizontalement.

Vous pouvez mettre des boîtes n'importe où dans un fichier XUL, même dans un élément HTML comme une table. Cependant, l'affichage sera en partie dépendant de l'élément HTML. Cela signifie que l'attribut `flex` risque de ne pas réagir exactement comme vous le voudriez. Rappelez vous que la flexibilité n'a de sens que pour les éléments qui sont directement à l'intérieur d'une boîte, ou d'un élément qui est assimilé à une boîte.

## Exemples

### 1. Utiliser les spacers

Exemple 3.3.3 :

```
<hbox>
  <button label="Un" />
  <spacer style="width: 5px" />
  <button label="Deux" />
</hbox>
```

Ici, un espacement est utilisé comme séparateur entre deux boutons, en précisant une largeur de 5 pixels. Vous pourriez aussi utiliser les marges (avec la propriété CSS `margin`).

### 2. Centrer des boutons

Exemple 3.3.4 :

```
<hbox pack="center" align="center" flex="1">
  <button label="Regardez Moi!" />
  <button label="Cliquez Moi!" />
</hbox>
```

Cet exemple propose une boîte horizontale flexible contenant deux boutons. La boîte possède l'attribut `pack` qui est utilisé pour centrer les boutons horizontalement. L'attribut `align` aligne quant à lui les boutons verticalement. Le résultat est que les boutons seront centrés dans la boîte dans les deux directions. Cet exemple fonctionnera aussi avec une boîte verticale, le second bouton sera alors sous le premier, au lieu d'être à côté de lui.

### 3. Une fenêtre de recherche dans le texte

Exemple 3.3.5 :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
```

```

<window id="findtext" title="Recherche dans un texte" orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

  <vbox flex="3">
    <label control="t1" value="Mots clés :"/>
    <textbox id="t1" style="min-width: 100px;" flex="1"/>
  </vbox>

  <vbox style="min-width: 150px;" flex="1" align="start">
    <checkbox id="c1" label="Ignorer la casse"/>
    <spacer flex="1" style="max-height: 30px;"/>
    <button label="Rechercher"/>
  </vbox>

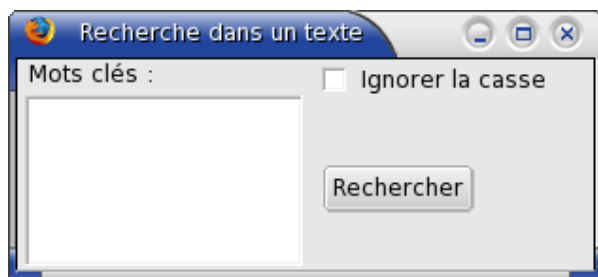
</window>

```

Ici, deux boîtes verticales sont créées, une pour le champs texte et l'autre pour la case à cocher et le bouton de recherche. La boîte de gauche a une flexibilité 3 fois plus grande que celle de droite donc elle prendra toujours 3 fois plus de place quand la taille de la fenêtre augmentera. La boîte de droite force une largeur minimum de 150 pixels.

Le champ de saisie est flexible donc il s'ajustera à la taille de la fenêtre. Il impose aussi une largeur minimum de 100 pixels. La case à cocher apparaît dans la boîte de droite avec son libellé. Juste en dessous de la case à cocher il y a un spacer (espacement). L'espacement s'agrandira ou retrécira mais n'excédera pas 30 pixels. Le résultat est que la case à cocher et le bouton de recherche seront séparés par un espace qui ne fera pas plus de 30 pixels.

La seconde boîte a un alignement de valeur *start*. Cela a pour effet que les trois éléments fils seront alignés sur le bord gauche. Si elle n'était pas précisée, la valeur par défaut serait *stretch*, ce qui aurait pour effet de positionner les éléments fils les uns à coté des autres horizontalement. Parce que nous ne voulons pas que le bouton de recherche change de taille, nous devons alors préciser un alignement.



Dans la prochaine section, nous allons en apprendre un peu plus sur un type de boîte plus spécifique, le groupe de boîte.

## 3.4 Les boîtes de groupe

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (13/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/titledbox.html>

Cette section décrit la façon d'inclure des éléments dans des groupes.

### Les boîtes de groupes

HTML fournit un élément, `fieldset`, qui peut être utilisé pour regrouper plusieurs éléments. Une bordure est affichée autour des éléments pour montrer qu'ils sont en relation. Un exemple pourrait être un groupe de cases à cocher. XUL fournit un élément équivalent, `groupbox`, qui peut être utilisé pour un usage similaire.

Comme son nom l'indique, groupbox est un type de boîte. Ce qui signifie que les éléments à l'intérieur sont alignés selon l'application des règles des boîtes. Il y a deux différences entre les boîtes de groupes et les boîtes normales :

1. Une bordure incrustée est affichée par défaut autour de la boîte. Vous pouvez changer ce comportement en changeant le style CSS.
2. Les boîtes de groupe ont une légende qui est placée sur la partie haute de la bordure.

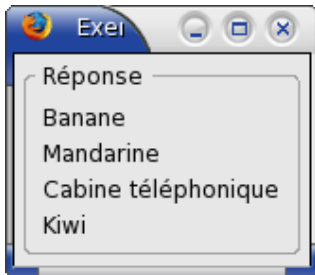
Puisque les boîtes de groupe sont des types de boîtes, vous pouvez utiliser les mêmes attributs tels que `orient` et `flex`. Vous pouvez mettre tous les éléments que vous voulez à l'intérieur de la boîte, mais il devra typiquement y avoir un rapport entre eux.

Le libellé en haut de la boîte de groupe peut être créé en utilisant l'élément caption. Il fonctionne comme l'élément HTML `legend`. Un simple élément caption placé en tant que première balise fille est suffisant.

L'exemple ci-dessous montre un groupbox simple :

Exemple 3.4.1 :

```
<groupbox>
  <caption label="Réponse"/>
  <description value="Banane"/>
  <description value="Mandarine"/>
  <description value="Cabine téléphonique"/>
  <description value="Kiwi"/>
</groupbox>
```



Cet exemple affiche quatre bouts de texte entourés par la boîte avec le libellé

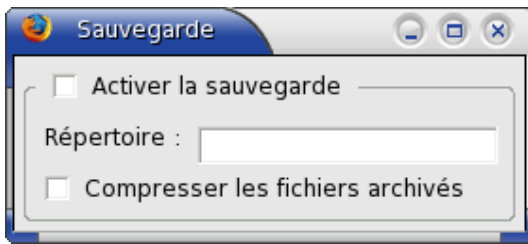
**Réponse.** Notez que la boîte de groupe a une orientation verticale par défaut nécessaire pour avoir les éléments texte empilés en une seule colonne.

Vous pouvez aussi ajouter des éléments fils à l'intérieur de l'élément caption pour créer une légende plus complexe. Par exemple, le panneau de préférence des polices de caractères de mozilla utilise une liste déroulante comme légende. Bien que vous puissiez utiliser n'importe quel contenu, vous utiliserez la plupart du temps une case à cocher ou une liste déroulante.

Exemple 3.4.2 :

```
<groupbox flex="1">
  <caption>
    <checkbox label="Activer la sauvegarde"/>
  </caption>
  <hbox>
    <label control="dir" value="Répertoire :"/>
    <textbox id="dir" flex="1"/>
  </hbox>
  <checkbox label="Compresser les fichiers archivés"/>
</groupbox>
```





Dans cet exemple, une case à cocher a été utilisée comme

légende. Nous devons utiliser un script pour activer et désactiver le contenu de la boîte de groupe, quand la case est cochée ou décochée. La boîte de groupe contient une boîte horizontale avec un libellé et un champ de saisie. Ces deux éléments sont flexibles donc le champ de saisie s'agrandit quand la fenêtre s'agrandit. La case à cocher supplémentaire apparaît en dessous du champ de saisie à cause de l'orientation verticale de la boîte de groupe. Nous ajouterons une boîte de groupe à notre exemple de recherche de fichiers dans la prochaine section.

## Les groupes de boutons radio

Vous pouvez utiliser l'élément `radiogroup` pour regrouper des éléments boutons radios. `radiogroup` est un type de boîte. Vous pouvez y mettre n'importe quel élément, et à part le comportement spécial sur les boutons radios, il fonctionne comme n'importe quelle autre boîte.

Tous les boutons radios placés à l'intérieur d'un `radiogroup` seront groupés ensemble, même s'ils sont à l'intérieur de boîtes imbriquées. Ce fonctionnement est utilisé pour ajouter des éléments supplémentaires à l'intérieur de la structure, comme dans l'exemple suivant :

Exemple 3.4.3 :

```
<radiogroup>
  <radio id="no" value="no" label="Pas de nombre"/>
  <radio id="random" value="random" label="Nombre aléatoire"/>
  <hbox>
    <radio id="specify" value="specify" label="Nombre spécifique :"/>
    <textbox id="specificnumber"/>
  </hbox>
</radiogroup>
```

Notez que l'élément `radiogroup` n'a pas de bordure. Vous devez placer un élément `groupbox` autour si vous voulez une bordure et une légende.

---

Maintenant, nous allons utiliser tout ce que nous avons appris pour ajouter des éléments supplémentaires dans la boîte de dialogue de recherche de fichiers.

## 3.5 Ajouter plus d'éléments

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (24/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/boxfinal.html>

Nous allons conclure ce chapitre en ajoutant des boîtes sur notre fenêtre de recherche de fichiers.

### Ajout d'éléments additionnels

Nous allons ajouter maintenant des éléments supplémentaires à notre boîte de dialogue de recherche de fichiers. Premièrement, nous allons ajouter la possibilité de faire une recherche sur d'autres informations comme la taille et la date du fichier.

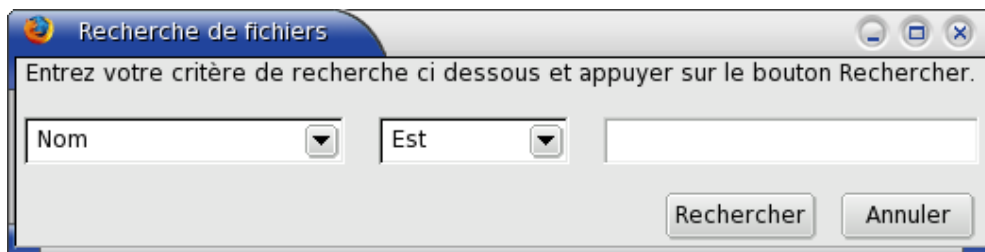
```
<hbox>
  <menulist id="searchtype">
```

```

<menupopup>
  <menuitem label="Nom"/>
  <menuitem label="Taille"/>
  <menuitem label="Date de modification"/>
</menupopup>
</menulist>
<spacer style="width: 10px;"/>
<menulist id="searchmode">
  <menupopup>
    <menuitem label="Est"/>
    <menuitem label="N'est pas"/>
  </menupopup>
</menulist>
<spacer style="width: 10px;"/>
<textbox id="find-text" flex="1" style="min-width: 15em;"/>
</hbox>

```

Deux listes déroulantes ont été ajoutées à la boîte de dialogue. Un espacement (spacer) a été ajouté entre chaque élément pour les séparer. Ces espacements ont une taille explicite de 10 pixels chacun. Vous noterez que si la fenêtre est redimensionnée, le champs de saisie s'agrandit mais pas les autres composants. Vous noterez également que le libellé a été enlevé.



Si vous redimensionnez la fenêtre verticalement, les éléments ne changeront pas de taille. C'est parce qu'ils sont à l'intérieur de boîtes horizontales. Ce serait mieux si les boutons "Rechercher" et "Annuler" restaient toujours en bas de la fenêtre. Il est facile de le faire en ajoutant un spacer entre les deux boîtes horizontales.

```

<spacer style="height: 10px;"/>
<hbox>
  <menulist id="searchtype">
    <menupopup>
      <menuitem label="Nom"/>
      <menuitem label="Taille"/>
      <menuitem label="Date de modification"/>
    </menupopup>
  </menulist>
  <spacer style="width: 10px;"/>
  <menulist id="searchmode">
    <menupopup>
      <menuitem label="Est"/>
      <menuitem label="N'est past"/>
    </menupopup>
  </menulist>
  <spacer style="width: 10px;"/>
  <textbox id="find-text" flex="1" style="min-width: 15em;"/>
</hbox>

<spacer style="height: 10px" flex="1"/>

<hbox>

```

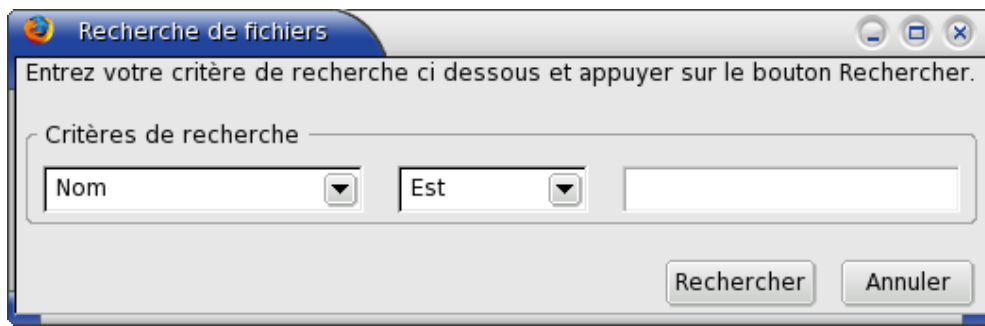
Maintenant, quand la boîte de dialogue est redimensionnée, les deux boutons sont placés le long du bas de la boîte de dialogue. Le premier spacer ajoute un espacement entre le titre du libellé et les éléments de

critères de recherche.

Il serait plus joli d'avoir une bordure autour des critères de recherche. Il y a deux moyens pour le faire. Vous pouvez utiliser la propriété CSS `border` ou vous pouvez utiliser l'élément `groupbox`. La première méthode nécessite que nous appliquions le style sur la boîte elle-même. Nous utiliserons l'autre méthode, à la place. Un élément `groupbox` a l'avantage de dessiner une boîte avec un joli effet d'incrustation, en adéquation avec le thème courant.

Changeons maintenant la boîte `box` en `groupbox` :

```
<groupbox orient="horizontal">
  <caption label="Critères de recherche"/>
  <menulist id="searchtype">
    .
    .
    .
  <spacer style="width: 10px;"/>
  <textbox id="find-text" flex="1" style="min-width: 15em;"/>
</groupbox>
```



Il reste d'autres problèmes cosmétiques. Nous pourrions avoir un `groupbox` qui s'étend verticalement vers le bas de la boîte. Mais aussi, nous pourrions modifier quelques marges afin de mieux positionner les éléments.

---

Nous verrons dans la suite comment créer des piles.

## 4. Éléments communs

### 4.1 Piles et Paquets

Écrit par Neil Deakin. Traduit par *Alain B.* (14/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/stacks.html>

Il se peut qu'il soit nécessaire d'afficher des éléments comme un empilement superposé de cartes. Les éléments stack et deck peuvent être utilisés à cet effet.

#### Containers

Chaque boîte XUL est un container qui peut contenir n'importe quel autre élément. Il y a un certain nombre d'éléments qui sont des types spécialisés de boîtes, tels que les barres d'outils et les onglets. La balise box crée la plus simple des boîtes sans propriétés spéciales. Toutefois, les types spécialisés de boîtes fonctionnent comme des boîtes normales dans le sens où elles orientent les éléments qu'elles contiennent, mais elles ont des fonctionnalités supplémentaires.

En fait, beaucoup de composants peuvent contenir d'autres éléments. Nous avons déjà vu que les boutons peuvent contenir d'autres choses que leurs contenus par défaut. Une barre de défilement est juste un type spécial de boîte qui crée ses propres éléments si vous ne les fournissez pas. Ceux-ci contrôlent également le déplacement de l'ascenseur de la barre de défilement.

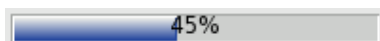
Dans les prochaines sections, nous allons introduire quelques éléments qui sont prévus pour le contrôle d'autres éléments. Ce sont tous des types spéciaux de boîtes et permettent d'inclure tous les attributs des boîtes.

#### Piles

L'élément stack est une simple boîte. Il fonctionne comme toute autre boîte mais a la propriété spéciale que ces enfants sont tous disposés les uns au dessus des autres. Le premier enfant de la pile est dessiné sur le dessous, le second enfant est dessiné ensuite, suivi du troisième et ainsi de suite. De nombreux éléments peuvent être empilés sur une pile.

La propriété orient n'a aucune signification particulière sur un élément stack, car les enfants sont empilés les uns sur les autres au lieu d'être côte à côte. La dimension d'une pile est déterminée par celle de son plus grand enfant, mais vous pouvez utiliser les propriétés CSS width, height, min-width ou d'autres propriétés similaires à la fois sur la pile et ses enfants.

L'élément stack pourrait être utilisé quand un indicateur d'état doit d'être ajouté au dessus d'un élément existant. Par exemple, un indicateur de progression pourrait être créé avec une barre et un libellé en surimpression.



Une utilisation pratique de l'élément stack est de pouvoir simuler un certain nombre de propriétés CSS avec. Par exemple, vous pouvez créer un effet similaire à la propriété text-shadow comme ceci :

Exemple 4.1.1 :

```
<stack>
  <description value="Ombré" style="padding-left: 1px; padding-top: 1px; font-size: 15pt"/>
  <description value="Ombré" style="color: red; font-size: 15pt;"/>
</stack>
```

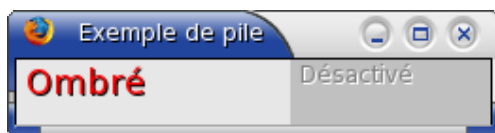
Les deux éléments `<description>` créent chacun un texte avec une taille de 15 points. Le premier est toutefois décalé d'un pixel vers la droite et vers le bas grâce à l'ajout d'une marge de texte sur la gauche et sur le haut. Le résultat est le dessin répété du même texte 'Ombré' mais en léger décalage. Le second élément `<description>` est dessiné en rouge pour que l'effet soit encore plus visible.

Cette méthode a des avantages sur l'emploi de `text-shadow` car vous pouvez complètement personnaliser les effets d'ombres de votre texte principal. Il peut avoir sa propre police de caractères, un soulignement ou une taille propre (vous pouvez même créer un texte ombré clignotant). C'est aussi utile car Mozilla ne supporte pas pour l'instant les textes ombrés en CSS. Un désavantage est que l'espace occupé par le texte ombré fait une pile plus grande. L'effet d'ombrage est très utile pour créer des boutons d'apparence désactivée :

Exemple 4.1.2 :

```
<stack style="background-color: #C0C0C0">
  <description value="Désactivé" style="color: white; padding-left: 1px; padding-top: 1px;"/>
  <description value="Désactivé" style="color: grey;"/>
</stack>
```

Cet arrangement de couleurs de texte et d'ombrage créent cet aspect désactivé que l'on retrouve sur certaines plates-formes.



Notez que les événements tels que les clics de la souris et les

touches du clavier sont transmis à l'élément sur le haut de la pile qui est le dernier élément de la pile. Cela signifie qu'un bouton ne fonctionnera correctement que s'il est le dernier élément de la pile.

## Paquets

Un élément `<deck>` place également ses enfants les uns au dessus des autres comme l'élément `<stack>`, toutefois les paquets n'affichent qu'un seul de leurs enfants à la fois. Ce comportement s'avère utile pour une interface d'assistant dans laquelle une série de panneaux similaires sont affichés en série. Au lieu de créer des fenêtres séparées et d'ajouter des boutons de navigation à chacune d'elles, vous n'avez qu'à créer une seule fenêtre et utiliser un paquet dans lequel le contenu changera.

Comme pour les piles, les enfants directs d'un élément `<deck>` forment les pages du paquet. S'il y a trois enfants dans l'élément `<deck>`, le paquet aura trois enfants. La page affichée du paquet peut être changée en définissant un attribut `selectedIndex` sur l'élément `<deck>`. L'index est un nombre qui identifie quelle page à afficher. Les pages sont numérotées à partir de zéro. Ainsi, le premier enfant du paquet est la page 0, le second est la page 1, etc.

Ce qui suit est un exemple de paquet :

Exemple 4.1.3 :

```
<deck selectedIndex="2">
  <description value="Ceci est la première page "/>
  <button label="Ceci est la seconde page "/>
  <box>
    <description value="Ceci est la troisième page "/>
    <button label="Ceci est également la troisième page "/>
  </box>
</deck>
```

Ici, il y a trois pages, celle affichée par défaut est la troisième. La troisième page est une boîte avec deux éléments qui y sont inclus. L'ensemble de cette boîte et de ses éléments forme une page. Le paquet sera aussi grand que le plus grand de ses enfants, qui dans le cas présent sera le troisième.

Vous pouvez afficher les pages en utilisant un script pour modifier l'attribut `selectedIndex`. Pour plus de renseignements là dessus, reportez vous aux sections sur les événements et le DOM.

La prochaine section décrit comment les piles peuvent être utilisées pour positionner les éléments.

## 4.2 Positionnement dans une pile

Écrit par Neil Deakin. Traduit par *Alain B.* (15/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/bulletins.html>

Cette section va décrire comment positionner des items dans une pile.

### Placement des enfants d'une pile

Normalement, les éléments enfants d'une pile s'étirent pour s'ajuster à la dimension de la pile. Toutefois, vous pouvez aussi placer ces enfants à des coordonnées spécifiées. Par exemple, si la pile a deux boutons comme enfants, l'un d'eux peut être placé à 20 pixels du bord gauche et 50 pixels du bord supérieur. Le second bouton peut être placé à 100 pixels du bord gauche et 5 pixels du bord supérieur.

La position d'un élément enfant doit être précisée en plaçant deux attributs sur chaque élément. Pour le positionnement horizontal, utilisez l'attribut `left` et pour le positionnement vertical, utilisez l'attribut `top`. Si vous ne mettez pas ces attributs sur un enfant de la pile, celui-ci va s'étirer pour s'ajuster à la dimension de la pile.

Exemple 4.2.1 :

```
<stack>
  <button label="Goblins" left="5" top="5"/>
  <button label="Trolls" left="60" top="20"/>
  <button label="Vampires" left="10" top="60"/>
</stack>
```



La pile stack contient trois éléments, chacun d'eux est positionné aux

coordonnées précisées par les attributs `left` et `top`. Ici, les trois éléments enfants sont des boutons, mais les éléments n'ont pas à être tous du même type. Il peut y avoir n'importe quels éléments, même des boîtes ou d'autres piles.

La dimension d'une pile est déterminée par les positions des ses éléments enfants. Elle est toujours dimensionnée pour que tous ses éléments enfants soient visibles. Ainsi, si vous initialisez à l'attribut `left` d'un enfant la valeur de `400`, la pile aura une largeur d'environ 400 pixels plus la largeur de cet élément enfant. Vous pouvez dépasser cette taille avec des propriétés de style variées telles que `width` et `max-width`.

Vous pouvez utiliser un script pour ajuster les valeurs des attributs `left` et `top` et de ce fait, rendre les éléments mobiles. Les piles ont l'avantage que lorsqu'un élément positionné de façon absolue change sa

position, la position des autres éléments n'est pas affectée. Si vous essayez de déplacer des éléments dans une boîte normale, les autres éléments vont bouger en réaction.

Il est également possible de placer des éléments enfants de telle manière qu'ils se superposent. Lorsque les éléments enfants se dessinent, ils sont affichés dans l'ordre où ils sont apparus dans la pile. De ce fait, le premier enfant d'une pile apparaît en arrière plan, suivi du second et ainsi de suite. Le dernier enfant apparaît au premier plan. Vous pouvez utiliser les fonctions DOM pour modifier l'ordre des éléments.

Lorsqu'ils répondent aux événements de la souris, seuls les éléments au premier plan vont capturer les événements en premier. Cela signifie que si deux boutons se superposent, le bouton au premier plan va capturer le clic de la souris à l'endroit où il couvre l'autre bouton.

---

La prochaine section décrit les onglets qui sont comme des piles mais fournissant leur propre navigation.

## 4.3 Onglets

Écrit par Neil Deakin. Traduit par *Alain B.* (18/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/tabpanel.html>

Il est courant de voir apparaître des pages d'onglets dans les boîtes de dialogues de préférences. Nous allons trouver ici comment les créer.

### Boîtes d'onglets

Les boîtes d'onglets sont typiquement utilisées dans une application de fenêtre de préférences. Une série d'onglets apparaît en travers du bord supérieur de la fenêtre. L'utilisateur peut cliquer sur chaque onglet pour voir la sélection des options. L'emploi d'onglets est très utile lorsque vous avez plus d'options qu'il ne peut en tenir sur l'écran.

XUL offre une méthode pour créer de telles boîtes de dialogues. Elles nécessitent cinq nouveaux éléments qui sont décrits brièvement ici et plus en détail par la suite.

#### tabbox

La boîte externe qui contient les onglets sur le haut et les pages correspondantes elles mêmes.

#### tabs

La boîte interne qui contient les onglets individuellement. En d'autres termes, il s'agit d'un groupement d'onglets.

#### tab

Un onglet spécifique. Cliquer sur un onglet remonte la page de l'onglet au premier plan.

#### tabpanel

Le container des pages.

#### tabpanel

Le corps d'une page seule. Vous allez placer le contenu d'une page dans cette élément. Le premier tabpanel correspond au premier onglet, le second tabpanel correspond au second onglet, et ainsi de suite.

L'élément tabbox est l'élément externe. Il est constitué de deux enfants, l'élément tabs qui contient les en-têtes des onglets, et l'élément tabpanel qui contient les pages d'onglets.

La syntaxe typique d'une boîte d'onglets est décrite ci dessous :

```
<tabbox id="tablist">
  <tabs>
    -- les éléments tab viennent ici --
  </tabs>
```

```

<tabpanel>
  -- les éléments tabpanel viennent ici --
</tabpanel>
</tabbox>

```

Les éléments `tab` sont placés à l'intérieur d'un élément `tabs` qui se comporte comme une boîte normale. L'élément `tabs` lui-même a été placé à l'intérieur d'un élément `tabbox`. Cet élément `tabbox` contient aussi l'élément `tabpanel` qui apparaît en dessous de l'élément `tabs` à cause de la disposition verticale de la boîte d'onglets.

Il n'y a réellement rien de spécial au sujet des éléments `tab` qui fassent d'eux quelque chose de différent des boîtes. La différence est que les onglets ont un rendu sensiblement différent et seulement le contenu de la page d'un seul onglet sera visible à la fois, comme peut le faire l'élément `deck`.

Le contenu des pages correspondant à chaque onglet doit être placé dans chaque élément `tabpanel` correspondant. Il ne doit pas être mis dans un élément `tab` qui contient, lui, le contenu descriptif de l'onglet sur le bord supérieur (et qui peut contenir n'importe quels éléments lui-même).

Chaque élément `tabpanel` devient une page de l'onglet affiché. La première page correspond au premier onglet, la seconde page correspond au second onglet, et ainsi de suite. Il y a une relation de un-à-un entre chaque élément `tab` et chaque élément `tabpanel`.

Pour déterminer la dimension d'une boîte d'onglets, la taille de la plus grande page est utilisée. Ainsi, si vous avez dix champs de saisie sur une page et seulement un sur une autre page, cette seconde page sera dimensionnée pour s'ajuster à celle qui contient les dix champs de saisie en occupant plus d'espace. La surface occupée par la page d'un onglet ne change pas lorsque l'utilisateur bascule vers une page d'un autre onglet.

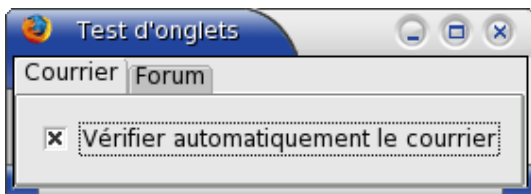
Regardons un exemple contenant deux onglets :

Exemple 4.3.1 :

```

<tabbox>
  <tabs>
    <tab label="Courrier"/>
    <tab label="Forum"/>
  </tabs>
  <tabpanel>
    <checkbox label="Vérifier automatiquement le courrier"/>
  </tabpanel>
  <tabpanel id="newstab">
    <button label="Effacer le cache des forums"/>
  </tabpanel>
</tabpanel>
</tabbox>

```



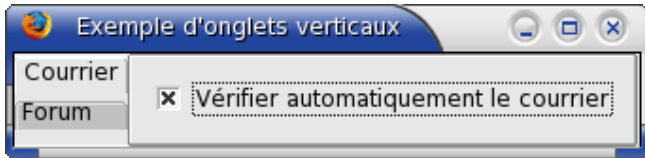
Ici, deux onglets ont été ajoutés. Le premier est intitulé

'Courrier' et l'autre est intitulé 'Forum'. Lorsque l'utilisateur clique sur l'onglet 'Courrier', le contenu de la première page est affiché. Dans ce cas, la boîte avec la case à cocher intitulé 'Vérifier automatiquement le courrier' apparaîtra sous cet onglet. Le second onglet, lorsqu'il est cliqué, affichera la boîte contenant le bouton intitulé 'Effacer le cache des forums'. Dans ce code, les deux pages d'onglet ont été nommées 'Courrier' et 'Forum'.



L'onglet courant sélectionné est donné par un attribut supplémentaire `selected` initialisé à `true` sur un élément `tab`. Il sert à donner à l'onglet par défaut un aspect différent de telle façon qu'il apparaisse sélectionné. Un seul onglet peut avoir cet attribut avec une valeur `true`.

Finalement, vous pouvez changer la position des onglets pour qu'ils apparaissent sur n'importe quel côté des pages d'onglets. Il n'y a pas de syntaxe spéciale pour faire cela. Vous n'avez qu'à simplement arranger la position des onglets et de spécifier l'attribut `orient` si nécessaire. Souvenez vous que les éléments `tab` sont des boîtes normales en terme de mise en page. Toutefois, vous devriez laisser les onglets sur le bord supérieur, autrement ils risquent de ne pas avoir une belle apparence avec certains thèmes graphiques.



Par exemple, pour mettre les onglets le long du bord

gauche, changez l'orientation de l'élément `tabs` pour qu'elle soit verticale. Faites ceci si vous voulez voir apparaître les onglets empilés les uns sur les autres. De plus, ajustez l'élément `tabbox` pour qu'il ait une orientation horizontale. Cet ajustement est nécessaire car vous voulez que les onglets apparaissent à côté de leurs pages. (NdT : )

Vous pouvez placer les onglets le long du bord droit en déplaçant l'élément `tabs` en dessous de l'élément `tabpanel`.

## Ajout d'onglets à la boîte de dialogue de recherche de fichiers

Ajoutons une seconde page à notre boîte de dialogue de recherche de fichiers. Nous allons créer un onglet 'Options' qui contiendra quelques options de recherche. Ce n'est peut être pas la meilleure interface pour faire cela, mais nous l'utiliserons pour la démonstration des onglets. Le libellé d'en-tête et la boîte de critères de recherche iront dans la première page d'onglet. Nous allons ajouter quelques options dans une seconde page d'onglet. La barre de progression et les boutons peuvent rester sur la boîte de dialogue principale, en dehors des onglets.

```
< vbox flex="1">
< tabbox>
  < tabs>
    < tab label="Recherche" selected="true"/>
    < tab label="Options"/>
  </ tabs>

  < tabpanels>
    < tabpanel id="searchpanel" orient="vertical">

      < description>
        Entrez votre critère de recherche ci dessous et appuyer sur le bouton Rechercher.
      </ description>

      < spacer style="height: 10px"/>

      < groupbox orient="horizontal">
        < caption label="Critère de recherche"/>

        < menulist id="searchtype">
          < menupopup>
            < menuitem label="Nom"/>
            < menuitem label="Taille"/>
            < menuitem label="Date de modification"/>
          </ menupopup>
        </ menulist>
        < spacer style="width: 10px;"/>
      </ groupbox>
    </ tabpanel>
  </ tabpanels>
</ vbox>
```

```

<menulist id="searchmode">
  <menupopup>
    <menuitem label="Est" />
    <menuitem label="N'est pas" />
  </menupopup>
</menulist>

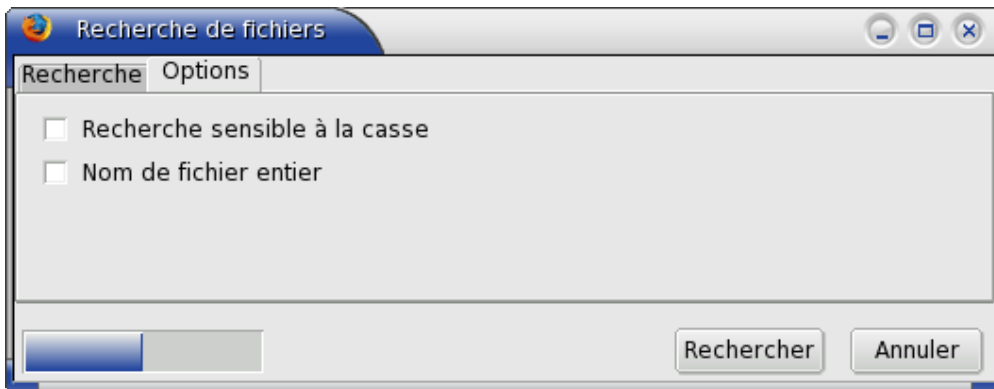
<spacer style="height: 10px" />
<textbox id="find-text" flex="1" style="min-width: 15em;" />

</groupbox>
</tabpanel>

<tabpanel id="optionspanel" orient="vertical">
  <checkbox id="casecheck" label="Recherche sensible à la casse" />
  <checkbox id="wordcheck" label="Nom de fichier entier" />
</tabpanel>

</tabpanel>
</tabbox>
</vbox>

```



Les éléments `tab` ont été placés autour du contenu principal de la fenêtre. Vous pouvez voir les deux onglets, 'Recherche' et 'Options'. En cliquant sur chacun d'eux, vous faites apparaître les pages correspondantes. Comme montré sur l'image ci-dessus, les deux options de recherche apparaissent sur le deuxième onglet. La première page ressemble beaucoup à ce qu'elle était avant, exceptée la présence des onglets sur le bord supérieur.

Exemple de recherche de fichiers :

---

Dans la section suivante, nous allons voir comment créer des grilles pour le placement de contenus.

## 4.4 Grilles

Écrit par Neil Deakin. Traduit par **Alain B.** (09/07/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/grids.html>

XUL dispose d'une série d'éléments pour créer des grilles sous forme de tableaux.

### Disposition tabulaire XUL

XUL dispose d'un jeu d'éléments pour la mise en page sous la forme d'une grille en utilisant l'élément `grid`. Il a quelques similitudes avec la balise HTML `table`. La grille n'affiche rien du tout ; elle ne sert qu'à positionner d'autres éléments en ligne et en colonne.

Une grille contient des éléments qui sont alignés comme avec des tableaux. À l'intérieur d'un élément grid, vous déclarez deux choses, les colonnes et les lignes qui sont utilisées. À l'instar des tableaux HTML, vous pouvez mettre du contenu tels que des libellés et des boutons à l'intérieur des lignes. Toutefois, la grille permet un arrangement soit en ligne, soit en colonne pour y mettre votre contenu. Il est fréquent de l'utiliser en ligne comme avec un tableau. Mais vous pouvez utiliser des colonnes pour définir la taille et l'apparence des colonnes dans une grille. Autrement, vous pouvez mettre du contenu à l'intérieur de colonnes, et utiliser les lignes pour définir l'apparence. Nous étudierons d'abord comment organiser les éléments par ligne.

Pour déclarer une série de lignes, utilisez l'élément rows, qui doit être un élément enfant de grid. À l'intérieur, vous devez ajouter les éléments row, qui représentent chacune des lignes. À l'intérieur d'un élément row, vous devez mettre le contenu que vous souhaitez sur cette ligne.

De même, les colonnes sont déclarées avec l'élément columns, qui doit être un élément enfant de grid. À l'intérieur de cet élément viennent les éléments individuels column, un pour chaque colonne que vous voulez dans la grille.

Il est plus simple de comprendre avec un exemple.

Exemple 4.4.1 :

```
<grid flex="1">
  <columns>
    <column flex="2"/>
    <column flex="1"/>
  </columns>

  <rows>
    <row>
      <button label="Lapin"/>
      <button label="Éléphant"/>
    </row>
    <row>
      <button label="Koala"/>
      <button label="Gorille"/>
    </row>
  </rows>
</grid>
```



Deux lignes et deux colonnes ont été ajoutées dans une grille.

Chaque colonne est déclarée avec l'élément column. Un attribut `flex` a été assigné à chacune des colonnes. Chaque ligne contient deux éléments qui sont des boutons. Le premier élément de chaque élément row est placé dans la première colonne de la grille, et le second élément de chaque ligne est placé dans la seconde colonne.

Notez que vous n'avez pas d'élément pour définir une cellule (il n'y a pas d'équivalent à l'élément HTML `td`). Au lieu de cela, vous placez vos contenus de cellules directement dans les éléments row.

Bien entendu, vous pouvez utiliser n'importe quel autre élément que l'élément button. Si vous voulez une cellule particulière contenant de multiples éléments, vous pouvez utiliser une boîte imbriquée hbox ou tout autre élément boîte. Une boîte hbox représente un seul élément, mais qui peut contenir autant d'éléments que vous le souhaitez. Par exemple :

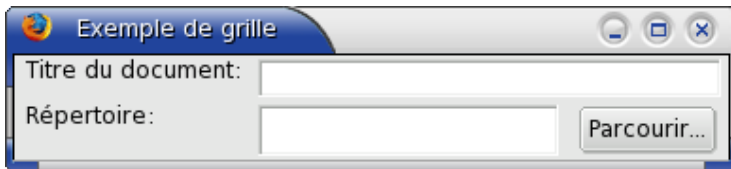
Exemple 4.4.2 :

```

<grid flex="1">
  <columns>
    <column/>
    <column flex="1"/>
  </columns>

  <rows>
    <row>
      <label control="doctitle" value="Titre du document:"/>
      <textbox id="doctitle" flex="1"/>
    </row>
    <row>
      <label control="docpath" value="Répertoire:"/>
      <hbox flex="1">
        <textbox id="docpath" flex="1"/>
        <button label="Parcourir..."/>
      </hbox>
    </row>
  </rows>
</grid>

```



Remarquez sur l'image ci-contre comment la

première colonne contenant les libellés a seulement un unique élément pour chaque ligne. La seconde colonne contient sur sa seconde ligne une boîte qui elle même contient deux éléments, textbox et button. Vous pouvez ajouter d'autres boîtes imbriquées ou une autre grille dans une simple cellule.

Si vous redimensionnez la fenêtre du dernier exemple, vous verrez que les champs de saisie s'ajustent en conséquence, mais pas les autres éléments. En effet, des attributs `flex` ont été ajoutés à ces champs de saisie et à la seconde colonne. La première colonne n'a pas besoin d'être flexible car les libellés n'ont pas besoin de changer de taille.

La largeur initiale d'une colonne est déterminée par le plus large de ses éléments. De même, la hauteur d'une ligne est déterminée par la taille des éléments sur cette ligne. Vous pouvez employer des propriétés CSS `minwidth`, `maxwidth` et propriétés similaires pour affiner les dimensions.

Vous pouvez également placer des éléments à l'intérieur des éléments column au lieu des éléments row. En procédant de la sorte, les lignes sont justes déclarées pour définir leur nombre.

Exemple 4.4.3 :

```

<grid>
  <rows>
    <row/>
    <row/>
    <row/>
  </rows>

  <columns>
    <column>
      <label control="first" value="Premier nom:"/>
      <label control="middle" value="Nom central:"/>
      <label control="last" value="Dernier nom:"/>
    </column>
    <column>
      <textbox id="first"/>
      <textbox id="middle"/>
      <textbox id="last"/>
    </column>
  </columns>
</grid>

```

```

    </column>
  </columns>

</grid>

```

Cette grille a trois lignes et deux colonnes. Les éléments `row` servent juste à définir combien de lignes la grille contient. Vous pouvez ajouter un attribut `flex` à une ligne pour la rendre flexible. Le contenu est placé dans chaque colonne. Le premier élément de chaque élément `column` est placé sur la première ligne, le second élément sur la seconde ligne et le troisième élément sur la troisième ligne.

Si vous placez du contenu en même temps sur les colonnes et sur les lignes, le contenu va se superposer l'un sur l'autre, même s'il est aligné correctement dans la grille. Cet effet correspondrait à une grille d'éléments stack.

L'ordre des éléments dans la grille détermine lequel est affiché au premier plan et lequel est affiché à l'arrière plan. Si l'élément `rows` est placé après l'élément `columns`, le contenu des lignes est affiché au premier plan. Si l'élément `columns` est placé après l'élément `rows`, le contenu des colonnes est affiché au premier plan. De même, les événements tels que les clics de souris et les touches de clavier sont seulement envoyés aux objets du premier plan. C'est pour cela que les colonnes sont définies après les lignes dans l'exemple ci dessus. Si les colonnes avaient été placées en premier, les lignes auraient capturées les événements et aucun texte n'aurait pu être saisi dans les champs de saisie.

Un des avantages des grilles sur une série de boîtes imbriquées est que vous pouvez créer des cellules qui sont flexibles aussi bien horizontalement que verticalement. Il vous suffit de mettre un attribut `flex` sur les lignes et colonnes concernées. L'exemple suivant en fait la démonstration :

Exemple 4.4.4 :

```

<grid flex="1">
  <columns>
    <column flex="5"/>
    <column/>
    <column/>
  </columns>
  <rows>
    <row flex="10">
      <button label="Cerise"/>
      <button label="Citron"/>
      <button label="Raisin"/>
    </row>
    <row flex="1">
      <button label="Fraise"/>
      <button label="Framboise"/>
      <button label="Pêche"/>
    </row>
  </rows>
</grid>

```

La première colonne et l'ensemble des lignes ont été rendus flexibles. Ainsi, chaque cellule de la première colonne est flexible horizontalement. De plus, chaque cellule est flexible verticalement car l'ensemble des lignes dispose de l'attribut `flex`. La cellule de la première colonne de la première ligne (le bouton *Cerise*) sera flexible d'un facteur 5 horizontalement et d'un facteur 10 verticalement. La cellule suivante (le bouton *Citron*) ne sera flexible que verticalement.

L'attribut `flex` a également été ajouté à l'élément `grid` afin que toute la grille soit flexible, autrement elle ne s'agrandirait que dans une seule direction.

## Étendre une colonne

Cela n'a aucun sens d'étendre une cellule sur un nombre particulier de colonnes ou de lignes multiples. Toutefois, il est possible de faire qu'une ligne ou qu'une colonne s'étende sur toute la largeur ou la hauteur de la grille. Il vous suffit d'ajouter un élément à l'intérieur d'un élément `<rows>` et qui ne soit pas à l'intérieur d'un élément `<row>`. Vous pouvez utiliser un type de boîte par exemple, d'y placer dedans d'autres éléments si vous devez en utiliser plusieurs. Voici un exemple simple :

Exemple 4.4.5 :

```
<grid>
  <columns>
    <column flex="1" />
    <column flex="1" />
  </columns>

  <rows>
    <row>
      <label value="Nord Ouest" />
      <label value="Nord Est" />
    </row>
    <button label="Équateur" />

    <row>
      <label value="Sud Ouest" />
      <label value="Sud Est" />
    </row>
  </rows>
</grid>
```

Le bouton va s'étendre pour s'ajuster sur toute la largeur de la grille comme s'il n'était pas à l'intérieur d'une ligne de la grille. Vous pouvez utiliser une technique similaire pour ajouter un élément entre deux colonnes. Il s'étendra pour s'ajuster sur toute la hauteur de la grille. Vous pouvez combiner les deux si vous le souhaitez.

---

Dans la section suivante, nous verrons comment ajouter des panneaux de contenu.

## 4.5 Cadres de contenu

Écrit par Neil Deakin. Traduit par **Alain B.** (19/02/2004), mise à jour par Gerard L. (26/03/2005) .

Page originale : <http://www.xulplanet.com/tutorials/xultu/cpanels.html>

Dans cette section, nous regarderons comment ajouter des cadres qui peuvent afficher des pages HTML ou d'autres fichiers XUL.

### Ajout de cadres enfants

Il y a des fois où vous souhaitez qu'une partie d'un document soit chargée à partir d'une autre page. D'autres fois, vous voulez changer une partie de la fenêtre. Un bon exemple est un assistant qui vous guide pas-à-pas à travers un certain nombre d'écrans, en vous posant une série de questions. Chaque fois que l'utilisateur clique sur le bouton suivant, l'écran suivant de l'assistant est affiché.

Vous pouvez créer une interface d'assistant en ouvrant une fenêtre différente pour chaque écran. Il y a trois problèmes avec cette approche. Premièrement, chaque fenêtre peut apparaître à une position différente sur l'écran (bien qu'il existe une alternative à cela). Deuxièmement, les éléments tels que les boutons suivant ou précédent sont les mêmes tout au long de l'interface. Ce serait bien mieux si seul le contenu de l'assistant changeait. Troisièmement, il sera difficile de coordonner les scripts tournant dans les différentes fenêtres.

Notez que XUL a un élément wizard qui peut être utilisé pour créer des assistants. Il sera décrit dans une prochaine section.

Une meilleure approche est d'utiliser l'élément iframe, qui fonctionne comme l'élément HTML du même nom. Il crée un document séparé dans la fenêtre. Il a l'avantage de pouvoir être placé n'importe où et son contenu chargé à partir d'un fichier différent. Indiquez l'URL que vous souhaitez afficher dans le cadre avec l'attribut `src`. Cette URL peut pointer sur n'importe quelle sorte de fichiers, bien qu'elle pointe habituellement sur un fichier HTML ou un autre fichier XUL. Vous pouvez utiliser un script pour changer le contenu de ce cadre sans affecter la fenêtre principale.

Dans la fenêtre du navigateur Mozilla, l'endroit, où la page web est affichée, est créé en utilisant un cadre iframe. Lorsque l'utilisateur entre une URL ou clique sur un lien du document, la source du cadre est changée.

L'exemple suivant utilise un cadre iframe :

Exemple 4.5.1 :

```
<toolbox>
  <toolbar id="nav-toolbar">
    <toolbarbutton label="Précédent"/>
    <toolbarbutton label="Suivant"/>
    <textbox id="urlfield"/>
  </toolbar>
</toolbox>

<iframe id="content-body" src="http://www.mozilla-europe.org/fr/" flex="1"/>
```



Ici, l'exemple crée une interface très simple pour un navigateur Web. Une boîte contenant deux éléments a été créée : un élément toolbox et un élément iframe. Un bouton 'Précédent', un bouton 'Suivant' et un champ de saisie des URLs ont été ajoutés sur la seule barre d'outils. Les pages Web apparaissent à l'intérieur du cadre iframe. Dans ce cas, la page d'accueil du site apparaîtra par défaut.

Cet exemple n'est pas fonctionnellement complet. Dans la suite, nous voudrions ajouter un script qui change l'attribut de `src` au moment désiré, par exemple quand l'utilisateur appuie sur la touche **Entrée**.

## Navigateurs

Il y a un deuxième type de cadre de contenu, utilisant la balise browser. Vous l'utiliserez quand vous voudrez créer un cadre qui montre le contenu comme un navigateur. En réalité, l'élément iframe peut aussi faire cela, mais le navigateur possède une variété de caractéristiques additionnelles. Par exemple, l'élément browser conserve un historique pour utiliser les boutons Précédent et Suivant. Le navigateur peut aussi charger des pages avec des `referers` et d'autres états. La balise browser devrait être utilisée essentiellement lorsque vous voulez créer une interface semblable au navigateur, mais l'iframe peut être utilisé lorsque vous avez juste besoin d'un cadre simple.

Un élément similaire, `tabbrowser`, fournit la fonctionnalité du `browser` mais fournit également une barre d'onglets pour basculer entre de multiples pages. C'est le composant graphique utilisé par le navigateur Mozilla pour son interface de navigation par onglets. L'élément `tabbrowser` est en réalité implémenté comme un `tabbox` contenant un ensemble d'éléments `browser`. Ces deux types de navigateur offrent un contrôle similaire sur les pages à afficher.

Voici un exemple de navigateur :

Exemple 4.5.2 : .

```
<browser src="http://www.mozilla.org" flex="1"/>
```

Comme avec l'`iframe`, vous pouvez indiquer l'URL dans un navigateur utilisant l'attribut `src`. Pour un `tabbrowser`, vous ne pouvez pas définir l'URL directement comme ceci, étant donné qu'il n'affiche pas qu'une seule URL. À la place, vous devez utiliser un script et appeler la fonction `loadURI`.

Il y a trois catégories de navigateurs, selon le genre de contenu que vous voulez y afficher à l'intérieur. Le type peut être indiqué en utilisant le `type`. Le premier type est le type par défaut utilisé si vous omettez cet attribut. Dans ce cas, le contenu chargé à l'intérieur du navigateur est traité comme s'il faisait partie de la même application et avait accès à la fenêtre extérieure. Cela signifie que lorsqu'un script chargé à l'intérieur du navigateur essaie d'obtenir la fenêtre la plus élevée, il obtiendra la fenêtre XUL extérieure.

Ce comportement conviendrait pour un cadre XUL fils qui fait partie de votre application, mais il ne serait pas adapté pour un navigateur qui chargerait des pages Web. À la place, vous voudrez restreindre la page Web pour obtenir seulement l'accès au contenu de la page Web. Vous pouvez noter qu'une fenêtre de navigateur Mozilla possède du contenu XUL pour les barres d'outils et de statut et ainsi de suite avec un `tabbrowser` formant la zone principale. Cette zone interne affiche une page Web, mais celle-ci ne peut pas accéder au XUL l'entourant. C'est parce le second type de navigateur est utilisé en définissant l'attribut `type` avec la valeur `content`. Il empêche le contenu de remonter jusqu'à la fenêtre XUL. Un exemple :

```
<browser src="http://www.mozilla.org" type="content" flex="1"/>
```

Il est important que vous définissiez l'attribut `type` correctement si vous envisagez d'afficher des sites Web distants à l'intérieur de l'élément `browser`. Le `tabbrowser` définit le type `content` automatiquement sur tous les onglets de navigation qu'il crée. Vous n'avez donc pas à le définir explicitement pour les onglets de navigation.

Le troisième type est utilisé pour indiquer le contenu primaire à l'intérieur de la fenêtre. Le navigateur par onglets le définit automatiquement pour le cadre de navigation qui est visible. Mais vous pouvez le définir sur un `browser` si vous en avez plus d'un dans une fenêtre, par exemple si vous avez une barre latérale affichant d'autres contenus. Définissez l'attribut `type` avec la valeur `content-primary` pour spécifier le contenu primaire. Il fonctionne comme avec la valeur `content` excepté que le contenu à l'intérieur est accessible en utilisant la propriété 'content' de la fenêtre XUL. L'accès au contenu du navigateur principal en utilisant un script en est simplifié. C'est une méthode particulièrement pratique lorsqu'on utilise un navigateur par onglets, étant donné que vous aurez toujours besoin d'accéder au contenu visible de cette façon.

---

Dans la section suivante, nous allons voir comment créer des séparateurs.

## 4.6 Séparateurs

Écrit par Neil Deakin. Traduit par *Alain B.* (20/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/splitter.html>



Ici, nous allons voir comment ajouter des séparateurs de fenêtre.

## Découpage d'une boîte

Il y a des fois où vous voulez avoir deux sections dans une fenêtre que l'utilisateur peut redimensionner à son gré. Un exemple est la fenêtre du navigateur Mozilla, où vous pouvez changer la taille du panneau latéral en déplaçant le trait d'intersection des deux cadres. Vous pouvez aussi cacher ce panneau latéral en cliquant sur la poignée de ce trait.

Cette fonctionnalité est accomplie en utilisant un élément appelé *splitter*. Il crée une barre étroite entre les deux sections en permettant de redimensionner chaque côté. Vous pouvez placer un séparateur où vous voulez pour vous permettre de redimensionner les éléments situés avant lui et les éléments situés après lui dans une même boîte.

Lorsqu'un séparateur est placé à l'intérieur d'une boîte horizontale, il permet un redimensionnement horizontal. Lorsqu'un séparateur est placé à l'intérieur d'une boîte verticale, il permet un redimensionnement vertical.

La syntaxe d'un séparateur est la suivante :

```
<splitter
  id="identifiant"
  state="open"
  collapse="before"
  resizebefore="closest"
  resizeafter="closest"/>
```

Les attributs sont les suivants :

*id*

L'identifiant unique d'un séparateur.

*state*

Indique l'état d'un séparateur. Mettez lui la valeur *open*, valeur par défaut, pour que le panneau redimensionnable soit initialement ouvert ou mettez lui la valeur *collapsed* pour qu'un des panneaux soit complètement réduit et que l'autre occupe toute la place.

*collapse*

Il indique de quel côté le panneau doit se réduire quand la poignée du séparateur est cliquée ou quand le séparateur est initialisée avec un état *collapsed*. Mettez lui la valeur *before* pour désigner l'élément avant le séparateur ou la valeur *after* pour désigner l'élément après le séparateur. Si vous l'initialisez avec la valeur *none*, qui est aussi la valeur par défaut, la poignée du séparateur ne réduira pas les panneaux quand elle est cliquée.

*resizebefore*

Lorsqu'un séparateur est déplacé, les éléments sur sa gauche sont redimensionnés. Cet attribut indique quel élément est concerné. Mettez la valeur *closest* pour que l'élément immédiatement à gauche du séparateur soit redimensionné. Mettez la valeur *farthest* pour que l'élément le plus éloigné à gauche du séparateur soit redimensionné (le premier élément de la boîte). La valeur par défaut est *closest*.

*resizeafter*

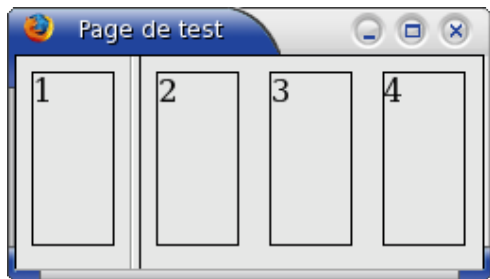
Lorsqu'un séparateur est déplacé, les éléments sur sa droite sont redimensionnés. Cet attribut indique quel élément est concerné. Mettez la valeur *closest* pour que l'élément immédiatement à droite du séparateur soit redimensionné. Mettez la valeur *farthest* pour que l'élément le plus éloigné à droite du séparateur soit redimensionné (le dernier élément de la boîte). Cet attribut peut aussi avoir la valeur *grow*, dans ce cas, les éléments à droite du séparateur ne changent pas de taille lorsque le séparateur est déplacé. La valeur par défaut est *closest*.

Si vous employez l'attribut `collapse`, vous devrez aussi ajouter l'élément `grippy` à l'intérieur de l'élément `splitter` afin que l'utilisateur puisse utiliser la poignée pour réduire un panneau.

Un exemple vous sera utile :

Exemple 4.6.1 :

```
<hbox flex="1">
  <iframe id="content-1" width="60" height="20" src="w1.html"/>
  <splitter collapse="before" resizeafter="farthest">
    <grippy/>
  </splitter>
  <iframe id="content-2" width="60" height="20" src="w2.html"/>
  <iframe id="content-3" width="60" height="20" src="w3.html"/>
  <iframe id="content-4" width="60" height="20" src="w4.html"/>
</hbox>
```



Ici, quatre cadres `iframe` ont été créés et un séparateur a été placé

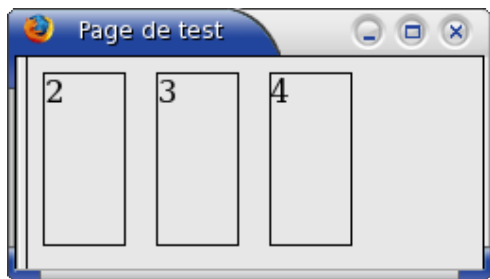
entre le premier et le second. L'attribut `collapse` a été affecté d'une valeur *before* pour signifier que si la poignée est cliquée, le premier cadre va disparaître, et le séparateur et les cadres restants vont glisser vers la gauche. La poignée du séparateur est centrée sur la barre de séparation.

La valeur *farthest* a été affectée à l'attribut `resizeafter` du séparateur. Elle signifie que lorsque le séparateur est déplacé, l'élément le plus éloigné après lui changera sa taille. Dans ce cas, il s'agit du quatrième cadre.

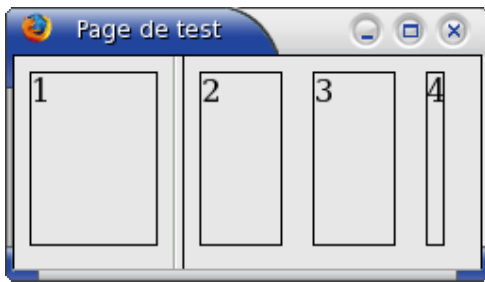
Aucune valeur n'a été spécifiée pour l'attribut `resizebefore` qui aura donc sa valeur *closest* par défaut. Dans ce cas, il n'y a qu'un cadre avant le séparateur, c'est donc le premier cadre qui changera de taille.

Les deuxième et troisième cadres ne changeront leur taille que si vous déplacez le séparateur suffisamment loin sur la droite jusqu'à ce que le quatrième cadre ait atteint sa largeur minimale.

Les quatre cadres avec le séparateur dans un état réduit :



Une image des quatre cadres avec le séparateur redimensionné vers la droite est montrée ci dessous. Notez que les deux cadres du milieu n'ont pas changé leur dimension. Seuls les premier et quatrième cadres ont changé de dimension. Vous pouvez seulement voir une partie du quatrième cadre. Si vous continuez à déplacer le séparateur vers la droite, les deux autres cadres vont se réduire.



Vous pouvez utiliser des propriétés de style telles que `min-width`, `max-width` sur les cadres pour spécifier leurs largeurs minimales ou maximales ou leurs hauteurs dans la boîte. Si vous faites cela, le séparateur va le détecter et ne permettra pas de le déplacer au-delà des tailles minimales ou maximales.

Par exemple, si vous spécifiez un minimum de 30 pixels en largeur sur le quatrième cadre, il ne se réduira pas en dessous de cette taille. Les deux autres cadres vont alors se réduire. Si vous mettez une largeur minimale de 50 pixels sur le premier cadre, vous ne pourrez déplacer le séparateur que de 10 pixels vers la gauche (car il démarre à 60 pixels de large). Toutefois, vous pouvez toujours faire disparaître le cadre.

Vous pouvez également placer plus d'un séparateur dans une boîte si vous le souhaitez, dans ce cas vous pourrez réduire les différentes parties de son contenu. De façon similaire, il n'y a pas que les cadres que vous pouvez réduire, n'importe quel élément peut l'être.

## Exemple de séparateur

Voyons ce que devient notre exemple de recherche de fichiers avec un séparateur. Une possibilité serait d'inclure les résultats de la recherche dans la boîte de dialogue. Nous ajouterons une zone entre les critères de recherche et les boutons du bas. Un séparateur permettra de réduire ou masquer les résultats de la recherche.

```
</tabbox>

<iframe src="results.html"/>
<splitter resizeafter="grow"/>

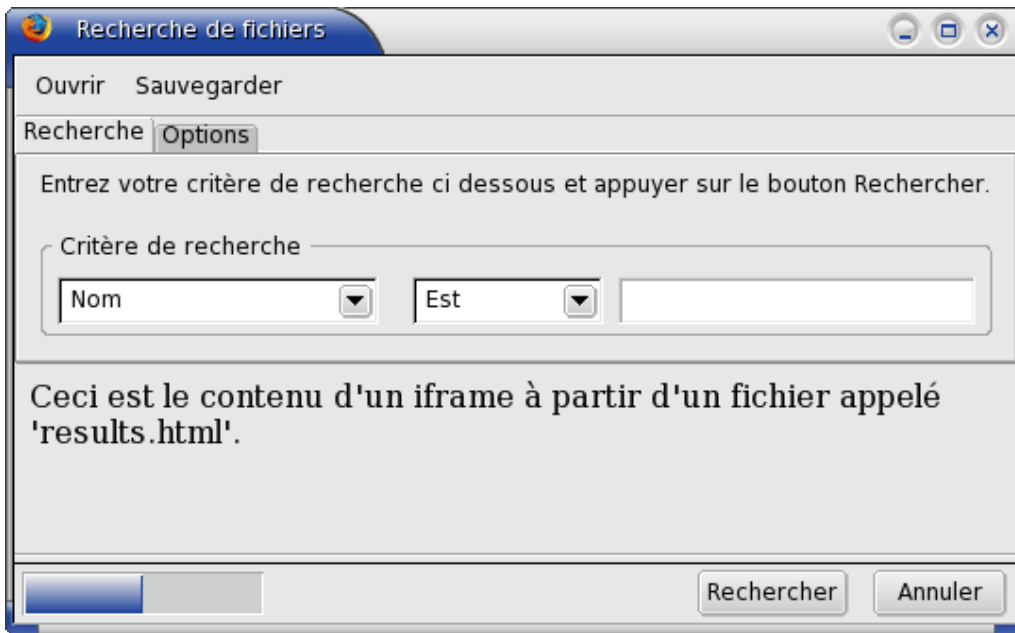
<hbox>
```

Ici, un séparateur et un cadre ont été ajoutés à la boîte de dialogue. Nous n'avons plus besoin de l'élément spacer après la boîte d'onglets, donc nous pouvons l'enlever. Le contenu du cadre provient d'un fichier appelé 'result.html'. Créez ce fichier et mettez ce que vous souhaitez dedans pour l'instant. Le cadre sera remplacé plus tard par la liste des résultats lorsque vous saurez comment la créer. Pour l'instant, il nous servira à expliquer le séparateur.

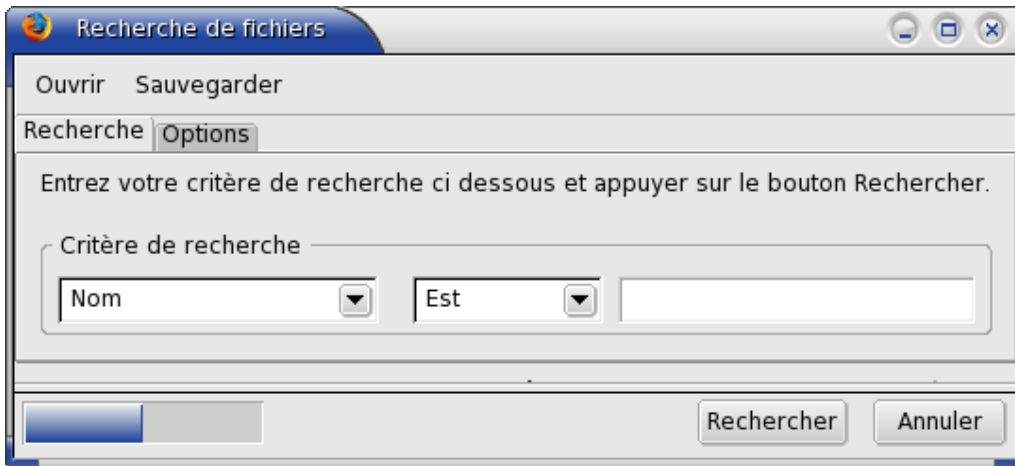
Le séparateur a été initialisé avec la valeur *before* sur l'attribut `collapse`, ce qui signifie que l'élément juste avant le séparateur va se réduire. Ici, il s'agit du cadre. Comme montré sur les images ci-dessous, lorsque la poignée est cliquée, le cadre est masqué et les boutons glissent vers le haut.

L'attribut `resizeafter` a été initialisé avec une valeur *grow* pour que les éléments situés après le séparateur se déplacent vers le bas lorsque celui-ci est déplacé vers le bas. Le contenu dans le cadre peut s'agrandir sans restriction. Vous noterez que la fenêtre ne se redimensionne pas automatiquement. Vous remarquerez également que c'est un séparateur horizontal parce qu'il a été placé dans une boîte verticale.

Etat normal :



Etat réduit :



Exemple de recherche de fichiers :

Dans la section suivante, nous verrons comment créer une barre de défilement.

## 4.7 Barres de défilement

Écrit par Neil Deakin. Traduit par *Alain B.* (09/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/scroll.html>

Maintenant, voyons comment ajouter des barres de défilement à une fenêtre.

### Ajout de barres de défilement

Une barre de défilement est typiquement employée pour permettre à un utilisateur de parcourir un grand document. Vous pouvez aussi l'utiliser quand vous avez besoin de demander une valeur comprise entre un certain intervalle. Les barres de défilement peuvent être créées de différentes manières. Sous XUL, l'une d'entre elle nécessite l'emploi de la balise `scrollbar`. Certains éléments, comme des champs de saisie, vont également ajouter des barres de défilement si nécessaire lorsque leur contenu devient trop grand.

Tout d'abord, nous allons aborder la création d'une barre de défilement seule. L'utilisateur choisira une valeur en déplaçant la barre de défilement. Vous n'utiliserez probablement pas très souvent cette méthode. Une barre de défilement est constituée de plusieurs parties, l'ascenseur, qui est la partie centrale de la barre, et les deux flèches boutons à chaque extrémités. Une barre de défilement crée tous ces éléments automatiquement.



La syntaxe d'une barre de défilement est la suivante :

```
<scrollbar
    id="identifiant"
    orient="horizontal"
    curpos="20"
    maxpos="100"
    increment="1"
    pageincrement="10"/>
```

Les attributs sont les suivants :

*id*

L'identifiant unique de la barre de défilement.

*orient*

Il spécifie l'orientation de la barre de défilement. Sa valeur par défaut *horizontal* crée une barre qui s'étend de gauche vers la droite. Vous pouvez également spécifier *vertical* qui crée une barre s'étendant du haut vers le bas.

*curpos*

Il indique la position actuelle de l'ascenseur (le rectangle que vous pouvez déplacer). La valeur s'étend de 0 à la valeur de *maxpos*. Cette valeur n'a pas besoin d'unité. La valeur par défaut est de 0.

*maxpos*

Il indique la valeur de la position maximale de l'ascenseur. Il s'agit d'une valeur numérique qui n'a pas d'unité. La valeur par défaut est de 100.

*increment*

Ici, le nombre spécifie de combien la valeur de *curpos* doit être changée quand l'utilisateur clique sur une des flèches de la barre de défilement. La valeur par défaut est de 1.

*pageincrement*

Ici, le nombre spécifie de combien la valeur de *curpos* doit être changée quand l'utilisateur clique sur la page à travers la barre de défilement, c'est à dire dans la zone entre l'ascenseur et les flèches. La valeur par défaut est de 10.

L'exemple donné ci-dessus crée une barre de défilement qui s'étend des valeurs de 0 à 100. La valeur de 100 peut être considérée comme le nombre de ligne d'une liste, mais elle peut représenter n'importe quoi d'autre que vous souhaitez. La valeur initiale de cet exemple est de 20. Quand l'une des flèches de la barre de défilement est cliquée, la valeur incrémente ou décrémemente de 1. En cliquant à travers la barre de défilement, la valeur change de 10.

Lorsque l'utilisateur clique sur l'une des flèches de la barre de défilement, l'ascenseur se déplace d'autant qu'indiqué par la valeur *increment*. Augmenter la valeur de cet attribut fera que la barre défilera plus vite à chaque clic. Les positions la plus à gauche et la plus élevée de l'ascenseur ont une valeur de 0, et celles la plus à droite et la plus basse ont une valeur donnée par *maxpos*.

En ajustant les valeurs de la barre de défilement, vous pouvez positionner et contrôler le déplacement de l'ascenseur comme vous le souhaitez.

---

Ensuite, nous verrons comment créer des barres d'outils.

## 5. Menus et barres d'outils

### 5.1 Barres d'outils

Écrit par Neil Deakin. Traduit par *Alain B.* (19/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/toolbar.html>

Une barre d'outils est habituellement placée sur le haut d'une fenêtre et contient un certain nombre de boutons réalisant des opérations communes. XUL a une méthode pour créer des barres d'outils.

#### Ajout d'une barre d'outils

Comme la plupart des éléments, les barres d'outils sous XUL sont un type de boîte. Habituellement, une rangée de boutons apparaît dans une barre d'outils, mais n'importe quel élément peut y être placé. Par exemple, la fenêtre du navigateur Mozilla contient une zone de texte qui permet l'affichage de l'URL de la page.

Les barres d'outils peuvent être placées sur n'importe quel côté d'une fenêtre, soit horizontalement, soit verticalement. Bien sur, vous veillerez à ne pas mettre de champs de saisie dans une barre d'outils verticale. En fait, comme les barres d'outils sont juste des boîtes, vous pouvez les mettre n'importe où, même au milieu d'une fenêtre. Habituellement, toutefois, le positionnement d'une barre d'outils se fait sur le haut d'une fenêtre. Lorsque plus d'une barre d'outils est placée l'une après l'autre, elles sont classiquement groupée ensemble dans une boîte d'outils (`toolbox`).

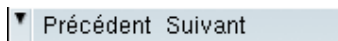
Sur le bord gauche d'une barre d'outils se trouve une petite encoche qui, si elle est cliquée, va réduire la barre d'outils et seule cette encoche restera visible. Cette encoche est appelée une poignée (grippy). Lorsque de multiples barres d'outils sont placées dans le même élément `toolbox`, les poignées vont les réduire à une simple ligne. La totalité de l'espace utilisé est ainsi libérée. Les barres d'outils verticales ont leur poignées sur leur bord supérieur. L'utilisateur réduira habituellement une barre d'outils s'il souhaite disposer de plus de place dans la fenêtre principale.

NdT : Les poignées décrites ici ne s'affichent pas sous Firefox.

Voici un exemple d'une simple barre d'outils dans une boîte d'outils.

Exemple 5.1.1 :

```
<toolbox>
  <toolbar id="nav-toolbar">
    <toolbarbutton label="Précédent"/>
    <toolbarbutton label="Suivant"/>
  </toolbar>
</toolbox>
```

 Cet exemple crée une barre d'outils contenant deux boutons, un bouton 'Précédent' et un bouton 'Suivant'. Cette barre d'outils a été placée à l'intérieur d'un élément `toolbox`. Cet exemple met en jeu quatre nouvelles balises qui sont décrites ici :

#### `toolbox`

Une boîte qui contient les barres d'outils.

#### `toolbar`

Une simple barre d'outils qui contient les items de la barre tels que des boutons. Les barres d'outils peuvent être réduites en utilisant la poignée située sur leur côté gauche ou leur bord supérieur.

#### `toolbarbutton`

Un bouton sur une barre d'outil, qui a exactement les mêmes fonctionnalités qu'un bouton normal mais qui est dessiné différemment.

#### toolbargrippy

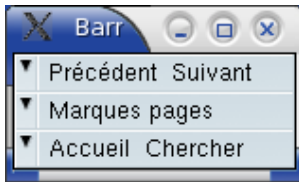
Cet élément crée l'encoche qui est utilisée pour réduire ou restaurer une barre d'outils. Il n'est pas nécessaire de l'utiliser directement car il est ajouté automatiquement.

toolbar est l'élément principal qui crée une barre d'outils. À l'intérieur sont placés des items individuels de la barre d'outils, habituellement des boutons, mais également d'autres éléments. La barre d'outils doit avoir un attribut `id` sinon la poignée risque de ne pas réduire ou restaurer la barre d'outils proprement.

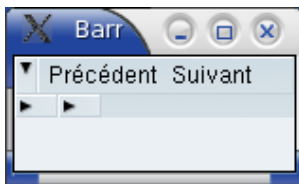
Dans l'exemple ci-dessus, une seule barre d'outils a été créée. De multiples barres d'outils peuvent être créées simplement en ajoutant plus éléments toolbar après le premier.

L'élément toolbox est le container des barres d'outils. Dans certaines applications, vous avez plusieurs barres d'outils sur le haut de la fenêtre. Vous pouvez toutes les placer à l'intérieur de l'élément toolbox.

Les poignées sur une barre d'outils sont créées en utilisant un autre élément appelé toolbargrippy. Cela n'a aucun sens de l'employer en dehors d'une barre d'outils car il n'aura aucun usage s'il n'a rien à réduire. Toutefois, vous pouvez souhaiter changer son style différemment. Vous pouvez cacher une poignée en ajoutant l'attribut `grippyhidden` à l'élément toolbar, et en lui affectant la valeur `true`.



Une boîte d'outils contenant trois barres d'outils. (NdT : )

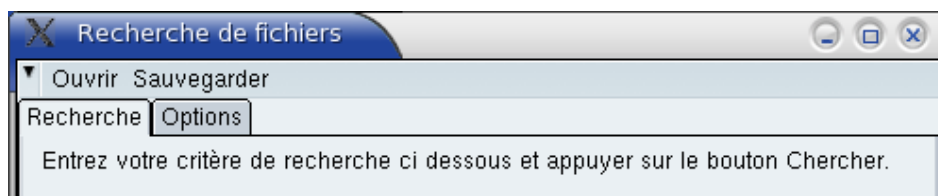


La même boîte où deux barres d'outils ont été réduites.

Ajoutons maintenant une barre d'outils à notre exemple de recherche de fichiers. Nous n'en avons pas réellement besoin mais nous l'ajouterons quand même pour montrer son usage. Deux boutons vont être ajoutés, un bouton 'Ouvrir' et un bouton 'Sauvegarder'. On peut penser qu'ils permettraient à l'utilisateur de sauvegarder les résultats de la recherche et de les rouvrir plus tard.

```
<vbox flex="1">
  <toolbox>
    <toolbar id="findfiles-toolbar">
      <toolbarbutton id="opensearch" label="Ouvrir"/>
      <toolbarbutton id="savesearch" label="Sauvegarder"/>
    </toolbar>
  </toolbox>
</tabbox>
```

Une barre d'outils avec deux boutons a été ajoutée ici. Dans l'image, vous pouvez les voir apparaître horizontalement en haut de la fenêtre. La poignée apparaît également sur le côté gauche de la barre d'outils. Notez que la barre d'outils a été placée à l'intérieur d'une boîte verticale juste au dessus de la boîte d'onglets, car nous avons besoin d'un placement vertical pour que la barre d'outils se place au dessus de tout le contenu.



Exemple de recherche de fichiers :

Nous allons maintenant voir comment ajouter une barre de menu à une fenêtre.

## 5.2 Barres de menu simples

Écrit par Neil Deakin. Traduit par **Vincent S.** (17/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/menubar.html>

Dans cette section, nous allons voir comment créer une barre de menu avec des menus dessus.

### Créer un menu

XUL a plusieurs façons différentes de créer des menus. Le moyen le plus basique est d'ajouter une barre de menu avec une ligne de menus dessus comme sur de nombreuses applications. Vous pouvez aussi créer des menus surgissants. Les fonctionnalités de menu XUL se composent de différents éléments qui vous permettent de créer des barres de menus ou des menus surgissants. Les items des menus peuvent être personnalisés assez facilement. Nous avons déjà vu en partie comment faire des menus en utilisant menulist. Cette section sera construite là-dessus.

Les barres de menu sont généralement créées de la même façon que les barres d'outils. La barre de menu peut-être placée dans une toolbox et une poignée (grippy) apparaîtra sur son côté gauche pour qu'elle puisse être cachée. Le menu fonctionnerait juste comme n'importe quel autre barre d'outil. XUL a quelques éléments spéciaux qui fournissent des fonctionnalités spéciales typiques des menus.

Il y a cinq éléments associés à la création d'une barre de menu et de ses menus, qui sont expliqués brièvement ici et en détail après :

#### menubar

Le container de la ligne de menu.

#### menu

En dépit du nom, il s'agit en fait seulement du titre du menu sur la barre de menu. Cet élément peut être placé sur une barre de menu ou séparément.

#### menupopup

La boîte déroulante qui apparaît quand vous cliquez sur le titre du menu. Cette boîte contient la liste des commandes de menu.

#### menuitem

Une commande individuelle sur un menu. Il est placé dans un menupopup.

#### menuseparator

Une barre de séparation sur un menu. Il est placé dans un menupopup.

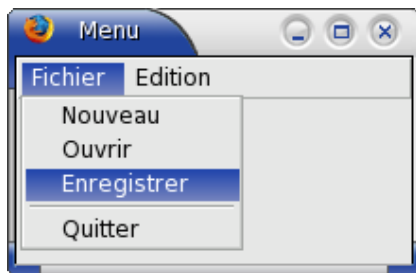
Vous pouvez personnaliser les menus sur la barre de menu comme vous le souhaitez pour toutes les plates-formes excepté pour le Macintosh. C'est parce que le Macintosh a son propre menu spécial en haut de l'écran contrôlé par le système. Bien que vous puissiez créer des menus personnalisés, toutes règles spéciales de style ou éléments non-menu que vous placez sur un menu peuvent ne pas être appliqués. Vous devez garder cette notion à l'esprit quand vous créez des menus.



Voici un exemple d'une simple barre de menu :

### Exemple 5.2.1 :

```
<toolbox flex="1">
  <menubar id="sample-menubar">
    <menu id="file-menu" label="Fichier">
      <menupopup id="file-popup">
        <menuitem label="Nouveau"/>
        <menuitem label="Ouvrir"/>
        <menuitem label="Enregistrer"/>
        <menuseparator/>
        <menuitem label="Quitter"/>
      </menupopup>
    </menu>
    <menu id="edit-menu" label="Edition">
      <menupopup id="edit-popup">
        <menuitem label="Annuler"/>
        <menuitem label="Refaire"/>
      </menupopup>
    </menu>
  </menubar>
</toolbox>
```

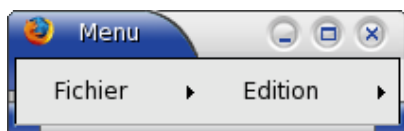


Ici, une simple barre de menu est créée en utilisant l'élément menubar.

Il va créer une ligne pour y placer des menus. Deux menus, Fichier et Edition ont été créés ici. L'élément menu crée le titre en haut du menu, qui apparaît sur la barre de menu. Les menus déroulants sont créés en utilisant l'élément menupopup. Le menu va s'ouvrir quand l'utilisateur cliquera sur le titre du menu parent. La taille du menu sera suffisamment large pour que les commandes tiennent à l'intérieur. Les commandes elles-mêmes sont créées en utilisant l'élément menuitem. Chacune d'elles représente une unique commande dans le menu déroulant.

Vous pouvez aussi créer des séparateurs sur les menus en utilisant l'élément menuseparator. Il est utilisé pour séparer des groupes d'items de menu.

La barre de menu est une boîte contenant des menus. Notez qu'elle a été placée dans une toolbox flexible. La barre de menu n'a pas d'attribut spécial mais c'est un type de boîte. De ce fait, vous pouvez créer une barre de menu verticale en affectant la valeur `vertical` à l'attribut `orient`.



L'élément menu est normalement placé sur une barre de menu, bien que

ce ne soit pas nécessaire. Cependant, il donnera une autre apparence. L'image ci-contre montre à quoi l'exemple précédent ressemblerait sans la barre de menu.

L'élément menu fonctionne comme l'élément button. Il accepte certains attributs semblables plus quelques autres :

*id*

L'identifiant unique du bouton de titre du menu.

#### *label*

Le texte qui apparaît sur le menu, comme Fichier ou Edition.

#### *disabled*

Cet attribut booléen détermine si le menu est désactivé. Bien que ce soit permis, il est rarement utile de désactiver un menu entier. Cet attribut peut être mis à *true* (vrai) ou *false* (faux). Bien sûr, cette dernière valeur est celle par défaut.

#### *accesskey*

Il s'agit de la touche que l'utilisateur peut presser pour activer le menu. Cette lettre est habituellement soulignée dans le titre du menu. Mozilla va regarder dans l'attribut *label* et ajouter un caractère *soulignement* au caractère spécifié ici. Pour cette raison, vous devez spécifier un caractère qui existe dans le texte (bien que la touche fonctionnera toujours si ce n'est pas le cas).

L'élément menupopup crée la fenêtre déroulante contenant les commandes de menu. C'est un type de boîte qui est orienté verticalement par défaut. Vous pouvez passer en orientation horizontale si vous le voulez, les menuitems seront alors placés sur une même ligne. Normalement seuls les menuitems et les menuseparators sont placés dans un menupopup. Vous pouvez placer n'importe quel élément dans un menupopup, cependant ils seront ignorés sur un Macintosh.

L'élément menuitem est comme l'élément menu et comporte quelques attributs semblables.

#### *id*

L'identifiant unique du bouton du menu.

#### *label*

Le texte qui apparaît sur l'item de menu, comme Ouvrir ou Enregistrer.

#### *disabled*

Cet attribut booléen détermine si l'item de menu est désactivé. Cet attribut peut être mis à *true* (vrai) ou *false* (faux), qui est la valeur par défaut.

#### *accesskey*

Il s'agit de la touche que l'utilisateur peut presser pour activer l'item de menu. Cette lettre est habituellement soulignée dans le libellé de l'item de menu. Mozilla va regarder dans l'attribut *label* et ajouter un caractère *soulignement* au caractère spécifié ici. Pour cette raison, vous devez spécifier un caractère qui existe dans le texte.

#### *accelettext*

Cet attribut spécifie le texte de la touche de raccourci qui apparaît près du texte de la commande de menu. Cependant, il n'associe pas de touche d'action avec l'item de menu. Nous verrons comment faire cela plus tard.

Le menuseparator n'a pas d'attribut spécial. Il crée juste une barre horizontale entre les items de menu.

---

Nous allons maintenant apprendre quelques autres fonctions de menu.

## 5.3 Plus de fonctionnalités de menu

Écrit par Neil Deakin. Traduit par **Vincent S.** (27/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/advmenu.html>

Dans cette section, nous allons voir comment créer des sous-menus et des coches de menus.

### Créer des sous-menus

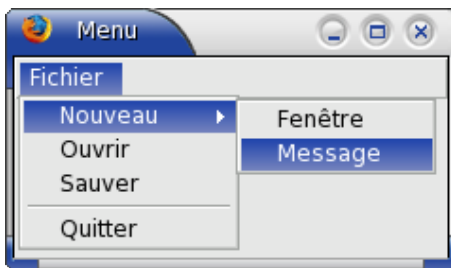
Vous pouvez créer des sous-menus à l'intérieur d'autres menus (menus imbriqués) en utilisant des éléments existants. Souvenez vous que vous pouvez mettre n'importe quel élément dans un menupopup. Nous avons vu comment placer des menuitem et des menuseparator dans des menupopup. Toutefois, vous pouvez créer des sous-menus en plaçant simplement l'élément menu à l'intérieur de l'élément menupopup.

Ce fonctionnement s'explique car l'élément menu est valide même quand il n'est pas placé directement dans une barre de menu.

L'exemple ci-dessous crée un simple sous-menu dans le menu Fichier :

Exemple 5.3.1 :

```
<toolbox flex="1">
  <menubar id="sample-menubar">
    <menu id="file-menu" label="Fichier">
      <menupopup id="file-popup">
        <menu id="new-menu" label="Nouveau">
          <menupopup id="new-popup">
            <menuitem label="Fenêtre"/>
            <menuitem label="Message"/>
          </menupopup>
        </menu>
        <menuitem label="Ouvrir"/>
        <menuitem label="Sauver"/>
        <menuseparator/>
        <menuitem label="Quitter"/>
      </menupopup>
    </menu>
  </menubar>
</toolbox>
```



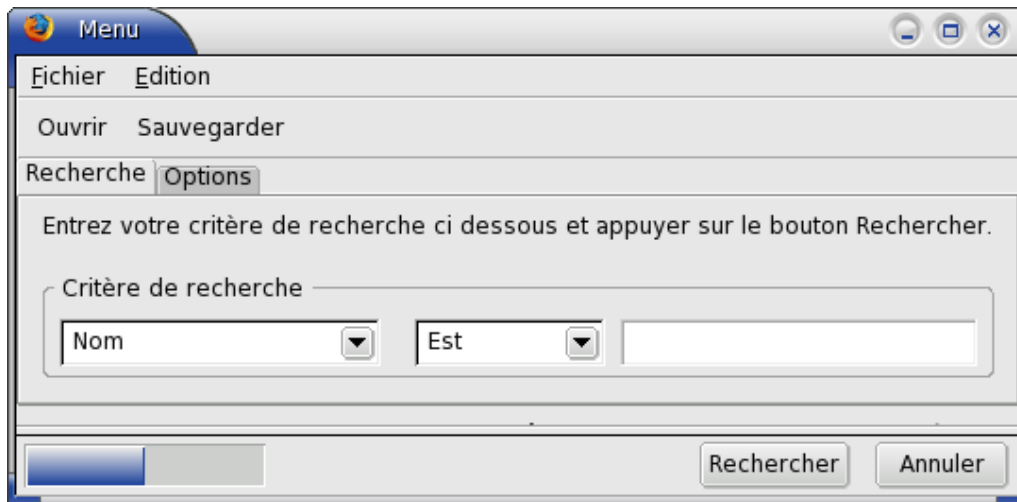
## Ajouter un menu Recherche de Fichiers

Ajoutons un menu à la boîte de dialogue de recherche de fichiers. Nous allons juste ajouter quelques commandes simples à un menu *Fichier* et à un menu *Edition*. Ces menus sont similaires à l'exemple ci-dessus.

```
<toolbox>

<menubar id="findfiles-menubar">
  <menu id="file-menu" label="Fichier" accesskey="f">
    <menupopup id="file-popup">
      <menuitem label="Ouvrir une Recherche..." accesskey="o"/>
      <menuitem label="Enregistrer une Recherche..." accesskey="s"/>
      <menuseparator/>
      <menuitem label="Fermer" accesskey="f"/>
    </menupopup>
  </menu>
  <menu id="edit-menu" label="Edition" accesskey="e">
    <menupopup id="edit-popup">
      <menuitem label="Couper" accesskey="c"/>
      <menuitem label="Copier" accesskey="p"/>
      <menuitem label="Coller" accesskey="l" disabled="true"/>
    </menupopup>
  </menu>
</menubar>
```

```
<toolbar id="findfiles-toolbar">
```



Ici nous avons ajouté deux menus contenant des commandes variées. Notez que la barre de menu a été ajoutée dans un toolbar. Sur l'image, vous pouvez voir la poignée (grippy) sur la barre de menu qui peut être utilisée pour cacher la barre de menu. Les trois points après *Ouvrir une Recherche* et *Enregistrer une Recherche* sont le moyen habituel pour indiquer à l'utilisateur qu'une boîte de dialogue va s'ouvrir quand il sélectionne cette commande. Des touches de raccourcis ont été ajoutées pour chaque menu et chaque item de menu. Vous verrez dans l'image que cette lettre a été soulignée dans le texte du menu. Vous verrez aussi que la commande *Coller* a été désactivée. Nous supposons qu'il n'y a rien à coller.

## Ajouter des coches aux menus

De nombreuses applications ont des items de menu ayant des coches. Par exemple, une fonctionnalité qui est active a une coche placée à côté de la commande et une fonctionnalité qui est désactivée n'a pas de coche. Quand l'utilisateur sélectionne le menu, l'état de la coche est inversé. Vous pouvez aussi créer des boutons radio sur les items de menu.

Les coches sont créées de manière similaire aux éléments checkbox et radio. Elles impliquent l'utilisation de deux attributs, `type` pour indiquer le type de coche et `name` pour grouper les commandes ensemble. L'exemple ci-dessous crée un menu avec un item coché.

Exemple 5.3.2 :

```
<toolbar>
  <menubar id="options-menubar">
    <menu id="options_menu" label="Options">
      <menupopup>
        <menuitem id="backups" label="Faire des sauvegardes" type="checkbox"/>
      </menupopup>
    </menu>
  </menubar>
</toolbar>
```

L'attribut `type` ajouté est utilisé pour rendre l'item de menu cochable. En mettant sa valeur à *checkbox*, l'item de menu peut être coché et décoché en le sélectionnant.

En plus des coches standard, vous pouvez créer des coches de style radio en mettant `type` à la valeur *radio*. Une coche radio est utilisée quand vous voulez un groupe d'items de menu où seul l'un d'entre eux peut être coché à la fois. Un exemple peut être un menu de police de caractères où une seule police peut être sélectionnée à la fois. Quand un autre item est sélectionné, l'item choisi précédemment est décoché.

Pour grouper plusieurs items de menu ensemble, vous devez placer un attribut name sur chacun d'eux. Mettez la même chaîne de caractère pour valeur. L'exemple ci-dessous vous en fait la démonstration :

Exemple 5.3.3 :

```
<toolbox>
  <menubar id="planets-menubar">
    <menu id="planet-menu" label="Planète">
      <menupopup>
        <menuitem id="jupiter" label="Jupiter" type="radio" name="ringed"/>
        <menuitem id="saturn" label="Saturne" type="radio" name="ringed"/>
        <menuitem id="uranus" label="Uranus" type="radio" name="ringed"/>
        <menuitem id="earth" label="Terre" type="radio" name="inhabited"/>
      </menupopup>
    </menu>
  </menubar>
</toolbox>
```

Si vous essayez cet exemple, vous verrez que sur les trois premiers items, un seul peut être coché à la fois. Ils sont groupés ensemble car ils ont le même nom. Le dernier item de menu, *Terre*, ne fait pas partie du groupe car il a un nom différent. Pourtant, c'est un bouton radio.

Bien sûr, les items groupés doivent tous être dans le même menu. Ils n'ont pas à être placés les uns à côté des autres, bien que cela n'ait pas autant de sens s'ils ne l'étaient pas.

---

Par la suite, nous allons voir comment créer des menus surgissants.

## 5.4 Menus surgissants

Écrit par Neil Deakin. Traduit par **Vincent S.** (01/04/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/popups.html>

Dans la section précédente, nous avons vu comment créer un menu sur une barre de menu. XUL a aussi la capacité de créer des menus surgissants. Les menus surgissants sont habituellement affichés lorsque l'utilisateur presse le bouton droit de la souris.

### Créer un menu surgissant

XUL a trois différents types de boîtes surgissantes, décrites ci-dessous. La différence majeure est leurs façons d'apparaître.

#### *Boîte surgissante simple*

La boîte surgissante simple est une fenêtre surgissante qui apparaît quand l'utilisateur presse le bouton gauche de la souris sur un élément. Elles sont assez semblables aux menus sur les barres de menu, excepté qu'elles peuvent être placées n'importe où et peuvent contenir n'importe quel contenu. Un bon exemple est le menu déroulant qui apparaît quand vous maintenez le bouton de la souris enfoncé sur les boutons "précédent" et "suivant" dans la fenêtre d'un navigateur.

#### *Boîte contextuelle*

La boîte contextuelle est une fenêtre surgissante qui apparaît quand l'utilisateur presse le bouton de menu contextuel, qui est habituellement le bouton droit de la souris. Sur certaines plates-formes, Il peut s'agir d'un bouton différent – mais c'est toujours le bouton ou une combinaison de touches et de bouton de souris qui invoque un menu spécifique au contexte. Sur le Macintosh par exemple, l'utilisateur doit soit presser la touche Control et le bouton de la souris, soit maintenir le bouton de la souris enfoncé un certain temps.

#### *Bulle d'aide*

Une fenêtre surgissante "bulle d'aide" va apparaître quand l'utilisateur survolera un élément avec la souris. Ce type de boîte surgissante est habituellement utilisé pour fournir la description d'un bouton de façon plus détaillée que le bouton le permet lui-même.

Ces trois types de boîtes surgissantes diffèrent dans la façon dont l'utilisateur les invoque. Elles peuvent contenir n'importe quel contenu, bien que des menus soient courants pour les boîtes simples et contextuelles, et qu'une simple chaîne de caractères soit courante pour une bulle d'aide. Le type de boîte surgissante est déterminé par l'élément qui invoque la boîte.

Une boîte surgissante est décrite en utilisant l'élément `popup`. C'est un type de boîte sans attributs spéciaux. Quand elle est invoquée, une fenêtre contenant tout ce que vous avez pu mettre dans le `popup` va s'afficher. Cependant, vous devez toujours insérer un attribut `id` sur le `popup` car il doit être associé à un élément. Nous verrons bientôt sa signification. D'abord, un exemple :

```
<popupset>
  <popup id="clipmenu">
    <menuitem label="Couper"/>
    <menuitem label="Copier"/>
    <menuitem label="Coller"/>
  </popup>
</popupset>
```

Comme on peut le voir ici, un simple menu surgissant contenant trois commandes a été créé. L'élément `popup` entoure les trois items de menu. C'est semblable à l'élément `menupopup`. Il s'agit d'un type de boîte ayant une orientation verticale par défaut. Vous remarquerez également que l'`id` a été mis sur l'élément `popup` lui-même.

L'élément `popupset` entoure l'entière déclaration de menu surgissant. Il s'agit d'un container générique pour les boîtes surgissantes, et est optionnel. Il ne s'affiche pas à l'écran mais il est utilisé comme une section dans laquelle vous pouvez déclarer tous vos menus surgissants. Comme le nom `popupset` sous-entend, vous pouvez placer plusieurs déclarations de menus surgissants à l'intérieur. Ajoutez en simplement d'autres après le premier élément `popup`. Vous pouvez avoir plus d'un `popupset` dans un fichier, mais habituellement vous n'en aurez qu'un.

Maintenant que nous avons créé la boîte surgissante, il est temps de la faire apparaître. Pour cela, nous avons besoin d'associer la boîte à un élément d'où elle devra apparaître. Nous faisons cela car nous voulons seulement que la boîte apparaisse quand l'utilisateur clique à un certain endroit de la fenêtre. Habituellement, il s'agira d'un bouton spécifique ou d'une boîte.

Pour associer le menu surgissant à un élément, ajoutez un de ces trois attributs à l'élément. L'attribut que vous ajoutez dépend du type de menu surgissant vous voulez créer. Pour les menus surgissants simples, ajoutez l'attribut `popup` à l'élément. Pour les menus contextuels, ajoutez l'attribut `context`. Enfin, pour les bulles d'aide, ajoutez l'attribut `tooltip`.

La valeur de l'attribut doit être celle de l'`id` du `popup` que vous voulez faire apparaître. C'est pour cela que vous devez mettre un `id` sur le `popup`. Par ce moyen, il est facile d'avoir plusieurs menus surgissants dans un fichier.

Dans l'exemple ci-dessus, nous voulons faire un menu surgissant contextuel. Nous devons donc utiliser l'attribut `context` et l'ajouter à l'élément sur lequel nous voulons associer le menu surgissant. L'exemple ci-dessous montre comment procéder :

Exemple 5.4.1 :

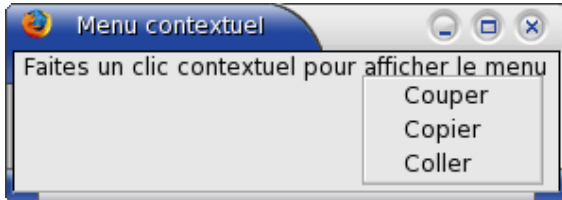
```
<popupset>
  <popup id="clipmenu">
```

```

    <menuitem label="Couper" />
    <menuitem label="Copier" />
    <menuitem label="Coller" />
  </popup>
</popupset>

<box context="clipmenu">
  <description value="Faites un clic contextuel pour afficher le menu" />
</box>

```



Ici, le menu contextuel a été associé à une boîte. À chaque

fois que vous faites un clic contextuel (clic droit) n'importe où dans la boîte, le menu surgissant apparaîtra. Le menu apparaîtra aussi même si vous cliquez sur un enfant de la boîte, donc il apparaîtra aussi si vous cliquez sur l'élément description. L'attribut `context` a été utilisé pour associer la boîte au menu contextuel de même id. Dans ce cas, le menu *clipmenu* va apparaître. De cette façon, vous pouvez disposer de plusieurs menus surgissants et les associer avec différents éléments.

Vous pouvez associer plusieurs menus surgissants avec le même élément en mettant plusieurs d'attributs de différents types sur un élément. Vous pouvez aussi associer le même menu surgissant à plusieurs éléments, ce qui est un avantage de l'utilisation de la syntaxe `popup`. Les menus surgissants ne peuvent être associés qu'avec des éléments XUL. Ils ne peuvent pas être associés à des éléments HTML.

## Bulles d'aide

Nous allons voir un moyen simple de créer des bulles d'aide. Il y a deux façons de créer des bulles d'aide. La méthode la plus simple, qui est la plus commune, est d'ajouter un attribut `tooltiptext` à un élément sur lequel vous voulez assigner une bulle d'aide.

La deuxième méthode consiste à utiliser un élément tooltip contenant le contenu d'une bulle d'aide. Il nécessite que vous ayez un bloc séparé de contenu pour chaque bulle d'aide ou que vous ayez un script contenant le contenu, bien que certains contenus hormis du texte dans une bulle d'aide ne sont pas permis.

Exemple 5.4.2 :

```

<button label="Save" tooltip="Cliquez ici pour enregistrer vos trucs" />

<popupset>
  <tooltip id="moretip" orient="vertical" style="background-color: #33DD00;">
    <description value="Cliquez ici pour voir plus d'information" />
    <description value="Vraiment!" style="color: red;" />
  </tooltip>
</popupset>

<button label="Plus" tooltip="moretip" />

```

Ces deux boutons ont chacun une bulle d'aide. Le premier utilise le style par défaut de bulle d'aide. Le second utilise une bulle d'aide modifiée qui a une couleur arrière-plan différente et un texte stylisé. La bulle d'aide est associée au bouton *Plus* en utilisant l'attribut `tooltip`, correspondant à l'id de l'élément tooltip.

## Alignement des menus surgissants

Par défaut, les menus surgissants et contextuels vont apparaître là où le pointeur de la souris se trouve. Les bulles d'aides seront placées légèrement sous l'élément pour que le pointeur de la souris ne les cache pas. Il y a des cas toutefois, où vous voudrez indiquer en détail l'emplacement du menu surgissant. Par exemple, le menu surgissant qui apparaît quand vous cliquez sur le bouton Précédent dans un navigateur doit apparaître sous le bouton Précédent, non pas là où se situe le pointeur de la souris.

Pour changer la position du menu, vous pouvez utiliser un attribut additionnel, `position`, sur le `popup`. Vous pouvez aussi l'ajouter à l'élément `menupopup`. Cet attribut est utilisé pour indiquer l'emplacement du menu relativement à l'élément invoquant la boîte. Ses différentes valeurs applicables sont décrites brièvement ci-dessous :

### *after\_start*

Le menu surgissant apparaît sous l'élément avec les bords gauche de l'élément et du menu alignés. Si le menu surgissant est plus large que l'élément, il s'étend à droite. C'est cette valeur qui est utilisée pour les menus déroulants associés avec les boutons Précédent et Suivant du navigateur.

### *after\_end*

Le menu surgissant apparaît sous l'élément avec les bords droit de l'élément et du menu alignés.

### *before\_start*

Le menu surgissant apparaît au-dessus de l'élément avec les bords gauche de l'élément et du menu alignés.

### *before\_end*

Le menu surgissant apparaît au-dessus de l'élément avec les bords droit de l'élément et du menu alignés.

### *end\_after*

Le menu surgissant apparaît à droite de l'élément avec les bords inférieurs de l'élément et du menu alignés.

### *end\_before*

Le menu surgissant apparaît à droite de l'élément avec les bords supérieurs de l'élément et du menu alignés.

### *start\_after*

Le menu surgissant apparaît à gauche de l'élément avec les bords inférieurs de l'élément et du menu alignés.

### *start\_before*

Le menu surgissant apparaît à gauche de l'élément avec les bords supérieurs de l'élément et du menu alignés.

### *overlap*

Le menu surgissant apparaît par dessus l'élément.

### *at\_pointer*

Le menu surgissant apparaît à la position du pointeur de la souris.

### *after\_pointer*

Le menu surgissant apparaît à la même position horizontale que le pointeur de la souris mais apparaît sous l'élément. C'est ainsi que les bulles d'aide apparaissent.

En ajoutant un ou plusieurs de ces attributs à un élément, vous pouvez spécifier précisément où le menu surgissant doit apparaître. Vous ne pouvez pas spécifier une position exacte en pixels. L'attribut `position` peut être utilisé pour les trois types de menus surgissants, bien que vous ne changerez probablement pas la valeur pour les bulles d'aide.

L'exemple ci-dessous montre la création d'un bouton Précédent avec un menu surgissant :

Exemple 5.4.3 :

```
<popupset>
```



```

<popup id="backpopup" position="after_start">
  <menuitem label="Page 1"/>
  <menuitem label="Page 2"/>
</popup>
</popupset>

<button label="Affiche moi" popup="backpopup"/>

```

## Exemple de menu surgissant

Ajoutons un simple menu contextuel à la boîte de dialogue de recherche de fichiers. Pour plus de simplicité, nous allons juste recopier le contenu du menu *Edition*. Le menu apparaîtra quand l'on clique sur le premier onglet :

```

<popupset>
  <popup id="editpopup">
    <menuitem label="Couper" accesskey="c"/>
    <menuitem label="Copier" accesskey="p"/>
    <menuitem label="Coller" accesskey="l" disabled="true"/>
  </popup>
</popupset>

<vbox flex="1">
.
.
.

<tabpanel id="searchpanel" orient="vertical" context="editpopup">

```

Ici un simple menu surgissant, similaire au menu Edition, a été ajouté au premier onglet. Si vous faites un clic droit (Control–clic sur Macintosh) n'importe où sur la page de ce premier onglet, le menu surgissant va apparaître. Cependant, le menu n'apparaîtra pas si vous cliquez autre part.

Notez que le champs de saisie a son propre menu surgissant qui supplantera celui que nous avons spécifié.

---

Par la suite, nous allons voir comment créer des menus défilants.

## 5.5 Menus défilants

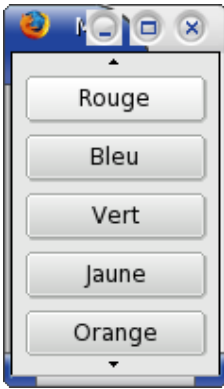
Écrit par Neil Deakin. Traduit par *Vincent S.* (13/04/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/menuscroll.html>

Cette section va décrire les menus défilants et comment utiliser le mécanisme avec d'autres éléments.

### Créer un grand menu

Vous vous demandez peut-être ce qu'il se passerait si vous créez un menu avec beaucoup de commandes, de telle manière que tous les items ne peuvent pas s'afficher tous à l'écran en même temps. Mozilla fournit un mécanisme de défilement permettant de faire défiler les items.



Si l'espace disponible est trop petit, des flèches vont apparaître sur chaque extrémité du

menu. Si vous bougez la souris sur les flèches, le menu va défiler vers le haut et vers le bas. Si l'espace disponible est assez grand, les flèches n'apparaîtront pas. Notez que le comportement exact du défilement dépendra du thème graphique utilisé.

Ce comportement est automatique. Vous n'avez pas à faire quoi que se soit pour avoir des menus défilants. Il va s'appliquer aux menus des barres de menu, aux menus surgissants ou listes déroulantes. Il est implémenté en utilisant un élément [arrowscrollbox](#). Cet élément peut être utilisé pour créer une boîte de défilement avec des flèches.

L'élément [arrowscrollbox](#) peut être utilisé n'importe où une boîte normale peut être utilisée. Vous n'êtes pas obligé de l'utiliser pour des menus. Il s'agit toujours une boîte verticale pouvant contenir n'importe quel élément à l'intérieur. Vous pouvez l'utiliser pour implémenter une liste que vous ne voulez pas déroulante.

L'exemple suivant montre comment créer une liste défilante de boutons (vous devrez redimensionner la fenêtre pour voir les boutons de flèches) :

Exemple 5.5.1 :

```
<arrowscrollbox orient="vertical" flex="1">
  <button label="Rouge"/>
  <button label="Bleu"/>
  <button label="Vert"/>
  <button label="Jaune"/>
  <button label="Orange"/>
  <button label="Argent"/>
  <button label="Lavande"/>
  <button label="Or"/>
  <button label="Turquoise"/>
  <button label="Pêche"/>
  <button label="Bordeaux"/>
  <button label="Noir"/>
</arrowscrollbox>
```

Si vous essayez cet exemple, il va d'abord s'ouvrir en pleine taille. Cependant, si vous réduisez la taille de la fenêtre, les flèches de défilement vont apparaître. Rendre la fenêtre plus grande à nouveau va faire disparaître les flèches.

Vous pouvez mettre une propriété CSS `max-height` sur les [arrowscrollbox](#) pour limiter la taille de la boîte de défilement et ainsi faire apparaître les flèches tout le temps.

L'élément [arrowscrollbox](#) est principalement utile dans les menus et boîtes surgissantes.

---

Par la suite, nous allons voir comment ajouter des gestionnaires d'évènements à des éléments XUL.

## 6. Évènements et scripts

### 6.1 Ajout de gestionnaires d'évènements

Écrit par Neil Deakin. Traduit par *Durandal* (18/07/2004), mise à jour par Julien Appert (15/06/2005) .  
Page originale : <http://www.xulplanet.com/tutorials/xultu/events.html>

La boîte de dialogue de recherche de fichiers est tout à fait correcte à ce stade. Nous ne l'avons pas beaucoup optimisé mais nous avons créé une interface utilisateur facilement. Dans ce qui suit, nous allons montrer comment lui ajouter des scripts.

#### Utiliser des scripts

Pour rendre la boîte de dialogue de recherche de fichiers fonctionnelle, nous avons besoin d'ajouter des scripts qui vont s'exécuter quand l'utilisateur interagira avec le dialogue. Nous voudrions ajouter un script pour gérer le bouton Rechercher, le bouton Annuler et pour gérer chaque commande du menu. Nous écrirons ces scripts en utilisant des fonctions Javascript de la même manière qu'en HTML.

Vous pouvez utiliser l'élément script pour inclure des scripts dans des fichiers XUL. Vous pouvez aussi inclure le script directement dans le fichier XUL entre les balises script ouvrante et fermante, mais il est préférable de placer le code dans un fichier à part. Votre fenêtre XUL se chargera un peu plus rapidement. Nous utiliserons alors l'attribut `src` pour lier un fichier de script externe.

Ajoutons un script au dialogue de recherche de fichiers. Bien que le nom du fichier de script n'ait pas d'importance, on lui donne habituellement le même nom que celui du fichier XUL, avec l'extension js. Dans ce cas, on utilisera `findfile.js`. Ajoutez la ligne ci-dessous juste après la balise window ouvrante et avant tout autre élément.

```
<script src="findfile.js"/>
```

Nous créerons le fichier de script plus tard, quand nous saurons quoi y mettre. Nous y définirons des fonctions que nous pourrions appeler dans des gestionnaires d'évènements.

Nous pouvons inclure de multiples scripts dans un fichier XUL en utilisant de multiples balises script, chacune pointant vers un script différent. Nous pouvons utiliser les URLs relative ou absolue. Par exemple, nous pouvons utiliser des URLs sous les formes suivantes :

```
<script src="findfile.js"/>  
<script src="chrome://findfiles/content/help.js"/>  
<script src="http://www.example.com/js/items.js"/>
```

Ce tutoriel n'a pas pour but de décrire comment écrire du JavaScript. C'est un sujet assez important et il existe déjà beaucoup de documentations traitant du sujet.

#### Répondre aux Évènements

Le script contiendra le code qui répondra aux différents événements déclenchés par l'utilisateur ou par d'autres situations. Il existe environ une trentaine d'évènements pouvant être gérés de différentes manières. Un événement typique est le bouton de souris ou la touche pressée par l'utilisateur. Chaque élément XUL a la capacité de déclencher certains événements dans des situations différentes. Quelques déclenchements d'évènements sont spécifiques à certains éléments.

Chaque événement a un nom, par exemple, *mousemove* est le nom de l'événement qui est déclenché quand l'utilisateur passe la souris par dessus un élément d'interface utilisateur. XUL utilise le mécanisme

d'événement défini par le DOM Events. Quand survient une action devant déclencher un événement, telle que le déplacement de la souris par l'utilisateur, un objet 'event' correspondant à ce type d'événement est créé. Des propriétés sont assignées à cet objet event telles que la position de la souris, la touche pressée, etc.

L'événement est ensuite envoyé au XUL par phases. La première est la phase de capture, durant laquelle l'événement est d'abord envoyé à la fenêtre, puis au document, suivi par chaque ancêtre de l'élément XUL visé, jusqu'à ce qu'il l'atteigne. Puis l'événement est envoyé à cet élément XUL. Enfin, pendant la phase de diffusion, l'événement est envoyé à chaque élément dans l'autre sens jusqu'à ce qu'il atteigne à nouveau la fenêtre. Vous pouvez répondre à un événement pendant l'une ou l'autre de ces deux phases. Une fois que l'événement a terminé de se propager, l'action correspondant au comportement natif de l'élément est déclenchée.

Par exemple, quand la souris passe sur un bouton inclus dans une boîte, un événement *mousemove* est généré et envoyé d'abord à la fenêtre, puis au document et ensuite à la boîte. Cela complète la phase de capture. Ensuite, l'événement *mousemove* est envoyé au bouton. Enfin la phase de diffusion se traduit par le renvoi de l'événement vers la boîte, au document et à la fenêtre. La phase de diffusion est essentiellement l'inverse de la phase de capture. Notez que certains événements ne passent pas par la phase de diffusion.

Vous pouvez attacher des scrutateurs à chaque élément pour être à l'écoute des événements pendant chaque étape de leur propagation. Étant donné qu'un simple événement est passé à tous les ancêtres, vous pouvez attacher un scrutateur à un élément spécifique ou à un élément supérieur dans la hiérarchie. Naturellement, un événement attaché à un élément parent recevra une notification de tous les éléments qu'il contient, alors qu'un événement attaché à un bouton recevra seulement des événements concernant ce bouton. C'est utile si vous désirez gérer plusieurs éléments avec un code identique ou similaire.

Une fois que vous gérez un événement, sans vous soucier du stade de sa propagation, vous voudrez probablement le stopper avant qu'il ne soit envoyé à d'autres éléments, essentiellement en empêchant les phases de capture et de diffusion de continuer. En fonction de la manière avec laquelle vous avez attaché le scrutateur d'événement à un élément, il y a différentes manières d'y arriver.

L'événement le plus utilisé est l'événement *command*. L'événement *command* est déclenché quand l'utilisateur active un élément, par exemple en pressant un bouton, en cochant une case ou en sélectionnant une entrée d'un menu. L'événement *command* est pratique car il gère automatiquement les différentes méthodes d'activation d'un élément. Par exemple, l'événement *command* se produira sans se soucier si l'utilisateur a utilisé sa souris pour cliquer le bouton ou s'il a pressé la touche Entrée.

Il y a deux manières d'attacher un scrutateur d'événement à un élément. Premièrement, en utilisant un attribut avec un script comme valeur. Deuxièmement, en appelant la méthode `addEventListener` d'un élément. La première manière peut seulement gérer les événements en phase de diffusion mais est plus simple à écrire. La seconde peut gérer les événements à chaque phase et peut aussi être utilisée pour attacher plusieurs scrutateurs pour un événement à un élément. L'utilisation de la forme par attribut est une manière plus commune pour la plupart des événements.

Pour utiliser la forme par attribut, placez un attribut sur un élément à l'endroit que vous voulez, son nom devra être le nom de l'événement précédé par le mot "on". Par exemple, l'attribut correspondant pour l'événement *command* est `oncommand`. La valeur de l'attribut devra être du script qui sera exécuté quand l'événement se produira. Typiquement, le code sera court et appellera une fonction définie dans un script séparé. Un exemple de réponse lorsqu'un bouton est pressé :

Exemple 6.1.1 :

```
<button label="OK" oncommand="alert('Le bouton a été pressé !');"/>
```

Puisque l'événement *command* va se diffuser, il est également possible de placer le scrutateur d'événement sur un élément conteneur. Dans l'exemple ci-dessous, le scrutateur a été placé sur une boîte et il recevra les

événements pour tous ses éléments.

#### Exemple 6.1.2 :

```
<vbox oncommand="alert(event.target.tagName);">
  <button label="OK"/>
  <checkbox label="Voir les images"/>
</vbox>
```

Dans cet exemple, l'événement *command* va se diffuser du bouton ou de la case à cocher jusqu'à la boîte, où il sera géré. Si un second scrutateur (l'attribut *oncommand*) est placé sur le bouton, son code sera appelé en premier, suivi par le gestionnaire de la boîte. L'objet *event* est transmis aux gestionnaires d'événement comme un argument implicite nommé *event*. Il est utilisé pour obtenir des informations spécifiques à propos de l'événement. Une des plus utilisées est la propriété *target* (cible) de l'événement, qui cible l'élément à partir duquel l'événement s'est produit. Dans l'exemple, nous affichons une alerte contenant le nom de l'élément cible. La cible est très utile lors de l'utilisation d'un événement en diffusion dans le cas où vous auriez une série de boutons gérée par un même script.

Vous pouvez noter que la syntaxe de l'attribut est identique à celle utilisée pour les événements dans les documents HTML. En fait, HTML et XUL partagent le même mécanisme d'événement. Une différence importante est que l'événement *click* (ou l'attribut *onclick*) est utilisé en HTML pour répondre aux boutons, alors qu'on devra utiliser l'événement *command* en XUL. XUL possède un événement *click*, mais il répond uniquement aux clics de souris, pas à l'utilisation du clavier. Ainsi, l'événement *click* devra être évité en XUL, sauf si vous avez une raison pour avoir un élément uniquement être géré avec la souris.

Un gestionnaire *oncommand* peut être placé sur les boutons Rechercher et Annuler dans le dialogue de recherche de fichiers. Cliquer sur le bouton Rechercher devrait démarrer la recherche. Comme nous n'implémenterons pas cette partie, nous l'omettrons. Par contre, cliquer sur le bouton Annuler devrait fermer la fenêtre. Le code ci-dessous montre comment faire cela. Tant que nous y sommes, ajoutons le même code à l'item de menu Fermer.

```
<menuitem label="Fermer" accesskey="c" oncommand="window.close();" />
...
<button id="cancel-button" label="Annuler"
  oncommand="window.close();" />
```

Deux gestionnaires ont été ajoutés ici. Le gestionnaire *oncommand* a été ajouté à l'item de menu Fermer. En utilisant ce gestionnaire, l'utilisateur pourra fermer la fenêtre en cliquant sur l'item de menu ou en le sélectionnant avec le clavier. Le gestionnaire *oncommand* a été ajouté au bouton Annuler.

## Scruteurs d'Événement DOM

La seconde façon d'ajouter un gestionnaire d'événement est d'appeler la méthode *addEventListener* d'un élément. Celle-ci vous permet d'attacher un scrutateur d'événement dynamiquement et de scruter les événements durant la phase de capture. La syntaxe est la suivante :

#### Exemple 6.1.3 :

```
<button id="okbutton" label="OK"/>

<script>
function buttonPressed(event)
{
  alert('Le bouton a été pressé !');
}
```

```
var button = document.getElementById("okbutton");
button.addEventListener('command', buttonPressed, true);
</script>
```

La fonction `getElementById` retourne l'élément ayant un identifiant `id`, dans notre cas, le bouton. La fonction `addEventListener` est appelée pour ajouter un nouveau scrutateur d'événement en phase de capture. Le premier argument est le nom de l'événement à scruter. Le second argument est la fonction du scrutateur d'événement qui sera appelée quand l'événement se produira. Enfin, le dernier argument devra être *true* pour les scrutateurs en phase de capture. Vous pouvez également scruter durant la phase de diffusion en donnant *false* comme dernier argument. La fonction scrutateur d'événement passée comme second argument devra avoir un argument, l'objet 'event', comme vous le voyez dans la déclaration de la fonction `buttonPressed` ci-dessus.

---

Dans la prochaine section, nous allons voir plus de détails à propos de l'objet `event`.

Le dialogue de recherche de fichiers à ce stade :

## 6.2 Plus sur les gestionnaires d'évènements

Écrit par Neil Deakin. Traduit par *Alain B.* (17/04/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/advevents.html>

Dans cette section, l'objet événement sera examiné et des événements additionnels seront décrits.

### L'objet événement

Chaque gestionnaire d'évènement dispose d'un unique argument qui contient un objet événement. Pour le fonctionnement des scrutateurs d'évènements, cet événement est un argument implicite auquel un script peut se référer en utilisant le nom *event*. Sous la forme `addEventListener`, le premier argument de la fonction scrutatrice sera un objet événement. L'objet *event* dispose d'un certain nombre de propriétés qui peuvent être examinées lors d'un événement. La liste complète est disponible dans les références objets.

Nous avons déjà vu la propriété *target* d'un événement dans la section précédente. Elle contient une référence de l'élément ayant déclenché l'évènement. Une propriété similaire *currentTarget* contient l'élément sur lequel est placé un scrutateur d'évènement. Dans l'exemple ci-dessous, *currentTarget* pointe toujours la boîte vbox, alors que la cible peut être un élément spécifique, soit le bouton ou soit la case à cocher, qui a été activé.

Exemple 6.2.1 :

```
< vbox oncommand="alert(event.currentTarget.tagName); ">
  < button label="OK" />
  < checkbox label="Voir les images" />
< / vbox>
```

Rappelez vous que la phase de capture intervient avant la phase de diffusion, donc tous les scrutateurs de capture sont déclenchés avant les scrutateurs de diffusion. Si un événement capturé stoppe la propagation événementielle, aucun des scrutateurs de capture suivants, ou de ceux de diffusion ne recevront jamais de notification d'un quelconque événement. Pour interrompre la propagation événementielle, appelez la méthode `stopPropagation` de l'objet événement, comme dans l'exemple suivant.

Exemple 6.2.2 :

```
< hbox id="outerbox">
  < button id="okbutton" label="OK" />
< / hbox>
```

```

<script>
function buttonPressed(event)
{
    alert('Le bouton a été pressé !');
}

function boxPressed(event)
{
    alert('La boîte a été pressée !');
    event.stopPropagation();
}

var button = document.getElementById("okbutton");
button.addEventListener('command',buttonPressed,true);

var outerbox = document.getElementById("outerbox");
outerbox.addEventListener('command',boxPressed,true);
</script>

```

Ici, un scrutateur d'évènement a été ajouté au bouton, et un autre à la boîte. La méthode `stopPropagation` est appelée dans le scrutateur de la boîte, donc le scrutateur du bouton ne sera jamais appelé. Si cet appel est enlevé, les deux scrutateurs seront appelés et les deux alertes apparaîtront.

Si aucun gestionnaire d'évènement n'a été enregistré pour un évènement, alors après avoir accompli les phases de capture et de diffusion, l'élément traitera l'évènement dans un mode par défaut. L'action dépendra de la nature de l'évènement et du type d'élément. Par exemple, l'évènement *popupshowing* est envoyé par un menu surgissant juste avant son affichage. L'action par défaut est l'affichage du menu surgissant. Si l'action par défaut est bloquée, l'affichage ne se fera pas. L'action par défaut peut être empêchée avec la méthode `preventDefault` de l'objet évènement, comme dans l'exemple ci-dessous.

Exemple 6.2.3 :

```

<button label="Types" type="menu">
  <menupopup onpopupshowing="event.preventDefault();">
    <menuitem label="Verre"/>
    <menuitem label="Plastique"/>
  </menupopup>
</button>

```

Alternativement, pour des scrutateurs d'évènement attribués, vous pouvez simplement faire renvoyer la valeur *false* par le code. Notez que l'empêchement de l'action par défaut n'est pas la même chose que d'interrompre la propagation événementielle avec la méthode `stopPropagation`. Même si l'action par défaut a été bloquée, l'évènement continue à se propager. De même, l'appel de la méthode `stopPropagation` ne bloquera pas l'action par défaut. Vous devrez appeler ces deux méthodes pour interrompre les deux actions.

Notez qu'une fois la propagation ou l'action par défaut bloquée, il n'est pas possible de les rendre actives de nouveau pour cet évènement.

Les sous-sections ci-dessous listent quelques uns des évènements pouvant être utilisés. Une liste complète est fournie dans [la référence des évènements](#).

## Évènements de la souris

Il y a plusieurs évènements pouvant être employés pour gérer les actions de la souris, listés dans le tableau suivant :

*click*

appelé lorsque la souris est appuyée et relâchée sur un élément.

*dblclick*

appelé lorsque la souris est double-cliquée.

*mousedown*

appelé lorsqu'un bouton de la souris est appuyé. Le gestionnaire d'évènement est appelé aussitôt que le bouton de la souris est appuyé, même s'il n'a pas été relâché.

*mouseup*

appelé lorsque la souris est relâchée sur un élément.

*mouseover*

appelé lorsque le pointeur de la souris survole un élément. Vous pourriez utiliser cet évènement pour mettre en valeur l'élément, toutefois CSS fournit une façon automatique de le faire, il est donc inutile de le faire avec un évènement. Vous pouvez toutefois l'utiliser pour afficher une aide dans la barre d'état.

*mousemove*

appelé lorsque le pointeur de la souris bouge au dessus d'un élément. L'évènement étant appelé à chaque mouvement de la souris par l'utilisateur, vous devriez éviter les tâches trop longues avec ce gestionnaire.

*mouseout*

appelé lorsque le pointeur de la souris quitte un élément. Vous pourriez annuler la mise en valeur de l'élément ou effacer le texte de la barre d'état.

Il existe également un jeu d'évènements relatifs au déplacement, qui intervient lorsque l'utilisateur maintient le bouton de la souris enfoncé et déplace la souris. Ces évènements seront décrits dans la section Glisser-Déposer.

Lorsqu'un évènement sur un bouton de la souris se produit, un nombre de propriétés additionnelles est disponible pour déterminer quels boutons ont été pressés et la position du pointeur de la souris. La propriété de l'évènement sur le bouton peut être utilisée pour déterminer quel bouton a été pressé, avec les valeurs possibles de *0* pour le bouton de gauche, *1* pour le bouton de droite, et *2* pour le bouton du milieu. Si vous avez configuré votre souris différemment, ces valeurs peuvent être différentes.

La propriété détaillée contient le nombre de fois que le bouton a été cliqué rapidement en séquence. Elle permet de tester des clics simples, doubles ou triples. Bien entendu, si vous ne souhaitez tester que les double-clics, vous pouvez également utiliser l'évènement *dblclick* à la place. L'évènement *click* sera lancé dès le premier clic, puis de nouveau pour le second clic, puis pour le troisième clic, mais l'évènement *dblclick* ne sera lancé que pour un double-clic.

Le bouton et les propriétés détaillées ne s'appliquent qu'aux évènements se rapportant aux boutons de la souris, et non aux mouvements de la souris. Pour l'évènement *mousemove*, par exemple, l'ensemble de ces propriétés aura une valeur de *0*.

Toutefois, tous les évènements de la souris disposent des propriétés contenant les coordonnées de la position de la souris lors du déclenchement de l'évènement. Il y a deux jeux de coordonnées. Le premier jeu est les propriétés *screenX* et *screenY* mesurées relativement par rapport au coin supérieur gauche de l'écran. Le second jeu, *clientX* et *clientY*, est calculé à partir du coin supérieur gauche du document. Voici un exemple qui affiche les coordonnées actuelle de la souris :

Exemple 6.2.4 :

```
<script>

function updateMouseCoordinates(event)
{
    var text = "X:" + event.clientX + " Y:" + event.clientY;
    document.getElementById("xy").value = text;
}
```



```

}
</script>

<label id="xy"/>
<hbox width="400" height="400" onmousemove="updateMouseCoordinates(event) ;"/>

```

Dans cet exemple, la taille de la boîte a été explicitement fixée pour que l'effet soit plus facile à voir. Le gestionnaire d'évènement récupère les propriétés `clientX` et `clientY` et crée une chaîne avec eux. Cette chaîne est affectée à la propriété `value` d'un libellé.

Notez que l'argument *event* doit être passé à la fonction `updateMouseCoordinate`. Si vous déplacez rapidement la souris autour de la bordure, vous noterez que les coordonnées ne s'arrêtent généralement pas exactement sur `400`. Ceci s'explique car l'évènement *mousemove* se déclenche selon un intervalle dépendant de la vitesse de déplacement de la souris, et celle-ci s'est généralement déplacée au delà de la bordure avant le lancement de l'évènement suivant. Évidemment, il ne serait pas efficace d'envoyer un évènement *mousemove* sur chacun des pixels parcourus par la souris.

Souvent, vous ne voulez obtenir les coordonnées relatives qu'à l'élément qui a déclenché l'évènement plutôt qu'à la fenêtre entière. Il vous suffit de soustraire la position de l'élément à la position de l'évènement, comme pour le code suivant.

```

var element = event.target;
var elementX = event.clientX - element.boxObject.x;
var elementY = event.clientY - element.boxObject.y;

```

Les éléments XUL ont un objet de boîte qui peut être accédé en utilisant la propriété `boxObject`. Nous en apprendrons plus sur l'objet de boîte dans une section ultérieure, mais sachez qu'il contient des informations sur comment est affiché l'élément, en incluant la position `x` et `y` de l'élément. Dans cet exemple de code, ces coordonnées sont soustraites de celles de l'évènement pour obtenir la position relative de l'élément.

## Évènements de chargement

L'évènement de chargement est envoyé au document (l'élément window) dès que le fichier XUL a fini son chargement et juste avant que son contenu ne soit affiché. Cet évènement est couramment utilisé pour initialiser les champs et réaliser d'autres tâches qui doivent être faites avant que l'utilisateur ne dispose de la fenêtre. Vous devriez utiliser un évènement de chargement pour faire ce genre de chose par opposition à l'ajout d'un script de niveau supérieur extérieur à une fonction. Cette préconisation vaut car les éléments XUL peuvent ne pas être chargés et être totalement initialisés, en engendrant des dysfonctionnements. Pour utiliser un évènement de chargement, placez l'attribut `onload` sur l'élément window. Appelez du code à l'intérieur du gestionnaire de chargement afin d'initialiser l'interface si nécessaire.

Il existe également un évènement `unload` qui est appelé dès que la fenêtre est fermée, ou dans un contexte de navigation, lorsque la page bascule vers une autre URL. Vous pouvez utiliser cet évènement pour sauvegarder des informations modifiées, par exemple.

Nous verrons ensuite comment ajouter des raccourcis clavier.

## 6.3 Raccourcis clavier

Écrit par Neil Deakin. Traduit par *Chaddaï Fouché* (19/07/2004), mise à jour par Cyril Trumpler (18/04/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/keyshort.html>

Pour réagir aux touches pressées, vous pourriez utiliser des gestionnaires d'évènements clavier, mais il serait fastidieux de le faire pour chaque bouton ou chaque item de menu.

## Créer un raccourci clavier

XUL fournit des méthodes par lesquelles vous pouvez définir des raccourcis clavier. Nous avons déjà vu dans la section sur les menus que nous pouvions définir un attribut appelé `accesskey` qui spécifie la touche à presser par l'utilisateur pour activer le menu ou l'item de menu. Dans l'exemple ci-dessous, le menu Fichier peut être sélectionné en pressant **Alt** et **F** (ou une autre combinaison de touches spécifique à une plate-forme). Une fois le menu Fichier ouvert, l'item de menu Fermer peut être sélectionné en pressant **F**.

Exemple 6.3.1 :

```
<menubar id="sample-menubar">
  <menu id="file-menu" label="Fichier" accesskey="f">
    <menupopup id="file-popup">
      <menuitem id="close-command" label="Fermer" accesskey="f"/>
    </menupopup>
  </menu>
</menubar>
```

Vous pouvez aussi utiliser l'attribut `accesskey` sur les boutons. Dans ce cas, quand la touche est pressée, le bouton est sélectionné.

Cependant, vous pourriez vouloir mettre en place des raccourcis clavier plus généraux, comme par exemple, la combinaison **Ctrl+C** pour copier du texte dans le presse-papiers. Bien que de tels raccourcis puissent ne pas être toujours valides, ils fonctionneront habituellement dès qu'une fenêtre est ouverte. Normalement, un raccourci sera autorisé à n'importe quel moment et vous pourrez vérifier via un script s'il doit faire quelque chose. Par exemple, copier du texte dans le presse-papiers ne devrait fonctionner seulement quand du texte est sélectionné.

XUL fournit un élément, `key`, qui vous permet de définir un raccourci clavier pour une fenêtre. Il comprend des attributs pour définir la touche qui doit être pressée et quels modificateurs de touches (tels que shift pour **Maj** ou control pour **Ctrl**) doivent l'accompagner. Un exemple :

```
<keyset>
  <key id="sample-key" modifiers="shift" key="R"/>
</keyset>
```

Cet exemple définit un raccourci clavier qui s'active lorsque l'utilisateur presse les touches **Maj** et **R**. L'attribut `key` (notez qu'il a le même nom que l'élément lui-même) peut être utilisé pour indiquer quelle touche doit être pressée, dans ce cas **R**. Vous pouvez ajouter n'importe quels caractères à cet attribut selon les combinaisons de touches devant être pressées. Les modificateurs de touches devant être pressés sont indiqués par l'attribut `modifiers`. Il s'agit d'une liste de modificateurs séparée par des espaces, et ils sont décrits ci-dessous :

*alt*

L'utilisateur doit presser la touche **Alt**

*control*

L'utilisateur doit presser la touche **Ctrl**

*meta*

L'utilisateur doit presser la touche **Meta**. Il s'agit de la touche **Command** sous Macintosh.

*shift*

L'utilisateur doit presser la touche **Shift** (**Maj**)

*accel*

L'utilisateur doit presser la touche de raccourci spécifique à sa plate-forme.

Votre clavier n'a pas forcément toutes ces touches, dans ce cas, elles seront s'activeront par d'autres touches de modification que vous possédez.

L'élément `key` doit être placé à l'intérieur d'un élément `keyset`. Cet élément est destiné à contenir un ensemble d'éléments `key` servant à grouper toutes les définitions de raccourcis dans un seul emplacement du fichier. Un élément `key` à l'extérieur d'un élément `keyset` ne sera pas pris en compte.

Généralement, chaque plate-forme utilise une touche différente pour les raccourcis clavier. Par exemple, Windows utilise la touche **Ctrl** tandis que Macintosh utilise la touche **Command**. Il serait peu commode de définir un élément `key` propre à chaque plate-forme. Heureusement, il y a une solution, le modificateur de touches `accel` se réfère à la touche de raccourci spécifique à la plate-forme. Il fonctionne exactement comme les autres modificateurs de touches excepté qu'il change selon la plate-forme.

Quelques exemples supplémentaires :

```
<keyset>
  <key id="copy-key" modifiers="control" key="C"/>
  <key id="explore-key" modifiers="control alt" key="E"/>
  <key id="paste-key" modifiers="accel" key="V"/>
</keyset>
```

L'attribut `key` est utilisé pour spécifier quelles touches doivent être pressées. Toutefois, il y aura aussi des cas où vous voudrez spécifier des touches qui ne peuvent être décrites par un simple caractère (telle que la touche **Enter** ou les touches de fonctions). L'attribut `key` peut seulement être utilisé pour des caractères imprimables. Un autre attribut, `keycode` peut être utilisé pour les caractères non imprimables.

La valeur de l'attribut `keycode` doit être un code spécial qui représente la touche souhaitée. Une liste de touches est disponible ci-dessous. Toutes les touches ne sont pas disponibles sur toutes les plate-formes.

- |                  |        |                    |                  |
|------------------|--------|--------------------|------------------|
| • VK_CANCEL      | • VK_0 | • VK_NUMPAD0       | • VK_F1          |
| • VK_BACK        | • VK_1 | • VK_NUMPAD1       | • VK_F2          |
| • VK_TAB         | • VK_2 | • VK_NUMPAD2       | • VK_F3          |
| • VK_CLEAR       | • VK_3 | • VK_NUMPAD3       | • VK_F4          |
| • VK_RETURN      | • VK_4 | • VK_NUMPAD4       | • VK_F5          |
| • VK_ENTER       | • VK_5 | • VK_NUMPAD5       | • VK_F6          |
| • VK_SHIFT       | • VK_6 | • VK_NUMPAD6       | • VK_F7          |
| • VK_CONTROL     | • VK_7 | • VK_NUMPAD7       | • VK_F8          |
| • VK_ALT         | • VK_8 | • VK_NUMPAD8       | • VK_F9          |
| • VK_PAUSE       | • VK_9 | • VK_NUMPAD9       | • VK_F10         |
| • VK_CAPS_LOCK   | • VK_A | • VK_MULTIPLY      | • VK_F11         |
| • VK_ESCAPE      | • VK_B | • VK_ADD           | • VK_F12         |
| • VK_SPACE       | • VK_C | • VK_SEPARATOR     | • VK_F13         |
| • VK_PAGE_UP     | • VK_D | • VK_SUBTRACT      | • VK_F14         |
| • VK_PAGE_DOWN   | • VK_E | • VK_DECIMAL       | • VK_F15         |
| • VK_END         | • VK_F | • VK_DIVIDE        | • VK_F16         |
| • VK_HOME        | • VK_G | • VK_COMMA         | • VK_F17         |
| • VK_LEFT        | • VK_H | • VK_PERIOD        | • VK_F18         |
| • VK_UP          | • VK_I | • VK_SLASH         | • VK_F19         |
| • VK_RIGHT       | • VK_J | • VK_BACK_QUOTE    | • VK_F20         |
| • VK_DOWN        | • VK_K | • VK_OPEN_BRACKET  | • VK_F21         |
| • VK_PRINTSCREEN | • VK_L | • VK_BACK_SLASH    | • VK_F22         |
| • VK_INSERT      | • VK_M | • VK_CLOSE_BRACKET | • VK_F23         |
| • VK_DELETE      | • VK_N | • VK_QUOTE         | • VK_F24         |
|                  | • VK_O | • VK_SEMICOLON     | • VK_NUM_LOCK    |
|                  | • VK_P | • VK_EQUALS        | • VK_SCROLL_LOCK |
|                  | • VK_Q |                    | • VK_HELP        |
|                  | • VK_R |                    |                  |

- `VK_S`
- `VK_T`
- `VK_U`
- `VK_V`
- `VK_W`
- `VK_X`
- `VK_Y`
- `VK_Z`

Par exemple, pour créer un raccourci qui est activé quand l'utilisateur presse les touches **Alt** et **F5**, faites ainsi :

```
<keyset>
  <key id="test-key" modifiers="alt" keycode="VK_F5"/>
</keyset>
```

L'exemple ci-dessous montre quelques raccourcis clavier supplémentaires :

```
<keyset>
  <key id="copy-key" modifiers="accel" key="C"/>
  <key id="find-key" keycode="VK_F3"/>
  <key id="switch-key" modifiers="control alt" key="1"/>
</keyset>
```

Le premier raccourci est déclenché lorsque l'utilisateur presse la touche de raccourci spécifique à sa plate-forme et **C**. Le second est invoqué quand l'utilisateur presse **F3**. Le troisième se déclenche sur une pression des touches **Ctrl**, **Alt** et **1**. Si vous voulez distinguer les touches de la partie centrale du clavier et les touches du pavé numérique, utilisez les touches `VK_NUMPAD` (telles que `VK_NUMPAD1`).

## Utiliser les raccourcis clavier

Maintenant que nous savons comment définir les raccourcis clavier, nous allons découvrir comment les utiliser. Il y a deux manières. La première est la plus simple et requiert seulement que vous utilisiez le gestionnaire d'événements clavier sur l'élément `key`. Quand l'utilisateur presse la (ou les) touche(s), le script est invoqué. Voici un exemple :

```
<keyset>
  <key id="copy-key" modifiers="accel" key="C" onkeypress="DoCopy();" />
</keyset>
```

La fonction `DoCopy` sera appelée quand l'utilisateur pressera les touches spécifiées par l'élément `key` qui sont, dans cet exemple, les touches pour copier vers le presse-papiers (telles que **Ctrl+C**). Ceci fonctionnera tant que la fenêtre sera ouverte. La fonction `DoCopy` devrait vérifier si du texte est sélectionné et le copier dans le presse-papiers si tel est le cas. Notez que les champs de saisie intègrent déjà des raccourcis pour utiliser le presse-papiers, de sorte que vous n'avez pas besoin de les implémenter vous-même.

Si vous assignez un raccourci à une commande qui existe déjà dans un menu, vous pouvez associer directement l'élément `key` avec la commande du menu. Pour cela, ajoutez un attribut `key` à l'élément `menuitem`. Donner lui comme valeur l'id du raccourci que vous voulez lui associer. L'exemple ci-dessous explique cette méthode.

Exemple 6.3.2 :

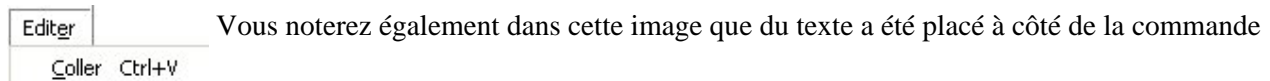
```
<keyset>
  <key id="paste-key" modifiers="accel" key="V"
    oncommand="alert('invoque Collier')"/>
```

```

</keyset>
<menubar id="sample-menubar">
  <menu id="edit-menu" label="Editer" accesskey="e">
    <menupopup id="edit-popup">
      <menuitem id="paste-command" accesskey="c" key="paste-key" label="Coller"
        oncommand="alert('invoke Coller')"/>
    </menupopup>
  </menu>
</menubar>

```

L'attribut `key` de l'item du menu, qui est ici `paste-key`, est égal à l'id du raccourci défini. Vous pouvez utiliser cette méthode pour définir des raccourcis supplémentaires à plusieurs items de menu.



`Coller` du menu pour indiquer le raccourci **Ctrl+V** pouvant être pressé pour invoquer la commande du menu. Cette indication est ajoutée automatiquement pour vous sur la base des touches de modification de l'élément `key`. Les raccourcis associés aux menus fonctionneront même si le menu n'est pas ouvert.

Une fonctionnalité supplémentaire des définitions de raccourcis est que vous pouvez les désactiver facilement. Il vous suffit d'ajouter un attribut `disabled` à l'élément `key` et lui affecter la valeur `true`. Cet attribut désactive le raccourci clavier de façon à ce qu'il ne puisse pas être invoqué. Il est utile de modifier l'attribut `disabled` par le biais d'un script.

## Exemples de raccourcis

Ajoutons des raccourcis clavier à la boîte de dialogue de recherche de fichiers. Nous en ajouterons quatre, un pour chacune des commandes Couper, Copier et Coller, et aussi un pour la commande Fermer quand l'utilisateur presse **Esc**

```

<keyset>
  <key id="cut_cmd" modifiers="accel" key="X"/>
  <key id="copy_cmd" modifiers="accel" key="C"/>
  <key id="paste_cmd" modifiers="accel" key="V"/>
  <key id="close_cmd" keycode="VK_ESCAPE" oncommand="window.close();" />
</keyset>

<vbox flex="1">
  <toolbox>
    <menubar id="findfiles-menubar">
      <menu id="file-menu" label="Fichier" accesskey="f">
        <menupopup id="file-popup">
          <menuitem label="Ouvrir une recherche..." accesskey="o"/>
          <menuitem label="Sauver une recherche..." accesskey="s"/>
          <menuseparator/>
          <menuitem label="Fermer" accesskey="c" key="close_cmd"
            oncommand="window.close();" />
        </menupopup>
      </menu>
      <menu id="edit-menu" label="Editer" accesskey="e">
        <menupopup id="edit-popup">
          <menuitem label="Couper" accesskey="c" key="cut_cmd" />
          <menuitem label="Copier" accesskey="p" key="copy_cmd" />
          <menuitem label="Coller" accesskey="l" key="paste_cmd" disabled="true" />
        </menupopup>
      </menu>
    </menubar>
  </toolbox>
</vbox>

```

Maintenant nous pouvons utiliser ces raccourcis pour activer les commandes. Évidemment les commandes du presse-papiers restent inactives puisque nous n'avons pas encore écrit leurs scripts.

## Évènements Clavier

Il y a trois types d'évènements clavier qui peuvent être utilisés si les dispositifs principaux décrits ci-dessus ne sont pas appropriés.

### *keypress*

Appelé quand une touche est pressée puis relachée avec l'élément qui a le focus (élément actif). Vous pouvez l'utiliser pour contrôler les caractères saisis dans un champ.

### *keydown*

Appelé quand une touche est pressée avec l'élément qui a le focus (élément actif). Remarquez que l'évènement sera appelé aussitôt la touche enfoncée, même si elle n'a pas été encore relachée.

### *keyup*

Appelé quand une touche est relachée avec l'élément qui a le focus (élément actif).

Les évènements clavier sont envoyés seulement à l'élément qui a le focus. Typiquement, ils incluent les champs de saisie, les boutons, les cases à cocher, et d'autres encore. Si aucun des éléments n'est actif, l'évènement sera dirigé vers le document XUL lui-même. Dans ce cas, vous pouvez associer un scrutateur d'évènements à la balise `window`. Cependant, si vous voulez réagir aux évènements de manière globale, vous devriez utiliser un raccourci clavier comme décrit plus haut.

L'objet `event` a deux propriétés qui définissent la touche pressée. La propriété `keyCode` contient le code de la touche qui peut être comparé à une des constantes de la table des codes de touche vue plus tôt dans cette section. La propriété `charCode` est utilisée pour les caractères imprimables et contient le caractère de la touche pressée.

Dans la prochaine section, nous allons découvrir comment gérer le focus et la sélection.

## 6.4 Focus et Selection

Écrit par Neil Deakin. Traduit par **Adrien Bustany** (19/07/2004), mise à jour par Cyril trumpler (15/04/2005)

Page originale : <http://www.xulplanet.com/tutorials/xultu/focus.html>

Cette section va décrire comment manipuler le focus et la sélection des éléments.

### Éléments focalisés

L'élément focalisé est l'élément qui reçoit les évènements d'entrée. Si il y a trois champs de saisie sur une fenêtre, celui qui détient le focus est celui dans laquelle l'utilisateur peut taper du texte. Seul un élément peut détenir le focus à la fois.

L'utilisateur peut changer le focus en cliquant sur un élément avec la souris ou en appuyant sur la touche **Tab** (tabulation). Lorsque la touche tabulation est appuyée, le focus passe à l'élément suivant. Pour revenir en arrière, il suffit d'appuyer sur les touches **Maj** (shift) et **Tab** simultanément.

Vous pouvez changer l'ordre dans lequel les éléments seront focalisés quand l'utilisateur appuiera sur la touche **Tab** en ajoutant un attribut `tabindex` à un élément. Cet attribut doit être défini avec un nombre. Lorsque l'utilisateur appuiera sur la touche **Tab**, le focus sera donné à l'élément ayant l'index de tabulation consécutif le plus haut. Cela implique que vous pouvez ordonner les éléments en définissant des indices pour séquencer les éléments. Toutefois, vous n'aurez normalement pas à définir l'attribut `tabindex`. Dans ce cas, un appui sur la touche tabulation donnera le focus à l'élément suivant affiché. Vous avez seulement besoin de définir des indices de tabulation si vous voulez utiliser un ordre différent. Voici un exemple :

Exemple 6.4.1 : [Source Voir](#)

```
<button label="Bouton 1" tabindex="2"/>
<button label="Bouton 2" tabindex="1"/>
<button label="Bouton 3" tabindex="3"/>
```

L'évènement focus est utilisé pour réagir lorsqu'un élément obtient le focus. L'évènement blur est utilisé pour réagir lorsqu'un élément perd le focus. Vous pouvez réagir aux changements de focus en ajoutant un attribut onfocus ou onblur à un élément. Ils fonctionnent de la même façon que leurs homologues HTML. Vous pouvez utiliser ces événements pour mettre un élément en surbrillance ou afficher un texte dans la barre d'état. L'exemple suivant peut s'appliquer à une fonction de gestion des événements de focus.

#### Exemple 6.4.2 :

```
<script>

function displayFocus()
{
    var elem=document.getElementById('sbar');
    elem.setAttribute('value','Entrez votre numéro de téléphone.');
```

```
}
```

```
</script>

<textbox id="tbx1"/>
<textbox id="tbx2" onfocus="displayFocus();" />
<description id="sbar" value=""/>
```

L'évènement focus, quand il a lieu, va appeler la fonction `displayFocus`. Cette fonction va changer la valeur de l'étiquette texte. Nous pourrions développer cet exemple pour effacer le texte quand l'évènement blur a lieu. Généralement vous utiliserez les événements focus et blur pour mettre à jour certaines parties de votre interface quand l'utilisateur sélectionne des éléments. Par exemple, vous pouvez mettre à jour un total quand l'utilisateur entre des valeurs dans d'autres champs, ou utiliser les événements de focus pour vérifier certaines valeurs. N'affichez pas de messages d'alerte pendant un événement focus ou blur, car ils pourraient distraire l'utilisateur et ils dégradent le design de l'interface.

Vous pouvez aussi ajouter des gestionnaires d'évènements dynamiquement en utilisant la fonction `DOM addEventListener`. Vous pouvez l'utiliser pour n'importe quel élément et type d'évènement. Elle prend trois paramètres, le type d'évènement, une fonction à exécuter quand l'évènement a lieu et un booléen indiquant si il faut capturer ou non l'évènement.

L'élément ayant le focus est pris en charge par un objet appelé répartiteur de commandes, et dont il ne peut y avoir qu'une instance par fenêtre. Le répartiteur de commandes garde la trace de l'objet qui a le focus pendant que l'utilisateur se sert de l'interface. Le répartiteur de commandes a d'autres rôles, qui seront abordés plus tard dans la section des commandes. Pour le moment, nous allons nous intéresser à quelques fonctions relatives au focus du répartiteur de commandes.

Vous pouvez récupérer le répartiteur de commandes d'une fenêtre en utilisant la propriété `commandDispatcher` du document. À partir de là, vous pouvez obtenir l'élément focalisé avec la propriété `focusedElement` du répartiteur. L'exemple ci-dessous illustre ce cas.

#### Exemple 6.4.3 :

```
<window id="focus-example" title="Exemple Focus"
    onload="init();"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script>
function init()
{
    addEventListener("focus",setFocusedElement,true);
```

```

}

function setFocusedElement()
{
    var focused = document.commandDispatcher.focusedElement;
    document.getElementById("focused").value = focused.tagName;
}
</script>

<hbox>
    <label control="username" value="Nom d'utilisateur:"/>
    <textbox id="username"/>
</hbox>

<button label="Bonjour"/>
<checkbox label="Se souvenir de ce choix"/>

<label id="focused" value="-pas de focus-"/>

</window>

```

Dans cet exemple, un gestionnaire d'évènement de focus est attaché à la fenêtre. Nous voulons utiliser un gestionnaire de capture d'évènements, donc la méthode `addEventListener` doit être utilisée. Elle associe un gestionnaire de capture d'évènements à la fenêtre qui appellera la méthode `setFocusedElement`. Cette méthode prend l'élément focalisé du répartiteur de commandes et modifie une étiquette avec le nom de sa balise. Quand l'élément focalisé change, l'étiquette montre le nom de l'élément. Voici quelques choses à noter. Premièrement, quand le champ de saisie est focalisé, le nom de la balise est 'html:input', et non 'textbox' comme on aurait pu s'y attendre. C'est parce que les champs de saisie XUL sont implémentés avec le composant HTML input, donc l'évènement focus est en fait reçu pour cet élément. Deuxièmement, un clic sur le libellé du champs de saisie donne le focus au champ de saisie. C'est parce que le libellé possède un attribut `control` pointant vers l'id du champ de saisie. Enfin, l'autre étiquette qui affiche la propriété `nom` n'a pas d'attribut `control`, donc un clic dessus n'affecte pas l'élément focalisé. Seuls les éléments focalisables peuvent être focalisés.

Si vous créez des éléments personnalisés, vous pourriez avoir besoin de décider si un élément peut prendre le focus ou non. Il vous suffit d'utiliser la propriété de style spéciale `-moz-user-focus`. Cette propriété détermine si un élément peut être focalisé ou non. Par exemple, vous pourriez rendre une étiquette focalisable, comme dans l'exemple ci-dessous.

Exemple 6.4.4 :

```

<label id="focused" style="-moz-user-focus: normal;"
      onkeypress="alert('Étiquette focalisée');" value="Focalise moi"/>

```

La propriété de style est réglée à *normal*. Vous pouvez aussi la définir à *ignore* pour désactiver le focus pour un élément. Toutefois, elle ne doit pas être utilisée pour désactiver un élément ; l'attribut ou la propriété `disabled` doit être utilisée dans ce cas, car elle a été conçue pour. Une fois que l'étiquette dans l'exemple est focalisée, elle peut réagir aux évènements du clavier. Évidemment, l'étiquette ne donne aucune indication comme quoi elle est focalisée, car elle n'est normalement pas prévue pour.

Il existe plusieurs façons de changer l'élément focalisé. La plus simple est d'appeler la méthode `focus` de l'élément XUL que vous voulez focaliser. La méthode `blur` peut être utilisée afin d'enlever le focus d'un élément. L'exemple suivant met en pratique ces principes :

Exemple 6.4.5 :

```

<textbox id="addr"/>

<button label="Focus" oncommand="document.getElementById('addr').focus()"/>

```



Sinon, vous pouvez utiliser les méthodes `advanceFocus` et `rewindFocus` du répartiteur de commandes. Ces méthodes changent le focus respectivement vers l'élément suivant ou précédent de la séquence de tabulation. Elles correspondent aux actions réalisées lorsque l'utilisateur appuie sur **Tab** ou **Maj+Tab**.

Pour les champs de saisie, un attribut spécial, `focused` est ajouté quand l'élément obtient le focus. Vous pouvez vérifier la présence de cet attribut pour déterminer si l'élément a le focus, soit depuis un script, soit depuis une feuille de styles. Il aura la valeur `true` si le champ de saisie a le focus, et le cas échéant, l'attribut ne sera pas présent.

## Capturer les modifications de texte

Il y a 2 événements pouvant être utilisés lorsque l'utilisateur modifie le contenu d'un champ de saisie. Naturellement, ces événements seront uniquement transmis au champ de saisie ayant le focus. L'événement d'insertion est transmis à chaque fois que le contenu du champ est modifié. La nouvelle valeur sera différente de l'ancienne. Vous devriez utiliser cet événement au lieu d'utiliser les événements clavier, car certaines touches, par exemple **Maj**, ne change pas la valeur. De plus, l'événement d'insertion ne se déclenche pas si une touche est enfoncée et s'il y a déjà plus de caractères dans le champ de saisie qu'il ne peut en contenir.

L'événement de changement est similaire dans le sens où il est transmis seulement lorsque le champ est modifié. Cependant, il est seulement transmis une fois que le champ texte perd le focus, ce qui veut dire, seulement une fois après une série de changements.

## Sélection de texte

Lorsque vous travaillez avec un champ de saisie, vous pouvez récupérer uniquement le texte que l'utilisateur a sélectionné. Ou alors vous pouvez changer la sélection.

Les champs de saisie XUL offrent une possibilité de récupérer et de modifier une sélection. La plus simple et de sélectionner tout le texte du champ. Cela implique l'utilisation de la méthode `select` du champ de saisie.

```
tbox.select();
```

Toutefois, vous pouvez aussi sélectionner seulement une partie du texte. Il vous suffit d'utiliser la fonction `setSelectionRange`. Elle prend deux paramètres, le premier représente le caractère de départ et le second le caractère suivant le dernier que vous voulez sélectionner. Les valeurs commencent à zéro, donc le premier caractère est 0, le second 1 et ainsi de suite.

```
tbox.setSelectionRange(4,8);
```

Cet exemple ca sélectionner le cinquième caractère affiché, ainsi que le sixième, le septième et le huitième. Si il n'y avait que six caractères présents dans le champ, seulement le cinquième et le sixième auraient été sélectionnés. Il n'y aurait pas d'erreur.

Si vous utilisez la même valeur pour les deux paramètres, le début et la fin de la sélection changent pour la même position. Le résultat revient à changer la position du curseur dans le champs de saisie. Par exemple, la ligne ci-dessous peut être utilisée pour bouger le curseur au début du texte.

```
tbox.setSelectionRange(0,0);
```

Vous pouvez récupérer la sélection en utilisant les propriétés `selectionStart` et `selectionEnd`. Ces propriétés définissent respectivement le début et la fin de la sélection. Si les deux sont définies à la même valeur, aucun texte n'est sélectionné, mais les valeurs représentent la position du curseur. Une fois que vous avez les valeurs de début et de fin, vous pouvez récupérer la section de chaîne du texte.

Vous pouvez récupérer et modifier le contenu du champ de saisie en utilisant la propriété `value`.

Une autre propriété utile des champs de saisie est la propriété `textLength` qui contient le nombre total de caractères dans le champ.

---

Dans la prochaine section, nous découvrirons comment utiliser les commandes.

## 6.5 Commandes

Écrit par Neil Deakin. Traduit par **Laurent Jouanneau** (15/11/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/commands.html>

Une commande est une opération qui peut être invoquée.

### Les éléments de commande

L'élément `command` est utilisé pour créer des commandes qui pourront être utilisées pour exécuter des opérations. Vous n'avez pas besoin d'utiliser les commandes si vous avez juste à appeler un script pour manipuler des choses. Cependant, une commande a l'avantage de pouvoir être désactivée automatiquement quand c'est nécessaire, et de pouvoir être invoquée de l'extérieur sans avoir besoin de savoir les détails de son implémentation. Les commandes fournissent un moyen pour séparer de façon abstraite les opérations et le code. Elles deviennent très utiles pour les grosses applications.

Par exemple, pour implémenter les commandes de menus du presse-papiers, *couper*, *copier* et *coller*, vous pouvez utiliser les commandes. Si vous ne les utilisiez pas, vous devriez trouver quel champ a le focus, ensuite s'assurer que l'opération est valable pour cet élément. De plus, les commandes de menus devraient être activées ou désactivées selon si l'élément cible a du texte sélectionné ou pas, et pour les opérations de collage, si il y a quelque chose de valable dans le presse-papiers, à coller. Comme vous pouvez le voir, cela devient compliqué. En utilisant les commandes, votre travail est simplifié.

Vous pouvez utiliser une commande pour n'importe quelle opération. Mozilla les utilise la plupart du temps pour les menus. De plus, les champs de saisie de texte et autres composants graphiques disposent de plusieurs commandes natives que vous pouvez donc invoquer. Vous devriez les utiliser quand les opérations dépendent de l'élément sélectionné.

Une commande est identifiée par son attribut `id`. Mozilla utilise une convention : les id de commandes commencent par `cmd_`. Vous voudrez probablement utiliser le même identifiant que celui d'une commande déjà utilisée, cependant, pour vos propres commandes, vous pouvez utiliser n'importe quel id de commande souhaité. Pour éviter les conflits, il est préférable d'inclure le nom de l'application dans l'id de la commande. Un moyen simple d'utilisation des commandes est montré ci-après :

Exemple 6.5.1 : .

```
<command id="cmd_openhelp" oncommand="alert('Aide !');"/>
<button label="Aide" command="cmd_openhelp"/>
```

Dans cet exemple, au lieu de placer l'attribut `oncommand` sur l'élément `button`, nous le plaçons sur un élément `command`. Les deux sont alors liés en utilisant l'attribut `command` qui a la valeur de l'id de la commande. Ainsi quand le bouton est pressé, la commande est invoquée.

Il y a deux avantages en utilisant cette approche. Premièrement, Elle déplace toutes les opérations dans des commandes qui peuvent être toutes regroupées ensemble dans une seule section de fichier XUL. Elle signifie que le code est au même endroit, et n'est pas éparpillé dans tout le code de l'interface utilisateur. Le deuxième avantage est que différents boutons et autres éléments de l'interface graphique peuvent être

rattachés à une même commande. Par exemple, vous pouvez avoir un item de menu, un bouton d'une barre de boutons et un raccourci clavier pour effectuer la même opération. Plutôt que de répéter le code trois fois, vous pouvez rattacher ces trois éléments à la même commande. Normalement, vous rattacherez seulement les éléments pouvant générer un événement de commande.

Si vous spécifiez l'attribut `disabled` sur une commande, elle sera désactivée et ne pourra pas être invoquée. En plus de cela, tous les boutons et les items de menus qui lui sont rattachés seront désactivés automatiquement. Si vous réactivez la commande, les boutons deviendront actifs de nouveau.

Exemple 6.5.2 :

```
<command id="cmd_openhelp" oncommand="alert('Aide');"/>
<button label="Aide" command="cmd_openhelp"/>
<button label="Plus d'aide" command="cmd_openhelp"/>

<button label="Désactiver"
  oncommand="document.getElementById('cmd_openhelp').setAttribute('disabled','true');"/>
<button label="Activer"
  oncommand="document.getElementById('cmd_openhelp').removeAttribute('disabled');"/>
```

Dans cet exemple, les deux boutons utilisent la même commande. Quand le bouton "Désactiver" est pressé, la commande est désactivée en mettant son attribut `disabled`, et les deux boutons seront aussi désactivés.

Habituellement, un groupe de commandes se place à l'intérieur d'un élément commandset, près du début du fichier XUL, comme dans l'exemple suivant :

```
<commandset>
  <command id="cmd_open" oncommand="alert('Ouvrir !');"/>
  <command id="cmd_help" oncommand="alert('Aide !');"/>
</commandset>
```

Une commande est invoquée quand l'utilisateur active le bouton ou les autres éléments rattachés à la commande. Vous pouvez aussi invoquer une commande en appelant la méthode `doCommand`, que ce soit de l'élément command ou d'un élément rattaché à la commande comme un bouton.

## Le répartiteur de commandes

Vous pouvez aussi utiliser les commandes sans utiliser l'élément command, ou, au moins, sans ajouter un attribut `oncommand` sur la commande. Dans ce cas, la commande n'invoquera pas un script directement, mais recherchera à la place un élément ou une fonction qui traitera la commande. Cette fonction peut être séparée du XUL lui-même, et peut être embarquée par un élément graphique en interne. Afin de trouver ce qui traitera la commande, XUL utilise un objet appelé répartiteur de commande (NdT : `command dispatcher`). Cet objet localise le gestionnaire d'une commande. Le gestionnaire d'une commande est appelé contrôleur. Ainsi, quand une commande est invoquée, le répartiteur de commande localise un contrôleur qui traite la commande. Vous pouvez déduire que l'élément command est un type de contrôleur pour une commande.

Le répartiteur de commandes localise un contrôleur en regardant l'élément sélectionné pour voir s'il a un contrôleur qui gère la commande. Les éléments XUL ont une propriété `controllers` qui est utilisée pour la vérification. Vous pouvez l'utiliser pour ajouter vos propres contrôleurs. Vous pourriez l'utiliser pour avoir une boîte de liste qui répond aux opérations de couper, copier et coller. Un exemple sera fourni plus tard. Par défaut, seuls les champs de saisie (`textbox`) ont un contrôleur fonctionnel. Ce contrôleur gère aussi bien les opérations de presse-papiers, sélection, défaire et refaire, que les opérations d'édition. Notez qu'un élément peut avoir plusieurs contrôleurs, qui seront alors tous pris en compte.

Si l'élément courant sélectionné n'a pas le contrôleur attendu, la fenêtre sera alors vérifiée. L'élément

window a aussi une propriété `controllers` que vous pouvez modifier comme bon vous semble. Si le focus est à l'intérieur d'un cadre `frame`, chaque cadre parent est également vérifié. Ainsi, les commandes fonctionneront même si le focus est à l'intérieur d'un cadre. Ce mécanisme fonctionne bien pour un navigateur ; les commandes d'édition invoquées à partir du menu principal fonctionneront à l'intérieur de la zone de contenu. Notez que HTML a aussi un système de commandes et de contrôleur, bien que vous ne pouvez pas l'utiliser sur des pages Web sans privilèges. Mais vous pouvez l'utiliser, par exemple, dans une extension du navigateur. Si la fenêtre ne fournit pas un contrôleur capable de gérer la commande, rien ne se passera.

Vous pouvez récupérer le répartiteur de commande en utilisant la propriété `commandDispatcher` de l'objet `document` ou pouvez le récupérer à partir des contrôleurs listés dans un élément ou la fenêtre. Le répartiteur de commande contient des méthodes pour récupérer les contrôleurs pour les commandes et pour récupérer et modifier le focus.

Vous pouvez implémenter vos propres contrôleurs pour répondre aux commandes. Vous pouvez tout aussi bien surcharger la gestion par défaut d'une commande en plaçant le contrôleur correctement. Un contrôleur doit implémenter quatre méthodes qui sont listées ci-dessous :

*supportsCommand (command)*

Cette méthode doit renvoyer *true* si le contrôleur gère la commande. Si vous renvoyez *false*, la commande n'est pas gérée et le répartiteur de commande interrogera un autre contrôleur. Un contrôleur peut gérer plusieurs commandes.

*isCommandEnabled (command)*

Cette méthode doit renvoyer *true* si la commande est activée, *false* sinon. Les boutons correspondants seront désactivés automatiquement.

*doCommand (command)*

exécute la commande. C'est ici que vous mettrez le code pour gérer la commande.

*onEvent (event)*

Cette méthode gère un événement.

Imaginons que nous voulions implémenter une boîte de liste (listbox) qui gère la commande "Supprimer". Quand un utilisateur sélectionne "Supprimer" dans le menu, la boîte de liste efface la ligne sélectionnée. Dans ce cas, vous avez juste à attacher un contrôleur à l'élément listbox qui exécutera la méthode `doCommand`.

Essayez d'ouvrir l'exemple qui suit dans une fenêtre du navigateur et sélectionnez des items de la liste. Vous noterez que la commande "Supprimer" du menu "Edition" du navigateur est activée et qu'elle effacera la ligne sélectionnée. L'exemple n'est cependant pas complet. Nous devrions nous assurer que la sélection et le focus soient ajustés comme il faut après l'effacement.

Exemple :

```
<window id="controller-example" title="Exemple de contrôleur" onload="init();"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script>
function init()
{
  var list = document.getElementById("theList");

  var listController = {
    supportsCommand : function(cmd){ return (cmd == "cmd_delete"); },
    isCommandEnabled : function(cmd){
      if (cmd == "cmd_delete") return (list.selectedItem != null);
      return false;
    },
    doCommand : function(cmd){
```

```

        list.removeItemAt(list.selectedIndex);
    },
    onEvent : function(evt){ }
};

list.controllers.appendController(listController);
}
</script>

<listbox id="theList">
    <listitem label="Océan"/>
    <listitem label="Desert"/>
    <listitem label="Jungle"/>
    <listitem label="Marécage"/>
</listbox>

</window>

```

Le contrôleur `listController` implémente les quatre fonctions décrites plus haut. La méthode `supportsCommand` renvoie *true* pour la commande `cmd_delete`, qui est le nom de la commande utilisée lorsque l'item de menu "Supprimer" est sélectionné. Pour les autres commandes, *false* est renvoyé puisque le contrôleur ne gère aucune autre commande. Si vous voulez gérer plusieurs commandes, indiquez les ici, dès lors que vous utiliserez souvent un seul contrôleur pour plusieurs commandes apparentées.

La méthode `isCommandEnabled` renvoie *true* si la commande est activée. Dans le cas présent, nous vérifions s'il y a un item sélectionné dans la liste et renvoyons *true* si c'est le cas. S'il n'y a pas de sélection, *false* est renvoyé. Si vous effacez toutes les lignes dans l'exemple, la commande "Supprimer" deviendra inactive. Vous devrez cliquer sur la liste pour mettre à jour le menu dans cet exemple simple. La méthode `doCommand` sera appelée lorsque l'item de menu "Supprimer" sera sélectionné, et elle provoquera l'effacement de la ligne sélectionnée dans la liste. Rien ne se passera pour la méthode `onEvent`, aussi nous n'ajouterons pas de code pour celle-ci.

Nous attachons le contrôleur à l'élément `listbox` en appelant la méthode `appendController` des objets contrôleurs de la liste. L'objet `controller` a un certain nombre de méthodes qui peuvent être utilisées pour manipuler les contrôleurs. Par exemple, il y a aussi une méthode `insertControllersAt` qui insère un contrôleur dans un élément avant les autres. Elle peut être utile pour surcharger des commandes. Par exemple, le code suivant désactivera le collage du presse-papiers dans un champ de saisie.

```

var tboxController = {
    supportsCommand : function(cmd){ return (cmd == "cmd_paste"); },
    isCommandEnabled : function(cmd){ return false; },
    doCommand : function(cmd){ },
    onEvent : function(evt){ }
};

document.getElementById("tbox").controllers.insertControllerAt(0,tboxController);

```

Dans cet exemple, nous insérons le contrôleur à l'index `0`, c'est-à-dire avant tous les autres. Le nouveau contrôleur supporte la commande `'cmd_paste'` et indique qu'elle est désactivée. Le contrôleur par défaut de `textbox` ne sera jamais appelé parce que le répartiteur de commande trouve un contrôleur avant celui-ci, prenant en charge la commande en premier.

---

Dans la section suivante, nous allons voir comment mettre à jour les commandes.

## 6.6 Mise à jour de commandes

Écrit par Neil Deakin. Traduit par *Nadine Henry* (20/07/04).

Page originale : <http://www.xulplanet.com/tutorials/xultu/commandupdate.html>

Dans cette section, nous verrons comment mettre à jour des commandes.

## Appel des commandes

Si une commande a un attribut `oncommand`, vous pouvez simplement l'appeler en utilisant la méthode `doCommand` de la commande ou un élément qui lui est attaché. Pour d'autres commandes, vous aurez besoin de quelques lignes de codes additionnelles. Vous devrez passer par ces étapes spéciales dans le cas où les commandes appelées sont implémentées par un contrôleur. De plus, vous aurez besoin de le faire dans le cas où vous créez votre propre menu de commandes, par exemple pour implémenter les commandes du menu d'édition dans votre propre application.

Heureusement, le code spécial est assez simple. Tout ce que nous avons besoin de faire est d'obtenir le contrôleur demandé et d'appeler la commande. Une manière simple de le faire est la suivante :

```
var controller = document.commandDispatcher.getControllerForCommand( "cmd_paste" );
if (controller && controller.isCommandEnabled( "cmd_paste" ))
    controller.doCommand( command );
}
```

Le code ci-dessus recherche d'abord le contrôleur pour la commande 'cmd\_paste' grâce au répartiteur de commandes. Puis, il vérifie pour voir si la commande est activée, et enfin exécute la commande utilisant la méthode `doCommand` du contrôleur. Notez que nous n'avons pas besoin de préciser l'élément ou le contrôleur à utiliser. Le répartiteur de commandes se charge de cette partie. En outre, nous pourrions juste appeler `doCommand` sans vérifier si la commande est activée ou non, bien que nous ne le devrions probablement pas.

Le code ci-dessus est tellement générique qu'il peut être une fonction qui prenne en paramètre un argument et exécute cette commande. Cette fonction pourrait être ainsi réutilisée pour toutes les commandes. En fait, c'est tellement commun que Mozilla inclue une bibliothèque qui ne fait que ça. Si vous incluez le script 'chrome://global/content/globalOverlay.js' dans un fichier XUL, vous pouvez appeler la méthode `goDoCommand` qui exécute la commande passée en argument. Le code pour cette fonction n'est long que de quelques lignes, ainsi vous pourriez l'inclure directement dans votre code si pour certaines raisons vous ne souhaitez pas inclure la bibliothèque.

```
<script src="chrome://global/content/globalOverlay.js"/>

<command id="cmd_paste" oncommand="goDoCommand( 'cmd_paste' );"/>
<button label="Coller" command="cmd_paste"/>
```

L'exemple ci-dessus va implémenter un bouton pour "Coller". Il est relié à la commande qui va appeler la commande du contrôleur concerné lorsqu'il est appelé. Le code ci-dessus est tout ce dont vous avez besoin pour implémenter la fonctionnalité de la commande Coller dans votre application. La seule autre chose dont vous avez besoin est de vous assurer que le statut de la commande Coller qui est activé, et donc du bouton, est mis à jour au bon moment, comme décrit ci-dessous.

## Dispositifs de mise à jour de commande

Un dispositif de mise à jour de commande est un dispositif spécial de l'élément `commandset` qui lui permet de mettre à jour les statuts activés d'une ou plusieurs commandes lorsque certains événements se passent. Vous devrez y penser lorsqu'une commande est valide et lorsqu'elle ne l'est pas. De plus, vous devrez considérer quand l'état pourrait changer et quand les commandes devraient être mises à jour.

Par exemple, la commande "Coller" est valide lorsque le champ de saisie de texte a le focus et qu'il y a quelque chose dans le presse-papiers à coller. La commande deviendra active chaque fois que le champ de saisie aura le focus et lorsque le contenu du presse-papiers changera. Un dispositif de mise à jour de contenu

surveillera ces situations et le code qui active et désactive les commandes pourra être exécuté selon les besoins.

Un simple dispositif de mise à jour de commandes ressemble à ceci :

```
<commandset id="updatePasteItem"
  commandupdater="true"
  events="focus"
  oncommandupdate="goUpdateCommand( 'cmd_paste' );" />
```

Un dispositif de mise à jour de commandes est indiqué en utilisant l'attribut `commandupdater`, qui devrait être déclaré à *true*. L'attribut `events` est utilisé pour lister les événements que le dispositif de mise à jour de commandes surveille. Vous pouvez spécifier de multiples événements en les séparant par des virgules. Dans l'exemple ci-dessus, le dispositif de mise à jour de commandes surveille les événements de focus. Il a pour effet de mettre à jour les commandes lorsqu'un élément reçoit le focus.

Lorsqu'un événement de focus se produit, le code dans l'attribut `oncommandupdate` est appelé. Dans l'exemple, la méthode `goUpdateCommand`, qui est une fonction provenant du script `globalOverlay.js` décrit plus tôt, est appelée. Elle va mettre à jour la commande et activer ou désactiver les items de boutons et de menus nécessaires. Le code qui est derrière est assez simple. Il appelle seulement le contrôleur nécessaire, appelle sa méthode `isCommandEnabled`, et enfin active ou désactive la commande. Si vous avez plusieurs commandes à mettre à jour, appelez la méthode `goUpdateCommand` une fois pour chaque commande.

Notez que le dispositif de mise à jour de commandes recevra les notifications de tous les événements de focus sur tous les éléments, même si d'autres gestionnaires d'événements répondent à l'événement. Essentiellement, un dispositif de mise à jour de commandes est comme un gestionnaire global d'événements.

Les dispositifs de mise à jour de commandes disposent de plusieurs événements pouvant répondre à ceux qui sont listés ci-dessous. Il est également possible de créer le votre.

- **focus** : se produit lorsque l'élément qui a le focus change.
- **select** : se produit lorsque le texte sélectionné change.
- **undo** : se produit lorsque le tampon d'annulation change.
- **clipboard** : se produit lorsque le contenu du presse-papiers change.

L'exemple suivant montre le dispositif de mise à jour de commandes utilisé dans le navigateur Mozilla pour mettre à jour le menu d'édition de commandes. Les fonctions utilisées sont disponibles dans le script '`chrome://communicator/content/utilityOverlay.js`'.

```
<commandset id="globalEditMenuItems"
  commandupdater="true"
  events="focus"
  oncommandupdate="goUpdateGlobalEditMenuItems()" />
<commandset id="selectEditMenuItems"
  commandupdater="true"
  events="select"
  oncommandupdate="goUpdateSelectEditMenuItems()" />
<commandset id="undoEditMenuItems"
  commandupdater="true"
  events="undo"
  oncommandupdate="goUpdateUndoEditMenuItems()" />
<commandset id="clipboardEditMenuItems"
  commandupdater="true"
  events="clipboard"
  oncommandupdate="goUpdatePasteMenuItems()" />
```

---

Ensuite, nous vous montrerons comment utiliser les observateurs.

## 6.7 Broadcasters et Observateurs

Écrit par Neil Deakin. Traduit par *BrainBooster* (20/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/broadob.html>

Il y a des moments où vous voulez que plusieurs éléments répondent à des événements ou changent d'état aisément. Pour cela, nous pouvons utiliser les « broadcasters » (diffuseurs).

### Commandes de transmission de paramètres

Nous avons déjà vu que les éléments tels que les boutons peuvent être ancrés à des commandes. De plus si vous placez l'attribut `disabled` sur l'élément `command`, tous les éléments ancrés sur celui-ci seront eux aussi désactivés automatiquement. C'est une façon utile de diminuer la taille du code nécessaire. Cette technique fonctionne aussi pour les autres attributs. Par exemple, si vous placez un attribut `label` sur un élément `command`, chaque bouton attaché à la commande partagera ce libellé.

Exemple 6.7.1 :

```
<command id="ma_commande" label="Ouvrir"/>

<button command="ma_commande"/>
<checkbox label="Ouvrir une nouvelle fenêtre" command="ma_commande"/>
```

Dans cet exemple le bouton n'a pas l'attribut `label`, néanmoins il est attaché à une commande qui en possède un. Le bouton va donc le partager avec la commande. La case à cocher a déjà un libellé, néanmoins, il va être surchargé par celui de la commande. Le résultat est que le bouton et la case à cocher auront le même libellé *Ouvrir*.

Si vous modifiez l'attribut `label` de la commande, les libellés du bouton et de la case à cocher changeraient eux aussi. Nous avons vu quelque chose comme ça dans une section précédente où l'attribut `disabled` était défini puis propagé aux autres éléments.

Cette transmission d'attribut est relativement utile pour plusieurs raisons. Par exemple, disons que nous voulons désactiver l'action "Page précédente" dans un navigateur. Nous aurions besoin de désactiver cette action dans le menu, dans la barre des tâches, le raccourci clavier (**Alt+Gauche** par ex.) et chaque commande "Page précédente" des menus déroulants. Bien que nous pourrions écrire un script pour faire ceci, ce n'est pas très simple. Le désavantage est de devoir prévoir tous les endroits où pourraient se trouver les boutons "Page précédente". Si quelqu'un a ajouté un nouveau bouton "Page précédente" en utilisant une extension, il ne serait pas pris en compte. Il est plus pratique de désactiver simplement l'action "Page précédente" et d'avoir tous les éléments utilisant cette action se désactiver eux même. Nous pouvons utiliser la transmission d'attribut des commandes pour accomplir cela.

### Broadcasters

Il y a un élément similaire appelé `broadcaster`. Les Broadcasters supportent la transmission d'attributs de la même manière que les commandes. Ils fonctionnent de la même manière excepté qu'une commande est utilisée pour les actions, alors qu'un broadcaster est utilisé pour contenir l'information d'un état. Par exemple, un élément `command` serait utilisé pour une action comme "Page précédente", "Couper" ou "Supprimer". Un `broadcaster` serait utilisé pour contenir, par exemple, un drapeau indiquant si l'utilisateur est en ligne ou non. Dans le premier cas, les éléments du menu et de la barre des tâches nécessiteraient d'être désactivés lorsqu'il n'y a pas de page de retour, ou aucun texte à couper, à effacer. Dans le second cas, plusieurs éléments de l'interface auraient besoin d'être mis à jour lorsque l'utilisateur passerait du mode en ligne au mode hors ligne.



Le broadcaster le plus simple est défini ci-dessous. Vous devriez toujours utiliser un attribut `id` afin qu'il puisse être référencé à partir d'autres éléments.

```
<broadcasterset>
  <broadcaster id="isOffline" label="Hors ligne"/>
</broadcasterset>
```

Tous les éléments qui observent le broadcaster seront modifiés automatiquement chaque fois que l'attribut `label` du broadcaster change. Ces éléments auront comme résultat un nouveau libellé. Tout comme d'autres éléments non affichés, l'élément `broadcasterset` est un conteneur pour les broadcasters. Vous devez déclarer tous vos broadcasters dans un élément `broadcasterset` afin de les réunir.

Les éléments qui observent le broadcaster sont appelés observateurs car ils observent l'état du broadcaster. Pour qu'un élément devienne un observateur, ajoutez lui un attribut `observes`. Il est analogue à l'attribut `command` utilisé pour attacher un élément à un élément `command`. Par exemple, pour qu'un bouton devienne un observateur du broadcaster décrit ci-dessus :

```
<button id="offline_button" observes="isOffline"/>
```

L'attribut `observes` a été placé sur le bouton et sa valeur a été affectée à la valeur de l'`id` du broadcaster à observer. Ici le bouton va observer le broadcaster avec l'`id` `isOffline` qui a été défini un peu plus haut dans le code. Si la valeur de l'attribut `label` sur le broadcaster change, les observateurs vont mettre à jour leur valeur de l'attribut `label` à leur tour.

Nous pourrions continuer avec des éléments supplémentaires. Autant d'éléments que vous voulez peuvent observer un simple broadcaster. Vous pouvez aussi n'en avoir qu'un seul mais cela ne servirait pas à grand chose puisque la raison principale d'utiliser les broadcasters est d'avoir des attributs transmis à de multiples endroits. Vous ne devriez utiliser les broadcasters que lorsque vous avez besoin que plusieurs éléments aient à observer un attribut. Ci-dessous quelques observateurs supplémentaires sont décrits :

```
<broadcaster id="offline_command" label="Hors ligne" accesskey="f"/>

<keyset>
  <key id="goonline_key" observes="offline_command" modifiers="accel" key="O"/>
</keyset>
<menuitem id="offline_menuitem" observes="offline_command"/>
<toolbarbutton id="offline_toolbarbutton" observes="offline_command"/>
```

Dans cet exemple, `label` et l'`accesskey` seront transmis par le broadcaster au raccourci clavier, à l'item de menu et au bouton de la barre d'outils. Le raccourci clavier n'utilisera aucun des attributs reçus, mais il sera désactivé lorsque le broadcaster le sera.

Vous pouvez utiliser un broadcaster pour observer n'importe quel attribut désiré. Les observateurs récupéreront toutes les valeurs de chaque attribut via les broadcasters si jamais ils changent. Si jamais la valeur d'un seul attribut change, les observateurs seront avisés et mettront à jour leurs valeurs afin de correspondre. Les attributs des observateurs que le broadcaster n'a pas lui même ne sont pas modifiés. Les seuls attributs qui ne sont pas modifiés sont les attributs `id` et `persist`, ces attributs ne sont jamais partagés. Vous pouvez également utiliser vos propres attributs si vous désirez.

Les broadcaster ne sont pas fréquemment utilisés, car les commandes peuvent en général convenir à la majorité des usages. Une chose à préciser est qu'il n'y a pas vraiment de différence entre l'élément `command` et l'élément `broadcaster`. Ils font tous les deux la même chose. La différence est plus sémantique. Utilisez les commandes pour les actions et utilisez les broadcasters pour les états. En fait, chaque élément peut agir comme un broadcaster, tant que vous l'observez en utilisant l'attribut `observes`.

## L'élément Observes

Il y a un moyen d'être plus spécifique quant à l'attribut du broadcaster à observer. Cela implique un élément observes. Tout comme son attribut l'indique, il vous permet d'indiquer à un élément qu'il est un observateur. L'élément observes doit être placé en tant qu'enfant de l'élément qui doit être l'observateur. Voici un exemple :

Exemple 6.7.2 :

```
<broadcasterset>
  <broadcaster id="isOffline" label="Hors ligne" accesskey="f"/>
</broadcasterset>

<button id="offline_button">
  <observes element="isOffline" attribute="label"/>
</button>
```

Deux attributs ont été ajoutés à l'élément observes. Le premier, `element`, spécifie l'identifiant du broadcaster à observer. Le second, `attribute`, spécifie l'attribut à observer. Le résultat est que le bouton recevra son libellé du broadcaster, et quand l'attribut `label` sera modifié, le libellé du bouton sera changé. L'élément observes ne change pas contrairement à l'élément qui le contient, qui est dans ce cas un button. Notez que l'attribut `accesskey` n'est pas transmis au bouton, puisque il n'est pas observé. Si vous voulez que ce soit le cas, un autre élément observes devra être ajouté. Si vous n'utilisez aucun élément observes, et qu'à la place vous utilisez l'attribut `observes` directement sur le bouton, tous les attributs seront observés.

Il existe un gestionnaire d'évènements supplémentaire, `onbroadcast`, que nous pouvons placer sur l'élément observes. L'évènement est appelé même si l'observateur détecte un changement dans l'attribut du broadcaster qu'il observe. Un exemple est décrit ci-dessous :

Exemple 6.7.3 :

```
<broadcasterset>
  <broadcaster id="colorChanger" style="color: black"/>
</broadcasterset>

<button label="Test">
  <observes element="colorChanger" attribute="style" onbroadcast="alert('La couleur a changé');"/>
</button>

<button label="Observateur"
  oncommand="document.getElementById('colorChanger').setAttribute('style','color: red');"/>
```

Deux boutons ont été créés, un nommé *Test* et l'autre *Observateur*. Si vous cliquez sur le bouton "Test", rien de spécial ne se produit. Néanmoins, si vous cliquez sur le bouton "Observateur", deux choses arrivent. Premièrement, le texte du bouton passe en rouge, deuxièmement, un message d'alerte apparaît avec le message *La couleur a changé*.

Ce qui arrive est que le gestionnaire `oncommand` du second bouton est appelé lorsque l'utilisateur appuie dessus. Le script dispose ici d'une référence au broadcaster et change le style de celui-ci afin qu'il ait une couleur (`color`) rouge. Le broadcaster n'est pas affecté par le changement de style car il n'est pas affiché à l'écran. Néanmoins, le premier bouton a un observateur qui rend compte du changement de style. Les attributs `element` et `attribute` sur la balise observes détecte le changement de style. Le style est appliqué automatiquement au premier bouton.

Ensuite, puisque la transmission se fait, le gestionnaire d'évènement `onbroadcast` est appelé. Il en résulte

l'affichage d'un message d'alerte. Notez que la transmission ne se fait que si l'attribut de style de l'élément broadcaster change. Changer le style du bouton directement ne déclenchera pas la diffusion et le message d'alerte ne s'affichera pas.

Si vous avez essayé de dupliquer le code du premier bouton (button) plusieurs fois, vous verriez une série de messages d'alerte, un pour chaque bouton, car chaque bouton est un observateur et sera prévenu du changement de style.

---

Nous verrons dans la section suivante l'utilisation du Modèle Objet de Document (DOM) avec les éléments XUL.

# 7. Modèle Objet de Document (DOM)

## 7.1 Document Object Model

Écrit par Neil Deakin. Traduit par *Chaddai Fouché* (19/07/2004), mise à jour par Julien Appert (17/06/2005) .

Page originale : <http://www.xulplanet.com/tutorials/xultu/dom.html>

Le Document Object Model (DOM, modèle objet d'un document) peut être utilisé pour obtenir des informations à propos d'éléments XUL ou les modifier.

### Introduction au DOM

Le DOM est utilisé pour stocker l'arbre des noeuds XUL. Quand un fichier XUL est chargé, les balises sont interprétées et converties dans une structure hiérarchique de noeuds du document, un pour chaque balise et bloc de texte. La structure DOM peut être examinée et modifiée en utilisant des méthodes diverses fournies à cette fin. Des éléments XUL spécifiques fournissent également des fonctions additionnelles pouvant être utilisées.

Chaque fichier XUL chargé aura son propre document affiché dans une fenêtre ou un cadre. Bien qu'il ne puisse y avoir qu'un document associé à une fenêtre à un moment donné, vous pouvez charger des documents additionnels en utilisant diverses méthodes.

Dans Mozilla, on peut accéder au DOM et le manipuler en utilisant JavaScript. Les divers objets DOM possèdent des fonctions accessibles par script, pourtant, il est important de noter que le DOM est une API qui est accessible par JavaScript. JavaScript lui-même n'est jamais qu'un langage de script pouvant accéder à ces objets parce que Mozilla fournit ces objets à l'utilisation.

Dans JavaScript, il y a un unique objet global qui est toujours disponible. Vous pouvez vous référer aux propriétés et méthodes de l'objet global sans avoir à les qualifier avec un objet. Par exemple, si l'objet global possède une propriété `name`, vous pouvez changer le nom avec le code `name=7` sans avoir à spécifier un objet à utiliser. Dans un contexte de navigateur, la fenêtre est l'objet global, et c'est également vrai pour le XUL. Naturellement, l'objet global sera différent pour chaque fenêtre. Chaque cadre aura également un objet `window` séparé.

On se réfère souvent à la fenêtre en utilisant la propriété `window`, bien que ce soit optionnel. Quelquefois, cette pratique sert uniquement à clarifier la portée de la méthode à laquelle vous vous référez. Par exemple, les deux lignes suivantes qui ouvrent une nouvelle fenêtre sont fonctionnellement équivalentes :

```
window.open( "test.xul", "_new" );  
open( "test.xul", "_new" );
```

Quand vous déclarez une fonction ou une variable tout en haut d'un script, en dehors d'une autre fonction, vous êtes en train de déclarer une propriété de l'objet global. En XUL, chaque fonction que vous déclarez sera définie comme une propriété de l'objet `window`. Par exemple, le code suivant affichera deux fois le texte *message* dans une alerte.

```
function getText()  
{  
    return "Message";  
}  
  
alert(getText());  
alert(window.getText());
```

Ainsi, si vous voulez accéder à des variables ou appeler une fonction déclarée dans un script utilisé par une autre fenêtre, vous pouvez y accéder juste en utilisant l'objet `window` de l'autre fenêtre. Par exemple, si nous avons combiné les deux derniers exemples dans un seul fichier, nous pourrions appeler la fonction `getText` au sein de l'autre fenêtre (la fenêtre `test.xul`). Pour cela, nous pouvons faire la chose suivante :

```
alert(window.opener.getText());
```

Chaque fenêtre possède une propriété `opener` contenant l'objet `window` l'ayant ouverte. Dans cet exemple, nous récupérons la fenêtre responsable de l'ouverture et appelons la fonction `getText` déclarée dans un script situé dans celle-ci. Notez que nous qualifions la propriété avec l'identifiant `window` uniquement pour plus de clarté.

La méthode `open` de la fenêtre retourne également une référence à la nouvelle fenêtre, ainsi vous pouvez appeler des fonctions de la nouvelle fenêtre à partir de l'ouvrante. Toutefois, il est important de noter que la méthode `open` renvoie sa valeur de retour avant que la fenêtre soit complètement chargée, donc les fonctions ne seront pas forcément disponibles pour autant.

L'objet `window` n'est défini par aucune spécification DOM, mais est quelquefois considéré, dans Mozilla, comme faisant parti du DOM niveau 0, un nom utilisé par des développeurs pour se référer aux fonctions assimilées DOM avant que celles-ci ne soient ajoutées aux spécifications. Le document actuel affiché dans une fenêtre peut être récupéré en utilisant la propriété `window` du document. Depuis qu'elle est devenue la propriété de la fenêtre la plus couramment utilisée, la propriété `document` est habituellement utilisée sans le qualifieur `window`.

Mozilla fournit divers objets de document en fonction du type de celui-ci. Les trois documents principaux sont les HTMLDocument, XMLDocument et XULDocument, respectivement pour les documents HTML, XML et XUL. Évidemment, c'est ce dernier type de document qui est utilisé pour le XUL. Les trois types de document sont très similaires, en fait ils partagent tous la même implémentation de base. Pourtant, il y a de nouvelles fonctions qui sont spécifiques à chacun des documents.

## Récupérer des éléments

La méthode la plus courante pour récupérer un élément dans un document est de lui donner un attribut `id` et d'utiliser la méthode `getElementById` du document. Nous avons ajouté l'attribut `id` à un certain nombre d'éléments dans la boîte de recherche de fichiers. Par exemple, nous pouvons obtenir l'état de la case à cocher en utilisant le code ci-dessous :

```
var state = document.getElementById('casecheck').checked;
```

La valeur `casecheck` correspond à l'`id` de la case à cocher définissant la sensibilité à la casse. Une fois que nous savons si elle est cochée ou non, nous pouvons utiliser cette indication pour effectuer la recherche. Nous pourrions procéder de façon similaire pour l'autre case à cocher, ou n'importe quel autre élément qui a un attribut `id`. Nous aurons besoin de récupérer le texte dans le champ de saisie par exemple.

Il n'est pas nécessaire d'afficher la barre de progression et l'arbre de données vide quand la boîte de dialogue pour la recherche de fichiers est affichée pour la première fois. Celles-ci ont été ajoutées de façon à ce que nous puissions les voir. Retirons les maintenant, et affichons les lorsque le bouton Rechercher est pressé. Tout d'abord, nous devons les rendre invisible au départ. L'attribut `hidden` est utilisé pour contrôler si un élément est visible ou pas.

Nous allons modifier la barre de progression de façon à ce qu'elle soit cachée au départ. Nous allons aussi lui ajouter un attribut `id` pour nous permettre de nous y référer dans un script, pour la cacher ou l'afficher. Tant que nous y sommes, cachons aussi le séparateur et l'arbre des résultats puisque nous n'avons besoin d'eux qu'après avoir effectué une recherche.

```

<tree id="results" hidden="true" flex="1">
  .
  .
  .
<splitter id="splitbar" resizeafter="grow" hidden="true"/>

<hbox>

  <progressmeter id="progmeter" value="50%"
    style="margin: 4px;" hidden="true"/>

```

Nous avons ajouté l'attribut `hidden` et mis sa valeur à `true`. L'élément est ainsi caché lors de sa première apparition.

Ensuite, ajoutons une fonction qui sera appelée quand le bouton Rechercher sera pressé. Nous mettrons les scripts dans un fichier séparé `findfile.js`. Dans la section précédente, nous avons ajouté l'élément `script` dans le fichier XUL. Si vous ne l'avez pas encore fait, faites le maintenant, comme ci-dessous. Nous ajouterons aussi un gestionnaire `oncommand` au bouton Rechercher.

```

<script src="findfile.js"/>
  .
  .
  .
<button id="find-button" label="Find"
  oncommand="doFind();" />

```

À présent, créez un autre fichier nommé `findfile.js` dans le même répertoire que `findfile.xul`. Nous ajouterons la fonction `doFind()` dans ce fichier. La balise `script` peut contenir du code directement à l'intérieur. Cependant, pour diverses raisons, notamment pour de meilleures performances, vous devriez toujours mettre vos scripts dans des fichiers séparés, excepté pour les courts morceaux de code qui peuvent se trouver directement dans les gestionnaires d'évènement.

```

function doFind()
{
  var meter = document.getElementById('progmeter');
  meter.hidden = false;
}

```

Cette fonction récupère d'abord une référence sur la barre de progression en utilisant son `id`, `progmeter`. La seconde ligne du corps de la fonction change l'état de `hidden` pour rendre l'élément visible à nouveau.

Finalement, ajoutons une boîte de dialogue qui affiche ce que nous sommes en train de rechercher. Évidemment nous n'en voudrions pas dans la version finale, mais ajoutons la maintenant pour nous assurer que quelque chose se produise.

```

function doFind()
{
  var meter = document.getElementById('progmeter');
  meter.hidden = false;
  var searchtext=document.getElementById('find-text').value;
  alert("Recherche de \""+searchtext+"\"");
}

```

Maintenant, avec cette boîte d'alerte placée ici, nous saurons ce qui se produit quand nous cliquons sur le bouton Rechercher. Nous pouvons ajouter du code pour obtenir aussi ce qui est sélectionné dans les listes déroulantes.

## Le DOM des éléments XUL

Chaque élément XUL possède un lot d'attributs, un lot de propriétés et un lot d'enfants. Les attributs sont déclarés dans la source, par exemple, `flex="1"` est un attribut `flex` déclaré avec la valeur `1`. Les propriétés sont disponibles en JavaScript en utilisant la syntaxe du point. Par exemple, `element.hidden` se réfère à la propriété `hidden` d'un élément. Les enfants sont les balises filles de l'élément et seront imbriqués à l'intérieur de l'élément dans la source. Il est possible de manipuler dynamiquement les attributs, propriétés et enfants d'un élément en utilisant les méthodes du DOM.

Il est important de noter que les attributs et les propriétés sont des choses différentes. Tout simplement car le fait qu'un attribut avec un nom donné existe ne signifie pas qu'il existe une propriété correspondante avec le même nom. Pourtant, c'est souvent le cas. Par exemple, pour obtenir le `flex` d'un élément, vous pouvez utiliser la propriété `flex`. Dans ce cas, le code implicite retourne simplement la valeur de l'attribut. Pour autres propriétés, XUL accomplira des calculs plus complexes.

Vous pouvez manipuler les attributs d'un élément en utilisant l'une des méthodes suivantes :

```
getAttribute( nom )
    Renvoie la valeur de l'attribut dont le nom est donné
hasAttribute( nom )
    Renvoie true si l'attribut dont le nom est donné a une valeur
setAttribute( nom , valeur )
    Fixe la valeur de l'attribut dont le nom est donné à la valeur donnée
removeAttribute( name )
    Supprime l'attribut dont le nom est donné
```

Ces fonctions vous permettent d'obtenir ou de modifier la valeur d'un attribut à tout moment. Par exemple, pour utiliser la valeur de l'attribut `flex`, vous pourriez utiliser le code suivant :

```
var box = document.getElementById('uneboite');
var flex = box.getAttribute("flex");

var box2 = document.getElementById('uneautreboite');
box2.setAttribute("flex", "2");
```

Pourtant, l'attribut `flex` a une propriété de script correspondante pouvant être utilisée à la place. Ce n'est pas plus efficace, mais c'est légèrement plus court à écrire. L'exemple suivant fait la même chose qu'au dessus, en utilisant la propriété `flex` à la place :

```
var box = document.getElementById('uneboite');
var flex = box.flex;

var box2 = document.getElementById('uneautreboite');
box2.flex = 2;
```

Une fois que vous avez une référence à un élément, vous pouvez appeler les propriétés de cet élément. Par exemple, pour obtenir la propriété `hidden` d'un élément, vous pouvez utiliser la syntaxe `element.hidden` où "element" est une référence à l'élément. Notez que la plupart des propriétés listées dans la référence est en corrélation avec les attributs communs des éléments. Il y a des différences, bien sûr, par exemple, alors que `getAttribute("hidden")` retournera la chaîne `"true"` pour un élément caché, la propriété `hidden` retournera une valeur `true` booléenne. Dans ce cas, la conversion du type est faite pour vous, donc la propriété est plus commode.

Comme pour chaque document, l'objet `element` pour les éléments XUL n'est pas le même que pour des éléments HTML et XML. Chaque élément XUL implémente l'interface [XULElement](#). Un élément XUL est un élément déclaré avec l'espace de nommage (namespace) XUL. Ainsi, les éléments XUL auront cette

interface même s'ils sont ajoutés à d'autres documents XML, et les éléments non-XUL ne l'auront pas. L'interface XULElement possède un certain nombre de propriétés et méthodes spécifiques aux éléments XUL, pour beaucoup héritées de l'interface générique des éléments DOM.

Un espace de nommage est un URI qui spécifie le type d'élément. Voici quelques exemples :

```
<button xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"/>
<button xmlns="http://www.w3.org/1999/xhtml" />
<html:button xmlns:html="http://www.w3.org/1999/xhtml" />
<html:button xmlns:html="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" />
```

Les espaces de nommages sont spécifiés en utilisant l'attribut `xmlns`. Le premier bouton est un élément XUL qui a été placé dans l'espace de nommage XUL. Le second élément est un élément XHTML auquel on a donné l'espace de nommage XHTML. Vous pouvez également utiliser la syntaxe en préfixe avec une colonne pour utiliser un espace de nommage spécifique. On y a recours lorsqu'on utilise plusieurs espaces de nommage dans un document et que l'on désire préciser quel espace de nommage est actuellement utilisé. Dans le troisième exemple, le préfixe "html" est donné à l'espace de nommage "http://www.w3.org/1999/xhtml". Dans ce cas, un bouton XHTML est produit. Le quatrième bouton est un peu confus, mais il faut préciser que c'est l'URI qui est important et non le préfixe. C'est une distinction importante. En fait, le texte utilisé pour le préfixe n'est pas significatif lorsqu'il détermine quel type d'élément est utilisé.

Le DOM fournit un certain nombre de fonctions relatives aux espaces de nommage similaires aux fonctions de base. Par exemple, la fonction `getAttributeNS` est similaire à la fonction `getAttribute`, excepté un argument supplémentaire pouvant être fourni pour spécifier un attribut dans un espace de nommage spécifique.

Quelques éléments XUL disposent de leurs propres propriétés qui leur sont spécifiques. Reportez vous à la [référence](#) pour un guide complet des attributs et propriétés disponibles pour un élément.

## Naviguer dans le DOM

Le DOM est une structure en arbre composé d'un unique noeud racine avec ses enfants. Vous pouvez obtenir une référence au noeud racine en utilisant la propriété `documentElement` du document. Le noeud racine est toujours un élément, mais ce n'est pas le cas pour tous les noeuds de l'arbre. Un élément correspond à une balise dans la source XUL, mais vous pouvez également trouver des noeuds de texte, des noeuds de commentaire et quelques autres types dans un arbre de document. Dans le cas de XUL, l'élément racine sera la balise `window` dans le document XUL. Chaque noeud de l'arbre peut avoir des enfants et ces enfants peuvent avoir des noeuds fils à leur tour. Comme le DOM est une structure en arbre, vous pouvez naviguer au sein de cet arbre en utilisant une grande variété de propriétés. Quelques méthodes les plus communes sont listées ci-après :

*firstChild*

Référence au premier noeud fils d'un élément

*lastChild*

Référence au dernier noeud fils d'un élément

*childNodes*

Contient la liste des enfants d'un élément

*parentNode*

Référence au père d'un noeud

*nextSibling*

Référence au prochain noeud de même niveau

*previousSibling*

Référence au noeud précédent de même niveau



Ces propriétés vous permettent de naviguer au sein d'un document de diverses manières. Par exemple, vous pourriez obtenir un premier enfant d'un élément en utilisant la propriété `firstChild` et ensuite, naviguer au sein de tous ses enfants en utilisant la propriété `nextSibling`. Ou, vous pourriez accomplir la même chose en parcourant les items du tableau `childNodes` listant tous les enfants. Dans Mozilla, la dernière méthode est plus efficace.

L'exemple suivant montre comment parcourir tous les enfants du noeud racine :

```
var childNodes = document.documentElement.childNodes;
for (var i = 0; i < childNodes.length; i++) {
    var child = childNodes[i];
    // faire quelque chose avec child
}
```

La variable *childNodes* contiendra les enfants de l'élément racine du document. Nous pouvons donc utiliser une boucle `for` pour parcourir les enfants, en accédant à chaque item comme pour un tableau.

---

Dans la prochaine section, nous découvrirons comment modifier le DOM.

## 7.2 Modification d'une interface XUL

Écrit par Neil Deakin. Traduit par *Alain B. et Paul Rouget* (01/07/2005).  
 Page originale : <http://www.xulplanet.com/tutorials/xultu/dommodify.html>

Le DOM fournit pleins de fonctions variées pour modifier un document.

### Création de nouveaux éléments

Vous pouvez créer de nouveaux éléments en utilisant la fonction `createElement` du document. Elle ne prend qu'un argument, le nom de la balise de l'élément à créer. Vous pouvez ensuite lui définir des attributs en utilisant la fonction `setAttribute` et ajouter cet élément au document XUL grâce à la fonction `appendChild`. L'exemple suivant ajoute un bouton à une fenêtre XUL :

Exemple 7.2.1 :

```
<script>
function addButton()
{
    var aBox = document.getElementById("aBox");

    var button = document.createElement("button");
    button.setAttribute("label", "Un bouton");
    aBox.appendChild(button);
}
</script>

<box id="aBox" width="200">
    <button label="Ajouter" onclick="addButton();" />
</box>
```

Le script récupère d'abord une référence de la boîte qui est le container dans lequel le nouveau bouton sera ajouté. La fonction `createElement` crée un nouveau bouton. Nous assignons un libellé *Un bouton* à ce bouton avec la fonction `setAttribute`. La fonction `appendChild` de la boîte est appelée pour lui ajouter le bouton.

La fonction `createElement` va créer l'élément type par défaut du document. Pour des documents XUL, il sera généralement question de création d'éléments XUL. Pour un document HTML, un élément HTML sera

créé, et donc, il aura les fonctionnalités et les fonctions d'un élément HTML à la place. La fonction `createElementNS` peut être utilisée pour créer des éléments dans un espace de nommage différent.

La fonction `appendChild` est utilisée pour ajouter un élément en tant qu'enfant d'un autre élément. Il existe trois fonctions associées qui sont les fonctions `insertBefore`, `replaceChild`, et `removeChild`. Leur syntaxe est la suivante :

```
parent.appendChild(child);
parent.insertBefore(child, referenceChild);
parent.replaceChild(newChild, oldChild);
parent.removeChild(child);
```

Le nom de ces fonctions suffit à comprendre ce qu'elles font. La fonction `insertBefore` insère un nouveau noeud enfant avant un autre existant. Elle est utilisée pour réaliser une insertion au milieu d'une série d'enfants plutôt qu'à la fin comme le fait la fonction `appendChild`. La fonction `replaceChild` efface un enfant existant et en ajoute un nouveau à sa place et à la même position. Pour finir, la fonction `removeChild` supprime un noeud.

Notez que pour toutes ces fonctions, l'enfant de référence ou l'enfant à supprimer doit exister sinon une erreur sera générée.

Il est fréquent que vous vouliez effacer un élément existant et l'ajouter autre part. Dans ce cas, vous pouvez simplement ajouter l'élément sans l'effacer préalablement. Puisqu'un noeud ne peut exister qu'à un seul emplacement à la fois, le mécanisme d'insertion se chargera toujours d'effacer le noeud d'abord de son emplacement initial. C'est une méthode pratique pour déplacer un noeud dans un document.

Toutefois, pour copier un noeud, vous devrez appeler la fonction `cloneNode`. Cette fonction réalise une copie d'un noeud existant, ce qui vous permet ensuite de l'ajouter autre part. Le noeud original restera à sa place. Elle prend un argument booléen indiquant si elle doit copier tous les noeuds enfants ou non. Si la valeur est *false*, seul le noeud est copié, comme s'il n'avait jamais eu aucun enfant. Si la valeur est *true*, tous les enfants sont également copiés. La copie est faite récursivement, donc pour de larges structures d'arbres, assurez vous de vouloir réellement passer cet argument *true* à la fonction `cloneNode`. Voici un exemple :

Exemple 7.2.2 :

```
<hbox height="400">
  <button label="Copier"
    oncommand="this.parentNode.appendChild(this.nextSibling.cloneNode(true));"/>

  <vbox>
    <button label="Premier"/>
    <button label="Deuxième"/>
  </vbox>
</hbox>
```

Lorsque le bouton *Copier* est appuyé, nous récupérerons l'élément voisin suivant de même niveau que le bouton, ce qui dans ce cas est l'élément vbox. Une copie de cet élément est effectuée en utilisant la fonction `cloneNode` et la copie est ajoutée à la fin.

Vous noterez que certains éléments, tels que listbox et menulist disposent de fonctions de modification spécialisées supplémentaires que vous devriez utiliser à la place lorsque vous le pouvez. Elles seront décrites dans une prochaine section.

## Manipulation d'éléments basiques

Les éléments principaux de XUL, tels que les boutons, les cases à cocher et les boutons radios, peuvent être

manipulés grâce à de nombreuses propriétés de script. Les propriétés disponibles sont listées sur la page [référence des éléments](#) car celles disponibles varient selon les éléments. Les propriétés communes que vous pouvez manipuler sont `label`, `value`, `checked` et `disabled`. Elles affectent ou effacent les attributs correspondants si nécessaire. Voici un exemple simple de changement d'un libellé sur un bouton :

Exemple 7.2.3 :

```
<button label="Bonjour" oncommand="this.label = 'Aurevoir';"/>
```

Lorsque le bouton est pressé, son libellé est modifié. Cette technique fonctionne pour une large majorité d'éléments ayant des libellés. Pour les champs de saisie, vous pouvez faire quelque chose de similaire pour sa propriété `value`.

Exemple 7.2.4 :

```
<button label="Ajouter" oncommand="this.nextSibling.value += '1';"/>
<textbox/>
```

Cet exemple ajoute un `1` dans le champ de saisie à chaque fois que le bouton est pressé. La propriété `nextSibling` permet d'atteindre l'élément suivant le bouton (`this`), le champ de saisie `textbox`. L'opérateur `+=` sert à ajouter un `1` à la fin du texte de la valeur courante. Notez que vous pouvez encore ajouter du texte dans ce champ de saisie. Vous pouvez récupérer le libellé courant ou la valeur en utilisant ses propriétés, comme dans l'exemple suivant :

Exemple 7.2.5 :

```
<button label="Bonjour" oncommand="alert(this.label);"/>
```

Les cases à cocher disposent d'une propriété `checked` qui sert à cocher ou à décocher une case. Il est facile de comprendre son usage. Dans l'exemple à suivre, nous inversons l'état de la propriété `checked` à chaque fois que le bouton est pressé. Tandis que les libellés et les valeurs sont des chaînes de caractères, vous noterez que la propriété `checked` est un booléen qui prend une valeur *true* ou *false*.

Exemple 7.2.6 :

```
<button label="Changer" oncommand="this.nextSibling.checked = !this.nextSibling.checked;"/>
<checkbox label="Cochez pour les messages"/>
```

Les boutons radios peuvent également être sélectionnés en utilisant les propriétés, toutefois, un seul est sélectionné à la fois dans un groupe, tous les autres étant décochés et un seul coché. Vous n'avez pas à réaliser cette gestion manuellement. La propriété `selectedIndex` du `radiogroup` peut être utilisée pour cela. La propriété `selectedIndex` sert à récupérer l'index du bouton radio sélectionné dans le groupe et également à le modifier.

Il est habituel de désactiver des champs particuliers qui ne servent pas dans une situation donnée. Par exemple, dans la boîte de dialogue des préférences, vous avez le choix entre plusieurs possibilités, mais seul un choix permet un paramétrage supplémentaire. Voici un exemple de création de ce type d'interface :

Exemple 7.2.7 :

```
<script>
function updateState()
{
    var name = document.getElementById("name");
    var sindex = document.getElementById("group").selectedIndex;
    if (sindex == 0) name.disabled = true;
    else name.disabled = false;
}
```

```

}
</script>

<radiogroup id="group" onselect="updateState();">
  <radio label="Nom aléatoire" selected="true"/>
  <hbox>
    <radio label="Spécifiez un nom :"/>
    <textbox id="name" value="Alain" disabled="true"/>
  </hbox>
</radiogroup>

```

Dans cet exemple, une fonction `updateState` est appelée à chaque fois qu'un évènement de sélection est déclenché depuis le groupe de boutons radios. Elle est exécutée lorsque qu'un bouton radio est sélectionné. La fonction retournera l'élément radio actuellement sélectionné en utilisant la propriété `selectedIndex`. Vous noterez que bien qu'un bouton radio se trouve à l'intérieur d'une boîte hbox, il reste attaché au groupe radio. Si le premier bouton radio est sélectionné (index de 0), le champ de saisie est désactivé en définissant sa propriété `disabled` à *true*. Si le second bouton radio est sélectionné, le champ de saisie est activé.

---

La section suivante fournira plus de détails sur la manipulation des groupes de boutons radios et la manipulation des listes.

## 7.3 Manipulation de listes

Écrit par Neil Deakin. Traduit par **Romain D.** (25/04/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/domlists.html>

La boîte de liste XUL fournit un certain nombre de méthodes spécialisées.

### Manipulation d'une liste

L'élément listbox fournit de nombreuses méthodes pour rechercher et manipuler ses items. Bien que les boîtes de liste puissent être manipulées en utilisant les fonctions standard de DOM, il est recommandé que les fonctions spécialisées de listbox soient employées autant que possible. Ces fonctions sont plus simples et feront correctement leur travail.

La fonction `appendItem` est utilisée pour ajouter un nouvel item à la fin d'une liste. Elle est similaire à la fonction `appendChild` de DOM sauf qu'elle prend un libellé, et que vous n'avez pas à vous soucier où placer votre item dans la structure de la liste. Voici un exemple :

Exemple 7.3.1 :

```

<script>
function addItem()
{
  document.getElementById('laliste').appendItem("Jeudi", "jeu");
}
</script>

<listbox id="laliste"/>

<button label="Ajouter" oncommand="addItem();"/>

```

La fonction `appendItem` prend deux arguments, le libellé, dans l'exemple *Jeudi*, et une valeur *jeu*. Les deux arguments correspondent aux attributs `label` et `value` dans l'élément listitem. L'attribut `value` est optionnel et sert à affecter à un item une valeur que vous pouvez réutiliser ensuite dans un script.

De même, il existe les fonctions `insertItemAt` et `removeItemAt`, qui respectivement, ajoutent un nouvel item et suppriment un item existant. La syntaxe est la suivante :

```
list.insertItemAt(3, "Jeudi", "jeu");
list.removeItemAt(3);
```

La fonction `insertItemAt` prend un argument supplémentaire, la position pour insérer le nouvel item. Le nouvel item est inséré à cet index, ainsi dans l'exemple, le nouvel item sera ajouté à la position 3 et l'item qui avait cette position aura maintenant à la position 4. Rappelez-vous que le premier item est 0. La fonction `removeItemAt` supprimera l'item à un index spécifique.

Ces trois méthodes sont également disponibles pour plusieurs autres éléments XUL et fonctionnent de la même manière. En fait, ces méthodes font parties de l'interface `nsIDOMXULSelectControlElement` donc tous les éléments XUL qui implémentent cette interface ont ces méthodes. Les éléments `menulist`, `radiogroup` et `tabs` en font partie. Par exemple, pour ajouter un nouvel item à un `menulist`, vous pouvez employer la même syntaxe qu'une `listbox`. Le bon type d'élément sera ajouté dans chaque cas.

## Sélection de liste

L'interface `nsIDOMXULSelectControlElement` fournit deux propriétés supplémentaires, `selectedIndex` et `selectedItem`. La première renvoie l'index de l'item sélectionné tandis que la deuxième renvoie l'élément sélectionné. Par exemple, la valeur de retour de `selectedItem` sera le `menuitem` sélectionné. Si aucun item n'est sélectionné, `selectedIndex` retournera `-1`, et `selectedItem` renverra `null`.

Ces deux propriétés sont généralement inspectées durant un évènement de sélection, comme dans l'exemple suivant :

Exemple 7.3.2 :

```
<listbox id="thelist" onselect="alert(this.selectedItem.label);">
  <listitem label="Petit"/>
  <listitem label="Moyen"/>
  <listitem label="Grand"/>
</listbox>
```

L'évènement de sélection est exécuté par une `listbox` quand un item de la liste est sélectionné. Le gestionnaire affiche ici une alerte contenant le libellé de l'item sélectionné dans la liste. Puisque l'évènement de sélection s'est exécuté, nous pouvons supposer qu'un item est sélectionné. Dans d'autres cas, vous devrez vous assurer que `selectedItem` n'est pas nul avant de poursuivre.

L'évènement de sélection est également exécuté quand un bouton radio dans un `radiogroup` est sélectionné et quand un onglet est sélectionné dans l'élément `tabs`. Cependant, les `menulists` ne génère pas d'évènement de sélection ; vous pouvez écouter l'évènement "command" à la place pour traiter la sélection d'un item.

Pour l'élément `tabs`, il est souvent plus commode d'employer les fonctions de l'élément `tabbox`. Il a aussi une fonction `selectedIndex` qui renverra l'index de l'onglet sélectionné. Cependant, pour récupérer l'item sélectionné, utilisez plutôt la fonction `selectedTab` de `tabbox`. Ou alors, utilisez la fonction `selectedPanel` pour récupérer la page d'onglet sélectionnée, ce qui renvoie le contenu associé à l'onglet.

Toutes les propriétés de sélection décrites ci-dessus peuvent également se voir assignées une nouvelle valeur pour changer la sélection. Dans l'exemple suivant, la propriété `selectedIndex` de l'élément `radiogroup` est changée avec la valeur entrée dans un champ de saisie. Ce code n'est cependant pas performant ; par exemple il ne vérifie pas si la valeur entrée est hors limite. Il est conseillé d'ajouter de genre

de vérification d'erreur.

#### Exemple 7.3.3 :

```
<script>
function doSelect()
{
    var val = document.getElementById('number').value;
    val = Number(val);
    if (val != null)
        document.getElementById('level').selectedIndex = val - 1;
}
</script>

<hbox align="center">
    <label value="Entrez un nombre compris entre 1 et 3 :"/>
    <textbox id="number"/>
    <button label="Sélectionnez" oncommand="doSelect();" />
</hbox>

<radiogroup id="level">
    <radio label="Excellent"/>
    <radio label="Bon"/>
    <radio label="Mauvais"/>
</radiogroup>
```

Les boîtes de liste supportent aussi les sélections multiples et les fonctions de l'interface [nsIDOMXULMultiSelectControlElement](#). Cette interface fournit un certain nombre de fonctions dédiées pour contrôler la sélection multiple. Par exemple, la propriété `selectedItems` contient une liste des items qui sont sélectionnés, alors que la propriété `selectedCount` contient le nombre d'items sélectionnés. En général, vous utiliserez ces propriétés pour parcourir la liste et y effectuer quelques opérations pour chaque item. Faites attention lorsque vous parcourez les items sélectionnés de la liste ; si vous modifiez les items dans la liste pendant que vous les parcourez, la liste sera modifiée et les propriétés de sélection pourraient retourner des valeurs différentes. C'est une raison pour laquelle il est utile de manipuler la liste par item plutôt que par l'index.

L'exemple suivant montre une méthode correcte de suppression des items sélectionnés :

#### Exemple 7.3.4 :

```
<script>
function deleteSelection()
{
    var list = document.getElementById('thelist');
    var count = list.selectedCount;
    while (count--){
        var item = list.selectedItems[0];
        list.removeItemAt(list.indexOfItem(item));
    }
}
</script>

<button label="Supprimer" oncommand="deleteSelection();" />

<listbox id="thelist" seltype="multiple">
    <listitem label="Cheddar"/>
    <listitem label="Cheshire"/>
    <listitem label="Edam"/>
    <listitem label="Gouda"/>
    <listitem label="Havartie"/>
</listbox>
```

À l'intérieur de la boucle `while`, nous récupérons d'abord l'item sélectionné à l'index 0. Le premier item sélectionné est toujours recherché car la taille du tableau diminuera au fur et à mesure que les items seront supprimés. Après, nous supprimons l'item en utilisant la fonction `removeItemAt`. Comme cette fonction nécessite un index, nous pouvons faire correspondre un item et son index en utilisant la fonction `getIndexOfItem`. La fonction inverse correspondante est `getItemAtIndex`.

L'interface `nsIDOMXULMultiSelectControlElement` fournit également des méthodes pour modifier les items sélectionnés. Par exemple, la fonction `addItemToSelection` ajoute un nouvel item à la liste des items sélectionnés, sans vider la sélection existante. La fonction `removeItemFromSelection` supprime un seul item dans la sélection.

## Défilement de liste

Si la boîte de liste contient plus de lignes qu'elle ne peut en afficher, une barre de défilement apparaîtra pour permettre à l'utilisateur de faire défiler la liste. La position du défilement peut être ajustée en utilisant quelques méthodes de `listbox`.

La méthode `scrollToIndex` fait défiler jusqu'à une ligne donnée. Cette boîte de liste défilera jusqu'à ce que la ligne soit la première ligne visible, à moins que la ligne ne soit proche de la fin de la liste des items. La méthode `ensureIndexIsVisible` est similaire puisqu'elle fait défiler la liste pour afficher une ligne, mais cette méthode ne défilera pas si l'item est déjà visible. Ainsi, la première fonction est utilisée pour faire défiler jusqu'à une ligne précise alors que la deuxième est utilisée pour être sûr que la ligne est visible. Il y a également `ensureItemIsVisible` qui nécessite un item en argument au lieu d'un index. Comparez l'effet de ces deux fonctions à des positions de défilement différentes dans cet exemple :

Exemple 7.3.5 :

```
<button label="scrollToIndex"
        oncommand="document.getElementById('thelist').scrollToIndex(4);" />
<button label="ensureIndexIsVisible"
        oncommand="document.getElementById('thelist').ensureIndexIsVisible(4);" />

<listbox id="thelist" rows="5">
  <listitem label="1"/>
  <listitem label="2"/>
  <listitem label="3"/>
  <listitem label="4"/>
  <listitem label="5"/>
  <listitem label="6"/>
  <listitem label="7"/>
  <listitem label="8"/>
  <listitem label="9"/>
  <listitem label="10"/>
  <listitem label="11"/>
  <listitem label="12"/>
</listbox>
```

---

Nous verrons ensuite les objets boîtes XUL.

## 7.4 Les objets boîtes

Écrit par Neil Deakin. Traduit par *Alain B.* (06/05/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/boxobject.html>

Cette section décrit l'objet boîte qui contient des informations relatives à l'affichage et à la mise en page concernant une boîte XUL aussi bien que des détails sur la mise en page XUL.

## À propos de la mise en page XUL

Mozilla divise son fonctionnement en deux séries d'arbres, l'arbre du contenu et l'arbre de mise en page. L'arbre de contenu stocke les noeuds trouvés tels quels dans le code source. L'arbre de mise en page contient un arbre différent des noeuds pour chaque composant individuel pouvant être affiché. L'arbre de mise en page contient la structure d'affichage des noeuds. Il n'y a pas forcément de relation un à un entre le contenu et la mise en page des noeuds. Certains noeuds de contenu peuvent avoir plusieurs objets de mise en page, par exemple, chaque ligne d'un paragraphe a un objet de mise en page séparé. Réciproquement, certains noeuds de contenu n'ont aucun objet de mise en page. Par exemple, l'élément `key` n'a aucun objet de mise en page puisqu'il n'est jamais affiché. De même, tout élément qui aurait été masqué n'aura non plus aucun objet de mise en page.

DOM est généralement utilisé pour récupérer et modifier des informations concernant le contenu ou la structure d'un document. Il est relativement simple de déterminer quelle sorte de noeud de l'arbre sera créée pour un élément donné. Un élément XUL, par exemple, aura un type de noeud de contenu `XULElement`.

Les objets de mise en page qui seront créés sont déterminés de manière plus complexe. Diverses parties d'information sont utilisées telles que le nom de la balise, les attributs d'un élément, diverses propriétés CSS, les éléments et les objets de mise en page autour de l'élément, et les liaisons XBL associées avec un élément (les XBL seront décrites dans une section ultérieure). À moins que vous ne changiez le style d'un élément, la plupart des éléments XUL utilisent habituellement l'objet de boîte de mise en page ou un de ses sous-types. Souvenez vous que tous les éléments XUL sont des types de boîtes, et que les boîtes sont la base de l'affichage de tous les éléments XUL. Toutefois, il y a un certain nombre de sous-types, environ 25 ou plus, pour des éléments XUL spécifiques. Certains de ces sous-types, tels que les piles ou les boîtes de liste ont besoin d'une mise en page plus complexe qu'une simple boîte, tandis que d'autres, tels que les boutons, ne sont utilisés que pour ajouter une gestion supplémentaire des événements pour la souris et le clavier.

L'objet de mise en page associé avec un élément peut être enlevé pour créer un objet de type complètement différent juste en changeant la propriété `display` CSS. Par exemple, l'affectation de la valeur `block` à la propriété `display` d'un élément bouton va effacer l'objet de mise en page et créer un objet `block` à la place. Naturellement, ce changement modifiera l'apparence et les fonctionnalités de cet élément.

Il n'est pas nécessaire de connaître les détails de la construction des objets de mise en page, mais il est quand même utile d'avoir la connaissance de comment est décrit la mise en page XUL pour aborder un développement avancé sous XUL.

## Les objets de boîte

Les objets de mise en page ne peuvent pas être manipulés par les développeurs. Ils font partie des composants internes à la mise en page XUL. Toutefois, XUL fournit quelques objets d'aide, appelés objets de boîte, qui fournissent quelques informations concernant la mise en page. Comme leurs noms l'indiquent, ils sont disponibles pour tous éléments de type boîte.

Il y a plusieurs sous-types d'objet boîte, bien que seul un couple d'entre eux est généralement employé. Les autres ont des fonctions qui sont plus accessibles par des méthodes liées directement sur l'élément, car ces types sont généralement seulement utilisés avec un élément particulier. L'objet de boîte, ou l'interface `BoxObject`, toutefois, a un nombre de propriétés pouvant être utile pour le développement XUL.

L'objet de boîte de base a deux fonctionnalités utiles. Premièrement, vous pouvez récupérer la position et les dimensions d'un élément XUL affiché, et deuxièmement, vous pouvez déterminer l'ordre d'affichage des éléments dans une boîte, au lieu de leurs ordres d'enregistrement dans le DOM.

L'objet de boîte fournit quatre propriétés, `x`, `y`, `width` et `height`, pour déterminer la position et les dimensions d'un élément. Les coordonnées `x` et `y` sont relatives au coin haut et gauche du document dans la



fenêtre (en excluant le bord de la fenêtre et son titre) et mesurées en pixels. Les propriétés `width` et `height` sont également mesurées en pixels et retournent la largeur et la hauteur d'un élément en incluant les styles CSS `padding` et `border`, s'ils existent.

Les valeurs sont toujours la position et les dimensions correspondant à l'affichage en cours, donc ces valeurs peuvent être différentes si l'élément est déplacé ou redimensionné. Par exemple, un élément flexible changera de dimension et les dimensions de son objet de boîte seront mises à jour en conséquence. L'exemple suivant en est une illustration :

#### Exemple 7.4.1 :

```
<button label="Cliquez ici"
        oncommand="alert('La largeur est ' + this.boxObject.width);"/>
```

Vous pouvez utiliser les attributs `width` et `height` pour définir respectivement la largeur et la hauteur d'un élément. Normalement, ces attributs ne sont pas utilisés et l'élément ajuste ses dimensions pour s'adapter au contenu. Ainsi, ces attributs remplacent la dimension par défaut par celle spécifiée. Les propriétés `width` et `height` correspondantes peuvent être employées pour ajuster les dimensions d'un élément à tout moment, si vous souhaitez afficher un élément à une dimension spécifique. La récupération des valeurs de ces propriétés vous donnera les dimensions explicitement indiquées. Notez que ces propriétés renverront une chaîne vide si les attributs ou propriétés `width` et `height` n'ont pas encore été initialisées. Du coup, vous ne pouvez pas récupérer les dimensions actuelles avec ces propriétés ; vous devrez utiliser les propriétés de l'objet de boîte à la place.

Cela doit vous sembler un peu confus, mais la clef est de se souvenir que les propriétés `width` et `height` d'un élément retournent les dimensions qui ont été définies dans le XUL, tandis que les propriétés `width` et `height` de l'objet de boîte retournent les dimensions réelles actuelles.

L'attribut `hidden` cachera un élément de telle sorte qu'il ne sera pas affiché. Comme il n'est pas affiché, les quatre propriétés de position et de dimensions de l'objet de boîte auront une valeur `0`. Lorsqu'un élément est caché, il est supprimé de l'affichage et ses objets de mise en page sont effacés. Ainsi, comme il n'est affiché nul part, il n'aura pas de position ou dimensions.

L'attribut `collapsed` aura le même effet visuel sur cet élément pour l'utilisateur, excepté qu'il laisse l'élément sur l'écran et conserve les objets de mise en page intacts, mais en mettant ses dimensions à `0`. Il signifie que même si les éléments cachés et réduits ont une largeur et une hauteur de `0`, les éléments cachés auront une position `x` et `y` de `0` tandis que les éléments réduits conserveront leur position dans la fenêtre.

Pour rechercher ou modifier les états `hidden` ou `collapsed`, utilisez leurs propriétés correspondantes comme dans l'exemple suivant.

#### Exemple 7.4.2 :

```
<script>
function showPositionAndSize()
{
    var labelbox = document.getElementById('thelabel').boxObject;

    alert("La position est (" + labelbox.x + "," + labelbox.y +
        ") et les dimensions sont (" + labelbox.width + "," +
        labelbox.height + ")");
}
</script>

<button label="Caché"
        oncommand="document.getElementById('thelabel').hidden = true;"/>
<button label="Montré"
```

```

        oncommand="document.getElementById('thelabel').hidden = false;"/>
<button label="Réduit"
        oncommand="document.getElementById('thelabel').collapsed = true;"/>
<button label="Non réduit"
        oncommand="document.getElementById('thelabel').collapsed = false;"/>
<button label="Montrer la position et les dimensions"
        oncommand="showPositionAndSize();"/>

<label id="thelabel" value="Je suis un libellé"/>

```

Notez que si vous cachez ou réduisez le libellé, il sera invisible. Vous devrez changer ses attributs `hidden` ou `collapsed` pour le voir réapparaître.

## Classement des éléments XUL

L'objet de boîte peut également être employé pour déterminer l'ordre d'affichage des éléments, qui peut ne pas être le même que dans la source. Souvenez vous que les propriétés DOM telles que `childNodes`, `firstChild` et `nextSibling` peuvent être utilisées pour parcourir l'arbre de document. L'objet de boîte permet de naviguer de façon similaire mais retourne les éléments dans leur ordre d'affichage.

L'objet de boîte fournit plusieurs propriétés, `firstChild`, `lastChild`, `nextSibling`, `previousSibling` et `parentBox`. Leurs noms devraient vous expliquer d'eux même leur fonctionnement. Ces propriétés renvoient des éléments, par exemple, la propriété `firstChild` renvoie le premier élément enfant affiché. Il n'y a pas de propriété `childNodes` correspondante pour naviguer entre les boîtes ; à la place, vous devez utiliser les propriétés `nextSibling` et `previousSibling` pour parcourir les frères et soeurs.

À la différence de naviguer à travers l'arbre DOM, les éléments cachés ne sont pas inclus dans la navigation avec les objets de boîte. Donc, une boîte avec six enfants où les deux premiers sont cachés, la propriété `firstChild` renverra le troisième élément. Toutefois, les éléments réduits seront inclus car ils sont affichés même sans avoir de dimensions. Par exemple, la boîte soeur suivant le bouton 1 de l'exemple suivant sera le bouton 3, parce que le bouton 2 est caché. Mais la boîte soeur suivant le bouton 3 sera le bouton 4 qui n'est seulement réduit.

Exemple 7.4.3 :

```

<hbox>
  <button label="Bouton 1"
    oncommand="alert('Le suivant est : ' + this.boxObject.nextSibling.label);"/>
  <button label="Bouton 2" hidden="true"/>
  <button label="Bouton 3"
    oncommand="alert('Le suivant est : ' + this.boxObject.nextSibling.label);"/>
  <button label="Bouton 4" collapsed="true"/>
</hbox>

```

Lorsqu'une boîte XUL est placée sur une fenêtre, les éléments sont ordonnés selon un certain nombre de propriétés, par exemple l'orientation, leur groupe ordinal et leur direction. L'orientation est communément modifiée en utilisant l'attribut `orient`. Il existe également une propriété CSS `-moz-box-orient` correspondante qui peut être utilisée à la place, en fonction de la situation. Cet attribut a été expliqué dans une section précédente sur les boîtes.

L'attribut ordinal d'un élément peut être placé pour spécifier un groupe ordonné d'éléments, ou vous pouvez utiliser la propriété CSS `-moz-box-ordinal-group`.

Les éléments avec une valeur ordinale inférieure sont placés dans la boîte avant les éléments ayant une plus grande valeur ordinale. Par exemple, un élément avec une valeur ordinale de 2 sera placé avant un élément

ayant une valeur ordinaire de 3 ou plus mais après un élément ayant un ordinal de 1. La valeur ordinaire par défaut, si elle n'est pas définie, est de 1. Elle signifie que si vous voulez modifier l'ordre d'affichage des éléments, vous aurez souvent besoin d'ajuster les ordinaux de plusieurs éléments.

L'ajustement de l'ordinal d'un élément ne se fait pas aussi simplement par le placement des éléments dans un certain ordre dans la source. Il peut être utilisé pour réarranger les items plus tard sans ajuster le DOM. L'exemple suivant en est une démonstration.

Exemple 7.4.4 :

```
<hbox>
  <button label="Un" oncommand="this.ordinal++;"/>
  <button label="Deux" oncommand="this.ordinal++;"/>
  <button label="Trois" oncommand="this.ordinal++;"/>
</hbox>
```

Si vous pressez le premier bouton, son ordinal augmentera de un, de 1 à 2. Il apparaîtra à la fin de la ligne puisque les autres boutons ont un ordinal de 1. Si vous pressez le second bouton, son ordinal sera augmenté de un et sera déplacé à la fin de la ligne. Les items de même ordinal apparaîtront dans le même ordre que dans la source. Si vous pressez une nouvelle fois le bouton libellé *Un*, son ordinal augmentera à 3 et il sera déplacé à la fin. Finalement, en pressant le bouton libellé *Trois* augmentera son ordinal à 2 et il apparaîtra entre les deux autres boutons.

L'attribut final d'ordonnement de boîte est l'attribut `dir`, ou la propriété CSS `–moz–box–direction`. S'il est initialisé à *reverse*, tous les enfants dans la boîte sont affichés dans l'ordre inverse. Dans une boîte horizontale, les éléments seront affichés de la droite vers la gauche et dans une boîte verticale, les éléments seront affichés du bas vers le haut. Voici un exemple :

Exemple 7.4.5 :

```
<hbox dir="reverse">
  <button label="Gauche"/>
  <button label="Centre"/>
  <button label="Droite"/>
</hbox>
```

Parcourir les noeuds en utilisant l'ordonnement des objets de boîte vous renverra les éléments dans leur ordre d'affichage en tenant compte des ajustements fait sur leur ordinaux. Ainsi, si vous changez l'ordinal d'un élément, celui-ci aura une position différente dans la boîte. Toutefois, le renversement de la direction n'affectera pas l'ordre de la boîte.

---

Ensuite, nous verrons comment utiliser les objets XPCOM à partir de XUL et des scripts.

## 7.5 Interfaces XPCOM

Écrit par Neil Deakin. Traduit par *Maximilien* (24/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xpcom.html>

Dans cette section, nous allons faire une brève présentation de XPCOM ("Modèle de composants objets multi plate-forme"), qui est le système d'objets utilisé par Mozilla.

### Appel des objets natifs

En utilisant XUL, nous pouvons construire des interfaces utilisateurs complexes. En y joignant des scripts, on peut modifier l'interface et réaliser des actions. Cependant, il y a un certain nombre de choses qui ne

peuvent pas être réalisées directement en javascript. Par exemple, si nous voulons créer une application gérant des courriels, nous avons besoin d'écrire des scripts permettant de se connecter au serveur de courriels, afin de les retirer ou d'en envoyer. Le langage Javascript ne permet pas de faire ce genre de choses.

Le seul moyen pour le faire est d'écrire du code natif implémentant ces fonctionnalités avancées. Nous avons aussi besoin d'un moyen pour pouvoir appeler ce code natif aisément à partir de nos scripts. Mozilla fournit une telle possibilité en utilisant XPCOM.

## À propos d'XPCOM

Mozilla est construit à partir d'une multitude de composants, où chacun d'eux réalise une tâche précise. Par exemple, il y a un composant pour chaque menu, bouton et élément. Ces composants sont construits à partir de plusieurs définitions appelées *interfaces*.

Une interface dans Mozilla est une définition d'un ensemble de fonctions que peuvent implémenter des composants. Les composants sont ce qui permet au code de Mozilla de réaliser des traitements. Chaque composant implémente les fonctions conforme à une interface. Un composant peut implémenter plusieurs interfaces. Et plusieurs composants peuvent implémenter la même interface.

Prenons l'exemple d'un composant de fichier. Une interface sera créée décrivant les propriétés et les fonctions que l'on veut pouvoir appliquer sur un fichier. Les propriétés seront le nom du fichier, sa date de dernière modification ou sa taille. Les fonctions permettront d'effacer, de déplacer ou de copier le fichier.

L'interface "Fichier" décrit uniquement les caractéristiques du fichier, elle ne les implémente pas. L'implémentation est laissée au composant. Celui-ci contiendra le code qui permettra de récupérer le nom du fichier, sa date, sa taille. Il contiendra également le code pour le copier ou le renommer.

Nous n'avons pas à nous intéresser sur la manière dont l'implémentation est faite par le composant, du moment qu'il respecte l'interface correctement. Bien sûr, nous aurons une implémentation différente pour chaque plate-forme. Entre les versions Macintosh et Windows, les composants de fichier seront très différents. Cependant ils implémentent la même interface et par conséquent on peut accéder au composant en utilisant les fonctions de cette interface.

Dans Mozilla, les interfaces sont préfixées par *nsI* ainsi elles sont facilement reconnaissables. Par exemple, *nsIAddressBook* est l'interface qui interagit avec le carnet d'adresses, *nsISound* est celle utilisée pour écouter des fichiers et *nsILocalFile* pour manipuler des fichiers.

Typiquement, les composants XPCOM sont implémentés nativement, ce qui signifie qu'ils font des choses que le langage Javascript ne peut pas réaliser. Par contre, on peut les appeler à partir de scripts. C'est ce que l'on va voir maintenant. Nous pouvons appeler n'importe laquelle des fonctions fournies par le composant telles que décrites par les interfaces qu'il implémente. Par exemple, si vous avez un composant à votre disposition, vous vérifiez alors s'il implémente l'interface *nsISound*, et si c'est le cas, vous pouvez jouer un son grâce lui.

Le processus d'appel de composants XPCOM à partir d'un script se nomme XPCConnect : une couche qui traduit les objets du script en objets natifs.

## Créer des objets XPCOM

L'appel d'un composant XPCOM se fait en trois étapes :

1. Récupérer un composant.
2. Récupérer la partie du composant qui implémente l'interface que l'on veut utiliser.
3. Appeler la fonction que l'on a besoin.

Une fois que les deux premières étapes sont réalisées, nous pouvons effectuer la dernière autant de fois que nécessaire. Prenons le cas du renommage d'un fichier. La première étape est de récupérer le composant "fichier". Puis on interroge ledit composant pour récupérer la portion qui implémente l'interface `nsILocalFile`. Enfin, on appelle les fonctions fournies par l'interface. Cette interface est utilisée pour représenter un unique fichier.

Nous avons vu que les noms d'interfaces commencent toujours par *nsI*. Par contre, la désignation des composants utilise la syntaxe URI. Mozilla stocke une liste de tous les composants disponibles dans son propre registre. Un utilisateur peut installer de nouveaux composants si besoin est. Ce mécanisme fonctionne comme les plugins.

Mozilla fournit un composant "fichier" c'est-à-dire implémentant `nsILocalFile`. Ce composant est désigné par l'URI `@mozilla.org/file/local;1`. Le schéma URI `component :` permet de spécifier un composant. D'autres composants peuvent être désignés de manière similaire.

L'URI du composant peut être utilisé pour récupérer l'objet composant correspondant. Voici en Javascript le code correspondant :

```
var aFile = Components.classes["@mozilla.org/file/local;1"].createInstance();
```

Le composant "fichier" est récupéré et stocké dans la variable `aFile`. Dans l'exemple, `Components` fait référence à un objet global fournissant les fonctions relatives à certains composants. Ici la classe d'un composant est récupérée en utilisant la propriété `classes`. Cette propriété est un tableau de tous les composants disponibles. Pour obtenir un composant différent, il suffit de remplacer l'URI par celui du composant voulu. Finalement, une instance est créée avec la fonction `createInstance`.

Vous devez vérifier que la valeur de retour de `createInstance` est différente de *null*, valeur qui indiquerait que le composant n'existe pas.

Pour l'instant, nous avons seulement une référence sur le composant "fichier". Pour appeler ses fonctions, nous avons besoin de récupérer une de ces interfaces, dans notre cas `nsILocalFile`. Une seconde ligne est ajoutée à notre code comme suit :

```
var aFile = Components.classes["@mozilla.org/file/local;1"].createInstance();
if (aFile) aFile.QueryInterface(Components.interfaces.nsILocalFile);
```

La fonction `QueryInterface` est fournie par tous les composants, elle permet d'obtenir une interface précise du composant. Elle prend un seul paramètre, le nom de l'interface souhaité. La propriété `interfaces` de `Components` contient une liste de toutes les interfaces des composants. Ici on utilise l'interface `nsILocalFile` que l'on passe en paramètre à `QueryInterface`. Ainsi `aFile` fera référence à la partie du composant qui implémente l'interface `nsILocalFile`.

Ces deux lignes de Javascript peuvent être utilisées pour obtenir n'importe quelle interface de n'importe quel composant. Il suffit de remplacer le nom du composant et le nom de l'interface que l'on veut utiliser. On peut bien sûr choisir n'importe quel nom pour la variable. Par exemple si l'on veut utiliser l'interface pour le son, notre code pourrait être comme suit :

```
var sound = Components.classes["@mozilla.org/sound;1"].createInstance();
if (sound) sound.QueryInterface(Components.interfaces.nsILocalFile);
```

Les interfaces `XPCOM` peuvent hériter d'autres interfaces. L'interface héritière possède ses propres fonctions mais aussi toutes celles des interfaces parentes. Ainsi toute interface hérite de l'interface principale `nsISupports` qui fournit la fonction `QueryInterface`. Comme tout composant doit implémenter `nsISupports`, la fonction `QueryInterface` est disponible sur tous les composants.

Plusieurs composants peuvent implémenter la même interface. Typiquement ce sont des sous-classes de l'original mais pas nécessairement. N'importe quel composant peut implémenter les fonctionnalités de `nsILocalFile`. De plus, un composant peut implémenter plusieurs interfaces. C'est pour ces raisons que l'on doit procéder en deux étapes pour appeler les fonctions d'une interface.

Cependant, il existe un raccourci pour réduire ces deux étapes en une seule ligne de code :

```
var aLocalFile = Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.
```

Ce code effectue la même action qu'avec les deux lignes, mais en une seule ligne. Il élimine le besoin de créer une instance et ensuite de l'interroger pour obtenir une interface précise, en deux étapes séparées.

Un appel à `QueryInterface` sur un objet qui ne fournit pas l'interface demandée lance une exception. Si vous n'êtes pas sûr que le composant supporte une interface, vous pouvez utiliser l'opérateur `instanceof` comme suit :

```
var aFile = Components.classes["@mozilla.org/file/local;1"].createInstance();
if (aFile instanceof Components.interfaces.nsILocalFile){
    // faire quelque chose si il s'agit d'une instance du bon type
}
```

L'opérateur `instanceof` renvoie *true* si `aFile` implémente l'interface `nsILocalFile`, et il effectue également l'appel de la méthode `QueryInterface`, ce qui fournit par la suite un objet `nsILocalFile` `aFile` valide.

## L'emploi des fonctions de l'interface

Maintenant que nous avons un objet qui fait référence à un composant avec l'interface `nsILocalFile`, nous pouvons appeler les fonctions de celle-ci à travers l'objet. La liste suivante montre quelques propriétés et méthodes de l'interface `nsILocalFile`.

*initWithPath*

Cette méthode est utilisée pour initialiser le chemin et le nom du fichier pour l'interface `nsILocalFile`. Le premier paramètre doit être le chemin du fichier, comme par exemple `/usr/local/mozilla`.

*leafName*

Le nom du fichier sans son chemin complet.

*fileSize*

La taille du fichier.

*isDirectory()*

renvoie *true* si `nsILocalFile` représente un répertoire.

*remove(recursif)*

efface un fichier. Si le paramètre `recursif` est *true*, le répertoire et tous ses fichiers et sous-répertoires sont effacés.

*copyTo ( repertoire, nouveauNom )*

Copie un fichier dans un autre répertoire, et optionnellement renomme le fichier. La variable `repertoire` doit être un objet `nsILocalFile` représentant le répertoire où l'on veut copier le fichier.

*moveTo ( repertoire, nouveauNom )*

déplace le fichier dans un autre répertoire ou le renomme. La variable `repertoire` doit être un objet `nsILocalFile` représentant le répertoire où l'on va mettre le fichier.

Pour effacer un fichier, on doit d'abord l'assigner à un objet `nsILocalFile`. Nous appelons la méthode `initWithPath` pour définir le fichier en question, en indiquant juste le chemin de celui-ci. Puis on appelle la fonction `remove` avec le paramètre `recursif` à *false*. Voici le code correspondant :

```
var aFile = Components.classes["@mozilla.org/file/local;1"].createInstance();
if (aFile instanceof Components.interfaces.nsILocalFile){
    aFile.initWithPath("/mozilla/testfile.txt");
    aFile.remove(false);
}
```

Ce code prend le fichier */mozilla/testfile.txt* et l'efface. Essayez cet exemple en ajoutant le code à un gestionnaire d'évènements. Vous devez changer le nom du fichier pour qu'il corresponde à un fichier existant que vous voulez effacer sur votre poste local.

Dans la liste ci-dessus, nous avons vu deux fonctions `copyTo` et `moveTo`. Ces fonctions sont utilisées pour respectivement copier et déplacer des fichiers.

Notez que ces fonctions ne prennent pas en paramètre une chaîne de caractères pour désigner un répertoire mais un objet `nsILocalFile`. Cela signifie que nous devons récupérer les deux composants "fichier".

L'exemple suivant montre comment copier un fichier :

```
function copyFile(sourcefile,destdir)
{
    // récupérer un composant pour le fichier à copier
    var aFile = Components.classes["@mozilla.org/file/local;1"]
        .createInstance(Components.interfaces.nsILocalFile);
    if (!aFile) return false;

    // récupérer un composant pour le répertoire où la copie va s'effectuer.
    var aDir = Components.classes["@mozilla.org/file/local;1"]
        .createInstance(Components.interfaces.nsILocalFile);
    if (!aDir) return false;

    // ensuite, on initialise les chemins
    aFile.initWithPath(sourcefile);
    aDir.initWithPath(destdir);

    // Au final, on copie le fichier sans le renommer
    aFile.copyTo(aDir,null);
}

copyFile("/mozilla/testfile.txt","/etc");
```

## Les services XPCOM

Certains composants XPCOM spéciaux sont appelés services. Vous ne pouvez pas créer plusieurs instances d'un service parce qu'il doit être unique. Les services fournissent des fonctions manipulant des données globales ou effectuent des opérations sur d'autres objets. Au lieu d'utiliser `createInstance`, on appelle `getService` pour récupérer une référence sur le composant de type "service". À part ça, les services ne diffèrent pas des autres composants.

Un exemple de service fournit par Mozilla est le service pour les marque-pages. Il vous permet d'ajouter un marque-page à la liste courante des marque-pages de l'utilisateur. Voici un exemple :

```
var bmarks = Components.classes["@mozilla.org/browser/bookmarks-service;1"].getService();
bmarks.QueryInterface(Components.interfaces.nsIBookmarksService);
bmarks.addBookmarkImmediately("http://www.mozilla.org","Mozilla",0,null);
```

Tout d'abord, le composant `@mozilla.org/browser/bookmarks-service;1` est récupéré et son service est placé dans la variable `bmarks`. Nous utilisons `QueryInterface` pour récupérer l'interface `nsIBookmarksService`. La fonction `addBookmarkImmediately` fournie par cette interface peut être utilisée pour ajouter des marque-pages. Les deux premiers paramètres de cette fonction sont l'URL et le titre

du marque–page. Le troisième paramètre est le type de marque–page qui doit normalement être *O*, et le dernier paramètre est l'encodage des caractères du document correspondant au marque–page, qui peut être nul.

Dans la section suivante, nous verrons quelques–unes des interfaces que l'on peut utiliser, fournies par Mozilla.

## 7.6 Exemples XPCOM

Écrit par Neil Deakin. Traduit par *Maximilien* (24/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xpcomex.html>

Cette section donne quelques exemples d'utilisation de la technologie XPCOM avec de nouvelles interfaces.

### Gestion de Fenêtres

La liste des fenêtres Mozilla ouvertes peut être utilisée comme une source de données RDF. Elle vous permet de créer dans votre application un menu donnant la liste des fenêtres courantes ouvertes. La source de données correspondante est *rdf:window–mediator* dont voici un exemple d'utilisation :

Exemple 7.6.1 :

```
<menubar id="windowlist-menubar">
  <menu label="Fenêtres">
    <menupopup id="window-menu" datasources="rdf:window-mediator" ref="NC:WindowMediatorRoot">
      <template>
        <rule>
          <menuitem uri="rdf:*" label="rdf:http://home.netscape.com/NC-rdf#Name"/>
        </rule>
      </template>
    </menupopup>
  </menu>
</menubar>
</toolbox>
```

Un menu contenant la liste de toutes les fenêtres ouvertes sera créé. Essayez cet exemple en ouvrant plusieurs fenêtres, et vous les verrez toutes dans le menu. Nous voudrions maintenant améliorer cet exemple de sorte que lorsqu'on sélectionne un élément du menu, on aille sur la fenêtre correspondante. Cette tâche sera rendue possible grâce au composant "window mediator" qui implémente l'interface *nsIWindowDataSource*. Le code suivant montre comment il fonctionne :

```
var wmdata = Components.classes["@mozilla.org/rdf/datasource;1?name=window-mediator"].getService(
wmdata.QueryInterface(Components.interfaces.nsIWindowDataSource));
```

Ce code récupère le composant "window mediator". Le composant utilisé ici est le même que celui qui gère la source de données RDF "Window–mediator". Vous pouvez également récupérer ce composant au travers du service RDF qui est un autre service de gestion des sources de données RDF.

L'interface *nsIWindowDataSource* possède une fonction *getWindowForResource* qui nous donne une fenêtre à partir d'une ressource. Dans un exemple précédent, nous avons généré une liste de fenêtres que nous avons ajoutée à un menu via une balise *template*. Celle–ci génère un attribut *id* pour chaque élément *menuitem*. La valeur de cet attribut peut être utilisée comme ressource. Ainsi pour donner le focus à la fenêtre sélectionnée, nous pouvons procéder de la manière suivante :

- Déterminer l'élément que l'utilisateur a sélectionné.
- Récupérer la valeur de l'attribut *id* de cet élément.



- Passer cette valeur à `getWindowForResource` pour avoir l'objet "window".
- Mettre le focus sur celle-ci.

L'exemple ci-dessous nous montre comment procéder :

```
<toolbox>
  <menubar id="windowlist-menubar">
    <menu label="Fenêtre" oncommand="switchFocus(event.target);">
      <menupopup id="window-menu" datasources="rdf:window-mediator" ref="NC:WindowMediatorRoot">
        <template>
          <rule>
            <menuitem uri="rdf:*" label="rdf:http://home.netscape.com/NC-rdf#Name"/>
          </rule>
        </template>
      </menupopup>
    </menu>
  </menubar>
</toolbox>
```

```
function switchFocus(elem)
{
  var mediator = Components.classes["@mozilla.org/rdf/datasource;1?name=window-mediator"].getService(
    Components.interfaces.nsIRDFDataSource);

  var resource = elem.getAttribute('id');
  switchwindow = mediator.getWindowForResource(resource);

  if (switchwindow){
    switchwindow.focus();
  }
}
```

Un gestionnaire de commande a été ajouté à l'élément menu. Ce gestionnaire appelle la fonction `switchFocus` avec comme paramètre l'élément du menu que nous avons sélectionné. Cette fonction récupère d'abord une référence du composant qui implémente l'interface du "window mediator" (NdT : plus exactement `nsIRDFDataSource` ). Puis nous récupérons l'attribut `id` de l'élément. Nous utilisons cette valeur comme ressource qui va être passée en paramètre à la fonction `getWindowForResource` pour renvoyer la fenêtre correspondante. Cette fenêtre est stockée dans la variable `switchwindow` comme objet javascript `window`. Par conséquent, toutes les fonctions de cet objet sont utilisables, comme la fonction `focus()`.

## Cookies

Maintenant, nous allons récupérer la liste des cookies sauvegardés par le navigateur. Nous allons utiliser le service "Cookie" qui implémente l'interface `nsICookieManager` utilisée pour énumérer tous les cookies. Voici un exemple qui alimente la liste d'un menu avec le nom de tous les cookies provenant du site MozillaZine.

```
<script>

function getCookies()
{
  var menu = document.getElementById("cookieMenu");
  menu.removeAllItems();

  var cookieManager = Components.classes["@mozilla.org/cookieManager;1"]
    .getService(Components.interfaces.nsICookieManager);

  var iter = cookieManager.enumerator;
  while (iter.hasMoreElements()){
```

```

var cookie = iter.getNext();
if (cookie instanceof Components.interfaces.nsICookie){
    if (cookie.host == "www.mozillazine.org")
        menu.appendItem(cookie.name, cookie.value);
    }
}
}
}
</script>

<hbox>
    <menulist id="cookieMenu" onpopupshowing="getCookies();" />
</hbox>

```

La fonction `getCookies` sera appelée à chaque ouverture du menu, comme indiqué par l'attribut `onpopupshowing` de l'élément `menulist`. Les deux premières lignes de `getCookies` récupèrent l'élément `menulist` et vident tous les items existants. En effet comme cette fonction est appelée à chaque fois que nous l'ouvrons, nous ne voulons pas garder les anciens éléments.

Ensuite, le gestionnaire de cookie est récupéré. Celui-ci a une méthode qui renvoie un objet énumérateur implémentant `nsISimpleEnumerator`. Il nous permet de parcourir tous les cookies. Un énumérateur dispose d'une méthode `hasMoreElements` retournant `true` jusqu'à ce que le dernier cookie soit récupéré. La méthode `getNext` renvoie un cookie et incrémente l'index de l'énumérateur. Comme l'énumérateur ne renvoie qu'un objet générique, nous devons lui indiquer que nous voulons utiliser l'interface `nsICookie`. Dans ce cas, l'opérateur `instanceof` permet d'accomplir cette vérification.

Finalement, un élément est ajouté au menu pour chaque cookie (NdT : dont le site hôte est "www.mozillazine.org"). Les propriétés hôte, nom et valeur du cookie sont alors utilisées. Les menus ont une fonction `appendItem` qui ajoute un élément avec un libellé et une valeur.

---

Dans la section suivante, nous allons voir comment utiliser le presse-papiers.

## 7.7 Utilisation du presse-papiers

Écrit par Neil Deakin. Traduit par *Nadine Henry* (22/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/clipboard.html>

Attention, cette page est maintenue dans la version française, mais elle a été enlevée par son auteur sur la version anglaise.

Cette section fournit des informations au sujet du couper, copier et coller dans le presse-papiers et à partir du presse-papiers.

### Le presse-papiers

Mozilla fournit un certain nombre d'interfaces pour accéder au presse-papiers. Le composant `@mozilla.org/widget/clipboardhelper;1` peut être utilisé pour copier du texte dans le presse-papiers. Ce composant implémente l'interface `nsIClipboardHelper`, dont la fonction `copyString` peut être utilisée pour copier une chaîne de caractères.

```

const gClipboardHelper = Components.classes["@mozilla.org/widget/clipboardhelper;1"]
    .getService(Components.interfaces.nsIClipboardHelper);
gClipboardHelper.copyString("Put me on the clipboard, please.");

```

Cet exemple va tout d'abord créer un outil d'aide pour le presse-papiers puis y copiera une courte chaîne de caractère. Cette méthode fonctionne uniquement pour insérer des chaînes de caractères dans le presse-papiers. Pour d'autres types de données, comme les URLs ou les images, vous aurez besoin d'utiliser

une méthode plus complexe.

Un composant `@mozilla.org/widget/clipboard;1` et une interface `nsIClipboard` fournissent un accès général au presse-papiers du système. Vous pouvez l'utiliser pour copier et coller n'importe quel type de données de votre application vers le presse-papiers. Trois objets XPCOM sont nécessaires pour manipuler les opérations concernant le presse-papiers. Le premier est un objet chargé de prendre les données pour les insérer dans le presse-papiers. Le deuxième est l'objet presse-papiers. Le troisième est un objet qui est utilisé pour transférer les données du premier objet vers le presse-papiers. Le modèle de presse-papiers dans Mozilla nécessite que vous suiviez les étapes suivantes pour copier les données :

1. Créez un container XPCOM pour les données que vous souhaitez insérer dans le presse-papiers. C'est nécessaire parce que vous pouvez y mettre n'importe quoi, du texte jusqu'à des images.
2. Créez un objet de transfert. Cet objet peut être le composant `@mozilla.org/widget/transerable;1` qui implémente l'interface `nsITranserable`.
3. Spécifiez à l'objet de transfert le type de données à copier.
4. Spécifiez à l'objet de transfert les données à copier.
5. Créez un objet presse-papiers qui se réfère au presse-papiers du système.
6. Spécifiez à l'objet presse-papiers de copier les données utilisant l'objet de transfert.

Vous pourriez vous demander pourquoi un objet de transfert est nécessaire au lieu de simplement insérer directement l'objet dans le presse-papiers. L'une des raisons est que certains systèmes ne copient pas les données tout de suite. Au lieu de cela, ils attendent jusqu'à ce que les données soient collées. Une autre raison est que l'objet de transfert peut contenir de multiples représentations d'une seule et même donnée. Par exemple, un extrait HTML peut être représenté autant dans sa forme originale en HTML qu'en texte brut. Si une application souhaite obtenir la donnée à partir du presse-papiers et ne comprend pas le HTML, elle peut utiliser la version en texte brut. Si elle peut comprendre le HTML, elle peut récupérer cette version. L'objet de transfert contiendra le contenu du presse-papiers jusqu'à ce que l'application ait décidé son besoin. Il permet au presse-papiers de pouvoir être utilisé par une autre application de suite.

Décomposons à présent les étapes nécessaires pour copier les données dans le presse-papiers. Tout d'abord, nous avons besoin de créer un objet XPCOM pour encapsuler ce que nous voulons copier. Nous supposons que nous voulons copier un peu de texte. Nous utiliserons l'interface `nsISupportsString` qui peut être utilisée pour représenter des chaînes de caractères (spécifiquement, des chaînes en Unicode).

```
var copytext="Texte à copier";
var str = Components.classes["@mozilla.org/supports-string;1"]
    .createInstance(Components.interfaces.nsISupportsString);
str.data=copytext;
```

La première ligne contient le texte que l'on veut copier. Ensuite, la variable `str` est assignée à un composant qui peut être utilisé pour contenir une chaîne de caractères. La troisième ligne assigne la chaîne de caractères au composant en utilisant la propriété `data`. Ici, la chaîne de caractères *Texte à copier* va être copiée mais vous pouvez la remplacer par la chaîne de caractères que vous souhaitez copier. À présent que l'on a l'objet à copier, un objet de transfert doit être créé :

```
var trans = Components.classes["@mozilla.org/widget/transerable;1"]
    .createInstance(Components.interfaces.nsITranserable);
trans.addDataFlavor("text/unicode");
trans.setTransferData("text/unicode",str,copytext.length * 2);
```

La première ligne reçoit le composant de transfert qui implémente `nsITranserable`. Ensuite, nous devons spécifier à l'objet de transfert quel type de données nous souhaitons utiliser. Le type de données est représenté par son type mime. La fonction `addDataFlavor` est utilisée pour indiquer à l'objet de transfert qu'il doit transférer les données d'un certain type. Dans le cas présent, nous transférons le type mime *text/unicode* qui indique une chaîne de caractères Unicode. Puis, la fonction `setTransferData` qui copie les données de la chaîne de caractères vers l'objet de transfert est appelée. Cette fonction prend en

compte trois paramètres. Le premier est le type mime que nous déclarons, le deuxième est l'objet qui contient la chaîne de caractères et le troisième est la longueur de la donnée, en octets. Ici, la longueur est multipliée par deux car nous utilisons une chaîne de caractères Unicode qui requiert deux octets par caractère.

Vous pouvez répéter les deux dernières lignes et appeler `addDataFlavor` et `setTransferData` pour de multiples types mime. De cette façon, vous pouvez avoir une version texte brut et une version HTML du contenu. L'objet de transfert va contenir sa propre copie des données. Lorsque vous avez ajouté tous les types souhaités, vous pouvez tous les mettre dans le presse-papiers immédiatement. L'objet de transfert va contenir toutes les données souhaitées jusqu'à ce que vous soyez prêt à les insérer dans le presse-papiers.

Ensuite, nous devons créer un objet presse-papiers qui se réfère au presse-papiers du système.

```
var clipid = Components.interfaces.nsIClipboard;
var clip = Components.classes["@mozilla.org/widget/clipboard;1"].getService(clipid);
clip.setData(trans,null,clipid.kGlobalClipboard);
```

Nous obtenons l'objet presse-papiers du système et le stockons dans la variable `clip`. Nous pouvons copier la donnée dans le presse-papiers en appelant la fonction `setData`. Le premier paramètre de cette fonction est l'objet de transfert. Le second paramètre peut habituellement être déclaré à *null* mais vous pourriez le déclarer à un `nsIClipboardOwner` de sorte que vous puissiez indiquer que la donnée copiée soit écrasée par une autre opération de copie. N'appellez `setData` que lorsque vous êtes prêt à copier dans le presse-papiers du système.

Le troisième paramètre de `setData` (ainsi que le paramètre de `emptyClipboard`) indique quel tampon de presse-papiers utiliser. Le code ci-dessus utilise la constante `kGlobalConstant` pour cela, qui se réfère au presse-papiers global. Ce serait la même constante qui coupe et colle les opérations dans le menu d'édition utilisé généralement. Si vous utilisez `kSelectionClipboard` à la place, vous allez copier dans le tampon de sélection, qui n'est en général disponible que sous des systèmes Unix.

Ce processus pluri-étapes a eu pour résultat la copie du texte dans le presse-papiers. Nous pouvons couper vers le presse-papiers au lieu de copier, en effaçant la donnée originale après la copie. Normalement, le texte serait dans un document ou une zone de texte. Le code est rassemblé ci-dessous, avec en plus une vérification des erreurs :

```
var copytext = "Texte à copier";

var str = Components.classes["@mozilla.org/supports-string;1"]
    .createInstance(Components.interfaces.nsISupportsString);
if (!str) return false;

str.data = copytext;

var trans = Components.classes["@mozilla.org/widget/transferrable;1"]
    .createInstance(Components.interfaces.nsITransferrable);
if (!trans) return false;

trans.addDataFlavor("text/unicode");
trans.setTransferData("text/unicode",str,copytext.length * 2);

var clipid = Components.interfaces.nsIClipboard;
var clip = Components.classes["@mozilla.org/widget/clipboard;1"].getService(clipid);
if (!clip) return false;

clip.setData(trans,null,clipid.kGlobalClipboard);
```

## Coller les contenus du presse-papiers

Pour coller la donnée à partir du presse-papiers nous pouvons utiliser un processus similaire, sauf que nous employons `getData` au lieu de `setData` et `getTransferData` au lieu de `setTransferData`. Voici les étapes pour le collage :

1. Créez un objet presse-papiers qui se réfère au presse-papiers du système.
2. Créez un objet de transfert qui implémente l'interface `nsITransferable`.
3. Spécifiez à l'objet de transfert quelle type de donnée vous souhaitez obtenir.
4. Recherchez la donnée du presse-papiers et l'insérez dans l'objet de transfert.
5. Obtenez la donnée à partir de l'objet de transfert.

La première étape est similaire à celle utilisée pour la copie :

```
var clip = Components.classes["@mozilla.org/widget/clipboard;1"]
    .createInstance(Components.interfaces.nsIClipboard);
if (!clip) return false;

var trans = Components.classes["@mozilla.org/widget/transerable;1"]
    .createInstance(Components.interfaces.nsITransferable);
if (!trans) return false;
trans.addDataFlavor("text/unicode");
```

Ce code reçoit l'objet presse-papiers du système et un objet de transfert. Le type mime est ajouté à ce dernier. Maintenant, nous avons besoin d'obtenir la donnée du presse-papiers :

```
clip.getData(trans, clip.kGlobalClipboard);

var str = new Object();
var strLength = new Object();

trans.getTransferData("text/unicode", str, strLength);
```

La première ligne exécute l'opposé de `setData`. La donnée actuellement dans le presse-papiers du système est placée dans l'objet de transfert. Ensuite, nous créons deux objets Javascript qui prendront la donnée et sa longueur. Notez que nous n'avons aucune idée du type de donnée qui est actuellement dans le presse-papiers. Elle a pu avoir été placée là par une autre application. C'est pour cela que nous utilisons des objets génériques pour `str` et `strLength`.

Puis nous utilisons `getTransferData` pour récupérer la donnée à partir de l'objet de transfert. Nous spécifions le type mime que nous voudrions obtenir. La donnée sera convertie si elle n'est pas du type désiré et si une conversion entre le type mime actuel et celui voulu est possible. Si à l'origine vous avez copié la donnée sous plusieurs types dans le presse-papiers, vous pouvez récupérer la donnée dans le meilleur format voulu. Par exemple, un champ de saisie de texte accepterait du texte unicode (ou du texte brut) tandis qu'une fenêtre d'édition pourrait accepter du HTML et des images.

La variable `str` contient maintenant la donnée du presse-papiers. Nous devons à nouveau convertir la donnée dans une chaîne de caractères Javascript à partir d'un objet XPCOM. Pour ce faire, le code ci-dessous peut être utilisé :

```
if (str) str = str.value.QueryInterface(Components.interfaces.nsISupportsString);
if (str) pastetext = str.data.substring(0, strLength.value / 2);
```

Du fait que la donnée de l'objet de transfert est un `nsISupportsString`, nous devons la convertir en une chaîne de caractère Javascript. La propriété `value` de l'objet complété par `getTransferData`, qui est `str` dans ce cas, fournit la valeur actuelle de l'objet.

Nous assignons la chaîne de caractères à la variable `pastetext`. Nous pouvons ainsi la mettre dans un champ texte ou à un autre endroit si nécessaire.

---

Dans la prochaine section, nous verrons comment faire du glisser-déposer.

## 7.8 Glisser-Déposer

Écrit par Neil Deakin. Traduit par *Gabriel de Perthuis* (27/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/dragdrop.html>

Attention, cette page est maintenue dans la version française, mais elle a été enlevée par son auteur sur la version anglaise.

Cette section indique comment implémenter des objets capables d'être déplacés, et lâchés sur d'autres objets.

### L'interface glisser-déposer

De nombreuses interfaces graphiques permettent de glisser-déposer des objets. Il est par exemple possible de déplacer des fichiers d'un dossier à l'autre, ou de lâcher une icône sur une fenêtre pour ouvrir le document qui lui correspond. Mozilla et XUL fournissent une panoplie d'évènements qui se déclenchent lorsque un utilisateur essaie de déplacer un objet.

L'utilisateur commence le glissement en cliquant et en déplaçant la souris sans relâcher le bouton. Le glissement finit quand l'utilisateur relâche le bouton. Des gestionnaires d'évènements sont appelés dans ces deux situations, ainsi que lors de certaines étapes intermédiaires.

Mozilla implémente le glisser-déposer à l'aide d'une session de glisser-déposer. Quand un utilisateur demande à faire glisser un objet, et que celui-ci est susceptible d'être glissé-déposé, une session est initiée. La session se charge de modifier le curseur de la souris là où l'objet peut être déposé. Si l'objet ne supporte pas le glisser déposer, la session n'est pas créée. L'utilisateur n'ayant en général qu'une souris, une seule session est active à un instant donné.

Notez que la session peut être créée depuis Mozilla, mais aussi par d'autres applications. Le cas échéant, Mozilla s'occupe de traduire la session.

La liste qui suit décrit tous les gestionnaires d'évènements pouvant être appelés, et que n'importe quel élément peut implémenter. Vous n'avez à définir que les évènements qui vous intéressent.

#### *ondraggesture*

Appelé quand l'utilisateur commence à glisser l'élément, ce qui arrive quand l'utilisateur garde le bouton appuyé et déplace la souris. Le script qui intercepte cet évènement doit mettre en place une session de glisser-déposer.

#### *ondragover*

Appelé pour un élément survolé par le glisser. Si l'objet peut être lâché ici, vous le signalez à la session.

#### *ondragenter*

Appelé quand le glisser entre dans l'espace aérien de l'élément. Il permet à l'élément de changer d'apparence pour signifier à l'utilisateur qu'il peut lâcher ici.

#### *ondragexit*

Appelé quand le glisser cesse de survoler l'élément, et également quand le glisser-déposer s'achève. Il permet ainsi de rétablir l'apparence de l'élément ou de faire d'autres choses.

#### *ondragdrop*

Appelé quand le glisser-déposer s'achève sur l'élément. À ce moment là, l'utilisateur a déjà relâché le bouton de la souris. L'élément peut choisir d'ignorer l'évènement, ou réagir en insérant l'objet

glissé par exemple.

Il y a deux façons de gérer les événements de glisser-déposer. La première passe par l'emploi direct des interfaces de glisser-déposer XPCOM. La seconde est de passer par un conteneur javascript qui vous décharge d'une partie du travail. Ce conteneur est inclus dans le paquet "toolkit" ou "global" de Mozilla.

## Le glisser-déposer avec XPCOM

Deux interfaces sont utilisées pour le glisser-déposer. La première est un service de glisser-déposer, `nsIDragService`, et la seconde est la session, `nsIDragSession`.

`nsIDragService` est responsable de la création de session quand le glissement commence, et de sa destruction quand le glissement s'achève. Sa fonction `invokeDragSession` doit être appelée pour commencer le glisser-déposer, dans le contexte d'un événement `ondraggesture`. Une fois cette fonction appelée, le glisser-déposer a commencé.

La fonction `invokeDragSession` prend quatre paramètres, décrits ci-après :

```
invokeDragSession(element, transferableArray, region, actions);
```

*element*

Une référence à l'élément à glisser-déposer. On l'obtient en accédant à la propriété `event.target` dans le gestionnaire d'évènement.

*transferableArray*

Un tableau d'objets `nsITransferable`, un par élément glisser-déposer. On utilise un tableau pour pouvoir déplacer plusieurs éléments d'un coup, comme un ensemble de fichiers.

*region*

Une région utilisée pour donner un retour à l'utilisateur. Mettre à *null* dans la majorité des cas.

*actions*

Les actions utilisées par le glisser-déposer. Elles peuvent prendre l'une des valeurs de constantes suivantes, ou une combinaison de ces valeurs. L'action peut être changée pendant le glisser-déposer selon l'élément qui est survolé :

`nsIDragSession.DRAGDROP_ACTION_NONE`

Indique qu'aucune action de glisser-déposer n'est valide.

`nsIDragSession.DRAGDROP_ACTION_COPY`

L'élément glisser-déposer peut être copié lors du lâcher.

`nsIDragSession.DRAGDROP_ACTION_MOVE`

L'élément peut être déplacé lors du lâcher.

`nsIDragSession.DRAGDROP_ACTION_LINK`

Un lien (raccourci ou alias) peut être créé là où a lieu le lâcher.

L'interface `nsIDragService` fournit également la fonction `getCurrentSession` qui peut être appelée dans le contexte d'un des événements de glisser-déposer pour accéder à, et modifier, l'état du glisser-déposer. Cette fonction renvoie un objet implémentant `nsIDragSession`.

L'interface `nsIDragSession` sert à modifier les propriétés du glisser-déposer en cours. Elle expose les propriétés et méthodes suivantes :

*canDrop*

Affectez *true* à cette propriété si l'élément en dessous du curseur pourrait accepter que l'élément glissé soit déposé, et *false* si ce n'est pas envisageable. Modifiez cette valeur dans le contexte d'évènements `ondragover` et `ondragenter`.

*dragAction*

Détermine l'action à entreprendre au lâcher, sous forme d'une combinaison des constantes décrites

plus haut. Permet de fournir un meilleur retour à l'utilisateur.

*numDropItems*

Le nombre d'objets déplacés. Par exemple, si l'on déplace cinq marque-pages, ce sera 5.

*getData (transfer, index)*

Obtenir des objets déplacés. Le premier argument est un objet `nsITransferable` qui recevra la réponse, le second est l'indice de l'élément à retourner.

*sourceDocument*

Le document où le glisser-déposer a commencé.

*sourceNode*

Le noeud DOM où le glisser-déposer a commencé.

*isDataFlavorSupported(flavour)*

Renvoie *true* si la donnée déplacée contient un exemplaire dand le type (flavour) indiqué.

Dans la prochaine section, nous verrons comment utiliser le conteneur Javascript pour le glisser-déposer.

## 7.9 Conteneur JavaScript pour le Glisser-Déposer

Écrit par Neil Deakin. Traduit par **Laurent Jouanneau** (11/11/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/dragwrap.html>

Attention, cette page est maintenue dans la version française, mais elle a été enlevée par son auteur sur la version anglaise.

Cette section décrit l'utilisation d'un conteneur JavaScript pour le glisser-déposer.

### Le conteneur JavaScript glisser-déposer

Le conteneur JavaScript pour le glisser-déposer simplifie le processus en appelant toutes les interfaces XPCOM pour vous. Il fonctionne en fournissant un objet qui implémente les gestionnaires d'événements. Tout ce que vous avez à faire est d'écrire quelques fonctions simples qui travaillent sur les données qui sont glissées.

L'interface glisser-déposer est stockée dans le paquetage "global", dans le fichier `chrome://global/content/nsDragAndDrop.js`. Vous pouvez inclure ce fichier dans votre fichier XUL avec la balise `<script>` de la même manière que pour vos scripts. La bibliothèque dépend aussi d'autres scripts, que vous aurez également à inclure, habituellement au début de votre page XUL. Vous pouvez regarder le contenu de ces fichiers pour voir comment fonctionne le glisser-déposer au plus bas niveau.

Notez que vous ne pouvez utiliser ces bibliothèques qu'à l'intérieur de fichiers XUL chargés avec un URL `chrome`.

```
<script src="chrome://global/content/nsDragAndDrop.js" />
<script src="chrome://global/content/nsTransferable.js" />
```

Cette bibliothèque glisser-déposer crée un objet stocké dans la variable `nsDragAndDrop`. L'objet contient une série de fonctions, une pour chaque gestionnaire d'événements (excepté pour `dragenter` où il n'y a rien de spécial à faire). Chacune de ces fonctions prend deux arguments : le premier est l'objet `event` et le deuxième est un objet observateur que vous créez. Vous aurez plus d'explications plus tard.

L'exemple suivant est un exemple d'appel de l'objet `nsDragAndDrop` :

```
<button label="Glissez moi" ondraggesture="nsDragAndDrop.startDrag(event,buttonObserver);" />
```



La fonction `startDrag` sera appelée quand le glisser-déposer débutera à partir du bouton. Le premier paramètre est l'objet `event`, disponible dans tous les gestionnaires d'évènements. Le second paramètre à cette fonction est l'observateur, que nous créerons bientôt. Dans cet exemple, nous ne faisons rien de spécial d'autre quand débute le glisser du bouton. Si nous voulions prendre aussi en compte les autres cas, nous pourrions appeler les autres fonctions, comme dans l'exemple suivant :

```
<description value="Cliquez et glissez ce texte."
  ondraggesture="nsDragAndDrop.startDrag(event,textObserver)"
  ondragover="nsDragAndDrop.dragOver(event,textObserver)"
  ondragexit="nsDragAndDrop.dragExit(event,textObserver)"
  ondragdrop="nsDragAndDrop.drop(event,textObserver)"/>
```

Comme mentionné plus haut, il n'y a rien à faire de spécial pendant l'évènement `dragenter`, aussi vous pouvez l'écrire vous même.

Les fonctions sont implémentées par l'objet `nsDragAndDrop`, qui est déclaré dans le fichier `nsDragAndDrop.js`, inclu par l'une des balises `script`. Elles prennent en charge les évènements, les appels aux interfaces XPCOM, et passent une structure de données simple aux fonctions de l'objet observateur.

L'observateur est un objet que vous déclarez vous-même. Dans les exemples ci-dessus, cet observateur est stocké dans les variables `buttonObserver` et `textObserver`. L'observateur est déclaré dans un script que vous devez inclure dans votre fichier XUL avec la balise `script`. Il doit avoir un certain nombre de propriétés, chacune s'occupant d'un aspect particulier du glisser-déposer. Cinq fonctions peuvent être définies. Vous avez juste à définir celle dont vous avez besoin.

*onDragStart (event , transferData, action)*

Définissez cette fonction pour déclencher une action quand le glisser commence. Elle prend trois arguments : l'objet `event` qui a été passé au gestionnaire d'évènement, les données à transférer, le type d'action du glisser. Cette fonction doit ajouter les données à transférer à l'objet `transferData`.

*onDragOver (event, flavour, session)*

Cette fonction doit être définie quand vous voulez que quelque chose arrive quand le glisser passe au dessus de l'élément. Le premier argument est l'objet `event`, le second est le type de donnée et le troisième est l'objet de session du glisser qui fournit plus de détails sur le glisser-déposer en cours. Vous devez définir cette fonction pour les éléments qui autorisent la dépose de données "glissées" sur eux-même.

*onDragExit (event, session)*

Cette fonction doit être définie quand quelque chose arrive lorsque le glisser quitte l'élément. Elle a deux arguments, l'objet `event` et la session du glisser-déposer.

*onDrop (event, dropData, session)*

Cette fonction doit être définie quand vous voulez faire quelque chose lorsque l'objet est déposé. Le premier argument est l'objet `event` et le second est la donnée qui était glissée. Le troisième argument est la session du glisser-déposer.

*getSupportedFlavours ( )*

Cette fonction doit retourner la liste des types de données que peut accepter l'objet sur lequel on fait le glisser. Cette fonction ne prend pas d'arguments. Elle est nécessaire car ainsi le conteneur peut déterminer le meilleur type de données à passer aux autres fonctions.

Pour un observateur lié à un élément qui peut débiter un glisser-déposer, vous devriez définir au moins la fonction `onDragStart`. Pour les éléments qui peuvent recevoir des objets glissés, vous devriez définir `onDragOver`, `onDrop` et `getSupportedFlavours` (et si vous le voulez, `onDragExit`).

Le type des données pouvant être glisser-déposer, est stocké comme un ensemble de type. Souvent, un objet glissé peut être disponible dans un certain nombre de type. Ce faisant, un élément cible peut accepter le type

qu'il trouve le mieux adapté. Par exemple, un fichier peut être transmis dans deux types, le fichier lui même et son nom. Si le fichier est glissé et déposé sur un répertoire, le type 'fichier' sera utilisé. Si le fichier est glissé sur un champs de saisie, le type 'nom de fichier' sera utilisé. Le texte du nom du fichier est par conséquent utilisé quand les fichiers ne peuvent être déposés directement.

Un type d'objet a un nom, qui est formaté comme un type MIME, comme *text/unicode*. À l'intérieur de la fonction `onDragStart`, vous spécifiez quels types sont disponibles pour l'item en cours de glisser-déposer. Pour faire cela, ajoutez les données et les types à l'objet `transferData`, qui est le second argument de `onDragStart`.

L'exemple ci-après devrait vous aider. La fonction `onDragStart` ajoute des données à l'objet `transferData`.

```
var textObserver = {
  onDragStart: function (evt , transferData, action){
    var htmlText="<strong>Cabbage</strong>";
    var plainText="Cabbage";

    transferData.data=new TransferData();
    transferData.data.addDataForFlavour("text/html",htmlText);
    transferData.data.addDataForFlavour("text/unicode",plainText);
  }
};
```

Ici, un observateur a été déclaré et stocké dans la variable `textObserver`. Il a une propriété appelée `onDragStart` (En JavaScript, les propriétés peuvent être déclarées avec la syntaxe `nom : valeur`). Cette propriété est une fonction qui définit les données qui seront transférées.

Une fois appelé, il commence le glisser-déposer pour la chaîne *Cabbage*. Bien sûr, vous voudrez calculer cette valeur à partir de l'élément sur lequel on a cliqué. Cet élément est disponible dans la propriété `target` de l'objet `event`. Cet objet `event` est passé en premier argument à `onDragStart`.

Nous créons un objet `transferData` qui peut être utilisé pour contenir toutes les données à transférer. Nous ajoutons deux données à celles-ci. La première est une chaîne de texte HTML et la seconde est une chaîne de texte brut. Si l'utilisateur dépose sur une zone qui accepte le HTML (comme la fenêtre d'édition HTML de Mozilla), le type HTML sera utilisé et le texte apparaîtra en gras. Sinon, la version texte brut sera utilisée à la place.

En général vous devrez fournir une version texte de la donnée, ainsi de nombreuses applications pourront l'accepter. L'ordre dans lequel vous définissez les types devra s'établir de la meilleure correspondance vers la moins bonne. Dans le cas ci-dessus, le type HTML (*text/html*) vient en premier, et le type texte (*text/unicode*) en second.

L'exemple ci-dessous montre comment spécifier les données à transférer à partir de l'attribut `label` de l'élément. Dans ce cas, nous fournissons la donnée dans un seul type.

```
var textObserver = {
  onDragStart: function (evt){
    var txt=evt.target.getAttribute("label");

    transferData.data=new TransferData();
    transferData.data.addDataForFlavour("text/unicode",txt);
  }
};
```

Il peut être utile lors de l'implémentation du glisser-déposer pour les cellules d'un arbre. Vous pouvez utiliser la valeur d'une cellule, ou d'une ressource du fichier RDF si l'arbre est construit à partir d'un gabarit,

comme valeur pour le glisser-déposer. Si vous la stockez dans une chaîne, n'importe quel objet acceptant les chaînes pour un glisser-déposer, peut récupérer cette valeur.

Vous aurez besoin d'ajouter un observateur à chaque élément qui peut soit démarrer une action glisser-déposer, soit accepter des objets glissés. Vous pouvez réutiliser le même observateur sur plusieurs éléments. Pour un élément qui démarre un glisser-déposer, `onDragStart` est juste ce qu'il faut à implémenter.

Pour un élément sur lequel on peut déposer, l'observateur aura besoin d'implémenter au moins les fonctions `getSupportedFlavours`, `onDragOver` et `onDrop`. Certains éléments pourraient être capable d'initier un glisser et d'accepter un déposer. Dans ce cas, `onDragStart` sera aussi nécessaire.

La fonction `getSupportedFlavours` doit retourner une liste de type que peut accepter pour une dépose l'élément sur lequel le glisser-déposer s'effectue. Une vue d'un répertoire de système de fichier pourrait accepter des fichiers et peut-être du texte, mais ne devrait pas accepter du texte HTML. Ci-dessous, nous définissons la fonction `getSupportedFlavours`. Nous n'autorisons qu'un seul type ici.

```
var textObserver = {
  getSupportedFlavours : function () {
    var flavours = new FlavourSet();
    flavours.appendFlavour("text/unicode");
    return flavours;
  }
}
```

La liste des types de données contient un seul type, qui est *text/unicode*. L'objet `FlavourSet` peut être utilisé pour contenir une liste de type. Dans certains cas, vous devez aussi fournir une interface XPCOM. Par exemple, pour les fichiers :

```
var textObserver = {
  getSupportedFlavours : function () {
    var flavours = new FlavourSet();
    flavours.appendFlavour("application/x-moz-file", "nsIFile");
    flavours.appendFlavour("text/unicode");
    return flavours;
  }
}
```

La fonction `onDragOver` définit ce qui arrive lorsqu'un objet est glissé au dessus. Vous pourriez alors changer l'apparence des éléments qui sont survolés. Dans la plupart des cas, la fonction ne fait rien. Cependant elle doit être définie pour les éléments qui acceptent des données glissées.

Ensuite, la fonction `onDrop` doit être créée. Son second argument est l'objet de transfert de données qui contient les données transférées. Avant d'appeler `onDrop`, le conteneur aura appelé `getSupportedFlavours` pour déterminer le meilleur type de données à déposer, aussi l'objet de transfert ne contient que les données du meilleur type déterminé.

L'objet de transfert a deux propriétés : `data` qui contient la donnée et `flavour` qui contient le type de la donnée. Une fois que vous avez la donnée, vous pouvez l'ajouter à l'élément de n'importe quelle façon. Par exemple, vous pourriez modifier la valeur d'un champ de saisie.

```
var textObserver = {
  onDrop : function (evt, transferData, session) {
    event.target.setAttribute("value", transferData.data);
  }
}
```

Le système de type utilisé permet à de multiples objets de types variés d'être glisser-déposer, et permet également à des formes alternatives de données de l'être. Le tableau suivant décrit quelques types de données que vous pourriez utiliser. Vous pouvez aussi définir votre propre type si nécessaire.

text/unicode	Text data
text/html	données HTML
application/x-moz-url	une URL
application/x-moz-file	Un fichier local

Dans la prochaine section, nous regarderons un exemple utilisant le glisser-déposer.

## 7.10 Exemple Drag and Drop

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (11/11/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/dragex.html>

Attention, cette page est maintenue dans la version française, mais elle a été enlevée par son auteur sur la version anglaise.

Un exemple de l'implémentation du glisser-déposer est montré dans cette section.

### Glisser-déposer des éléments

Ici, nous créons un simple panneau où des items peuvent y être glisser-déposer à partir d'une palette. L'utilisateur peut cliquer sur l'un des nombreux éléments XUL de la palette et les glisser au dessus d'un élément stack pour créer un élément d'un type particulier.

Tout d'abord, nous ajouterons les scripts du conteneur javascript :

```
<script src="chrome://global/content/nsDragAndDrop.js"/>
<script src="chrome://global/content/nsTransferable.js"/>

<script src="dragboard.js"/>
```

Un script supplémentaire `dragboard.js` est inclus et contiendra le code que nous allons écrire nous-même.

Le panneau sera créé en utilisant un élément stack. Nous utiliserons quelques propriétés de style pour spécifier la largeur et la hauteur de la pile. Une taille maximum est aussi spécifiée, ainsi elle ne sera pas redimensionnée lorsque de nouveaux éléments seront déposés dessus.

Le panneau devra répondre à l'évènement `dragdrop`, en créant un élément lorsque l'utilisateur en déposera un dessus.

```
<stack id="board"
      style="width:300px; height: 300px; max-width: 300px; max-height: 300px"
      ondragover="nsDragAndDrop.dragOver(event,boardObserver)"
      ondragdrop="nsDragAndDrop.drop(event,boardObserver)">
</stack>
```

Le panneau a juste besoin de répondre aux évènements `dragdrop` et `dragover`. Nous ajouterons un observateur `boardObserver` dans le fichier `dragboard.js` dans un moment.

Ensuite, une palette sera ajoutée sur le côté droit de la fenêtre. Elle contiendra trois boutons, un pour créer des nouveaux boutons, un pour créer des cases à cocher, et un autre pour créer des champs de saisie. Ces boutons répondront à l'évènement `draggesture` et débiteront un glisser-déposer.

```
<vbox>

<button label="Bouton"
        elem="button" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
<button label="Case à cocher"
        elem="checkbox" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
<button label="Champ texte"
        elem="textbox" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
</vbox>
```

L'objet `nsDragAndDrop` sera appelé pour faire la plupart du travail. Nous créerons un observateur `listObserver` qui définira la donnée à transférer. Notez que chaque bouton ici a un attribut supplémentaire `elem`. Il s'agit d'un attribut inventé. XUL ne le reconnaît pas et l'ignorera, mais nous pourrions toujours le récupérer avec la fonction DOM `getAttribute`. Nous en avons besoin pour savoir de quel type est l'élément à créer lors du glisser-déposer.

Ensuite nous définirons deux observateurs. Premièrement, `listObserver` qui a besoin d'une fonction pour gérer le démarrage du glisser.

```
var listObserver = {
  onDragStart: function (evt,transferData,action){
    var txt=evt.target.getAttribute("elem");
    transferData.data=new TransferData();
    transferData.data.addDataForFlavour("text/unicode",txt);
  }
};
```

Une seule fonction a été définie, `onDragStart`, et elle sera appelée par l'objet `nsDragAndDrop` lorsque nécessaire. La fonction ajoute la donnée à transférer, à l'objet `transferData`. L'attribut `elem` est récupéré à partir de la cible de l'évènement du glisser-déposer. La cible sera l'élément sur lequel le glisser-déposer a commencé. Nous utiliserons la valeur de cet attribut comme donnée pour le glisser.

L'objet `boardObserver` aura besoin de trois fonctions, `getSupportedFlavours`, `onDragOver`, et `onDrop`. La fonction `onDrop` récupérera la donnée à partir de la session du glisser-déposer et créera un nouvel élément du type approprié.

```
var boardObserver = {
  getSupportedFlavours : function () {
    var flavours = new FlavourSet();
    flavours.appendFlavour("text/unicode");
    return flavours;
  },
  onDragOver: function (evt,flavour,session){},
  onDrop: function (evt,dropdata,session){
    if (dropdata.data!=""){
      var elem=document.createElement(dropdata.data);
      evt.target.appendChild(elem);
      elem.setAttribute("left",""+evt.pageX);
      elem.setAttribute("top",""+evt.pageY);
      elem.setAttribute("label",dropdata.data);
    }
  }
};
```

La fonction `getSupportedFlavours` a seulement besoin de retourner une liste de type que la pile peut accepter lors de la dépose. Dans notre cas, elle accepte juste du texte. Nous n'avons pas besoin de faire

quelque chose de spécial pour la fonction `onDragOver`, ainsi aucun code ne sera ajouté dans son contenu.

Le gestionnaire `onDrop` utilise tout d'abord la fonction `createElement` pour créer un nouvel élément du type stocké dans la session. Ensuite, `appendChild` est appelée pour ajouter un nouvel élément à la pile, qui est la cible de l'évènement. Enfin, nous ajoutons quelques attributs à ce nouvel élément.

La position des éléments dans la pile est déterminée par les attributs `left` et `top`. Les valeurs des propriétés `pageX` et `pageY` contiennent les coordonnées du pointeur de la souris sur la fenêtre lorsque la dépose a lieu. Ils nous permettent de placer le nouvel élément à la même position que la souris quand le bouton a été relâché. Ce n'est pas tout à fait la bonne méthode puisque nous devons en fait calculer les coordonnées de l'évènement relativement à celles de la pile. Mais elle fonctionne ici parce que le panneau est dans le coin en haut à gauche de la fenêtre.

L'attribut `label` est défini avec la donnée issue du glisser-déposer, ainsi le bouton a un libellé par défaut.

Cet exemple est assez simple. Un changement possible est d'utiliser un type personnalisé pour les données plutôt que du texte. Le problème avec l'utilisation du texte est que si le texte provenant d'un glisser-déposer externe est le mot *button*, un bouton sera créé sur le panneau. Un type personnalisé signifie que le panneau acceptera uniquement les glisser-déposer en provenance de la palette.

Le code final est montré ci-dessous :

#### Exemple 7.10.1 :

```
<window title="Composant à déplacer" id="test-window"
  orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script src="chrome://global/content/nsDragAndDrop.js"/>
<script src="chrome://global/content/nsTransferable.js"/>
<script src="dragboard.js"/>

<stack id="board"
  style="width:300px; height: 300px; max-width: 300px; max-height: 300px"
  ondragover="nsDragAndDrop.dragOver(event,boardObserver)"
  ondragdrop="nsDragAndDrop.drop(event,boardObserver)">

</stack>

<vbox>

<button label="Bouton"
  elem="button" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
<button label="Case à cocher"
  elem="checkbox" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
<button label="Champ de saisie"
  elem="textbox" ondraggesture="nsDragAndDrop.startDrag(event,listObserver)"/>
</vbox>

</window>
```

#### Exemple 7.10.2 :

```
var listObserver = {
  onDragStart: function (evt,transferData,action){
    var txt=evt.target.getAttribute("elem");
    transferData.data=new TransferData();
    transferData.data.addDataForFlavour("text/unicode",txt);
  }
};
```

```
var boardObserver = {
  getSupportedFlavours : function () {
    var flavours = new FlavourSet();
    flavours.appendFlavour("text/unicode");
    return flavours;
  },
  onDragOver: function (evt,flavour,session){},
  onDrop: function (evt,dropdata,session){
    if (dropdata.data!=""){
      var elem=document.createElement(dropdata.data);
      evt.target.appendChild(elem);
      elem.setAttribute("left",""+evt.pageX);
      elem.setAttribute("top",""+evt.pageY);
      elem.setAttribute("label",dropdata.data);
    }
  }
};
```

---

Dans la section suivante, nous allons voir comment créer des arbres.

## 8. Arbres

### 8.1 Arbres

Écrit par Neil Deakin. Traduit par *Damien Hardy* (10/04/2004), mise à jour par Alain B. (08/07/2005) .  
Page originale : <http://www.xulplanet.com/tutorials/xultu/trees.html>

XUL fournit un moyen de créer des listes tabulaires ou hiérarchiques en utilisant les arbres.

#### L'élément tree

Un des éléments le plus complexe de XUL est l'arbre. Un arbre peut être utilisé pour afficher des lignes de texte en colonnes. Il peut servir pour des listes tabulaires ou arrangées hiérarchiquement. Un arbre permet également à l'utilisateur de réarranger, redimensionner et masquer certaines colonnes individuellement. Les messages dans une application courrier ou les marque-pages dans Mozilla sont des exemples d'utilisation d'arbres.

D'une certaine manière, un arbre a des similitudes avec une boîte de liste listbox. Tous les deux peuvent être utilisés pour créer des tableaux de données avec des lignes et colonnes multiples, et ils peuvent contenir des en-têtes de colonnes. Les arbres supportent également les lignes imbriquées, alors que les boîtes de liste ne le peuvent pas. Toutefois, les boîtes de liste peuvent contenir n'importe quel type de contenu, alors que les arbres ne peuvent uniquement contenir du texte et des images.

Un arbre comporte deux parties : un ensemble de colonnes, et le corps de l'arbre. L'ensemble de colonnes est défini avec plusieurs éléments treecol, un pour chaque colonne. Chaque colonne apparaîtra comme un en-tête en haut de l'arbre. La seconde partie, le corps de l'arbre, contient les données apparaissant dans l'arbre et est créé avec l'élément treechildren.

L'arbre est unique dans le sens où son corps consiste en un seul composant graphique qui dessine toutes les données dans l'arbre. Cette définition contraste avec la boîte de liste où des balises individuelles listitem et listcell sont utilisées pour spécifier les lignes dans l'élément listbox. Dans un arbre, toutes les données à afficher sont fournies par un objet séparé, appelé la vue d'arbre. Lorsqu'une cellule doit être affichée, le composant graphique de l'arbre fait appel à cet objet de vue d'arbre pour déterminer ce qui doit être affiché, et le dessine ensuite dans l'arbre. L'arbre est suffisamment intelligent pour ne solliciter les informations de la vue que des lignes qui ont besoin d'être affichées. Ainsi, l'affichage est optimisé par le chargement des données le nécessitant réellement. Par exemple, un arbre peut contenir des milliers de lignes, la plupart d'entre elles étant en dehors du cadre de l'arbre, cachées à la vue. Ainsi, l'arbre peut être étendu sur autant de lignes sans problèmes de perte de performance. Bien entendu, ceci est indépendant de la performance de l'objet de vue lui même.

Une vue d'arbre est un objet qui implémente l'interface nsITreeView. Cette interface contient trente propriétés et fonctions que vous pourrez implémenter. Ces fonctions seront appelées par l'arbre au besoin pour récupérer les données et les états de l'arbre. Par exemple, la fonction `getCellText` sera appelée pour obtenir le libellé d'une cellule particulière dans l'arbre.

L'utilisation d'une vue a l'avantage de vous permettre de stocker vos données d'une façon plus adaptée à l'arbre, ou de charger les données sur demande seulement lorsque les lignes sont affichées. Elle offre une plus grande souplesse avec l'utilisation des arbres.

Naturellement, devoir implémenter une vue d'arbre avec une trentaine de propriétés et méthodes peut être très encombrant, surtout pour de simples arbres. Fort heureusement, XUL fournit un ensemble d'implémentations natives réalisant le gros du travail pour vous. Pour la plupart des arbres, surtout lorsque vous débutez à utiliser les arbres, vous utiliserez un de ces types natifs. Cependant, vous pouvez créer une



vue entièrement à partir d'un brouillon si nécessaire. Ainsi, il vous est possible de stocker vos données dans un tableau ou une structure JavaScript, ou de charger les données à partir d'un fichier XML.

Comme le corps de l'arbre dans sa totalité est un unique élément graphique, vous ne pouvez pas modifier le style de lignes ou de cellules individuellement de manière classique. En fait, il n'existe pas d'éléments affichant des cellules individuelles, comme il en existe avec les boîtes de liste. À la place, tout l'affichage est effectué par le corps de l'arbre grâce aux données fournies par la vue de l'arbre. Ce point important a tendance à dérouter bien des développeurs XUL. Pour modifier l'apparence d'une cellule d'un arbre, la vue doit associer un jeu de mots clefs pour une ligne et une cellule. Une syntaxe CSS spéciale est employée entre les composants de styles du corps d'un arbre avec ces mots clefs. Dans un sens, le mécanisme est similaire aux classes CSS. L'application d'un style à un arbre sera détaillée dans une section ultérieure.

## Les éléments d'arbre

Les arbres sont créés avec l'élément `tree` qui sera décrit dans les sections suivantes. Il existe également deux éléments définissant l'affichage des colonnes dans l'arbre.

### tree

L'élément entourant d'un arbre

### treecols

Cet élément déclare une colonne d'un arbre. Avec l'utilisation de cet élément, vous pouvez spécifier des informations supplémentaires sur le tri des données en colonne et si l'utilisateur pourra redimensionner les colonnes. Vous devez toujours placer un attribut `id` sur une colonne, car Mozilla utilise cet identifiant pour les colonnes à réarranger ou à masquer. Il n'est plus nécessaire sous les versions 1.8 et suivante de Mozilla, mais c'est une bonne habitude à conserver.

### treechildren

Cet élément contient le corps principal de l'arbre, là où les lignes individuelles de données seront affichées

Voici un exemple d'arbre avec deux colonnes :

Exemple 8.1.1 :

```
<tree flex="1">
  <treecols>
    <treecol id="nameColumn" label="Nom" flex="1"/>
    <treecol id="addressColumn" label="Adresse" flex="2"/>
  </treecols>
  <treechildren/>
</tree>
```

Tout d'abord, le tableau entier est entouré avec l'élément `tree`. Il déclare un élément qui servira de tableau ou d'arbre. Comme avec les tables HTML, les données d'un arbre sont toujours organisées en lignes. Les colonnes sont spécifiées grâce à la balise `treecols`.

Vous pouvez mettre autant de colonnes que vous le souhaitez dans un arbre. Comme pour les boîtes de listes, une ligne d'en-têtes apparaîtra avec les libellés des colonnes. Un menu déroulant apparaîtra dans le coin supérieur droit de l'arbre et permettra à l'utilisateur d'afficher ou de masquer les colonnes individuellement. Chaque colonne est créée avec l'élément `treecols`. Vous pouvez définir le libellé d'en-tête en utilisant l'attribut `label`. Vous pouvez également rendre vos colonnes flexibles si votre arbre l'est aussi, ainsi les colonnes s'ajusteront en fonction de l'arbre. Dans cet exemple, la seconde colonne sera deux fois plus large que la première. Toutes les colonnes doivent être placées à l'intérieur de l'élément `treecols`.

Dans ce cas, nous n'avons pas indiqué à la vue les données de l'arbre, seuls les en-têtes de colonnes et un arbre vide seront visibles. Vous pouvez redimensionner la fenêtre, mais rien ne sera visible car il n'y a

aucune donnée à afficher. Comme l'arbre a été marqué comme flexible, son corps s'ajustera à l'espace disponible. La flexibilité d'un arbre est couramment appliquée, car les données de l'arbre sont souvent les informations les plus significatives affichées, donc il est logique que l'arbre puisse ajuster sa dimension. Toutefois, vous pouvez spécifier un nombre de lignes à afficher dans l'arbre en affectant l'attribut `rows` sur l'élément `tree`. Cet attribut indique le nombre de lignes qui seront affichées dans l'interface utilisateur et non le nombre de lignes de données. Le nombre total de lignes est fourni par la vue d'arbre. S'il y a trop de lignes de données à afficher dans l'arbre, une barre de défilement apparaîtra pour permettre à l'utilisateur de visualiser le reste. Si vous ne spécifiez aucun attribut `rows`, la valeur par défaut sera `0` signifiant qu'aucune ligne ne s'affichera. Dans ce cas, vous devrez rendre votre arbre flexible. Si votre arbre est flexible, il n'a pas besoin d'un attribut `rows` puisqu'il s'ajustera automatiquement à l'espace disponible.

## Le contenu de la vue d'arbre

Nous avons vu que les données à afficher dans un arbre proviennent d'une vue et non de balises XUL, en passant par une construction interne de la vue d'arbre à partir des balises XUL. Ce mécanisme peut être légèrement confus, mais retenez que la construction des vues d'arbre emploie une série de balises servant à définir l'information sur les données dans l'arbre. Les éléments suivants sont utilisés :

### treeitem

Il contient une unique ligne de niveau supérieur et tous ses descendants. Il sert également d'item pouvant être sélectionné par l'utilisateur. La balise `treeitem` entoure une ligne entière en permettant de la sélectionner entièrement.

### treerow

Une seule ligne d'un arbre devant être placée à l'intérieur d'une balise `treeitem`.

### treecell

Une seule cellule d'un arbre. Cet élément est placé à l'intérieur d'un élément `treerow`.

Par convention, ces balises peuvent être placées directement à l'intérieur de la balise `treechildren`, imbriquées dans l'ordre mentionné ci-dessus. Ces balises définissent les données à afficher dans le corps de l'arbre. Dans ce cas, l'arbre utilise la construction interne de la vue d'arbre, appelée le contenu de la vue d'arbre, qui utilise les libellés et les valeurs spécifiés sur ces éléments comme données pour l'arbre. Lorsque l'arbre a besoin d'afficher une ligne, il demande à la vue d'arbre le libellé de la cellule en appelant la fonction `getCellText` de la vue, qui, dans la continuité, obtient la donnée de l'élément `treecell` approprié.

Cependant, les trois éléments listés ci-dessus ne sont pas affichés directement. Ils ne sont utilisés que comme source de données pour la vue. Ainsi, seuls des attributs utiles sont appliqués sur l'élément `treeitem` et les éléments associés. Par exemple, vous ne pouvez pas modifier l'apparence des lignes d'un arbre en utilisant un attribut `style` ou d'autres propriétés CSS, et les fonctionnalités existantes pour les boîtes, telles que la flexibilité et l'orientation, ne peuvent pas être utilisées.

En fait, à part quelques attributs spécifiques aux arbres, les seuls qui auront un effet sont l'attribut `label` pour définir un texte libellé d'une cellule et l'attribut `src` pour définir une image. Toutefois, dans les sections ultérieures, nous verrons des moyens spéciaux de modifier le style d'un arbre et d'appliquer d'autres fonctionnalités.

De même, les événements ne sont pas générés par un élément `treeitem` et ses enfants ; en revanche, ils seront générés par l'élément `treechildren`.

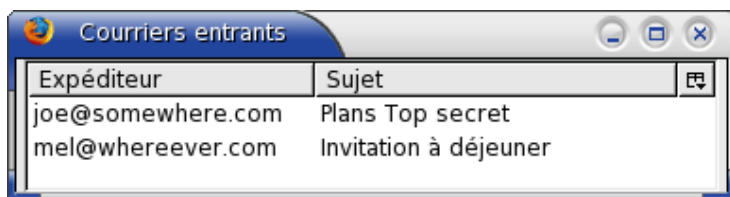
Le fait que les éléments `treeitem` soient si différents des autres éléments XUL est une source classique de confusion pour les développeurs XUL. Essentiellement, le contenu de la vue d'arbre est une vue où les données des cellules sont fournies à partir des balises placées à l'intérieur de l'arbre. Naturellement, si vous utilisez un type différent de vue, les données seront fournies par une autre source, et il n'y aura aucun élément `treeitem` du tout.

Commençons par regarder comment créer un arbre simple avec des colonnes multiples en utilisant la vue d'arbre de contenu. Il pourrait servir à créer une liste de messages mail. Il y aura plusieurs colonnes, telles que l'expéditeur et le sujet.

Exemple 8.1.2 :

```
<tree flex="1">
  <treecols>
    <treecol id="sender" label="Expéditeur" flex="1"/>
    <treecol id="subject" label="Sujet" flex="2"/>
  </treecols>

  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="joe@somewhere.com"/>
        <treecell label="Plans Top secret"/>
      </treerow>
    </treeitem>
    <treeitem>
      <treerow>
        <treecell label="mel@whereever.com"/>
        <treecell label="Invitation à déjeuner"/>
      </treerow>
    </treeitem>
  </treechildren>
</tree>
```



Comme vous pouvez le voir sur cette image,

l'arbre a été créé avec deux lignes de données.

Cet arbre a deux colonnes dont la seconde occupe plus de place que la première. Vous rendrez généralement les colonnes flexibles. Vous pouvez également imposer les largeurs grâce à l'attribut `width`. Vous devez inclure le même nombre d'éléments `treecol` qu'il y a de colonnes dans l'arbre. Dans le cas contraire, des choses étranges peuvent se produire.

Les en-têtes sont créés automatiquement. Le bouton situé dans le coin supérieur droit sert à afficher ou à masquer les colonnes. Vous pouvez placer un attribut `hidecolumnpicker` sur l'élément `tree` et le définir à `true` si vous désirez masquer ce bouton. Si ce bouton est masqué, l'utilisateur ne sera pas en mesure de masquer des colonnes.

Assurez vous d'avoir défini un attribut `id` sur chaque colonne, sinon les actions de masquage et d'affichage ne fonctionneront pas avec toutes les versions de Mozilla.

L'élément `treechildren` entoure toutes les lignes. Les lignes individuelles à l'intérieur du corps peuvent contenir d'autres lignes. Pour l'arbre le plus simple, chaque ligne est créée avec les éléments `treeitem` et `treerow`. L'élément `treerow` entoure toutes les cellules d'une ligne, tandis que l'élément `treeitem` entoure une ligne et tous ses enfants. Les arbres avec des lignes imbriquées seront décrits dans la section suivante.

Dans les cellules, vous placerez les cellules individuelles. Elles sont créées avec l'élément `treecell`. Vous pouvez définir un texte dans une cellule en utilisant l'attribut `label`. Le premier élément `treecell` dans une ligne détermine le contenu qui apparaîtra dans la première colonne, le second élément `treecell`

détermine le contenu qui apparaîtra dans la seconde colonne, et ainsi de suite.

L'utilisateur peut sélectionner les items de l'arbre en cliquant sur eux avec la souris, ou en mettant en surbrillance avec le clavier. Il peut en sélectionner plusieurs en maintenant appuyée la touche **Maj** ou **Ctrl** et en cliquant sur d'autres lignes. Pour désactiver la sélection multiple, placez un attribut `seltype` sur l'élément `tree` avec la valeur *single*. Ainsi, l'utilisateur ne pourra sélectionner qu'une seule ligne à la fois.

## Exemple d'arbre

Ajoutons un arbre à notre exemple de recherche de fichiers dans lequel les résultats de la recherche seront affichés. L'arbre utilisera une vue de contenu d'arbre. Le code suivant doit être ajouté à la place de la balise `iframe`.

Exemple :

```
<tree flex="1">
  <treecols>
    <treecol id="name" label="Nom de fichier" flex="1"/>
    <treecol id="location" label="Emplacement" flex="2"/>
    <treecol id="size" label="Taille" flex="1"/>
  </treecols>

  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="mozilla"/>
        <treecell label="/usr/local"/>
        <treecell label="2520 bytes"/>
      </treerow>
    </treeitem>
  </treechildren>
</tree>

<splitter collapse="before" resizeafter="grow"/>
```

Nous avons ajouté un arbre avec les trois colonnes *Nom de fichier*, *Emplacement* et *Taille*. La seconde colonne sera deux fois plus large grâce à une flexibilité plus grande. Une seule ligne a été ajoutée pour les besoins de la démonstration de l'apparence du tableau avec une ligne. Dans une implémentation réelle, les lignes seront ajoutées par un script à l'issue de la recherche, ou une vue personnalisée sera créée pour contenir les données.

---

Nous verrons ensuite comment créer des arbres plus complexes

## 8.2 Autres caractéristiques des arbres

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (24/06/2004), mise à jour par Gerard L. (25/03/2005)

Page originale : <http://www.xulplanet.com/tutorials/xultu/advtrees.html>

Nous allons voir ici des caractéristiques supplémentaires sur les arbres.

### Arbres hiérarchiques

L'élément `tree` est aussi utilisé pour créer des listes hiérarchiques, comme on en trouve dans un gestionnaire de fichiers ou dans les listes de marque-pages d'un navigateur. La vue arborescente a plusieurs fonctions qui spécifient la hiérarchie des éléments dans un arbre. Chaque élément dans l'arbre possède un niveau commençant à 0. Les éléments les plus élevés dans l'arbre auront un niveau à 0, Les fils de ces

éléments auront un niveau à 1, les fils en dessous auront un niveau à 2, et ainsi de suite. L'arbre demandera l'affichage pour le niveau de chaque élément afin de déterminer comment dessiner les rangées.

L'arbre dessinera les flèches d'ouverture et de fermeture pour ouvrir et fermer un élément parent ainsi que les lignes reliant les fils à leurs parents. L'arbre se chargera aussi de dessiner les rangées avec le niveau correct d'indentation. Cependant, la vue devra s'assurer de conserver l'état des niveaux de chaque ligne si nécessaire. Ce travail peut parfois être un peu délicat, mais heureusement, le module natif de vue d'arbre réalise tout le travail difficile pour nous.

Pour créer un ensemble de rangées imbriquées, tout ce que nous devons faire est d'ajouter un second `treechildren` élément à l'intérieur du parent `treeitem`. Vous pouvez ensuite ajouter des éléments à l'intérieur pour spécifier les rangées fils d'un élément. Ne mettez pas d'élément `treechildren` à l'intérieur de `treerow` car ceci ne marchera pas.

Vous pouvez répéter ce processus pour créer les arbres profondément imbriqués. Essentiellement, un élément `treeitem` peut contenir soit une simple ligne déclarée avec l'élément `treerow` ou soit un ensemble de lignes déclarées avec l'élément `treechildren`

Il y a deux autres choses que vous devez faire pour vous assurer que la hiérarchie fonctionne correctement. Premièrement, vous devez marquer l'élément `treeitem` possédant des fils comme conteneur. Il vous suffit d'ajouter l'attribut `container` comme ceci :

```
<treeitem container="true"/>
```

Cet attribut permet à l'utilisateur de double-cliquer sur le `treeitem` pour déplier ou replier les lignes intérieures. Vous pouvez faire que les lignes filles soient initialement affichées en ajoutant l'attribut `open`. Quand l'utilisateur déplie ou replie le parent, la fonction `toggleOpenState` de la vue est appelée pour basculer l'item de l'état ouvert vers fermé. Pour une vue d'arbre de type contenu, il sera ajouté un attribut `open` pour refléter l'état courant.

Le second changement vous impose de mettre l'attribut `primary` sur la première colonne. Il fait apparaître un petit triangle ou un signe + devant les cellules dans cette colonne pour indiquer les cellules pouvant être dépliées. De plus, les lignes filles sont indentées. Notez aussi que l'utilisateur ne peut pas cacher cette colonne via le petit menu déroulant à droite des colonnes.

Voici un exemple simple :

Exemple 8.2.1 :

```
<tree rows="6">
  <treecols>
    <treecol id="firstname" label="Prénoms" primary="true" flex="3"/>
    <treecol id="lastname" label="Noms" flex="7"/>
  </treecols>

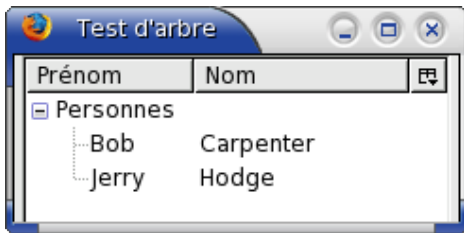
  <treechildren>
    <treeitem container="true" open="true">
      <treerow>
        <treecell label="Personnes"/>
      </treerow>

      <treechildren>
        <treeitem>
          <treerow>
            <treecell label="Bob"/>
            <treecell label="Carpenter"/>
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>
  </treechildren>
</tree>
```

```

</treeitem>
<treeitem>
  <treerow>
    <treecell label="Jerry" />
    <treecell label="Hodge" />
  </treerow>
</treeitem>
</treechildren>
</treeitem>
</treechildren>
</tree>

```



Cet exemple crée un arbre hiérarchique. Comme ce que l'on peut voir

sur l'image, un petit signe plus ou moins (souvent appelé "twisty") est apparu sur la première ligne, indiquant qu'elle contient des lignes filles. En double-cliquant sur la ligne, l'utilisateur peut ouvrir ou fermer la liste. Les lignes filles sont indentées. Notez comment la ligne "Personnes" n'a besoin que d'une colonne.

L'élément `treeitem` externe contient une seul élément `treerow` et un élément `treechildren`. Le premier crée la donnée pour la ligne parente et le deuxième contient les items fils.

Vous pouvez imbriquer autant de lignes que vous le désirez. Souvenez-vous que vous devez utiliser l'attribut `container` sur les lignes qui ont des lignes filles. La simple présence de lignes filles n'est pas suffisante pour l'indiquer, car vous pourriez avoir un conteneur sans enfants mais qui devrait quand même être traité comme un conteneur. Par exemple, un répertoire sans fichier devrait être traité comme un conteneur alors qu'un fichier ne le devrait pas.

## Plus sur les colonnes d'arbres

`enableColumnDrag` est un attribut supplémentaire que vous pouvez ajouter à l'arbre. S'il est mis à `true`, l'utilisateur peut alors déplacer les en-têtes de colonne afin de réarranger l'ordre des colonnes.

Un utilisateur voudra également changer la largeur des colonnes. Il vous suffit de placer un élément `splitter` entre chaque élément `treecol`. Une petite encoche apparaissant entre chaque en-tête de colonne pourra être déplacée par l'utilisateur pour changer la largeur d'une colonne. Vous pouvez utiliser la classe de style `tree-splitter` pour masquer l'encoche bien que la colonne puisse encore être retaillée.

Vous pouvez définir une largeur minimale ou maximale à une colonne, en utilisant les attributs `minwidth` et `maxwidth` sur les en-têtes de colonnes.

Vous pouvez mettre l'attribut `hidden` à `true` pour avoir une colonne cachée par défaut. L'utilisateur peut choisir d'afficher la colonne en la sélectionnant dans la liste déroulante à la fin de la ligne des en-têtes de colonne.

Comme avec tous les éléments, l'attribut `persist` peut être utilisé pour sauvegarder l'état des colonnes entre chaque session. Ainsi, la façon dont l'utilisateur aura choisi l'affichage des colonnes, sera automatiquement sauvegardé pour sa prochaine session. Vous pouvez sauvegarder plusieurs attributs comme indiqué dans l'exemple ci-dessous :

Exemple 8.2.2 :

```

<tree enableColumnDrag="true" flex="1">

  <treecols>
    <treecol id="runner" label="Coureur" flex="2" persist="width ordinal hidden"/>
    <splitter class="tree-splitter"/>
    <treecol id="city" label="Ville" flex="2" persist="width ordinal hidden"/>
    <splitter class="tree-splitter"/>
    <treecol id="starttime" label="Heure de départ" flex="1" persist="width ordinal hidden"/>
    <splitter class="tree-splitter"/>
    <treecol id="endtime" label="Heure d'arrivée" flex="1" persist="width ordinal hidden"/>
  </treecols>

  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="Joshua Granville"/>
        <treecell label="Vancouver"/>
        <treecell label="7:06:00"/>
        <treecell label="9:10:26"/>
      </treerow>
    </treeitem>
    <treeitem>
      <treerow>
        <treecell label="Robert Valhalla"/>
        <treecell label="Seattle"/>
        <treecell label="7:08:00"/>
        <treecell label="9:15:51"/>
      </treerow>
    </treeitem>
  </treechildren>
</tree>

```

Trois attributs des colonnes doivent être persistants, l'attribut `width` pour sauver les largeurs de colonnes, l'attribut `ordinal` contenant la position de la colonne, et l'attribut `hidden` indiquant si la colonne est visible ou invisible.

---

Dans la prochaine section, nous allons voir comment récupérer ou spécifier une sélection sur un arbre.

## 8.3 Sélection dans les arbres

Écrit par Neil Deakin. Traduit par *Medspix* (17/07/2004), mise à jour par Alain B. (03/07/2005) .  
 Page originale : <http://www.xulplanet.com/tutorials/xultu/seltree.html>

Cette section décrit comment obtenir et modifier les items sélectionnés dans un arbre.

### Accéder aux items sélectionnés d'un arbre

Chaque élément `treeitem` d'un arbre peut être sélectionné individuellement. Si vous ajoutez l'attribut `seltype` à l'arbre en lui attribuant la valeur *single*, l'utilisateur ne pourra sélectionner qu'une ligne en même temps. Sinon, l'utilisateur pourra sélectionner plusieurs lignes qui ne seront pas nécessairement contiguës. L'arbre fournit plusieurs fonctions qui peuvent être utilisées pour déterminer si un item est sélectionné.

D'abord, voyons comment nous pouvons déterminer qu'un item est sélectionné. Le gestionnaire d'évènement `onselect` peut être ajouté à l'élément `tree`. Lorsque l'utilisateur sélectionne un item de l'arbre, le gestionnaire d'évènement est appelé. L'utilisateur peut également changer la sélection en utilisant les touches de déplacement. Si l'utilisateur appuie en continu sur une touche de déplacement afin de se déplacer rapidement dans les items, le gestionnaire d'évènement n'est pas appelé jusqu'à ce que l'utilisateur arrête d'appuyer sur la touche. Ce mode de fonctionnement conduit à améliorer les performances. Il signifie

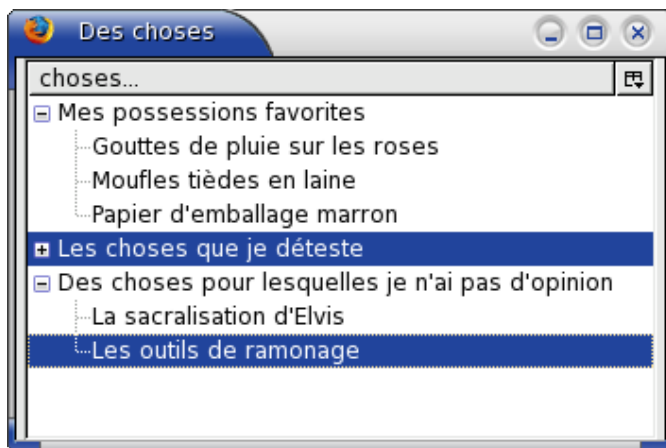
également que l'illumination du texte apparaîtra sur plusieurs items même si l'évènement de sélection n'est pas déclenché pour ces items.

La syntaxe du gestionnaire d'évènement `onselect` est présentée ci-dessous.

```
<tree id="treeset" onselect="alert('Vous avez sélectionné quelquechose !');">
```

L'arbre possède une propriété `currentIndex` qui peut être utilisée pour atteindre l'item sélectionné, où la première ligne est `0`.

Les éléments fils sont inclus dans le comptage juste après leurs parents. Ainsi, s'il y a trois items racines et que chacun a deux items fils, il y aura un total de six objets. Le premier item (à l'index `0`) sera le premier item racine. L'item suivant à l'index `1` sera son premier fils. Le second fils sera à l'index `2` et le second item parent sera à la position `3` et ainsi de suite.



Dans l'image ci-contre, il y a huit lignes affichées,

dont deux sont sélectionnées. La première ligne sélectionnée a un index de `4` et la seconde, un index de `7`. Les lignes qui ne sont pas affichées ne sont pas incluses dans le comptage d'index.

Pour les arbres qui permettent une sélection multiple, obtenir la liste des lignes sélectionnées est un peu plus compliqué. La vue de l'arbre a une propriété de sélection qui contient l'information sur les lignes sélectionnées. Cette sélection sera un objet `TreeSelection`. La vue n'a pas besoin d'implémenter cet objet elle-même, car l'arbre aura assigné un objet de sélection à la propriété de sélection de la vue lorsque celle-ci a été attachée à l'arbre. À partir de l'arbre, vous pouvez obtenir la sélection en utilisant la propriété `view` de l'arbre et ensuite la propriété `selection` de la vue. Vous pouvez utiliser les méthodes de l'objet de sélection pour récupérer l'ensemble des items sélectionnés ou pour modifier la sélection.

Comme les items sélectionnés dans une sélection multiple ne sont pas nécessairement contigus, vous pouvez récupérer chaque bloc de sélection contiguë en utilisant les fonctions `getRangeCount` et `getRangeAt`. La première fonction retourne le nombre de plages de sélection effectuées. Si seule une valeur a été sélectionnée, la valeur retournée sera `1`. Vous pourrez alors écrire une boucle sur le nombre de plages, en appelant `getRangeAt` pour obtenir les bons index du début et de la fin d'une plage.

La fonction `getRangeAt` comporte trois arguments. Le premier est la plage d'index. Le second est un objet qui sera rempli par la fonction avec l'index du premier item sélectionné. Le troisième argument est un objet qui sera rempli avec l'index du dernier item sélectionné.

Par exemple :

```
var start = new Object();
var end = new Object();
var numRanges = tree.view.selection.getRangeCount();

for (var t=0; t<numRanges; t++){
```



```
tree.view.selection.getRangeAt(t,start,end);
for (var v=start.value; v<=end.value; v++){
    alert("Item "+v+" sélectionné.");
}
}
```

Nous avons créé deux objets appelés `start` et `end`. Ensuite, nous lançons une boucle sur les plages de la sélection dont le nombre a été retourné par la fonction `getRangeCount`. La fonction `getRangeAt` est appelée avec l'index de la plage, et les objets `start` et `end`. Cette fonction renseignera les index `start` et `end` en leur attribuant la propriété `value`. Donc, si la première plage commence au troisième item et se termine au septième, `start.value` vaudra **2** (souvenez-vous que les index commencent à 0, donc nous en décrémenteons de un) et `end.value` vaudra **6**. Un message d'alerte est affichée pour chaque index.

Si vous souhaitez seulement connaître si une ligne précise a été sélectionnée, vous pouvez utiliser la fonction `isSelected`. Elle prend l'index de la ligne en argument et retourne *true* si la ligne est sélectionnée.

```
alert(tree.view.selection.isSelected(3));
```

## Modifier la sélection dans un arbre

L'objet de sélection dispose de plusieurs fonctions qui peuvent être utilisées pour changer une sélection. La fonction la plus simple est `select` qui dé-sélectionne n'importe quelles lignes actuellement sélectionnées et peut en sélectionner une spécifique. Par exemple, le code suivant sélectionnera la ligne d'index **5** :

```
tree.view.selection.select(5);
```

Notez que vous ne devez pas modifier la propriété `currentIndex` pour changer la sélection. Vous devez plutôt utiliser la fonction `select` de sélection comme indiqué dans l'exemple précédent. Vous pouvez sélectionner toutes les lignes avec la fonction `selectAll`. Notez que les lignes imbriquées à l'intérieur de containers qui n'ont pas été ouverts, ne seront pas sélectionnées. Naturellement, cette fonction n'est valable que pour des arbres à sélections multiples. Il y a aussi les fonctions `clearSelection` pour effacer la sélection et `invertSelection` pour inverser la sélection, c'est-à-dire dé-sélectionner toutes les lignes sélectionnées et sélectionner les autres.

Pour sélectionner des lignes spécifiques, utiliser la fonction `rangedSelect` qui sélectionne toutes les lignes entre deux indices. Voici un exemple de sélection des lignes comprises entre les index **2** et **7**. Notez que les lignes **2** et **7** seront incluses dans la sélection.

```
tree.view.selection.rangedSelect(2,7,true);
```

L'argument final indique s'il s'agit d'un ajout à la sélection courante. Si la valeur est *true*, la plage sera ajoutée à la sélection existante. Si la valeur est *false*, toutes les lignes préalablement sélectionnées seront d'abord dé-sélectionnées. Finalement, la fonction `clearRange` peut être utilisée pour dé-sélectionner une plage de lignes, les lignes en dehors de cette plage n'étant pas affectées.

```
tree.view.selection.clearRange(2,7);
```

---

Nous allons maintenant apprendre comment créer une vue personnalisée pour un arbre.

## 8.4 Vues d'arbre personnalisées

Écrit par Neil Deakin. Traduit par *Chaddai Fouché* (20/07/2004), mise à jour par Alain B. (08/07/2005) .  
Page originale : <http://www.xulplanet.com/tutorials/xultu/treeview.html>

Les vues sous formes d'arbres permettent d'afficher des données dans une arborescence.

## Créer une vue personnalisée

Jusqu'à présent, nous avons utilisé la construction interne d'une vue d'arbre. Dans cette section, nous verrons la création d'une vue personnalisée. Elle devient nécessaire dès lors que la quantité de données devient trop importante ou que celles-ci soient arrangées de manière trop complexe. Par exemple, l'utilisation d'éléments `treeitem` ne serait plus viable en terme de performance avec plusieurs milliers de lignes. Vous pouvez également utiliser une vue personnalisée pour afficher des données obtenues par calculs. Puisque la vue peut stocker et récupérer de la meilleure manière possible les données en fonction de leur type, l'arbre peut être utilisé même si plusieurs milliers de lignes doivent être affichées.

Pour implémenter une vue personnalisée, vous devez créer un objet qui implémente l'interface `nsITreeView`. Vous pouvez créer ces objets en javascript, mais vous aurez besoin d'un objet séparé pour chaque arbre. Naturellement, comme la vue d'arbre personnalisée est utilisée, la vue de contenu d'arbre ne le sera pas, donc les éléments `treeitem`, `treerow` et `treecell` n'auront aucun sens car la vue d'arbre obtient ses données ailleurs. Ainsi, vous pouvez simplement laisser l'élément `treechildren` vide. L'exemple suivant vous le montre :

```
<tree id="my-tree" flex="1">
  <treecols>
    <treecol id="namecol" label="Nom" flex="1"/>
    <treecol id="datecol" label="Date" flex="1"/>
  </treecols>
  <treechildren/>
</tree>
```

Pour assigner les données à afficher dans l'arbre, un objet de visualisation doit être créé, il sera utilisé pour indiquer la valeur de chaque cellule, le nombre total de lignes plus d'autres informations optionnelles. L'arbre appellera des méthodes de visualisation pour obtenir les informations dont il a besoin pour son affichage.

En général, bien que la vue d'arbre dispose de plus d'une trentaine de fonctions pouvant être implémentées, il vous suffira de ne définir que celles appelées par l'arbre. Les trois méthodes que vous devrez implémenter sont décrites ci-dessous :

*rowCount*

Cette propriété doit se voir assigner le nombre total de lignes dans l'arbre.

*getCellText( ligne, colonne )*

Cette méthode doit retourner le texte contenu à la ligne et la colonne spécifiées. Elle sera appelée pour afficher les données de chaque cellule. Les lignes sont spécifiées par des valeurs numériques qui commencent à 0. Les colonnes sont spécifiées par les valeurs de l'attribut `id` de la colonne. À partir de Mozilla 1.8, les colonnes sont définies par un objet `TreeColumn`.

*setTree( arbre )*

Cette méthode est appelée une seule fois pour affecter l'objet arbre à la vue.

Voici un exemple de définition d'un tel objet qui peut avoir le nom que vous souhaitez :

```
var treeView = {
  rowCount : 10000,
  getCellText : function(row,column){
    if (column == "namecol") return "Ligne "+row;
    else return "20 juillet";
  },
  setTree: function(treebox){ this.treebox = treebox; },
  isContainer: function(row){ return false; },
  isSeparator: function(row){ return false; },
  isSorted: function(row){ return false; },
  getLevel: function(row){ return 0; },
  getImageSrc: function(row,col){ return null; },
  getRowProperties: function(row,props){},
}
```

```

    getCellProperties: function(row,col,props){},
    getColumnProperties: function(colid,col,props){}
};

```

Les fonctions de l'exemple ci-dessus que nous n'avons pas décrites n'ont pas besoin d'effectuer une quelconque action, mais elles doivent être définies car l'arbre les appelle pour rassembler des informations supplémentaires.

Cet exemple peut-être utilisé pour un arbre avec 10000 lignes. Le contenu des cellules de la première colonne sera *Ligne X* où X sera le numéro de la ligne. Le contenu des cellules de la seconde colonne sera *20 juillet*. La structure if dans la fonction `getCellText` compare la colonne au texte *namecol*. Ce texte *namecol* correspond à l'id du premier élément *treecol* dans l'exemple précédent. Cet exemple est très simplifié bien sûr – en réalité vous aurez des données bien plus complexes dans chaque cellule.

L'étape finale est d'associer l'objet de vue avec l'arbre. L'arbre a une propriété `view`, à laquelle on peut assigner l'objet déclaré ci-dessus. Nous pouvons assigner une valeur à cette propriété à tout moment pour attribuer une vue à l'arbre, ou en changer.

```

function setView()
{
    document.getElementById('my-tree').view = treeView;
}

```

Le code source suivant présente l'exemple en entier. Un script intégré au fichier XUL a été utilisé ici pour simplifier l'exemple. En temps normal vous mettriez le script dans un fichier de script externe.

#### Exemple 8.4.1 :

```

<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window title="Exemple d'arbre" id="tree-window"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  onload="setView();" >

<script>
var treeView = {
  rowCount : 10000,
  getCellText : function(row,column){
    if (column == "namecol") return "Ligne "+row;
    else return "20 juillet";
  },
  setTree: function(treebox){ this.treebox = treebox; },
  isContainer: function(row){ return false; },
  isSeparator: function(row){ return false; },
  isSorted: function(row){ return false; },
  getLevel: function(row){ return 0; },
  getImageSrc: function(row,col){ return null; },
  getRowProperties: function(row,props){},
  getCellProperties: function(row,col,props){},
  getColumnProperties: function(colid,col,props){}
};

function setView()
{
  document.getElementById('my-tree').view=treeView;
}
</script>

<tree id="my-tree" flex="1">

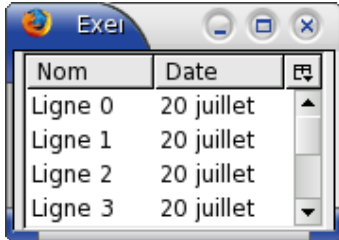
```

```

<treecols>
  <treecol id="namecol" label="Nom" flex="1" />
  <treecol id="datecol" label="Date" flex="1" />
</treecols>
<treechildren/>
</tree>

</window>

```



Sur l'image, vous voyez deux colonnes, chacune obtenant ses données par

l'intermédiaire de la fonction `getCellText`. La fonction `setView` a été appelée par le gestionnaire `onload` de la fenêtre (`window`), mais vous pouvez aussi changer la vue plus tard si vous le souhaitez. Vous pouvez la changer à n'importe quel instant.

Il faut noter que la fonction `getCellText` est seulement appelée quand cela est nécessaire pour afficher le contenu. Dans l'exemple de 10000 lignes ci-dessus, `getCellText` est seulement appelée pour les cellules qui sont actuellement affichées. Sur cette image, seules quatre lignes sont affichées, donc la fonction `getCellText` a été appelée 8 fois. Elle sera appelée pour les autres lignes lorsque l'utilisateur les fera défiler. Cette méthode rend l'arbre beaucoup plus efficace.

Notez que l'objet de vue est aussi disponible pour des arbres prédéfinis dans votre installation. Vous pouvez les utiliser pour récupérer les libellés de leurs cellules et d'autres informations.

L'[interface nsITreeView](#) liste toutes les méthodes et propriétés que vous pouvez implémenter pour la vue d'un arbre.

Dans la prochaine section, nous verrons comment RDF sert à remplir des arbres automatiquement.

## 8.5 Détails sur les vues d'arbres

Écrit par [Neil Deakin](#). Traduit par **Romain D.** (03/05/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/treeviewdet.html>

Cette section décrira quelques fonctionnalités supplémentaires des vues d'arbre.

### Créer une vue hiérarchique personnalisée

Dans la précédente section, nous avons créé une vue d'arbre simple qui était implémentée avec seulement un minimum de fonctionnalités. À présent, regardons quelques fonctions supplémentaires que les vues peuvent implémenter. Ici, nous examinerons comment créer un ensemble hiérarchique d'items utilisant la vue. C'est un processus relativement astucieux qui implique de conserver une trace des items qui sont des enfants et également des lignes qui sont ouvertes et fermées.

Chaque ligne dans l'arbre possède un niveau d'imbrication. Les lignes les plus hautes ont un niveau 0, les enfants de ces lignes ont un niveau 1, leurs enfants le niveau 2 et ainsi de suite. L'arbre interroge la vue pour chaque ligne en appelant sa méthode `getLevel` pour connaître le niveau de cette ligne. La vue devra retourner 0 pour les lignes les plus à l'extérieur et des valeurs plus élevées pour les lignes intérieures. L'arbre utilisera cette information pour déterminer la structure hiérarchique de ces lignes.

En complément de la méthode `getLevel`, la fonction `hasNextSibling` retourne pour une ligne donnée la valeur *true* si la ligne qui suit existe et est de même niveau. Cette fonction est utilisée, spécifiquement, pour dessiner l'imbrication des lignes le long du côté de l'arbre.

La méthode `getParentIndex` est supposée retourner la ligne parente d'une ligne donnée, c'est-à-dire, la ligne précédente avec une valeur d'imbrication inférieure. Toutes ces méthodes doivent être implémentées par la vue pour que les enfants soient proprement manipulés.

Il y a également trois fonctions, `isContainer`, `isContainerEmpty` et `isContainerOpen` qui sont utilisées pour manipuler un item parent dans l'arbre. Naturellement, la méthode `isContainer` devrait retourner *true* si une ligne est un conteneur pouvant contenir des enfants. La méthode `isContainerEmpty` devrait renvoyer *true* si une ligne est un conteneur vide, par exemple, un répertoire sans fichiers à l'intérieur. La vue a besoin de conserver une trace des items qui sont ouverts ou fermés, la méthode `isContainerOpen` est donc utilisée pour le déterminer. L'arbre appellera cette méthode pour déterminer quels conteneurs sont ouverts et lesquels sont fermés. Notez que l'arbre n'appellera ni `isContainerEmpty`, ni `isContainerOpen` pour les lignes qui ne sont pas conteneurs en se basant sur la valeur de retour de la méthode `isContainer`.

Un conteneur peut être affiché différemment qu'un non-conteneur. Par exemple, un conteneur peut avoir un icône de dossier devant lui. Une feuille de styles peut être utilisée pour styler des items en se basant sur diverses propriétés telles que l'ouverture d'une ligne conteneur. Le stylage sera décrit dans une prochaine section. Un conteneur non vide apparaîtra avec une poignée (NdT : "twisty", petit '+' ou '-' permettant de développer et replier l'arborescence) à côté permettant à l'utilisateur d'ouvrir ou de fermer la ligne pour voir les items enfants. Les conteneurs vides n'auront pas de poignées, mais seront toujours considérés comme des conteneurs.

Lorsque l'utilisateur clique sur la poignée pour ouvrir une ligne, l'arbre appellera la méthode de vue `toggleOpenState`. La vue met en oeuvre les opérations nécessaires pour intégrer les lignes enfants et mettre à jour l'arbre avec les nouvelles lignes.

Voici un récapitulatif des méthodes nécessaires pour implémenter des vues hiérarchiques :

```
getLevel(ligne)
hasNextSibling(ligne, apresIndex)
getParentIndex(ligne)
isContainer(ligne)
isContainerEmpty(ligne)
isContainerOpen(ligne)
toggleOpenClose(ligne)
```

L'argument `apresIndex` de la fonction `hasNextSibling` est utilisée pour une raison d'optimisation, afin de démarrer la recherche à partir de la prochaine ligne soeur (ligne au même niveau) après ce point. Par exemple, l'appelant pourrait déjà connaître la position de la prochaine ligne soeur. Imaginez une situation où une ligne possède des sous-lignes et que ces sous-lignes aient des lignes enfants dont quelques unes sont ouvertes. Dans ce cas, la détermination de l'index de la prochaine ligne soeur prendrait du temps dans certaines implémentations.

Regardons cela ensemble dans un exemple simple qui construit un arbre à partir d'un tableau. Nous l'examinerons morceau par morceau.

```
<window onload="init();"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<tree id="elementList" flex="1">
  <treecols>
    <treecol id="element" label="Élément" primary="true" flex="1"/>
  </treecols>
```

```

    <treechildren/>
</tree>

</window>

```

Nous utilisons un arbre simple qui ne contient pas de données dans `treechildren`. La fonction `init` est appelée au chargement de la fenêtre pour initialiser l'arbre. Elle définit simplement la vue personnalisée en récupérant l'arbre et en définissant sa propriété `view`. Nous définirons `treeView` plus tard.

```

function init() {
    document.getElementById("elementList").view = treeView;
}

```

La vue d'arbre personnalisée aura besoin d'implémenter un certain nombre de méthodes lesquelles parmi les plus importantes seront examinées individuellement. Cet arbre supporte uniquement un seul niveau de parenté avec un niveau enfant interne, mais il peut être étendu pour supporter des niveaux supplémentaires sans trop d'efforts. Tout d'abord nous définirons deux structures pour conserver les données de l'arbre, la première contiendra une carte relationnelle entre les parents et leurs éventuels enfants, et la seconde contiendra un tableau des items visibles. Souvenez vous qu'une vue doit conserver elle même une trace des items qui sont visibles.

```

var treeView = {
    childData : {
        Solides: ["Argent", "Or", "Plomb"],
        Liquides: ["Mercure"],
        Gaz: ["Hélium", "Azote"]
    },

    visibleData : [
        ["Solides", true, false],
        ["Liquides", true, false],
        ["Gaz", true, false]
    ],
};

```

La structure `childData` contient un tableau des enfants pour chacun des trois noeuds parents. Le tableau `visibleData` commence avec seulement trois items visibles, les trois items de haut niveau. Des items seront ajoutés et supprimés depuis ce tableau quand les items sont ouverts ou fermés. Essentiellement, quand une ligne parente est ouverte, l'enfant sera récupéré depuis la carte `childData` et inséré dans le tableau `visibleData`. Par exemple, si la ligne *Liquides* est ouverte, le tableau correspondant depuis `childData`, lequel dans ce cas contient seulement l'enfant *Mercure*, sera inséré dans le tableau `visibleData` après *Liquides* mais avant *Gaz*. La taille du tableau sera incrémentée de un. Les deux valeurs booléennes présentes dans chaque ligne dans la structure `visibleData` indiquent respectivement si une ligne est un conteneur et si elle est ouverte. Évidemment, le nouvel enfant inséré aura ces deux valeurs initialisées à *false*.

Ensuite, nous avons besoin d'implémenter l'interface de vue de l'arbre. Tout d'abord, les fonctions simples :

```

treeBox: null,
selection: null,

get rowCount() { return this.visibleData.length; },
setTree: function(treeBox) { this.treeBox = treeBox; },
getCellText: function(idx, column) { return this.visibleData[idx][0]; },
isContainer: function(idx) { return this.visibleData[idx][1]; },
isContainerOpen: function(idx) { return this.visibleData[idx][2]; },
isContainerEmpty: function(idx) { return false; },
isSeparator: function(idx) { return false; },
isSorted: function() { return false; },
isEditable: function(idx, column) { return false; },

```

La fonction `rowCount` retournera la taille du tableau `visibleData`. Notez qu'elle devrait retourner le nombre courant de lignes visibles, pas le total. Donc, au début, seulement trois items sont visibles et la valeur retournée par `rowCount` devrait être trois, même si six lignes sont cachées.

La fonction `setTree` sera appelée pour définir l'objet boîte de l'arbre. L'objet boîte de l'arbre est un type spécialisé d'objet boîte spécifique aux arbres qui sera examiné en détail dans la prochaine section. Il est utilisé pour aider à la représentation graphique de l'arbre. Dans cet exemple, nous avons seulement besoin d'une fonction de l'objet boîte capable de redessiner l'arbre quand des items sont ajoutés ou supprimés.

Les fonctions `getCellText`, `isContainer` et `isContainerOpen` retournent juste l'élément correspondant dans le tableau `visibleData`. Enfin, les fonctions restantes peuvent retourner *false* puisque nous n'avons pas besoin de leurs fonctionnalités. Si nous avions eu une ligne qui n'aurait pas d'enfant, nous aurions implémenté la fonction `isContainerEmpty` pour laquelle retourne *true* pour ces éléments.

```
getParentIndex: function(idx) {
    if (this.isContainer(idx)) return -1;
    for (var t = idx - 1; t >= 0 ; t--) {
        if (this.isContainer(t)) return t;
    }
},
```

La fonction `getParentIndex` sera nécessaire pour trouver le parent d'un index donné. Dans notre exemple simple, il y a juste deux niveaux, donc nous savons que les conteneurs n'ont pas de parents, la valeur `-1` est retournée pour ces items. Autrement, nous devrions parcourir les lignes en arrière pour rechercher celle qui est un conteneur. Ensuite, la fonction `getLevel`.

```
getLevel: function(idx) {
    if (this.isContainer(idx)) return 0;
    return 1;
},
```

La fonction `getLevel` est simple. Elle retourne juste `0` pour une ligne conteneur et `1` pour une ligne non-conteneur. Si nous voulions ajouter un niveau supplémentaire d'enfants, ces lignes auraient un niveau de `2`.

```
hasNextSibling: function(idx, after) {
    var thisLevel = this.getLevel(idx);
    for (var t = idx + 1; t < this.visibleData.length; t++) {
        var nextLevel = this.getLevel(t);
        if (nextLevel == thisLevel) return true;
        else if (nextLevel < thisLevel) return false;
    }
},
```

La fonction `hasNextSibling` doit retourner *true* s'il existe une ligne suivant une autre de même niveau. Le code ci-dessus utilise une méthode basique qui consiste à parcourir les lignes après celle donnée, en retournant *true* si une ligne de même niveau existe et *false* si une ligne de niveau inférieur est rencontrée. Dans cet exemple simple, cette méthode est bonne, mais un arbre avec davantage de données aura besoin d'utiliser une méthode optimisée pour déterminer si une ligne suivante soeur existe.

La dernière fonction est `toggleOpenState` qui est la plus complexe. Elle a besoin de modifier le tableau `visibleItems` lorsqu'une ligne est ouverte ou fermée.

```
toggleOpenState: function(idx) {
    var item = this.visibleData[idx];
    if (!item[1]) return;
```

```

if (item[2]) {
    item[2] = false;

    var thisLevel = this.getLevel(idx);
    var deletecount = 0;
    for (var t = idx + 1; t < this.visibleData.length; t++) {
        if (this.getLevel(t) > thisLevel) deletecount++;
        else break;
    }
    if (deletecount) {
        this.visibleData.splice(idx + 1, deletecount);
        this.treeBox.rowCountChanged(idx + 1, -deletecount);
    }
}
else {
    item[2] = true;

    var label = this.visibleData[idx][0];
    var toinsert = this.childData[label];
    for (var i = 0; i < toinsert.length; i++) {
        this.visibleData.splice(idx + i + 1, 0, [toinsert[i], false]);
    }
    this.treeBox.rowCountChanged(idx + 1, toinsert.length);
}
},

```

D'abord nous vérifions si la ligne est un conteneur. Si elle ne l'est pas, la fonction retourne juste que les non-conteneurs ne peuvent pas être ouverts ou fermés. Comme le troisième élément du tableau (celui avec l'index 2) indique si une ligne est ouverte ou fermée, nous utilisons deux blocs de code, le premier pour fermer une ligne et le second pour ouvrir une ligne. Examinons chaque bloc de code, mais en commençant par le second chargé d'ouvrir une ligne.

```

item[2] = true;

var label = this.visibleData[idx][0];
var toinsert = this.childData[label];
for (var i = 0; i < toinsert.length; i++) {
    this.visibleData.splice(idx + i + 1, 0, [toinsert[i], false]);
}
this.treeBox.rowCountChanged(idx + 1, toinsert.length);

```

La première ligne de code définit la ligne item comme étant ouverte dans le tableau, ainsi le prochain appel de la fonction `toggleOpenState` aura l'indication de devoir fermer la ligne. Ensuite, regardons les données pour la ligne dans la carte `childData`. Le résultat est que la variable 'toinsert' sera définie avec un des tableaux enfants, par exemple `["Argent", "Or", "Plomb"]` si la ligne *Solides* est celle qui est en train d'être ouverte. Ensuite, nous utilisons la fonction de tableau `splice` pour insérer une nouvelle ligne pour chaque item. Pour *Solides*, trois items seront insérés.

Enfin, la fonction de boîte d'arbre `rowCountChanged` a besoin d'être appelée. Rappelez-vous que l'objet `treeBox` est un objet de boîte d'arbre qui a été défini plus tôt par un appel de la fonction `setTree`. L'objet de boîte d'arbre sera créé par l'arbre pour vous et vous pourrez appeler ses fonctions. Dans ce cas, nous utilisons la fonction `rowCountChanged` pour informer l'arbre que quelques lignes de données ont été ajoutées. L'arbre redessiner son contenu avec comme résultat que les lignes enfants apparaîtront à l'intérieur du conteneur. Les autres fonctions implémentées ci-dessus telles que `getLevel` et `isContainer` sont utilisées par l'arbre pour déterminer son affichage.

La fonction `rowCountChanged` prend deux arguments, l'index où la première ligne a été insérée et le nombre de lignes à ajouter. Dans le code ci-dessus nous indiquons que la ligne de départ est la valeur de `idx` plus un, elle sera la première ligne enfant sous le parent. L'arbre utilisera cette information et ajoutera l'espace nécessaire pour le nombre approprié de lignes en poussant les lignes suivantes vers le bas. Assurez



vous de fournir le nombre correct ou bien l'arbre pourrait se redessiner incorrectement ou essayer de dessiner plus de lignes que nécessaire.

Le code suivant est utilisé pour supprimer des lignes quand une ligne est fermée.

```
item[2] = false;

var thisLevel = this.getLevel(idx);
var deletecount = 0;
for (var t = idx + 1; t < this.visibleData.length; t++) {
    if (this.getLevel(t) > thisLevel) deletecount++;
    else break;
}
if (deletecount) {
    this.visibleData.splice(idx + 1, deletecount);
    this.treeBox.rowCountChanged(idx + 1, -deletecount);
}
```

Premièrement, l'item est déclaré comme fermé dans le tableau. Ensuite, nous scannons les lignes suivantes jusqu'à ce que nous arrivions à une ligne de même niveau. Toutes celles qui ont un niveau supérieur auront besoin d'être supprimées, mais une ligne de même niveau sera le prochain conteneur qui ne devra pas être supprimée.

Enfin, nous utilisons la fonction `splice` pour supprimer les lignes du tableau `visibleData` et appelons la fonction `rowCountChanged` pour redessiner l'arbre. Lors de la suppression des lignes, vous aurez besoin de fournir un chiffre négatif correspondant au nombre de lignes à supprimer.

Il existe plusieurs autres fonctions de vue pouvant être implémentées mais nous n'en avons pas l'utilité dans cet exemple, donc nous créons des fonctions qui ne font rien ici. Elles sont placées à la fin de notre exemple complet, comme montré ici :

Exemple 8.5.1 :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window onload="init();"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<tree id="elementList" flex="1">
    <treecols>
        <treecol id="element" label="Élément" primary="true" flex="1"/>
    </treecols>
    <treechildren/>
</tree>

<script>
<![CDATA[

var treeView = {
    childData : {
        Solides: ["Argent", "Or", "Plomb"],
        Liquides: ["Mercure"],
        Gaz: ["Hélium", "Azote"]
    },

    visibleData : [
        ["Solides", true, false],
        ["Liquides", true, false],
        ["Gaz", true, false]
    ],
```

```

treeBox: null,
selection: null,

get rowCount() { return this.visibleData.length; },
setTree: function(treeBox) { this.treeBox = treeBox; },
getCellText: function(idx, column) { return this.visibleData[idx][0]; },
isContainer: function(idx) { return this.visibleData[idx][1]; },
isContainerOpen: function(idx) { return this.visibleData[idx][2]; },
isContainerEmpty: function(idx) { return false; },
isSeparator: function(idx) { return false; },
isSorted: function() { return false; },
isEditable: function(idx, column) { return false; },

getParentIndex: function(idx) {
    if (this.isContainer(idx)) return -1;
    for (var t = idx - 1; t >= 0 ; t--) {
        if (this.isContainer(t)) return t;
    }
},
getLevel: function(idx) {
    if (this.isContainer(idx)) return 0;
    return 1;
},
hasNextSibling: function(idx, after) {
    var thisLevel = this.getLevel(idx);
    for (var t = idx + 1; t < this.visibleData.length; t++) {
        var nextLevel = this.getLevel(t);
        if (nextLevel == thisLevel) return true;
        else if (nextLevel < thisLevel) return false;
    }
},
toggleOpenState: function(idx) {
    var item = this.visibleData[idx];
    if (!item[1]) return;

    if (item[2]) {
        item[2] = false;

        var thisLevel = this.getLevel(idx);
        var deletecount = 0;
        for (var t = idx + 1; t < this.visibleData.length; t++) {
            if (this.getLevel(t) > thisLevel) deletecount++;
            else break;
        }
        if (deletecount) {
            this.visibleData.splice(idx + 1, deletecount);
            this.treeBox.rowCountChanged(idx + 1, -deletecount);
        }
    }
    else {
        item[2] = true;

        var label = this.visibleData[idx][0];
        var toinsert = this.childData[label];
        for (var i = 0; i < toinsert.length; i++) {
            this.visibleData.splice(idx + i + 1, 0, [toinsert[i], false]);
        }
        this.treeBox.rowCountChanged(idx + 1, toinsert.length);
    }
},

getImageSrc: function(idx, column) {},
getProgressMode : function(idx, column) {},
getCellValue: function(idx, column) {},
cycleHeader: function(col, elem) {},

```

```

selectionChanged: function() {},
cycleCell: function(idx, column) {},
performAction: function(action) {},
performActionOnCell: function(action, index, column) {},
getRowProperties: function(idx, column, prop) {},
getCellProperties: function(idx, column, prop) {},
getColumnProperties: function(column, element, prop) {}
};

function init() {
    document.getElementById("elementList").view = treeView;
}

]]></script>

</window>

```

---

Ensuite, nous verrons plus en détails l'objet de boîte d'arbre.

## 8.6 Les objets boîtes des arbres

Écrit par Neil Deakin. Traduit par *Alain B.* (15/06/2005).

Page originale : <http://www.xulplanet.com/tutorials/xultu/treeboxobject.html>

Cette section va décrire l'objet de boîte d'arbre qui est utilisé pour gérer l'affichage d'un arbre.

### À propos de l'objet de boîte

Les objets de boîte ont été décrits dans une section précédente. L'objet de boîte d'arbre est un objet de boîte spécial utilisé spécifiquement pour les arbres. La boîte d'arbre implémente l'interface `TreeBoxObject`.

Nous avons déjà vu la fonction `rowCountChanged` de l'objet de boîte d'arbre dans la section précédente. Elle est employée pour indiquer qu'une ou plusieurs lignes de l'arbre ont été ajoutées ou enlevées. L'arbre rafraîchira l'affichage de la zone affectée. Vous n'avez pas besoin d'appeler la fonction `rowCountChanged` lorsqu'une ligne a simplement été modifiée, comme par exemple lors du changement du libellé d'une cellule. Dans ce cas, d'autres fonctions d'affichage peuvent être utilisées. La plus simple est la fonction `invalidateRow` qui rafraîchit l'affichage d'une ligne spécifique d'un arbre. L'arbre appellera la vue pour obtenir les données mises à jour et actualise son contenu à l'écran.

Les autres fonctions de redessinage sont `invalidateCell` pour l'actualisation d'une unique cellule, `invalidateColumn` pour l'actualisation d'une colonne, `invalidateRange` pour l'actualisation d'une plage de lignes, ou la fonction `invalidate` pour l'actualisation de l'arbre entier. Notez que l'actualisation de l'affichage n'aura pas lieu avant que les scripts aient terminé car Mozilla n'effectue pas le redessinage en tâche de fond.

Vous pouvez également faire défiler l'arbre en utilisant quatre différentes méthodes similaires à celles disponibles pour les menus déroulants. La fonction `scrollToRow` peut être utilisée pour faire le défilement jusqu'à une ligne particulière. Voici un simple exemple :

Exemple 8.6.1 :

```

<script>
function doScroll()
{
    var value = document.getElementById("tbox").value;
    var tree = document.getElementById("thetree");

    var boxobject = tree.boxObject;

```

```

    boxobject.QueryInterface(Components.interfaces.nsITreeBoxObject);
    boxobject.scrollToRow(value);
}
</script>

<tree id="thetree" rows="4">
  <treecols>
    <treecol id="row" label="Ligne" primary="true" flex="1"/>
  </treecols>
  <treechildren>
    <treeitem label="Ligne 0"/>
    <treeitem label="Ligne 1"/>
    <treeitem label="Ligne 2"/>
    <treeitem label="Ligne 3"/>
    <treeitem label="Ligne 4"/>
    <treeitem label="Ligne 5"/>
    <treeitem label="Ligne 6"/>
    <treeitem label="Ligne 7"/>
    <treeitem label="Ligne 8"/>
    <treeitem label="Ligne 9"/>
  </treechildren>
</tree>

<hbox align="center">
  <label value="Défile jusqu'à la ligne :"/>
  <textbox id="tbox"/>
  <button label="Défile" oncommand="doScroll();" />
</hbox>

```

Notez que nous utilisons l'attribut `rows` sur l'élément `tree` pour définir que seulement quatre lignes sont affichées à la fois. Ainsi, il est plus facile de se représenter l'exemple. Notez également que la première ligne commence à 0.

La fonction `doScroll` récupère l'objet de boîte et appelle la fonction `scrollToRow` avec comme argument la valeur saisie dans le champ texte. Vous noterez que l'objet de boîte d'arbre peut être obtenu de la même manière qu'avec d'autres objets de boîte, en utilisant la propriété `boxObject`, mais nous devons toutefois appeler `QueryInterface` pour invoquer l'objet de boîte très spécifique qu'est l'arbre. Les fonctions les plus générales de l'objet de boîte sont également disponibles pour les arbres.

Les méthodes supplémentaires de défilement incluent les fonctions `scrollByLines`, `scrollByPages`, et `ensureRowIsVisible`. la fonction `scrollByLines` fait défiler vers le haut ou vers le bas d'un certain nombre de lignes ; un nombre positif fait défiler vers le bas, et un nombre négatif fait défiler vers le haut. La fonction `scrollByPages` fait défiler d'un nombre de pages. Elle est appelée automatiquement lorsque l'utilisateur appuie sur une touche **Page Up** ou **Page Down** avec un arbre qui a le focus. Une page est égale au nombre de lignes visibles. Par exemple, si un arbre affiche 10 lignes en même temps, une page sera équivalente à 10 lignes. C'est une méthode pratique dès lors que l'utilisateur redimensionne un arbre flexible, la taille de la page augmentera ou diminuera automatiquement sans avoir à la recalculer manuellement. Il n'est pas trop difficile de calculer cette taille manuellement car l'objet de boîte d'arbre fournit également une fonction `getPageLength` qui retourne le nombre de lignes dans une page. Dans l'exemple de défilement ci-dessus, `getPageLength` retournerait 4.

Notez que dans Firefox 1.0 et Mozilla 1.7 et les versions plus récentes, la fonction `getPageLength` est plutôt appelée `getPageCount`. Le nom a été changée en `getPageLength` afin d'éviter les confusions avec une fonction qui ne retourne pas le nombre de pages d'un arbre, mais la taille de chaque page. Vous pouvez déterminer le nombre de pages en divisant le nombre total de lignes par la taille d'une page.

La fonction `ensureRowIsVisible` fera défiler l'arbre jusqu'à une ligne comme avec la fonction `scrollToRow` seulement si la ligne n'est actuellement pas visible.

## Coordonnées d'une cellule

Certaines des fonctions les plus intéressantes d'un objet de boîte d'arbre sont utilisées pour obtenir les parties d'un arbre à des coordonnées spécifiques. La fonction `getCellAt` peut être utilisée pour obtenir une cellule précise située à un emplacement défini en pixels, tandis que la fonction `getRowAt` peut être utilisée pour obtenir une ligne à un emplacement spécifique. La fonction `getRowAt` prend deux arguments qui sont les coordonnées `x` et `y`.

```
tree.boxObject.getRowAt( 50, 100 );
```

Cet exemple retournera l'index de la ligne ayant une position horizontale de *50* et verticale de *100*. Naturellement, la coordonnée `x` semble ne pas avoir beaucoup de sens dès lors que la ligne occupe tout l'espace horizontal de l'arbre. Une chose importante est de noter que les coordonnées sont mesurées à partir du coin en haut et à gauche du document et non de l'arbre lui même. Il est donc facile de passer à ces fonctions les coordonnées événementielles de l'objet `event`, comme avec la fonction `getCellAt` dans l'exemple suivant.

Exemple 8.6.2 :

```
<script>
function updateFields(event)
{
    var row = {}, column = {}, part = {};
    var tree = document.getElementById("thetree");

    var boxobject = tree.boxObject;
    boxobject.QueryInterface(Components.interfaces.nsITreeBoxObject);
    boxobject.getCellAt(event.clientX, event.clientY, row, column, part);

    if (typeof column.value != "string") column.value = column.id;

    document.getElementById("row").value = row.value;
    document.getElementById("column").value = column.value;
    document.getElementById("part").value = part.value;
}
</script>

<tree id="thetree" flex="1" onmousemove="updateFields(event);">
    <treecols>
        <treecol id="ustensile" label="Ustensiles" primary="true" flex="1"/>
        <treecol id="nombre" label="Nombre" flex="1"/>
    </treecols>
    <treechildren>
        <treeitem>
            <treerow>
                <treecell label="Fourchette"/>
                <treecell label="5"/>
            </treerow>
        </treeitem>
        <treeitem>
            <treerow>
                <treecell label="Couteau"/>
                <treecell label="2"/>
            </treerow>
        </treeitem>
        <treeitem>
            <treerow>
                <treecell label="Cuillère"/>
                <treecell label="8"/>
            </treerow>
        </treeitem>
    </treechildren>
```

```

</tree>

<label value="Ligne:" />
<label id="row" />
<label value="Colonne:" />
<label id="column" />
<label value="Type enfant:" />
<label id="part" />

```

La fonction `getCellAt` prend cinq arguments, les coordonnées où regarder et trois autres paramètres. Un argument par référence est utilisé parce que la fonction a besoin de retourner plus d'une valeur. Vous verrez de nombreuses interfaces utilisant des arguments par référence avec [les objets disponibles](#). Ces arguments sont marqués avec un préfixe 'out'. Pour ceux-ci, vous devez transmettre un objet vide et la fonction remplira sa propriété `value` avec la valeur nécessaire.

Les trois paramètres par référence seront renseignés avec la ligne, la colonne et le type enfant. L'objet `row` contient l'index de la ligne survolée par la souris au moment où la fonction a été appelée par un événement `mousemove` avec les coordonnées de cet événement. Si les coordonnées ne sont pas au dessus d'une ligne de l'arbre, la valeur `row.value` sera égale à `-1`. La variable `column` est un objet `column` tel que défini dans Mozilla 1.8 et supérieur. Dans les versions plus anciennes, les colonnes étaient identifiées avec un chaîne de caractères, l'identifiant `id` de la colonne. Avec les versions plus récentes, un objet séparé de colonne existe et permet de réaliser des requêtes sur les données en colonne.

La ligne suivante est utilisée pour que l'exemple ci-dessus puisse fonctionner avec toutes les versions.

```
if (typeof column.value != "string") column.value = column.id;
```

Si la colonne est une chaîne de caractères, nous tournons sur Mozilla 1.7 ou inférieur, mais pour les versions récentes, nous obtenons l'identifiant de la colonne à partir de l'objet `column`. Si vous écrivez du code pour des versions multiples, vous devrez effectuer un test comme indiqué ci-avant.

Le dernier argument de la fonction `getCellAt` est le type enfant renseigné avec une chaîne dépendante de la partie de la cellule pointée par les coordonnées. Si vous déplacez la souris dans l'exemple précédent, vous noterez que le libellé passe de *text* à *cell*. La valeur *text* indique la zone où le texte est dessiné et la valeur *cell* indique la zone autour du texte, par exemple, la marge de gauche où sont habituellement dessinées les poignées ouvrantes et fermantes. Toutefois, s'il y avait une poignée, la valeur aurait plutôt été *twisty*. Cette information est pratique pour vous permettre de déterminer si l'utilisateur a cliqué sur une poignée plutôt que sur une autre partie de la ligne. En fait, lorsque l'utilisateur double-clique sur la poignée, le code natif sous-jacent utilise cette méthode. La dernière valeur qui peut être retournée est *image* si une image se trouve dans la cellule et que les coordonnées correspondent à celle de l'image. Bien entendu, dans la plupart des cas, vous ne désirez pas connaître quelle partie de la cellule pointe les coordonnées, mais seulement la ligne et la colonne concernées.

Pour inverser la recherche et obtenir les coordonnées spécifiques à une cellule, utilisez la fonction `getCoordsForCellItem`. Elle prend sept arguments tels que décrit ci-dessous.

```
var x = {}, y = {}, width = {}, height = {};
tree.boxObject.getCoordsForCellItem( row, column, part, x, y, width, height );
```

Les arguments 'row', 'column' et 'part' sont similaires à ceux retournés par la fonction `getCellAt`. De nouveau, l'argument `column` doit soit être une chaîne de caractères, ou soit être un objet `column`, selon la version que vous utilisez. Le type de la zone de la cellule peut être utilisé pour obtenir les coordonnées, soit du texte, soit de toute la cellule, soit de la poignée ou soit de l'image dans la cellule. Les mêmes valeurs que la fonction `getCellAt` sont utilisées. La fonction `getCoordsForCellItem` retourne, au travers des arguments passés en référence, les coordonnées `x` et `y` accompagnées de la largeur et la hauteur.

Par la suite, nous verrons comment RDF peut être utilisé automatiquement pour peupler des arbres et d'autres éléments.

## 9. RDF et templates

### 9.1 Introduction à RDF

Écrit par Neil Deakin. Traduit par *Vincent S.* (08/06/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/intrordf.html>

Dans cette section, nous allons nous intéresser à RDF (Resource Description Framework).

#### Resource Description Framework

Nous pouvons utiliser l'élément tree pour afficher un ensemble de données, telles que des marque-pages ou des courriels. Cependant, il ne serait pas pratique de le faire en entrant les données directement dans le fichier XUL. Il serait très difficile de modifier les marque-pages s'ils étaient directement dans le fichier XUL. Le moyen de résoudre cette difficulté est d'utiliser des sources de données RDF.

RDF (Resource Description Framework) est un format qui peut être utilisé pour stocker des ressources telles que des marque-pages ou des courriels. Alternativement, on peut utiliser des données dans d'autres formats et écrire du code qui va lire le fichier et créer le fichier de données RDF. C'est de cette façon que Mozilla fonctionne quand il lit des données telles que les marque-pages, l'historique ou les messages de courriel. Mozilla fournit des sources de données pour ces données communes pour que vous puissiez facilement les utiliser.

Vous pouvez utiliser n'importe quelles sources de données RDF fournies pour peupler les arbres tree avec des données ou vous pouvez désigner un fichier RDF au format XML contenant les données. Elles sont très commodées pour afficher des arbres contenant beaucoup de lignes. RDF peut aussi peupler d'autres éléments XUL comme les listbox et les menu. Nous verrons cela dans la prochaine section.

Un très bref aperçu de RDF sera fourni ici. Pour un guide de RDF plus détaillé, lisez Introduction to the RDF Model (en). Il est recommandé de lire ce guide si vous êtes débutant en RDF. Pour voir quelques exemples de fichiers RDF/XML, regardez ceux fournis avec Mozilla. Ils ont une extension .rdf.

RDF consiste en un modèle, qui est une représentation des données sous forme de graphe. RDF/XML est un langage XML utilisé pour représenter des données RDF. Il contient un ensemble assez simple d'éléments. L'exemple ci-dessous montre un gabarit RDF minimal.

```
<?xml version="1.0"?>
<RDF:rdf
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  ...
</RDF:rdf>
```

Il a quelques similitudes avec l'en-tête XUL. À la place de l'élément window, l'élément RDF est utilisé. Vous pouvez voir que l'espace de nommage pour RDF a été déclaré pour que les éléments RDF soient reconnus proprement. À l'intérieur de l'élément RDF, vous placerez les données.

Une brève description de RDF sera donnée ici. Pour plus d'informations à propos de RDF, consultez les spécifications RDF (NdT : RDF est une recommandation du W3C). Prenons l'exemple d'une liste de marque-pages générée à partir de RDF. Une liste de marque-pages contient un ensemble d'enregistrements, chacun avec un ensemble de données associées, telles que l'URL, le titre et une date de visite.

Pensez aux marque-pages comme une base de données qui est stockée comme une grande table avec de nombreux champs. Dans le cas de RDF cependant, les listes peuvent être hiérarchisées. C'est nécessaire pour que nous puissions avoir des dossiers ou des catégories de marque-pages. Chacun des champs dans la base



de données RDF est une ressource, avec un nom associé. Le nom est décrit par un URI.

Par exemple, une selection de champs dans la liste de marque–pages de Mozilla est décrite par les URIs ci-dessous :

Name	<a href="http://home.netscape.com/NC-rdf#Name">http://home.netscape.com/NC-rdf#Name</a>	Nom du marque–page
URL	<a href="http://home.netscape.com/NC-rdf#URL">http://home.netscape.com/NC-rdf#URL</a>	URL correspondante
Description	<a href="http://home.netscape.com/NC-rdf#Description">http://home.netscape.com/NC-rdf#Description</a>	description du marque–page
Last Visited	<a href="http://home.netscape.com/WEB-rdf#LastVisitDate">http://home.netscape.com/WEB-rdf#LastVisitDate</a>	Date de dernière visite

Ils sont générés en prenant le nom de l'espace de nommage (NdT : exemple : <http://home.netscape.com/NC-rdf>) et en ajoutant le nom du champ (NdT : exemple : [#Name](http://home.netscape.com/NC-rdf#Name)). Dans la prochaine section, nous verrons comment les utiliser pour remplir les valeurs des champs automatiquement.

Notez que la dernière date de visite a un espace de nommage légèrement différent des trois autres.

Voici maintenant un exemple de fichier RDF/XML listant une table avec trois enregistrements et trois champs.

```
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ANIMALS="http://www.www.some-fictitious-zoo.com/rdf#">

  <RDF:Seq RDF:about="http://www.www.some-fictitious-zoo.com/all-animals">
    <RDF:li>
      <RDF:Description RDF:about="http://www.www.some-fictitious-zoo.com/mammals/lion">
        <ANIMALS:name>Lion</ANIMALS:name>
        <ANIMALS:species>Panthera leo</ANIMALS:species>
        <ANIMALS:class>Mammifère</ANIMALS:class>
      </RDF:Description>
    </RDF:li>
    <RDF:li>
      <RDF:Description RDF:about="http://www.www.some-fictitious-zoo.com/arachnids/tarantula">
        <ANIMALS:name>Tarantule</ANIMALS:name>
        <ANIMALS:species>Avicularia avicularia</ANIMALS:species>
        <ANIMALS:class>Arachnide</ANIMALS:class>
      </RDF:Description>
    </RDF:li>
    <RDF:li>
      <RDF:Description RDF:about="http://www.www.some-fictitious-zoo.com/mammals/hippopotamus">
        <ANIMALS:name>Hippopotame</ANIMALS:name>
        <ANIMALS:species>Hippopotamus amphibius</ANIMALS:species>
        <ANIMALS:class>Mammifère</ANIMALS:class>
      </RDF:Description>
    </RDF:li>
  </RDF:Seq>
</RDF:RDF>
```

NdT : les spécifications du format RDF précisent qu'il faut toujours indiquer l'espace de nom RDF pour les attributs du langage RDF comme ID about, resource etc. Pour être en conformité avec ces spécifications, vous devrez toujours déclarer un alias d'espace de nom (comme **RDF** dans l'exemple précédent : `xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"`) et l'utiliser pour les attributs RDF (ex : **RDF**:about=". . .").

Ici, trois enregistrements ont été décrits, un pour chaque animal. Chaque balise `RDF:Description` décrit un seul enregistrement. À l'intérieur de chaque enregistrement, trois champs sont décrits, *name*, *species* et *class*. Il n'est pas nécessaire que tous les enregistrements aient les mêmes champs mais cela donne plus de sens.

À chacun des trois champs a été donné un espace de nommage appelé *ANIMALS*, dont l'URL a été déclarée dans la balise `RDF`. Ce nom a été choisi pour sa signification dans ce cas précis, mais nous aurions pu en choisir un autre. La fonctionnalité d'espace de nommage est utile car le champ *class* peut entrer en conflit avec celui utilisé pour les styles.

Les éléments `Seq` et `Li` sont utilisés pour indiquer que les enregistrements sont dans une liste. C'est la même façon dont les listes HTML sont déclarées. L'élément `Seq` est utilisé pour indiquer que les éléments sont ordonnés. À la place de l'élément `Seq`, vous pouvez aussi utiliser l'élément `Bag` pour indiquer des données non ordonnées, et `Alt` pour indiquer des données où chaque enregistrement spécifie des valeurs alternatives (telles que des URLs miroirs).

À l'intérieur d'un fichier XUL, il est fait référence aux ressources en combinant l'URL de l'espace de nommage suivi du nom du champ. Dans l'exemple ci-dessus, les URIs suivants générés peuvent être utilisés pour référer aux champs spécifiques :

Nom	<a href="http://www.www.some-fictitious-zoo.com/rdf#name">http://www.www.some-fictitious-zoo.com/rdf#name</a>
Espèce	<a href="http://www.www.some-fictitious-zoo.com/rdf#species">http://www.www.some-fictitious-zoo.com/rdf#species</a>
Classe	<a href="http://www.www.some-fictitious-zoo.com/rdf#class">http://www.www.some-fictitious-zoo.com/rdf#class</a>

Dans la suite, nous allons voir comment utiliser RDF pour peupler des éléments XUL

## 9.2 Gabarits

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (12/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/templates.html>

Dans cette section, nous allons voir comment peupler des éléments avec des données.

### Peuplement des éléments

XUL apporte une méthode permettant de créer des éléments à partir de données fournies par RDF, que ce soit à partir d'un fichier RDF ou à partir d'une source de données interne. Plusieurs sources de données sont déjà fournies avec Mozilla comme les marque-pages, l'historique et les messages mails. Plus de détails seront donnés dans la prochaine section.

Habituellement, les éléments tels que les `treeitem` et `menuitem` seront peuplés avec des données. Cependant, vous pouvez utiliser d'autres éléments si vous le voulez, bien qu'ils soient utilisés pour des cas spécifiques. Néanmoins nous commencerons avec ces autres éléments parce que les arbres et menus nécessitent plus de code.

Pour permettre la création d'éléments basés sur des données RDF, vous avez besoin de fournir un gabarit simple (Ndt : `template`) qui sera dupliqué pour chaque élément devant être créé. En gros, vous fournissez juste le premier élément et les suivants seront construits sur le même modèle.

Le gabarit est créé en utilisant l'élément `template`. À l'intérieur de celui-ci, vous pouvez placer les éléments que vous voulez utiliser pour chaque élément construit. L'élément `template` doit être placé à l'intérieur du conteneur qui contiendra les éléments construits. Par exemple, si vous utilisez un arbre `tree`, vous devez placer l'élément `template` à l'intérieur de l'élément `tree`.

C'est mieux d'expliquer avec un exemple. Prenons un exemple simple où nous voulons créer un bouton pour chaque marque-page principal. Mozilla fournit une source de données pour les marque-pages, pouvant être ainsi utilisée pour récupérer les données. Cet exemple ne récupérera que les marque-pages principaux car nous allons créer des boutons. Pour les marque-pages fils, nous devrions utiliser un élément qui affiche une hiérarchie tel qu'un arbre ou un menu.

Cet exemple et tous les autres qui font référence à des sources de données RDF interne, et ils ne fonctionneront que si vous les chargez à partir d'une url chrome. Pour des raisons de sécurité, Mozilla ne permet pas d'y accéder à partir de fichiers extérieurs.

Pour voir cet exemple, vous devrez créer un paquet chrome contenant le fichier à charger. Vous pouvez alors entrer l'URL chrome dans le champ de saisie des URLs du navigateur.

Exemple 9.2.1 :

```
<vbox datasources="rdf:bookmarks" ref="NC:BookmarksRoot" flex="1">
  <template>
    <button uri="rdf:*" label="rdf:http://home.netscape.com/NC-rdf#Name"/>
  </template>
</vbox>
```



Ici une boîte verticale a été créée contenant une colonne de boutons, un pour

chaque marque-page principal. Vous pouvez voir que le template ne contient qu'un seul button. Cet unique bouton est utilisé comme modèle pour tout les autres boutons qu'il sera nécessaire de créer. Vous pouvez voir sur l'image qu'un ensemble de boutons a été créé, un pour chaque marque-page.

Essayez d'ajouter un marque-page dans le navigateur pendant que vous avez cet exemple ouvert dans une fenêtre. Vous noterez que les boutons seront mis à jour instantanément (Vous devez donner le focus à la fenêtre pour voir le changement).

Le gabarit lui-même est placé à l'intérieur d'une boîte verticale. La boîte a deux attributs qui lui permet d'être utilisée pour les gabarits, indiquant d'où les données proviennent. Le premier attribut de la boîte est `datasources`. Il est utilisé pour déclarer la source de données RDF qui fournira les données pour créer les éléments. Dans le cas présent, `rdf:bookmarks` est indiqué. Vous devinez probablement qu'il signifie qu'il faut utiliser la source de données des marque-pages. Cette source de données est fournie par Mozilla. Pour utiliser votre propre source de données, spécifiez l'URL d'un fichier RDF dans l'attribut `datasources`, comme le montre l'exemple suivant :

```
<box datasources="chrome://zoo/content/animals.rdf"
  ref="http://www.some-fictitious-zoo.com/all-animals">
```

Vous pouvez spécifier plusieurs sources de données à la fois, en les séparant avec un espace dans la valeur de l'attribut. Ainsi, l'affichage de données provenant de multiples sources est possible.

L'attribut `ref` indique l'endroit dans la source de données à partir duquel vous voulez récupérer les données. Dans le cas des marque-pages, la valeur `NC:BookmarksRoot` est utilisée pour indiquer la racine de la

hiérarchie des marque–pages. Les autres valeurs que vous pouvez indiquer dépendront de la source de données que vous utiliserez. Si vous utilisez votre propre fichier RDF, la valeur correspondra à la valeur d'un attribut `about` d'un élément RDF `Bag`, `Seq` ou `Alt`.

En ajoutant ces deux attributs à la boîte du dessus, vous permettez la génération d'éléments en utilisant le gabarit. Cependant, les éléments à l'intérieur du gabarit nécessite une déclaration différente. Vous noterez dans l'exemple du dessus que le bouton a un attribut `uri` et a une valeur inhabituelle pour l'attribut `label`.

Un attribut à l'intérieur d'un gabarit qui commence par *rdf:* indique que la valeur doit être prise à partir de la source de données. Dans l'exemple plus haut, c'est le cas de l'attribut `label`. Le reste de la valeur réfère au nom de la propriété dans la source de données. Elle est construite en prenant l'URL de l'espace de nom utilisé par la source de données et en y ajoutant le nom de la propriété. Ici, nous utilisons seulement le nom du marque–page mais d'autres champs sont disponibles.

L'attribut `label` des boutons est renseigné avec cet URI spécial parce que nous voulons que les libellés des boutons aient le nom des marque–pages. Nous pouvons mettre cet URI sur n'importe quel attribut de l'élément `button`, ou n'importe quel élément. Les valeurs de ces attributs sont remplacées par les données fournies par la source de données qui, ici, sont les marque–pages.

L'exemple du dessous montre comment nous pourrions assigner d'autres attributs d'un bouton à partir de la source de données. Bien sûr, cela implique que la source de données fournisse les ressources appropriées. Si une ressource particulière est inexistante, la valeur de l'attribut sera une chaîne vide.

```
<button class="rdf:http://www.example.com/rdf#class"
  uri="rdf:*"
  label="rdf:http://www.example.com/rdf#name"
  crop="rdf:http://www.example.com/rdf#crop" />
```

Comme vous pouvez le voir, vous pouvez générer dynamiquement une liste d'éléments avec les attributs fournis par une source de données séparée.

L'attribut `uri` est utilisé pour spécifier l'élément où la génération du contenu commencera. Le contenu extérieur ne sera généré qu'une seule fois, tandis que le contenu intérieur sera généré pour chaque ressource. Nous en verrons plus à ce propos quand nous créerons des gabarits pour les arbres.

En ajoutant ces fonctionnalités au conteneur dans lequel est le gabarit, qui dans ce cas est une boîte, et aux éléments à l'intérieur du gabarit, nous pouvons générer de multiples listes de contenu à partir de données externes. Nous pouvons bien sûr mettre plus d'un élément à l'intérieur du gabarit, et ajouter une référence RDF spéciale dans les attributs sur n'importe quel élément. L'exemple suivant le montre :

Exemple 9.2.2 :

```
< vbox datasources="rdf:bookmarks" ref="NC:BookmarksRoot" flex="1">
  < template>
    < vbox uri="rdf:*">
      < button label="rdf:http://home.netscape.com/NC-rdf#Name" />
      < label value="rdf:http://home.netscape.com/NC-rdf#URL" />
    < /vbox>
  < /template>
< /vbox>
```

Cet exemple crée une boîte verticale avec un bouton et un libellé pour chaque marque–page. Le bouton contiendra le nom du marque–page et le libellé contiendra l'URL.

Les nouveaux éléments qui sont créés ne sont fonctionnellement pas différents de ceux que vous mettez directement dans le fichier XUL. L'attribut `id` est ajouté à tous les éléments créés au travers du gabarit, et il

est assigné à la valeur qui identifie la ressource. Vous pouvez l'utiliser pour identifier la ressource.

Vous pouvez aussi spécifier de multiples ressources dans le même attribut en les séparant avec un espace, comme dans l'exemple qui suit ([en savoir plus sur la syntaxe des ressources](#)).

Exemple 9.2.3 :

```
<vbox datasources="rdf:bookmarks" ref="NC:BookmarksRoot"
    flex="1">
  <template>
    <label uri="rdf:*" value="rdf:http://home.netscape.com/NC-rdf#Name" rdf:http://home.netscape.com/NC-rdf#Name"/>
  </template>
</vbox>
```

## Comment sont construits les gabarits

Quand un élément a un attribut `datasources`, cela indique que l'élément est susceptible d'être généré à partir d'un gabarit. Notez que ce n'est pas la balise `template` qui détermine si le contenu sera généré, mais bien l'attribut `datasources`. Quand cet attribut est présent, un objet que l'on appelle *un constructeur* est ajouté à l'élément. C'est cet objet qui est responsable de la génération du contenu à partir du gabarit. En javascript, vous pouvez accéder à l'objet constructeur par la propriété `builder`, bien qu'habituellement vous en aurez seulement besoin pour régénérer le contenu dans les situations où il ne le fait pas automatiquement.

Il y a deux différents types de constructeur. Le premier est un *constructeur de contenu* utilisé dans la plupart des situations, et l'autre est un *constructeur d'arbres* utilisé uniquement avec les éléments `tree`.

Le constructeur de contenu prend le contenu situé à l'intérieur de l'élément `template` et le duplique pour chaque ligne. Par exemple, si l'utilisateur a dix marque-pages dans l'exemple du dessus, dix éléments `label` seront créés et ajoutés en tant que fils à l'élément `vbox`. Si vous utilisez les fonctions DOM pour traverser l'arbre, vous trouverez ces éléments à ces emplacements et pourrez récupérer leurs propriétés. Ces éléments sont affichés, mais pas l'élément `template`, bien qu'il existe encore dans l'arbre du document. De plus, l'attribut `id` de chaque libellé sera initialisé avec la ressource RDF de la ligne correspondante.

Le constructeur de contenu démarre toujours à partir de l'élément qui a l'attribut `uri="rdf:*"`. Si l'attribut `uri` est placé à l'intérieur d'autres éléments, ces derniers ne seront créés qu'une seule fois. Dans l'exemple ci-dessous, un `hbox` sera créé et rempli avec un `label` pour chaque item.

```
<template>
  <hbox>
    <label uri="rdf:*" value="rdf:http://home.netscape.com/NC-rdf#Name" />
  </hbox>
</template>
```

S'il y a du contenu à l'intérieur de l'élément qui a l'attribut `datasources` mais en dehors de l'élément `template`, ce contenu apparaîtra également. Ce faisant, vous pouvez mélanger du contenu statique et dynamique dans un gabarit.

Le constructeur d'arbres, d'autre part, ne génère pas d'éléments DOM pour chaque ligne. À la place, il récupère les données directement à partir de la source de données RDF quand il en a besoin. Comme les arbres ont souvent besoins d'afficher des centaines de lignes de données, c'est plus efficace comme ceci. Créer un élément pour chaque cellule serait trop coûteux. Cependant, en contre partie, ces arbres ne peuvent afficher que du texte (NdT : ainsi que des images et des cases à cocher), et comme aucun élément n'est créé, vous ne pouvez pas utiliser les propriétés CSS de manière habituelle pour décorer les cellules de l'arbre.

Le constructeur d'arbre est utilisé seulement pour les arbres. Les autres éléments n'utilisent que le constructeur de contenu. Ceci n'est pas un problème étant donné que les autres éléments comme les menus n'ont généralement pas besoin d'afficher beaucoup d'items. Il est possible également d'utiliser le constructeur de contenu pour les arbres, ainsi un élément `treeitem` et d'autres seront créés pour chaque ligne.

## Règles

Dans l'image du précédent exemple, vous avez pu noter que le troisième bouton est simplement un bouton avec des tirets comme libellé. C'est un séparateur dans la liste des marque-pages. Au cas où nous l'utiliserions, la source de données RDF des marque-pages stocke les séparateurs comme si ils étaient des marque-pages normaux. Ce que nous voulons faire est d'ajouter un petit espace à la place d'un bouton pour les ressources "séparateur". Ce qui signifie que nous voulons avoir deux différents types de contenu à créer, un type pour les marque-pages normaux, et un autre type pour les séparateurs.

Nous pouvons le faire en utilisant un élément `rule`. Nous définissons une règle pour chaque version d'élément que nous avons à créer. Dans notre cas, nous aurons besoin d'une règle pour les marque-pages, et une règle pour les séparateurs. Les attributs placés sur l'élément `rule` déterminent quelle règle s'applique sur quelle ressource RDF.

Pour savoir quelle règle s'applique sur les données, chaque élément `rule` est analysé en séquence pour une vérification de concordance. De fait, l'ordre dans lequel vous définissez les règles est important. Les règles du début ont priorité sur les règles suivantes.

L'exemple suivant modifie l'exemple précédant avec deux règles.

Exemple 9.2.4 :

```
<window
  id="example-window"
  title="Liste des marque-pages"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

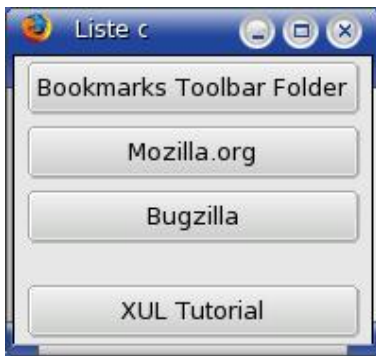
  < vbox datasources="rdf:bookmarks" ref="NC:BookmarksRoot" flex="1">
    < template>

      < rule rdf:type="http://home.netscape.com/NC-rdf#BookmarkSeparator">
        < spacer uri="rdf:*" height="16"/>
      < /rule>

      < rule>
        < button uri="rdf:*" label="rdf:http://home.netscape.com/NC-rdf#Name"/>
      < /rule>

    < /template>
  < /vbox>

< /window>
```



En utilisant deux règles, nous permettons au contenu du gabarit d'être généré

sélectivement. Dans le premier rule, les séparateurs de marque-pages sont sélectionnés, comme nous pouvons le voir par l'attribut `rdf:type`. Le second rule n'ayant aucun attribut, il sélectionne tout le reste.

Tout les attributs placés dans l'élément rule, sont utilisés comme critère de sélection. Dans notre cas, la source de données des marque-pages fournit une propriété `rdf:type` pour distinguer les séparateurs. Cet attribut contient une valeur spéciale pour les séparateurs dans la source de données RDF. C'est ainsi qu'on peut les distinguer des marque-pages. Vous pouvez utiliser une technique similaire pour n'importe quel attribut que l'on peut mettre sur un élément RDF *Description*.

La valeur spéciale d'URL donnée dans l'exemple du dessus pour la première règle, est utilisée pour les séparateurs. Elle signifie que les séparateurs s'appliqueront à la première règle en générant un élément spacer d'une hauteur de 16 pixels. Les éléments qui ne sont pas des séparateurs ne correspondront pas à la première règle, et atterriront dans la deuxième règle. Celle-ci n'a aucun attribut. Elle correspond à n'importe quelle donnée. Ce qui bien sûr, est ce que nous voulons pour le reste des données.

Vous avez dû noter que, parce que nous voulons un attribut provenant de l'espace de nom du RDF (`rdf:type`), nous avons besoin d'ajouter la déclaration de cet espace de nom dans la balise window. Si nous n'avions pas fait cela, l'attribut serait attribué à l'espace de nom XUL. Parce qu'il n'existe pas dans cet espace, la règle ne s'appliquerait pas. Si nous utilisons des attributs provenant de notre propre espace de nom, vous devez ajouter la déclaration de votre espace de nom pour qu'ils soient reconnus.

Vous devez deviner ce qui arriverait si la seconde règle était enlevée. Le résultat serait alors un simple spacer, sans aucun marque-page puisqu'ils ne correspondent à aucune règle.

Dit plus simplement, une règle correspond si tous les attributs placés dans l'élément rule correspondent aux attributs de la ressource RDF. Dans le cas d'un fichier RDF, les ressources seraient les éléments *Description*.

Il y a cependant quelques petites exceptions. Vous ne pouvez pas faire la correspondance avec les attributs `id`, `rdf:property` ou `rdf:instanceOf`. Mais puisque vous utiliserez vos propres attributs dans votre propre espace de noms, ces exceptions n'auront probablement pas d'importance de toute façon.

Notez qu'un gabarit sans règle, comme dans le premier exemple, est équivalent fonctionnellement à un gabarit qui possède un seul rule sans attribut.

Nous allons voir maintenant l'utilisation des gabarits avec les arbres.

## 9.2bis Exemples de syntaxe de gabarits

Écrit par Neil Deakin. Traduit par *Alain B.* (11/09/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/templateex.html>

Les exemples suivants vous montrent l'utilisation de la syntaxe des gabarits.

## Substitution des sources de données RDF

Les exemples ci dessous proposent d'illustrer les mécanismes de substitution des ressources RDF par les règles de gabarits. Lorsque la valeur d'un attribut de n'importe quel élément placé dans une règle rule contient un symbole interrogatif ou une sous-chaîne *rdf:*, elle est interprétée et substituée par les valeurs de la base de données du gabarit.

Un symbole interrogatif indique une variable de substitution déclarée à partir de conditions ou de liaisons de l'élément. À la suite de ce symbole interrogatif devrait se trouver le nom de la variable. La valeur de la variable est remplacée en lieu et place du symbole interrogatif et du nom de la variable. Si la variable n'est pas définie, le symbole interrogatif et la variable sont simplement ignorés.

Lors de l'emploi de règles, tout texte commençant par *rdf:* indique l'adresse URI d'une ressource de la base de données. La valeur de cette ressource remplace *rdf:* et l'adresse URI. Si la valeur n'existe pas, ce préfixe et cette URI sont ignorés.

```
<button value="?count"/>
```

Le libellé du bouton aura la valeur de la variable 'count', qui doit être définie dans la partie des conditions ou des liaisons de la règle de gabarit. Si elle n'est pas définie, la valeur affectée sera une chaîne vide.

```
<button value="Nombre: ?count"/>
```

Le libellé du bouton aura la valeur *Nombre:* suivie par la valeur de la variable count.

```
<button value="rdf:http://home.netscape.com/NC-rdf#Name"/>
```

La valeur de la ressource 'Name' est affectée comme libellé du bouton.

```
<button value="Mon nom est: rdf:http://home.netscape.com/NC-rdf#Name"/>
```

Le libellé du bouton aura la valeur *Mon nom est:* suivie par la valeur de la ressource 'Name'. De cette façon, le texte littéral et celui de la ressource sont concaténés.

```
<button value="rdf:http://home.netscape.com/NC-rdf#Name est mon nom."/>
```

Le libellé du bouton aura la valeur de la ressource suivie par *est mon nom..* Notez qu'il y aura un espace entre ces deux textes.

```
<button value="rdf:http://home.netscape.com/NC-rdf#Name^'s my name."/>
```

Le caractère ^ placé après la ressource 'Name' signifie une concaténation sans espace. Dans cet exemple, le résultat sera la valeur de la ressource suivie du texte *'s my name..* Il n'y a pas d'espace intermédiaire. Ce caractère a cette signification spéciale que lorsqu'il est placé à la fin d'une ressource URI. Pour afficher ce caractère, vous devez l'échapper en l'écrivant deux fois sur la ligne.

```
<button value="rdf:http://home.netscape.com/NC-rdf#Name rdf:http://home.netscape.com/NC-rdf#URL"/>
```

Le libellé du bouton aura la valeur de la ressource Name suivie par la valeur de la ressource URL séparée par un espace. Vous pouvez utiliser la caractère ^ à la place de l'espace pour attacher ces deux valeurs.



## 9.3 Arbres et Gabarits

Écrit par Neil Deakin. Traduit par *Cyril Delalande et Laurent Jouanneau* (15/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/treetempl.html>

Nous allons voir maintenant comment utiliser un gabarit avec un arbre.

### Ajouter des sources de données aux arbres

Quand vous utilisez un arbre, vous utilisez souvent un gabarit pour construire son contenu, pour manipuler un grand volume de données hiérarchisées. L'utilisation d'un gabarit avec un arbre est très proche au niveau de la syntaxe aux autres éléments. Vous avez besoin d'ajouter un attribut `datasources` et `ref` à la balise `tree`, ce qui spécifie la source des données et le noeud racine à afficher. De nombreuses règles peuvent être utilisées pour indiquer différents contenus pour différents types de données.

L'exemple suivant utilise l'historique comme source de données :

```
<tree datasources="rdf:history" ref="NC:HistoryByDate" flags="dont-build-content">
```

Comme il est décrit dans la section précédente, l'arbre peut utiliser un *constructeur* d'arbre pour la génération du gabarit à la place du constructeur normal de contenu. Les éléments ne seront pas créés pour chacune des lignes dans l'arbre, afin de le rendre plus performant. Lorsque l'attribut `flags` a pour valeur *dont-build-content*, comme dans l'exemple ci-dessus, il indique que le constructeur de l'arbre doit être utilisé. Si l'attribut n'est pas renseigné, le constructeur de contenu sera utilisé. Vous pouvez voir la différences en utilisant l'inspecteur DOM de Mozilla sur un arbre avec, puis sans l'attribut.

Si vous utilisez le constructeur normal à la place, notez que le contenu ne sera pas construit avant qu'il soit nécessaire. Avec les arbres hiérarchiques, les enfants ne sont pas générés avant que le noeud parent ne soit ouvert par l'utilisateur.

Dans le gabarit, il n'y aura qu'une cellule `treecell` pour chaque colonne dans l'arbre. Les cellules devront avoir un attribut `label` afin de mettre un libellé à la cellule. Normalement, une propriété RDF se charge de récupérer le libellé à partir de la source de données.

L'exemple suivant montre un arbre construit à partir d'un gabarit, dans ce cas le système de fichier.

Exemple 9.3.1 :

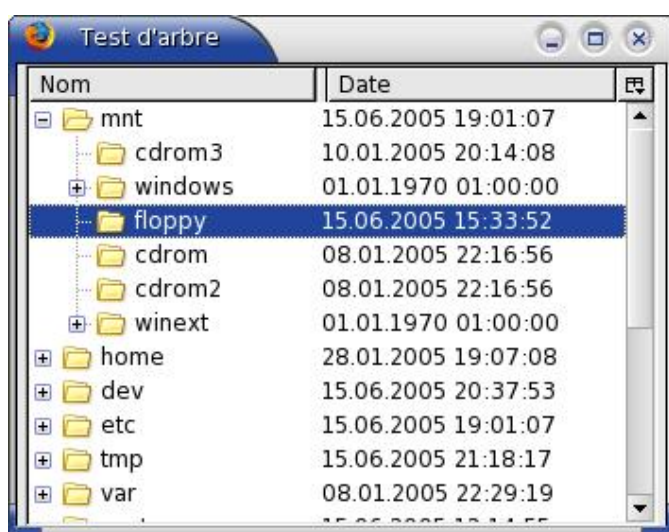
```
<tree id="my-tree" flex="1"
  datasources="rdf:files" ref="file:/// " flags="dont-build-content">
  <treecols>
    <treecol id="Name" label="Nom" primary="true" flex="1"/>
    <splitter/>
    <treecol id="Date" label="Date" flex="1"/>
  </treecols>

  <template>
    <rule>
      <treechildren flex="1">
        <treeitem uri="rdf:*">
          <treerow>
            <treecell label="rdf:http://home.netscape.com/NC-rdf#Name"/>
            <treecell label="rdf:http://home.netscape.com/WEB-rdf#LastModifiedDate"/>
          </treerow>
        </treeitem>
      </treechildren>
    </rule>
```

```
</template>
</tree>
```

Ici, un arbre est créé avec deux colonnes, pour le nom et la date d'un fichier. L'arbre doit afficher une liste de fichiers situés dans le répertoire racine. Une seule règle est utilisée, mais vous pouvez en ajouter d'autres si vous en avez besoin. Comme avec les autres gabarits, l'attribut `uri` d'un élément indique où commencer pour générer du contenu. Les deux cellules puisent le nom et la date dans la source et place les valeurs dans le libellé de la cellule.

Cet exemple montre pourquoi l'attribut `uri` devient utile. Notez comment il a été placé dans le `treeitem` dans l'exemple, même si ce n'est pas un descendant direct de l'élément `rule`. Nous avons besoin de mettre cet attribut seulement sur les éléments que nous voulons répéter pour chaque ressource. Comme nous ne voulons pas des éléments `treechildren` multiples, l'attribut est placé ailleurs. Nous le mettons plutôt dans l'élément `treeitem`. En fait, les éléments à l'extérieur (ou au-dessus) de l'élément qui a l'attribut `uri` ne sont pas dupliqués tandis que les éléments avec l'attribut `uri` et les éléments à l'intérieur sont répétés pour chaque ressource.



Notez dans l'image que des éléments fils additionnels ont été ajoutés automatiquement sous les éléments du niveau supérieur. XUL sait comment ajouter des éléments fils quand les modèles ou règles contiennent des éléments arbre ou menu. Il génère alors les éléments d'arbre imbriqués selon les données disponibles dans le RDF.

Une partie intéressante des sources de données RDF est que les valeurs sont déterminées seulement quand les données sont requises. Les valeurs qui sont plus profondes dans la hiérarchie de ressource ne sont pas déterminées jusqu'à ce que l'utilisateur atteigne ce noeud dans l'arbre. Ce mécanisme devient utile pour certaines sources où les données sont déterminées dynamiquement.

## Trier les colonnes

Si vous essayez l'exemple précédent, vous pouvez noter que la liste de dossiers n'est pas triée. Les arbres qui produisent leurs données à partir d'une source ont la capacité facultative de trier leurs données. Vous pouvez trier de façon croissante ou décroissante sur n'importe quelle colonne. L'utilisateur peut changer la colonne de tri et la direction de tri en cliquant sur les en-têtes de colonne. Ce dispositif de tri n'est pas disponible pour des arbres dont le contenu est statique, bien que vous puissiez écrire un script pour trier ces données.

Trier implique trois attributs, qui doivent être placés sur les colonnes. Le premier attribut, `sort`, doit être placé sur une propriété de RDF qui est employée alors comme critère de tri. Habituellement, c'est la même que celle utilisée dans l'étiquette de la cellule de cette colonne. Si vous le placez sur une colonne, les données seront triées dans cette colonne. L'utilisateur peut changer la direction de tri en cliquant sur l'en-tête

de colonne. Si vous ne placez pas l'attribut `sort` sur une colonne, les données ne pourront pas être triées par cette colonne.

L'attribut `sortDirection` (notez la casse mixte) est utilisé pour définir la direction dans laquelle la colonne sera triée par défaut. Trois valeurs sont possibles :

- *ascending* : les données sont affichées par ordre croissant.
- *descending* : les données sont affichées par ordre décroissant.
- *natural* : les données sont affichées dans le sens "naturel", c'est-à-dire l'ordre dans lequel elles sont apparaissent dans la source RDF.

Le dernier attribut, `sortActive`, doit être défini à *true* sur une seule colonne, celle qui sera triée par défaut.

Bien que le tri fonctionnera correctement avec seulement ces attributs, vous pouvez également utiliser la classe de style `sortDirectionIndicator` sur une colonne qui peut être triée. Un petit triangle apparaîtra dans l'en-tête de colonne et indiquera le sens du tri. Sans cela, l'utilisateur pourra toujours trier les colonnes mais il n'aura pas d'indication sur la colonne triée.

L'exemple suivant modifie les colonnes de l'exemple précédent pour inclure les fonctionnalités supplémentaires :

```
<treecols>
  <treecol id="Name" label="Name" flex="1" primary="true"
    class="sortDirectionIndicator" sortActive="true"
    sortDirection="ascending"
    sort="rdf:http://home.netscape.com/NC-rdf#Name" />
  <splitter/>
  <treecol id="Date" label="Date" flex="1" class="sortDirectionIndicator"
    sort="rdf:http://home.netscape.com/WEB-rdf#LastModifiedDate" />
</treecols>
```

## L'état persistant des colonnes

Une chose supplémentaire que vous voudrez faire est de rendre persistant la colonne qui est actuellement triée, ainsi cet état est mémorisé entre chaque session. Pour ce faire, nous utilisons l'attribut `persist` sur chaque élément `treecol`. Il peut être utile de rendre persistant cinq attributs : la taille de la colonne, l'ordre des colonnes, la visibilité de la colonne, quelle colonne est actuellement triée et dans quel ordre. L'exemple suivant montre une simple colonne :

```
<treecol id="Date" label="Date" flex="1"
  class="sortDirectionIndicator"
  persist="width ordinal hidden sortActive sortDirection"
  sort="rdf:http://home.netscape.com/WEB-rdf#LastModifiedDate" />
```

## Attributs supplémentaires pour les règles

Deux attributs supplémentaires peuvent être ajoutés sur l'élément `rule`, lui permettant d'appliquer des correspondances dans certaines circonstances. Les deux sont des booléens.

*iscontainer*

Si cet attribut est mis à *true*, alors la règle s'appliquera sur toutes les ressources qui peuvent avoir des enfants. Par exemple, nous pouvons appliquer cette règle pour récupérer les dossiers des marque-pages. Il est utile car la source de données n'a pas besoin d'inclure un attribut spécial pour l'indiquer.

*isempty*

Si cet attribut est mis à *true*, alors la règle s'appliquera sur toutes les ressources qui n'ont pas d'enfants.

Les deux attributs du dessus sont vraiment l'opposé l'un de l'autre. Une ressource pourrait être un conteneur et être vide en même temps. Toutefois, c'est différent pour une ressource qui n'est pas un conteneur. Par exemple, un dossier de marque–pages est un conteneur mais il peut avoir ou ne pas avoir d'enfants. Cependant, un simple marque–page ou un séparateur n'est pas un conteneur.

Vous pouvez combiner ces deux éléments avec d'autres attributs de critères pour des règles plus spécifiques.

---

Dans la section suivante, nous verrons quelques unes des sources de données fournies par Mozilla.

## 9.4 Sources de données RDF

Écrit par Neil Deakin. Traduit par *Caffeine* (16/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/datasrc.html>

Nous nous intéresserons ici aux sources de données additionnelles, ainsi qu'à la manière d'utiliser vos propres fichiers RDF comme sources de données.

### Autres sources de données pour Mozilla

Mozilla fournit nativement plusieurs sources de données. Certaines sont indiquées ici, avec quelques exemples. Leur fonctionnement est très similaire à celui des marque–pages, bien que les champs soient différents à chaque fois.

Ndt : Les ressources RDF fournies par mozilla ne sont utilisables que par les applications qui sont enregistrées dans le chrome. Vous ne verrez donc pas les données dans certains des exemples proposés.

### La liste d'historique

Cette source de données fournit l'accès à la liste d'historique de l'utilisateur, qui est une liste d'URLs que l'utilisateur a consulté récemment. On peut se référer à cette ressource en utilisant `rdf:history` comme source de données. La table ci–dessous montre les ressources (ou champs) que vous pouvez récupérer depuis la source de donnée "history". Utilisez les URLs ci–dessous là où vous souhaitez que la valeur de la ressource soit utilisée.

Date	<code>rdf:http://home.netscape.com/NC-rdf#Date</code>	Date de la dernière visite
Name	<code>rdf:http://home.netscape.com/NC-rdf#Name</code>	Titre de la page
Page	<code>rdf:http://home.netscape.com/NC-rdf#Page</code>	Nom de la page
Referrer	<code>rdf:http://home.netscape.com/NC-rdf#Referrer</code>	Page d'origine (referrer)
URL	<code>rdf:http://home.netscape.com/NC-rdf#URL</code>	URL de la page
Visit Count	<code>rdf:http://home.netscape.com/NC-rdf#VisitCount</code>	Nombre de visites de la page

Une liste d'historique typique affichera un arbre doté d'une sélection de ces champs. Pour les utiliser, placez les URL ci–dessus dans l'attribut `label` des boutons ou des cellules d'un arbre. Vous pouvez utiliser `NC:HistoryRoot` comme valeur de l'attribut `ref`. Vous pouvez également utiliser la valeur `NC:HistoryByDate` pour obtenir la liste d'historique triée par jour.

Voyons un exemple d'affichage de la liste d'historique. Nous afficherons l'historique dans un arbre avec les trois colonnes Nom, URL et Date.

## Exemple 9.4.1 :

```

<tree flex="1" datasources="rdf:history" ref="NC:HistoryRoot">

  <treecols>
    <treecol id="name" label="Nom" flex="1"/>
    <treecol id="url" label="URL" flex="1"/>
    <treecol id="date" label="Date" flex="1"/>
  </treecols>

  <template>

    <rule>
      <treechildren flex="1">
        <treeitem uri="rdf:*">
          <treerow>
            <treecell label="rdf:http://home.netscape.com/NC-rdf#Name" />
            <treecell label="rdf:http://home.netscape.com/NC-rdf#URL" />
            <treecell label="rdf:http://home.netscape.com/NC-rdf#Date" />
          </treerow>
        </treeitem>
      </treechildren>
    </rule>

  </template>
</tree>

```

## Autres sources de données

Les tableaux ci-dessous listent quelques-unes des autres sources de données disponibles avec Mozilla. Vous pouvez les utiliser comme vous voulez.

Marque-pages (*rdf:bookmarks*) :

Les marque-pages sont générés depuis la liste de marque-pages de l'utilisateur.

## Ressources

Date d'Ajout	<code>rdf:http://home.netscape.com/NC-rdf#BookmarkAddDate</code>	Date à laquelle le marque-page a été ajouté
Description	<code>rdf:http://home.netscape.com/NC-rdf#Description</code>	Description du marque-page
Dernière Modification	<code>rdf:http://home.netscape.com/WEB-rdf#LastModifiedDate</code>	Date de la dernière modification
Dernière Visite	<code>rdf:http://home.netscape.com/WEB-rdf#LastVisitDate</code>	Date de la dernière visite
Nom	<code>rdf:http://home.netscape.com/NC-rdf#Name</code>	Nom du marque-page
Raccourci URL	<code>rdf:http://home.netscape.com/NC-rdf#ShortcutURL</code>	Champ de mots-clés personnalisés
URL	<code>rdf:http://home.netscape.com/NC-rdf#URL</code>	L'URL vers laquelle pointe le lien

## Racines Possibles pour les marque-pages

*NC:BookmarksRoot*

La racine de la hiérarchie des marque-pages

*NC:IEFavoritesRoot*

Le dossier de marque-pages correspondant aux Favoris IE de l'utilisateur

*NC:PersonalToolbarFolder*

Le dossier de marque-pages correspondant au dossier de la barre d'outils personnelle

## Fichiers(*rdf:files*) :

Une vue des fichiers de l'utilisateur.

### Ressources

Nom	<code>rdf:http://home.netscape.com/NC-rdf#Name</code>	Nom du fichier
URL	<code>rdf:http://home.netscape.com/NC-rdf#URL</code>	URL du fichier

### Racine possible des fichiers

#### *NC:FilesRoot*

Racine du système de fichier (habituellement une liste de disques)

#### *URL d'un fichier*

En utilisant une URL de fichier pour l'attribut `ref`, vous pouvez choisir un répertoire à retourner.  
Par exemple, vous pouvez utiliser `file:///windows` ou `file:///usr/local`

La source de données de fichiers est un exemple de source de données qui ne détermine ses ressources que lorsque cela est nécessaire. Nous ne voulons pas que tous les fichiers du système de fichier soient évalués avant d'afficher les données. Au lieu de cela, seuls les fichiers et répertoires que l'arbre (ou tout autre élément) doit afficher à un instant donné seront déterminés.

## Sources de données composites

Vous pouvez spécifier des sources de données multiples dans l'attribut `datasources` en les séparant par des espaces, comme dans l'exemple ci-dessous. Les données seront ainsi lues dans toutes les sources de données mentionnées.

```
<tree datasources="rdf:bookmarks rdf:history animals.rdf" ref="NC:BookmarksRoot">
```

Cet exemple lit les ressources des marque-pages, de l'historique et d'un fichier nommé *animals.rdf*. Celles-ci sont combinées en une source de données composite et peuvent être utilisées comme si elles ne faisaient qu'une.

La source de donnée spéciale `rdf:null` correspond à une source vide. Vous pouvez utiliser cette source de données si vous voulez gérer dynamiquement la source via un script, sans avoir à l'initialiser tout de suite ou parce que vous ne connaissez pas son URL exacte.

## Sources de données RDF personnalisées

Vous pouvez utiliser chacune des sources de données internes ci-dessus si vous le désirez. Il en existe plusieurs autres, pour le courrier électronique, les carnets d'adresses, les recherches... Cependant, vous pouvez utiliser votre propre source de données RDF, enregistrée dans un fichier RDF. Le fichier peut être local ou distant. Il suffit d'indiquer l'URL du fichier RDF dans l'attribut `datasources`.

L'utilisation des fichiers RDF apporte les mêmes fonctionnalités que n'importe quelle des sources de données internes. Vous pouvez utiliser des règles pour retrouver un type spécifique de contenu. Les attributs de l'élément *rule* correspondront s'ils correspondent aux attributs d'un élément RDF *Description*. Vous pouvez également créer des fichiers RDF hiérarchiques.

Ce qui suit est un exemple d'utilisation d'un fichier RDF comme source de données. Le fichier RDF est relativement gros et peut être vu séparément :

## Exemple 9.4.2 :

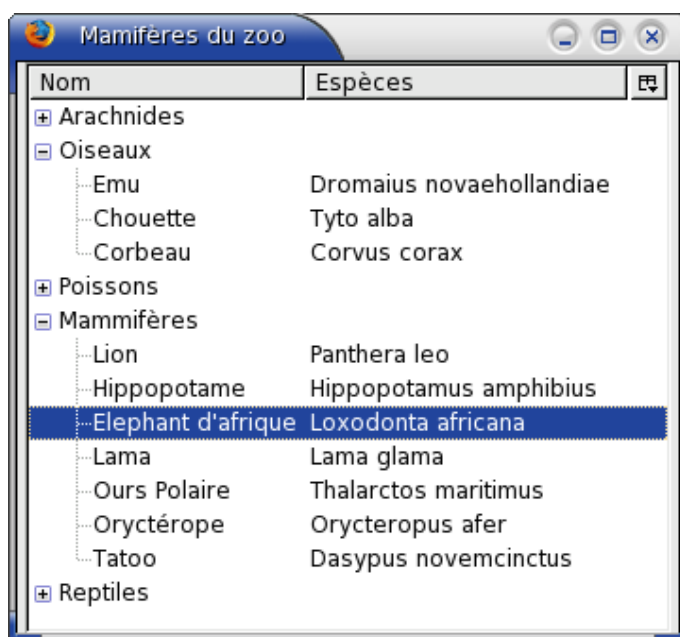
```

<tree flex="1" width="200" height="200" datasources="animals.rdf"
ref="http://www.some-fictitious-zoo.com/all-animals">

  <treecols>
    <treecol id="name" label="Nom" primary="true" flex="1"/>
    <treecol id="species" label="Espèces" flex="1"/>
  </treecols>

  <template>
    <rule>
      <treechildren>
        <treeitem uri="rdf:*">
          <treerow>
            <treecell label="rdf:http://www.some-fictitious-zoo.com/rdf#name"/>
            <treecell label="rdf:http://www.some-fictitious-zoo.com/rdf#species"/>
          </treerow>
        </treeitem>
      </treechildren>
    </rule>
  </template>
</tree>

```



Ici, les données sont générées depuis le fichier.

L'attribut `ref` a été positionné sur l'élément racine du fichier RDF, qui est la balise de premier niveau `Seq`. Nous obtenons une liste complète d'animaux. Si nous avions voulu, nous aurions pu positionner l'attribut `ref` sur n'importe quelle valeur de l'attribut `about` afin de filtrer la quantité de données retournée. Par exemple, pour afficher seulement les reptiles, nous pouvons utiliser la valeur <http://www.some-fictitious-zoo.com/reptiles>.

L'exemple ci-dessous montre comment afficher des éléments particuliers d'une source de données RDF en positionnant l'attribut `ref`.

## Exemple 9.4.3 :

```

<window
  id="example-window"
  title="Mammifères du zoo"
  xmlns:ANIMALS="http://www.some-fictitious-zoo.com/rdf#"

```

```

xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<button label="Cliquez ici pour voir les mammifères du zoo" type="menu"
        datasources="animals.rdf" ref="http://www.some-fictitious-zoo.com/mammals">
  <template>
    <rule ANIMALS:specimens="0"></rule>
    <rule>
      <menupopup>
        <menuitem uri="rdf:*" label="rdf:http://www.some-fictitious-zoo.com/rdf#name"/>
      </menupopup>
    </rule>
  </template>
</button>

</window>

```

Dans le cas présent, nous voulons seulement la liste des mammifères, nous sélectionnons donc l'URI de la liste des mammifères. Vous remarquerez que la valeur de l'attribut `ref` de notre exemple est `http://www.some-fictitious-zoo.com/mammals`, ce qui correspond à l'un des éléments `Seq` du . Seuls les descendants de cette liste seront retournés.

Nous avons utilisé deux règles ici. La première filtre toutes les ressources dont l'attribut `ANIMALS:specimen` est positionné à `0`. Vous pouvez voir cet attribut dans le fichier RDF pour chacun des éléments `Description`. Certains d'entre eux ont une valeur de `0`. Dans ce cas, la première règle s'applique. Puisque cette règle n'a pas de contenu, rien ne sera affiché pour les éléments concernés. Ainsi, nous pouvons cacher les données que nous ne voulons pas afficher.

La seconde règle s'applique aux autres ressources et crée une rangée dans un menu surgissant. Le résultat final est un menu surgissant contenant tous les mammifères dont le specimen n'est pas positionné à `0`.

Dans la prochaine section, nous examinerons la syntaxe des règles.

## 9.5 Règles avancées

Écrit par Neil Deakin. Traduit par **Julien Etaix** (22/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/advrules.html>

Cette section décrit la syntaxe des règles les plus complexes.

### La syntaxe complète des règles

La syntaxe des règles décrites jusqu'ici est utile pour certaines sources de données, mais parfois les données doivent être affichées de manière plus sophistiquée. La syntaxe de règle simple n'est en fait qu'un raccourci pour la syntaxe de règle complète qui est décrite ci-dessous. Comme pour la syntaxe de règle simple, la règle complète est placée entre des balises rule.

Une syntaxe de règles complètes contient trois balises filles, une balise conditions, une balise bindings et une balise action, bien que la balise bindings ne soit pas toujours nécessaire.

L'élément conditions spécifie le critère qui doit correspondre à une ressource donnée. Vous pouvez spécifier plusieurs conditions qui doivent toutes correspondre à la ressource donnée. En utilisant la syntaxe de règle simple, les conditions sont directement situées sur l'élément rule.

Si les conditions correspondent à une ressource, le contenu placé entre les balises action est généré. Dans la syntaxe de règle simple, le contenu est directement placé dans la balise rule.



## Conditions d'une règle

Lorsque qu'un arbre, un menu ou tout autre élément avec une source de données génère son contenu, le générateur de modèle cherche en premier la ressource marquée par l'attribut `ref`. L'opération est ensuite répétée pour l'ensemble des ressources filles. Il compare chaque ressource aux conditions. Si celles-ci sont vérifiées, le contenu de l'élément `action` est généré pour cette ressource. Si elles ne sont pas vérifiées, rien n'est généré.

L'élément `conditions` contient trois sous éléments. Le premier est l'élément `content` qui doit toujours être présent une fois seulement. Il sert de marqueur lorsque le générateur de modèle parcourt les ressources. Il indique le nom de la variable dans laquelle est placée une référence à la ressource racine pendant que les conditions sont analysées. La ressource racine est celle indiquée par l'attribut `ref` dans l'élément contenant le modèle.

La syntaxe d'un élément `content` est la suivante :

```
<content uri="?var" />
```

La point d'interrogation indique que le texte qui suit est une variable. Vous pouvez alors utiliser la variable `var` dans le reste des conditions. Bien entendu, vous pouvez nommer la variable comme vous le voulez.

L'élément suivant est l'élément `member` qui est utilisé pour parcourir un ensemble de ressources filles. En termes RDF, il est comparable à `Seq`, `Bag` ou `Alt`. Disons que vous avez une liste de villes comme dans l'extrait RDF/XML suivant :

```
<RDF:Seq RDF:about="http://www.xulplanet.com/rdf/weather/cities">
  <RDF:li RDF:resource="http://www.xulplanet.com/rdf/weather/city/paris"/>
  <RDF:li RDF:resource="http://www.xulplanet.com/rdf/weather/city/Manchester"/>
  <RDF:li RDF:resource="http://www.xulplanet.com/rdf/weather/city/Melbourne"/>
  <RDF:li RDF:resource="http://www.xulplanet.com/rdf/weather/city/Kiev"/>
</RDF:Seq>

<RDF:Description RDF:about="http://www.xulplanet.com/rdf/weather/city/paris">
  <cityset:name>paris</cityset:name>
</RDF:Description>

.
.
.
```

Vous voulez afficher une ligne dans une arborescence pour chaque ville. Pour accomplir cela, utilisez l'élément `member` comme ceci :

```
<tree id="citiesTree" datasources="weather.rdf"
  ref="http://www.xulplanet.com/rdf/weather/cities">
  <template>
    <rule>
      <conditions>
        <content uri="?list"/>
        <member container="?list" child="?city"/>
      </conditions>
    </rule>
  </template>
</tree>
```

Le générateur de modèle commence par récupérer la valeur de l'attribut `ref` qui dans ce cas est `http://www.xulplanet.com/rdf/weather/cities`. Cette ressource va être mise dans la variable `list` comme il est indiqué par la balise `content`. Vous obtiendrez les ressources associées à la ressource racine en utilisant la variable `list`.

Le générateur de modèle s'intéresse ensuite à l'élément member. Le générateur est forcé de parcourir les fils d'un élément. Le parent est indiqué par l'attribut `container` et les fils par l'attribut `child`. Dans l'exemple ci-dessus, la valeur de l'attribut `container` est la variable `list`. Ainsi le parent sera la valeur de la variable `list` qui a la valeur de la ressource racine `http://www.xulplanet.com/rdf/weather/cities`. L'effet induit va être de parcourir à travers la liste des fils de 'http://www.xulplanet.com/rdf/weather/cities'.

Si vous regardez en détail le RDF ci-dessus, la ressource 'http://www.xulplanet.com/rdf/weather/cities' a quatre filles, une pour chaque ville. Le générateur de modèle parcourt chacune d'elle, comparant la fille avec la valeur de l'attribut `child`. Dans le cas présent, celui-ci contient la valeur `city`. Donc le générateur va donner à la variable `city` la valeur des ressources filles au fur et à mesure.

Comme il n'y a pas d'autres conditions, la condition correspond à chacune des quatre ressources et le générateur va créer du contenu pour chacune des quatre. Bien sûr, l'exemple ci-dessus n'a aucun contenu. On l'ajoutera par la suite.

L'élément suivant est l'élément triple. Il est utilisé pour vérifier l'existence d'un triplet (ou assertion) dans la source de données du RDF. Un triplet est comme la propriété d'une ressource. Par exemple, un triplet existe entre un marque-page et son URL associée. Il peut-être exprimé ainsi :

Un marque-page vers mozilla.org -> URL -> www.mozilla.org

Cela signifie qu'il existe un triplet entre le marque-page *Un marque-page vers mozilla.org* et *www.mozilla.org* par la propriété URL. La première partie est appelée le sujet, la seconde, le prédicat, et la dernière, l'objet. Exprimé avec l'élément triple, ce mécanisme est décrit comme ceci :

```
<triple subject="Un marque-page vers mozilla.org"
         predicate="URL"
         object="www.mozilla.org"/>
```

Ce code a été un peu simplifié par rapport au code réel. Le prédicat devrait normalement inclure les espaces de nom (namespace), et le sujet devrait être l'identifiant ressource du marque-page, et non pas le titre du marque-page comme ici. En fait, le titre du marque-page devrait être un autre triplet dans la source de données qui utiliserait le prédicat 'nom'.

Vous pouvez remplacer le sujet et l'objet dans l'élément triple avec des références aux variables, auquel cas les valeurs seront substituées aux variables. Si aucune valeur n'est définie pour une variable, le générateur de modèle va attribuer la valeur de la source de données à la variable.

Disons par exemple, que l'on veuille ajouter une prédiction météo à la source de données des villes. Les conditions suivantes peuvent être utilisées :

```
<conditions>
  <content uri="?list"/>
  <member container="?list" child="?city"/>
  <triple subject="?city"
          predicate="http://www.xulplanet.com/rdf/weather#prediction"
          object="?pred"/>
</conditions>
```

Le générateur de modèle va parcourir chaque ville comme précédemment. Lorsqu'il va en arriver au triplet, il va s'intéresser aux assertions de la source de données RDF pour une prédiction météo. La prédiction météo est attribuée à la variable *pred*. Le générateur va répéter cette opération pour chacun des quatre villes. Une comparaison a lieu et le générateur va créer du contenu pour chaque ville qui a une prédiction météo associée. Si la ville n'a pas de ressource de prédiction, la condition ne correspond pas et aucun contenu ne sera créé pour cette ville. Remarquez que vous n'avez pas besoin de mettre 'rdf:' au début du prédicat, car il est sous-entendu.

On peut aussi remplacer l'attribut `object` avec une valeur statique. par exemple :

```
<conditions>
  <content uri="?city"/>
  <triple subject="?city"
    predicate="http://www.xulplanet.com/rdf/weather#prediction"
    object="Nuageux"/>
</conditions>
```

Cet exemple est similaire mais nous spécifions que nous voulons une comparaison qui s'effectue sur *Nuageux*. Le résultat obtenu est que la condition ne va s'appliquer que pour les villes dont la prédiction météo est *Nuageux*.

On peut ajouter davantage de triplet pour réaliser plus de comparaisons. Par exemple, dans l'exemple ci-dessus, on peut également vouloir vérifier la température et la vitesse du vent. Pour cela, il suffit d'ajouter un autre triplet qui vérifiera les ressources supplémentaires. La condition va correspondre si et seulement si l'intégralité des triplets retournent des valeurs.

L'exemple ci-dessous va vérifier un triplet supplémentaire concernant le nom de la ville. Il lui sera attribué une variable `name`. La condition va correspondre si la ville a à la fois un nom et une prédiction météo.

```
<conditions>
  <content uri="?list"/>
  <member container="?list" child="?city"/>
  <triple subject="?city"
    predicate="http://www.xulplanet.com/rdf/weather#name"
    object="?name"/>
  <triple subject="?city"
    predicate="http://www.xulplanet.com/rdf/weather#prediction"
    object="?pred"/>
</conditions>
```

## Générer du contenu

Le contenu à générer pour une règle est spécifié dans l'élément `action`. Il peut être le contenu des lignes d'un arbre, des items de menu ou tout ce que vous souhaitez générer. À l'intérieur du contenu, vous pouvez vous référer aux variables qui ont été définies dans les conditions. Ainsi, dans l'exemple météo ci-dessus, vous pouvez utiliser les variables `name` ou `pred` pour afficher la ville ou la prédiction météo. Vous pouvez aussi utiliser les variables `list` ou `city` mais elles contiennent des ressources et non du texte, donc elles n'auront pas de sens pour les utilisateurs.

Dans la syntaxe de règle simple, on utilise la syntaxe `uri="rdf: *"` pour indiquer l'emplacement où le contenu doit être généré. Dans la syntaxe de règles complètes, vous définissez la valeur de l'attribut `uri` à une variable que l'on a utilisé dans la partie des conditions. Normalement, ce sera la variable assignée à l'attribut `child` de l'élément `member`.

L'exemple suivant montre un arbre complet avec des conditions et une action. Vous pouvez consulter le fichier RDF séparément ( ).

### Exemple 9.5.1

```
<tree id="weatherTree" flex="1" datasources="weather.rdf"
  ref="http://www.xulplanet.com/rdf/weather/cities">
  <treecols>
    <treecol id="city" label="Ville" primary="true" flex="1"/>
    <treecol id="pred" label="Prédiction Météo" flex="1"/>
  </treecols>
```

```

<template>
  <rule>
    <conditions>
      <content uri="?list"/>
      <member container="?list" child="?city"/>
      <triple subject="?city"
        predicate="http://www.xulplanet.com/rdf/weather#name"
        object="?name"/>
      <triple subject="?city"
        predicate="http://www.xulplanet.com/rdf/weather#prediction"
        object="?pred"/>
    </conditions>
    <action>
      <treechildren>
        <treeitem uri="?city">
          <treerow>
            <treecell label="?name"/>
            <treecell label="?pred"/>
          </treerow>
        </treeitem>
      </treechildren>
    </action>
  </rule>
</template>
</tree>

```

Deux colonnes apparaissent dans cet arbre, l'une qui affiche la valeur de 'name' pour chacun ligne, et l'autre qui affiche le résultat de la prédiction météo.

Si vous utilisez le marqueur *dont-build-content* (ne-pas-générer-de-contenu) sur un arbre, remplacez l'élément content par un élément treeitem

## Ajouter des liens supplémentaires

Le dernier élément que vous pouvez ajouter à l'intérieur d'une règle est l'élément bindings. À l'intérieur de celui-ci, vous pouvez mettre un ou plusieurs éléments binding. Un lien dans une règle a la même syntaxe qu'un triplet et remplit quasiment la même fonction. Par exemple, dans l'exemple météo précédent, on peut ajouter le lien suivant :

```

<bindings>
  <binding subject="?city"
    predicate="http://www.xulplanet.com/rdf/weather#temperature"
    object="?temp"/>
</bindings>

```

Le lien va prendre la ressource "température" de chaque ville et l'attribuer à la variable `temp`. C'est tout à fait similaire à ce qu'un triplet accomplit. La différence se situe dans le fait qu'un lien n'est pas examiné lorsque quand les conditions sont vérifiées. Ainsi, chaque ville doit avoir un nom et une prédiction météo associée pour être affichée, mais cela n'a aucune importance si elle n'a pas de température. Cependant, si elle en a une, la valeur sera attribuée à la variable `temp` afin qu'elle soit utilisée dans une action. Si une ville n'a pas de température, la variable `temp` sera une chaîne de caractère vide.

---

Par la suite, nous verrons comment sauvegarder les états des éléments XUL.

## 9.6 Données persistantes

Écrit par Neil Deakin. Traduit par *Alain B.* (20/02/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/persist.html>

Cette section décrit comment sauvegarder l'état d'une fenêtre XUL.

## Mémorisation d'un état

Lors de la création d'une application importante, vous souhaitez souvent vouloir sauvegarder l'état d'une fenêtre tout au long des sessions. Par exemple, la fenêtre doit se souvenir quelles étaient les barres d'outils masquées après que l'utilisateur l'ait quittée.

Une possibilité serait d'écrire un script chargé de collecter l'information que vous voulez sauvegarder et de l'enregistrer dans un fichier. Toutefois, cette opération deviendrait pénible à chaque application. Heureusement, XUL propose un mécanisme pour sauvegarder les états d'une fenêtre.

L'information est collectée et stockée dans un fichier RDF (localstore.rdf) dans le même répertoire que les autres préférences de l'utilisateur. Ce fichier retient les états de chaque fenêtre. Cette méthode a l'avantage de fonctionner avec les profils utilisateurs de Mozilla, ainsi chaque utilisateur a ses propres paramètres.

XUL vous permet de sauvegarder l'état de n'importe quel élément. Typiquement, vous voulez sauvegarder les états des barres d'outils, les positions des fenêtres et si certains panneaux sont affichés ou non, mais vous pouvez sauvegarder presque tout.

Pour permettre la sauvegarde d'états, vous devez simplement ajouter l'attribut `persist` à l'élément qui contient la valeur que vous voulez sauvegarder. L'attribut `persist` doit être affecté par une liste d'attributs séparés par des espaces. L'élément concerné doit également avoir un attribut `id` pour permettre de l'identifier.

Par exemple, pour sauvegarder la position d'une fenêtre, vous devrez procéder comme ceci :

```
<window
  id="someWindow"
  width="200"
  height="300"
  persist="width height"
  .
  .
  .
```

Les deux attributs de l'élément `window`, `width` et `height`, seront sauvegardés. Vous pourriez ajouter à l'attribut `persist` des attributs supplémentaires à mémoriser tout en les séparant par un espace. Vous pouvez ajouter l'attribut `persist` à n'importe quel élément et mémoriser n'importe quel attribut. Vous pouvez utiliser des valeurs inhabituelles si vous ajustez les attributs par un script.

Ajoutons un attribut `persist` à quelques éléments de notre boîte de dialogue de recherche de fichiers. Pour sauvegarder la position de la fenêtre, nous devons modifier l'élément `window` :

```
<window
  id="findfile-window"
  title="Recherche de fichiers"
  persist="screenX screenY width height"
  orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Cette modification va sauvegarder la position x et y de la fenêtre, sa largeur et sa hauteur. Nous poursuivrons plus tard la mémorisation de l'état du sélecteur. La mémorisation de l'onglet sélectionné n'aurait aucun sens.

Exemple de recherche de fichiers :

Dans la section suivante, nous verrons comment appliquer des feuilles de style à des fichiers XUL.

# 10. Thèmes et localisation

## 10.1 Ajouter des feuilles de styles

Écrit par Neil Deakin. Traduit par *Dkoo* (19/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/style.html>

Jusqu'à présent, nous avons à peine modifier l'aspect visuel des éléments que nous avons créés. XUL utilise CSS (Cascading Style Sheets) pour personnaliser les éléments.

### Feuilles de styles

Une feuille de styles est un fichier qui contient des informations de style pour les éléments. Les styles ont été conçus au départ pour des éléments HTML mais ils peuvent également être appliqués à des éléments XUL ou à n'importe quels éléments XML. La feuille de styles contient des informations telles que les polices, couleurs, bordures et taille des éléments.

Mozilla applique une feuille de styles par défaut pour chaque fenêtre XUL. Dans la plupart des cas, il sera suffisant de laisser les valeurs par défaut telles quelles. Toutefois, vous pouvez fournir une feuille de styles personnalisée. En général vous associerez une seule feuille de styles à chaque fichier XUL.

Vous pouvez placer une feuille de styles où vous le désirez. Si votre fichier XUL se trouve sur un serveur distant et doit être accédé via une URL HTTP, vous pouvez également stocker la feuille de styles à distance. Si vous créez un paquetage XUL destiné à faire partie du système chrome, vous avez deux choix. Premièrement, vous pouvez placer la feuille de styles dans le même répertoire que le fichier XUL. Cette méthode a l'inconvénient d'interdire le changement du thème graphique de votre application. La seconde méthode consiste à placer vos fichiers à l'intérieur d'un thème.

Imaginons que nous construisions la boîte de dialogue de recherche de fichiers pour permettre le choix d'un thème. Comme la fenêtre peut être appelée par l'URL `chrome://findfile/content/findfile.xul`, la feuille de styles sera enregistrée dans `chrome://findfile/skin/findfile.css`.

Tous les exemples XUL ont utilisé une feuille de styles jusqu'à présent. La seconde ligne a toujours été :

```
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
```

Cette ligne indique que nous voulons utiliser le style fourni par `chrome://global/skin/`. Sous Mozilla, elle sera traduit par le fichier `global.css` qui contient les informations du style par défaut pour les éléments XUL. Si vous enlevez cette ligne, les éléments fonctionneront, toutefois ils apparaîtront dans un style plus simple. La feuille de styles utilise diverses polices, couleurs et bordures pour rendre l'apparence des éléments plus appropriée.

### Changer les styles

Il arrivera toutefois que l'apparence par défaut des éléments ne soit pas celle désirée. Dans ces cas, nous devons ajouter notre propre feuille de styles. Jusqu'à présent, nous avons appliqué différents styles en utilisant l'attribut `style` sur des éléments. Bien que ce soit une technique fonctionnelle, elle n'est pas la meilleure. Il est de loin préférable de créer une feuille de styles séparée. La raison est que des styles ou des thèmes différents peuvent être appliqués très facilement.

Il peut y avoir des cas où l'utilisation de l'attribut `style` est acceptable. Un bon exemple serait lorsqu'un script change le style d'un élément, ou quand une différence d'agencement pourrait changer la signification de l'élément. Cependant vous devriez l'éviter autant que possible.

Pour des fichiers installés, vous aurez à créer ou à modifier un fichier manifeste et installer le thème.

Modifions la boîte de dialogue de recherche de fichiers pour que son style provienne d'un fichier style séparé. Tout d'abord, voici les lignes modifiées de findfile.xul :

```
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="findfile.css" type="text/css"?>
...
<spacer class="titlespace"/>
  <groupbox orient="horizontal">
    <caption label="Critères de recherche"/>

    <menulist id="searchtype">
      <menupopup>
        <menuitem label="Nom"/>
        <menuitem label="Taille"/>
        <menuitem label="Date de modification"/>
      </menupopup>
    </menulist>
    <spacer class="springspace"/>
    <menulist id="searchmode">
      <menupopup>
        <menuitem label="Est"/>
        <menuitem label="N'est pas"/>
      </menupopup>
    </menulist>

    <spacer class="springspace"/>
    <menulist id="find-text" flex="1"
      editable="true"
      datasources="file:///mozilla/recents.rdf"
      ref="http://www.xulplanet.com/rdf/recent/all"/>
    ...
  <spacer class="titlespace"/>
</hbox>

<progressmeter id="progmeter" value="50%" style="display:none;"/>
```

La nouvelle ligne `xml-stylesheet` est utilisée afin d'importer la feuille de styles. Elle contiendra les styles au lieu de les avoir directement dans le fichier XUL. Vous pouvez inclure un nombre indéterminé de feuilles de styles de la même façon. Ici la feuille de styles est placée dans le même répertoire que `findfile.xul`.

Certains des styles dans le code ci-dessus ont été enlevés. La propriété `display` du `progressmeter` ne l'a pas été. Elle sera changée par un script donc, le style a été maintenu, car la barre de progression n'a pas à être visible au lancement. Vous pouvez toujours mettre le style dans une feuille de styles séparée si vous le souhaitez vraiment. Une classe a été ajoutée aux éléments `spacer` pour qu'ils puissent être appelés.

Une feuille de styles a également besoin d'être créée. Créez un fichier `findfile.css` dans le même répertoire que le fichier XUL (Il devrait normalement être mis dans un thème séparé). Dans ce fichier, nous allons ajouter la déclaration de styles, comme indiqué ci-dessous :

```
#find-text {
  min-width: 15em;
}

#progmeter {
  margin: 4px;
}

.springspace {
  width: 10px;
```



```

}

.titlespace {
    height: 10px;
}

```

Remarquez que ces styles sont équivalents aux styles que nous avons précédemment. Cependant, il est beaucoup plus facile pour quelqu'un de changer l'apparence de la boîte de dialogue de recherche de fichiers maintenant car il est possible d'ajouter ou modifier la déclaration de styles en modifiant le fichier ou en changeant le thème. Si l'utilisateur change le thème, les fichiers dans un répertoire autre que celui par défaut seront utilisés.

## Importer des feuilles de styles

Nous avons déjà vu comment importer des feuilles de styles. Un exemple est montré ci-dessous :

```
<?xml-stylesheet href="chrome://bookmarks/skin/" type="text/css"?>
```

Cette ligne peut être la première d'une fenêtre *bookmarks*. Elle importe la feuille de style *bookmarks*, qui est *bookmarks.css*. Le système de thème de Mozilla est assez intelligent pour savoir quelle feuille de styles utiliser, car le nom spécifique du fichier n'a pas été indiqué ici. Nous avons fait une chose similaire avec la feuille de styles globale (*chrome://global/skin*).

Une feuille de styles peut importer des styles d'une autre feuille en utilisant la directive *import*. Normalement, vous n'importerez qu'une seule feuille de styles de chaque fichier XUL. La feuille de styles globale peut être importée à partir de celle associée avec le fichier XUL. Ceci peut être fait grâce au code ci-dessous, vous permettant de retirer l'import du fichier XUL :

Importation de styles à partir de XUL :

```
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
```

Importation de styles à partir de CSS :

```
@import url(chrome://global/skin/);
```

La seconde syntaxe est préférable car elle réduit le nombre de dépendances à l'intérieur du fichier XUL lui-même.

Retirez l'importation de la feuille de styles globale dans *findfile.xul* et ajoutez l'importation dans *findfile.css*.

Tous les éléments peuvent être décorés à l'aide de CSS. Vous pouvez utiliser des sélecteurs pour sélectionner l'élément que vous souhaitez styler (Le sélecteur est la partie avant l'accolade dans une règle de style). La liste suivante résume quelques-uns des sélecteurs disponibles :

*button*

Désigne toutes les balises *button*.

*#special-button*

Désigne les éléments avec un id de *special-button*

*.bigbuttons*

Désigne tous les éléments avec une classe *bigbuttons*

*button.bigbuttons*

Désigne tous les éléments *button* avec une classe à *bigbuttons*

*toolbar > button*

Désigne tous les boutons directement insérés dans un élément *toolbar*.

*toolbar > button.bigbuttons*

Désigne tous les éléments button avec une classe *bigbuttons*, directement insérés dans un élément toolbar.

*button.bigbuttons: hover*

Désigne tous les éléments button avec une classe *bigbuttons* mais seulement lorsque la souris se trouve au dessus d'eux.

*button#special-button: active*

Désigne tous les éléments button avec un id *special-button* mais seulement lorsqu'ils sont actifs (en train d'être cliqués).

*box[orient="horizontal"]*

Désigne tous les éléments box avec un attribut *orient* réglé sur *horizontal*.

Vous pouvez combiner ces règles comme vous le désirez. C'est toujours une bonne idée d'être aussi précis que possible lorsque vous spécifiez ce qui doit être décoré et comment. C'est bien plus efficace et réduit également les risques de décorer un mauvais élément.

---

Dans la prochaine section, nous verrons comment appliquer des styles aux arbres.

La boîte de dialogue de recherche de fichiers à ce stade :

## 10.2 Styler un arbre

Écrit par Neil Deakin. Traduit par *Durandal* (13/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/treestyle.html>

Cette section explique comment styler un arbre.

### Styler l'arbre

Vous pouvez styler la bordure de l'arbre et les en-têtes de colonnes de la même manière que pour d'autres éléments. Les styles ajoutés à l'élément tree seront appliqués à l'arbre entier. L'ajout d'un style à l'élément treecol ne l'applique pas à la colonne mais seulement à son en-tête.

Le corps de l'arbre doit être stylé d'une manière un peu différente des autres éléments, parce que le corps de l'arbre est stocké d'une manière différente des autres éléments. Le treechildren extérieur est le seul vrai élément du corps de l'arbre. Les éléments intérieurs n'ont qu'un rôle de conteneurs.

Vous devez donc utiliser l'attribut *properties* sur les lignes ou les cellules pour modifier une ou plusieurs propriétés nommées. Cet attribut peut être utilisé avec des arbres à contenu statique, produit par RDF ou avec des arbres à vue personnalisée. Supposons que nous voulions donner une couleur de fond bleue à une ligne particulière, comme il est possible de le remarquer sur les libellés dans la messagerie de Mozilla. Nous allons utiliser une propriété appelée *makeItBlue*. Vous pouvez utiliser le nom que vous voulez. Vous pouvez modifier plusieurs propriétés en les séparant par des espaces.

Modifiez la propriété sur une ligne ou une cellule, comme dans l'exemple suivant :

```
<treerow properties="makeItBlue">
```

La feuille de styles peut récupérer cette propriété et l'utiliser pour changer l'apparence d'une ligne pour les messages non lus ou les libellés. Les propriétés fonctionnent en quelque sorte comme les classes de style, bien qu'elles nécessitent une syntaxe un peu plus complexe à utiliser dans une feuille de styles. En effet, il est possible de spécifier un style pour chaque partie d'une cellule. Vous pouvez styler non seulement la cellule et son texte, mais aussi le "twisty" (NdT : petit '+' ou '-' permettant de développer et replier l'arborescence) et l'indentation. La syntaxe à utiliser est la suivante :

```
treechildren::-moz-tree-row(makeItBlue)
{
    background-color: blue;
}
```

Ce pseudo-style spécial est utilisé pour styler la couleur de fond des lignes qui ont la propriété *makeItBlue*. Cette syntaxe spéciale est nécessaire parce que les cellules elles-mêmes ne sont pas des éléments séparés. Tout le contenu intérieur au corps de l'arbre obtient son rendu par l'élément treechildren (Notez que les styles de la règle ci-dessus concernent l'élément treechildren). Le pseudo-style définit des règles de style pour des parties particulières de ce qu'il affiche. Cette règle de style signifie, dans un élément treechildren : mettre une couleur de fond bleue à toutes les lignes de l'arbre qui ont la propriété *makeItBlue*.

Le texte `::-moz-tree-row` spécifie quelle est la zone de contenu désirée, qui est dans ce cas une ligne. Vous pouvez aussi utiliser les valeurs suivantes :

```
::-moz-tree-cell
    une cellule. Utilisez ceci pour modifier les couleurs des bordures et du fond.
::-moz-tree-cell-text
    le texte d'une cellule. Utilisez ceci pour modifier la police et la couleur du texte.
::-moz-tree-twisty
    l'apparence du "twisty" utilisé pour développer et replier les lignes filles.
::-moz-tree-image
    l'image pour une cellule. Vous pouvez indiquer l'image avec la propriété list-style-image.
::-moz-tree-row
    une ligne. Utilisez ceci pour choisir la couleur de fond d'une ligne.
::-moz-tree-indentation
    l'indentation à gauche des lignes filles.
::-moz-tree-column
    une colonne.
::-moz-tree-line
    les lignes dessinées pour connecter les lignes filles aux lignes parentes.
::-moz-tree-separator
    un séparateur dans un arbre.
::-moz-tree-progressmeter
    contenu pour des cellules à indicateur de progression. Vous pouvez créer une colonne avec des
    indicateurs de progression en mettant l'attribut type de la colonne à progressmeter.
::-moz-tree-drop-feedback
    le retour du glisser-déposer.
```

Vous pouvez tester la présence de plusieurs propriétés en les séparant par des virgules. L'exemple ci-dessous met une couleur de fond grise aux lignes qui ont les propriétés *readonly* et *unread*. Pour les propriétés qui sont *readonly*, il ajoute une bordure rouge autour de la ligne. Notez que la première règle s'appliquera à toute ligne qui est *readonly*, peu importe la présence d'autres propriétés comme *unread*.

```
treechildren::-moz-tree-row(readonly)
{
    border: 1px solid red;
}

treechildren::-moz-tree-row(readonly, unread)
{
    background-color: rgb(80%, 80%, 80%);
}
```

La liste des propriétés des éléments tree contient un petit nombre de propriétés par défaut que vous pouvez aussi utiliser dans une feuille de style. Vous pouvez utiliser ces propriétés supplémentaires pour modifier

l'apparence des conteneurs ou des lignes sélectionnées. Les propriétés suivantes sont modifiées automatiquement en cas de besoin :

*focus*

cette propriété est mise si l'arbre a le focus.

*selected*

cette propriété est mise pour les lignes ou les cellules actuellement sélectionnées.

*current*

cette propriété est mise si le curseur est sur la ligne. Seule une ligne à la fois peut avoir cette propriété.

*container*

cette propriété est mise pour les lignes ou les cellules qui ont des lignes filles.

*leaf*

cette propriété est mise pour les lignes ou les cellules qui n'ont pas de lignes filles.

*open*

cette propriété est mise pour les lignes ou les cellules qui sont développées.

*closed*

cette propriété est mise pour les lignes ou les cellules qui sont repliées.

*primary*

cette propriété est mise pour les cellules de la colonne primaire.

*sorted*

cette propriété est mise pour les cellules de la colonne actuellement triées.

*even*

cette propriété est mise pour les lignes paires.

*odd*

cette propriété est mise pour les lignes impaires. Cette propriété ainsi que la propriété *even* vous permettent, par exemple, d'alterner les couleurs entre chaque ligne.

*dragSession*

cette propriété est mise si quelque chose est en train d'être déplacé.

*dropOn*

si un déplacement a lieu au-dessus de l'arbre, cette propriété est mise pour la ligne en train d'être survolée par le déplacement, tant que le pointeur de la souris est au-dessus de la ligne.

*dropBefore*

cette propriété est mise si le pointeur de la souris survole avant la ligne en cours de déplacement.

*dropAfter*

cette propriété est mise si le pointeur de la souris survole après la ligne en cours de déplacement.

*progressNormal*

cette propriété est mise pour les cellules à indicateur de progression.

*progressUndetermined*

cette propriété est mise pour les cellules à indicateur de progression indéterminé.

*progressNone*

cette propriété est mise pour les cellules sans indicateur de progression.

Les propriétés sont mises pour les lignes ou les cellules d'une ligne à l'état correspondant. Pour les colonnes et les cellules, une propriété additionnelle, l'id de la colonne ou la colonne dans laquelle est la cellule, sera mise.

Pour les arbres générés par RDF, vous pouvez utiliser la même syntaxe. Cependant, vous affecterez souvent les propriétés en vous basant sur des valeurs de la source de données.

Pour des arbres avec un script de vue personnalisée, vous pouvez modifier des propriétés en fournissant les fonctions *getRowProperties*, *getColumnProperties* et *getCellProperties* dans la vue. Elles renvoient des informations à propos d'une ligne, d'une colonne et d'une cellule individuelle. Les arguments à ces fonctions indiquent quelle ligne et/ou colonne. Le dernier argument de chacune de ces

fonctions est une liste de propriétés que la vue est supposée remplir avec une liste de propriétés. La fonction `getColumnProperties` fournit aussi l'élément `treecol` correspondant pour la colonne.

```
getRowProperties : function(row,prop){}
getColumnProperties : function(column,columnElement,prop){}
getCellProperties : function(row,column,prop){}
```

Regardons un exemple de modification d'une cellule spécifique. Rendons le texte bleu une ligne sur quatre, en utilisant l'exemple d'une section précédente. Nous allons avoir besoin d'ajouter du code à la fonction `getCellProperties` pour ajouter une propriété *makeItBlue* aux cellules toutes les quatre lignes (Nous n'utilisons pas `getRowProperties` puisque la couleur du texte ne sera pas héritée dans chaque cellule).

L'objet `properties` qui est passé en dernier argument de `getCellProperties` est un objet XPCOM qui implémente `nsISupportsArray`. Il s'agit en fait une version XPCOM d'un tableau. Il contient une fonction `AppendElement` qui peut être utilisée pour ajouter un élément au tableau. Nous pouvons utiliser l'interface `nsIAtomService` pour construire des atomes de chaînes pour les propriétés.

```
getCellProperties: function(row,col,props){
    if ((row % 4) == 0){
        var aserv=Components.classes["@mozilla.org/atom-service;1"].
            getService(Components.interfaces.nsIAtomService);
        props.AppendElement(aserv.getAtom("makeItBlue"));
    }
}
```

Cette fonction sera définie comme partie d'un objet de vue. Elle vérifie d'abord quelle ligne est demandée et affecte une propriété pour les cellules toutes les quatre lignes. La liste des propriétés nécessite un tableau d'objets `atom` qui peuvent être considérés comme des chaînes de caractères constantes. Nous les créons en utilisant l'interface XPCOM `nsIAtomService` et nous les ajoutons au tableau en utilisant la fonction `AppendElement`. Ici, nous créons un atome *makeItBlue*. Vous pouvez appeler `AppendElement` à nouveau pour ajouter des propriétés additionnelles.

Dans la suite, nous allons voir comment modifier le thème par défaut.

## 10.3 Modification du thème par défaut

Écrit par Neil Deakin. Traduit par *Durandal* (21/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/defskin.html>

Cette section décrit comment modifier le thème graphique d'une fenêtre.

### Les bases d'un thème

Un thème est un ensemble de feuilles de styles, d'images et de comportements qui est appliqué à un fichier XUL. En appliquant un thème différent, vous pouvez changer l'apparence d'une fenêtre sans changer ses fonctionnalités. Mozilla fournit deux thèmes par défaut, Classic et Modern, et vous pouvez en télécharger d'autres. Le XUL pour les deux est le même, par contre les feuilles de styles et les images utilisées sont différentes.

Pour une simple personnalisation de l'apparence d'une fenêtre Mozilla, vous pouvez facilement changer les feuilles de styles qui lui sont associées. Des changements plus importants peuvent être faits en créant un thème complètement nouveau. La fenêtre des préférences de Mozilla comporte un panneau pour changer le thème par défaut.

Un thème est décrit en utilisant des CSS, ce qui vous permet de définir les couleurs, les bordures et les images utilisées pour dessiner des éléments. Les fichiers `classic.jar` et `modern.jar` contiennent les définitions

des thèmes. Le répertoire "global" inclus dans ces archives contient les définitions principales des styles concernant la manière d'afficher les différents éléments XUL. En modifiant ces fichiers, vous pouvez changer l'apparence des applications XUL.

Si vous placez un fichier appelé *userChrome.css* dans le répertoire "chrome" dans le répertoire de votre profil utilisateur, vous pouvez remplacer des paramètres sans changer les archives elles-mêmes. Ce répertoire devrait être créé quand vous créez un profil et quelques exemples placés ici. Le fichier *userContent.css* permet de personnaliser les pages Web, tandis que *userChrome.css* permet de personnaliser les fichiers chrome.

Par exemple, en ajoutant ce qui suit à la fin de ce fichier, vous pouvez changer tous les éléments menubar pour leur donner un fond rouge.

```
menubar {
    background-color: red;
}
```

Si vous ouvrez une fenêtre Mozilla après avoir effectué ce changement, les barres de menu seront rouges. Comme ce changement a été fait à la feuille de styles utilisateur, il affecte toutes les fenêtres. Cela signifie que la barre de menu du navigateur, la barre de menu des marque-pages et même la barre de menu du dialogue de recherche de fichiers seront rouges.

Pour que le changement n'affecte qu'une fenêtre, changez la feuille de styles associée avec ce fichier XUL. Par exemple, pour ajouter une bordure rouge autour des commandes de menu dans la fenêtre du carnet d'adresses, ajoutez ce qui suit au fichier *addressbook.css* dans l'archive *modern.jar* ou *classic.jar*.

```
menuitem {
    border: 1px solid red;
}
```

Si vous regardez dans les archives de thème, vous remarquerez que chacune contient un certain nombre de feuilles de styles et d'images. Les feuilles de styles font référence aux images. Vous devriez éviter de faire directement référence aux images dans les fichiers XUL si vous voulez que votre contenu puisse être modifié par un thème, parce qu'un certain thème peut ne pas utiliser d'images et avoir besoin d'un design plus complexe. En faisant référence aux images avec les CSS, on peut facilement les enlever. Cela enlève aussi la dépendance sur les noms de fichier spécifiques des images.

Vous pouvez attribuer des images à un bouton, une case à cocher et à d'autres éléments en utilisant la propriété `list-style-image` comme suit :

```
checkbox {
    list-style-image: url("chrome://findfile/skin/images/check-off.jpg");
}

checkbox:checked {
    list-style-image: url("chrome://findfile/skin/images/check-on.jpg");
}
```

Ce code change l'image associée à une case à cocher. Le premier style met une image pour une case à cocher à l'état normal et le second style pour une case à cocher cochée. Le modificateur 'checked=true' n'applique le style qu'aux éléments qui ont leur attribut `checked` à *true*.

Dans la prochaine section, nous allons voir comment créer un nouveau thème.

## 10.4 Créer un thème

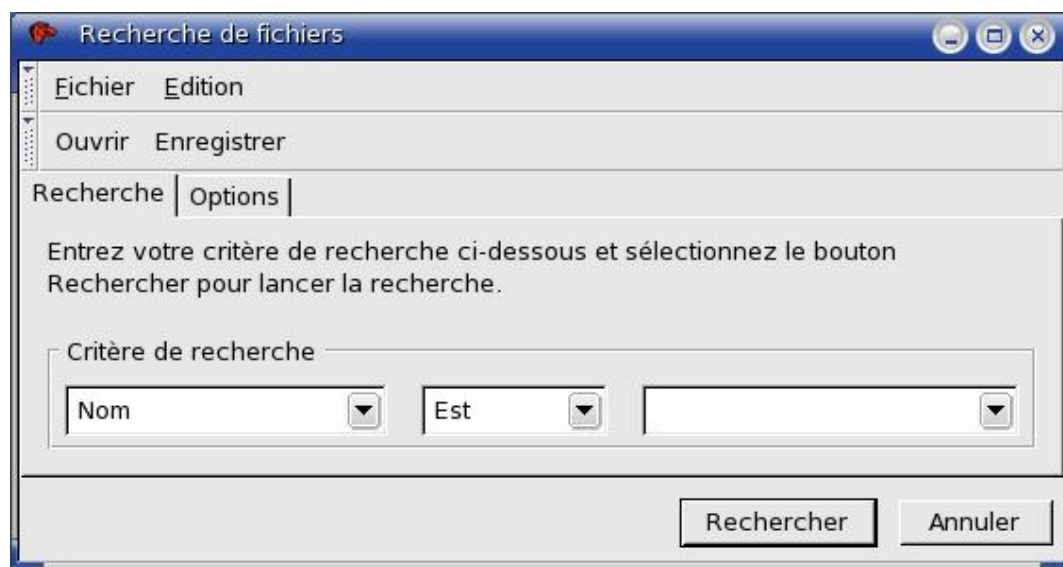
Écrit par Neil Deakin. Traduit par *Durandal* (29/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/cskin.html>

Cette section décrit comment créer un thème simple. Nous ne l'appliquerons qu'à la fenêtre de recherche de fichiers.

### Un thème simple

L'image ci-dessous montre la boîte de dialogue de recherche de fichiers actuelle. Créons un thème que nous pourrions lui appliquer. Normalement, un thème doit s'appliquer à l'application entière, mais, pour plus de facilité, nous allons nous concentrer sur la boîte de dialogue de recherche de fichiers. Pour cette raison, nous n'allons modifier que le fichier `findfile.css` plutôt que le fichier `global.css`. Cette section suppose que vous démarriez avec le thème Classic. Si vous le souhaitez, faites une copie des fichiers utilisés par la boîte de dialogue de recherche de fichiers avant de les modifier.



Note du traducteur : Pour les versions récentes de mozilla (> 1.6) et de firefox (>0.8), avec le fichier `findfile.css` tel qu'il est, il se peut que vous n'obteniez pas un design identique à ce que montre l'image. En effet, dans le thème classic, des styles `-moz-appearance` ont été ajoutés depuis aux différents éléments, leur permettant d'avoir le même aspect que les éléments natifs de l'interface graphique utilisée (windows ou kde, macos, etc..). Ce style empêche certains styles d'être appliqués correctement.

Pour pouvoir modifier complètement l'apparence des éléments utilisés dans notre exemple et afin de suivre le tutoriel, il a été rajouté le code suivant dans la feuille de style de l'exemple :

```
tab, button, menulist, menubar,menupopup,toolbar,tabpanel {
  -moz-appearance:none;
}
```

Vous devez créer un fichier '`findfile.css`' dans un thème personnalisé. Ou vous pouvez le placer temporairement dans le répertoire "content" et y faire référence en utilisant une directive de la feuille de styles. Vous pouvez modifier le fichier `findfile.css` existant directement pour voir à quoi cela ressemble, ou vous pouvez créer un thème personnalisé et y faire un lien. Pour créer un thème, faites ce qui suit :

1. Créez un répertoire quelque part où vous placerez les fichiers du thème.
2. Copiez un fichier manifeste (`contents.rdf`) du thème Classic ou Modern dans ce nouveau répertoire.

3. Modifiez les références dans le fichier manifeste à un nom personnalisé pour votre thème. Par exemple, changez les références de *classic/1.0* vers *blueswayedshoes/1.0*.
4. Ajoutez une ligne de la forme suivante au fichier *chrome/installed-chrome.txt* :  
`skin,install,url,file:///stuff/blueswayedshoes/` où la dernière partie pointe vers le répertoire que vous avez créé. Vérifiez bien que vous avez mis un slash à la fin.

Copiez le fichier *findfile.css* original dans le nouveau répertoire. Nous l'utiliserons comme base pour le nouveau thème. Nous pourrions y faire référence par la suite en utilisant l'URL *chrome://findfile/skin/findfile.css*. Décidons d'abord quels types de changements nous voulons effectuer. Nous allons faire de simples changements de couleurs, modifier les styles des boutons et modifier un peu l'espacement. Commençons par les menus, les barres d'outils et le panneau global des onglets.

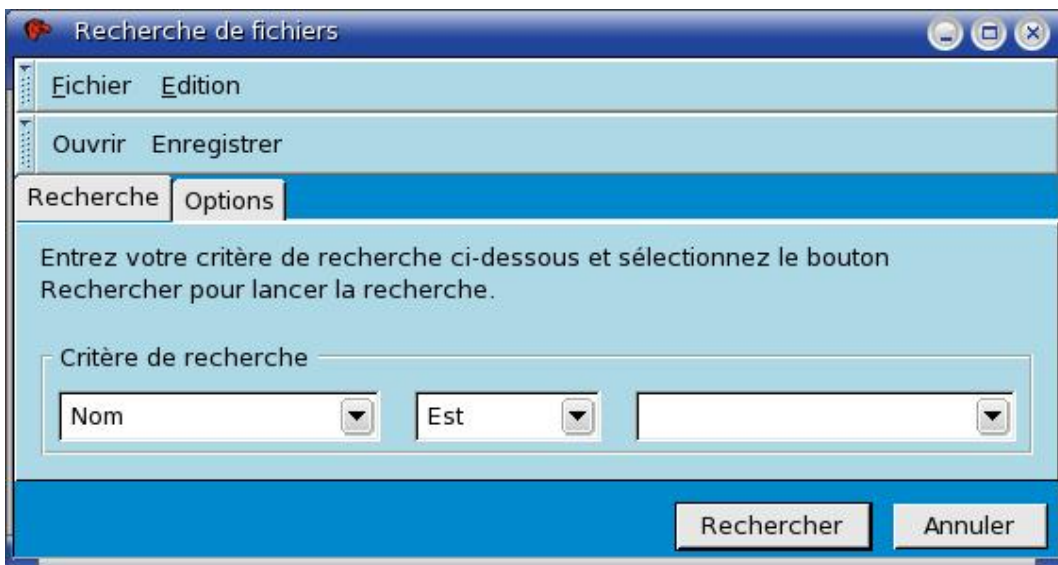
L'ajout des règles de styles suivantes à *findfile.css* provoquera les changements montrés dans l'image qui suit.

```

window > vbox {
  background-color: #0088CC;
}

menubar,menupopup,toolbar,tabpanel {
  background-color: lightblue;
  border-top: 1px solid white;
  border-bottom: 1px solid #666666;
  border-left: 1px solid white;
  border-right: 1px solid #666666;
}

caption {
  background-color: lightblue;
}
    
```



La boîte intérieure de la fenêtre (qui entoure en fait tout le contenu de la fenêtre) a été changée pour avoir une couleur bleue. Vous pouvez voir ce bleu derrière la bande des onglets et le long du bas de la fenêtre. Les quatre éléments menubar, menupopup, toolbar et tabpanel apparaissent en bleu clair. La bordure autour de ces quatre éléments a été modifiée pour donner une apparence 3D plus forte. Vous pouvez le voir si vous regardez attentivement. La couleur de fond du caption a aussi été modifiée pour correspondre avec le fond.

La première règle au-dessus (pour `window > vbox`) spécifie que la boîte enfant de la fenêtre principale a une couleur différente. Ce n'est probablement pas la meilleure façon de procéder. Nous devrions modifier la



fenêtre et utiliser une classe de style à la place. C'est ce que nous allons faire. De cette manière, nous pourrions modifier le code XUL sans avoir besoin de conserver la boîte comme premier élément enfant de la fenêtre.

```
.findfilesbox {
  background-color: #0088CC;
}
```

**XUL:**

```
<vbox class="findfilesbox" orient="vertical" flex="100%">
<toolbox>
```

Ensuite, modifions les onglets. Nous allons rendre l'onglet sélectionné en gras et changer les arrondis sur les onglets.

```
tab:first-child {
  -moz-border-radius: 4px 0px 0px 0px;
}

tab:last-child {
  -moz-border-radius: 0px 4px 0px 0px;
}

tab[selected="true"] {
  color: #000066;
  font-weight: bold;
  text-decoration: underline;
}
```

Deux règles changent l'apparence normale de l'élément `tab`, la première met un arrondi sur le premier onglet et la deuxième met un arrondi sur le dernier. On utilise ici une règle de styles spéciale de Mozilla, `-moz-border-radius`, qui crée des coins arrondis. La bordure supérieure gauche du premier onglet et la bordure supérieure droite du deuxième onglet sont arrondies de quatre pixels et les autres coins sont arrondis de zéro pixels, ce qui est équivalent à aucun arrondi. Augmentez ces valeurs pour un arrondi plus prononcé et diminuez-les pour une apparence plus rectangulaire.



La dernière règle ne s'applique qu'aux onglets qui ont leur attribut `selected` à la

valeur `true`. Elle met le texte d'un onglet sélectionné en gras, souligné et bleu foncé. Notez dans l'image que ce style n'est appliqué qu'au premier onglet, puisqu'il est sélectionné.

Il est assez difficile de distinguer les boutons de la barre d'outils des commandes du menu. Nous pourrions ajouter des icônes aux boutons pour les rendre plus clairs. L'éditeur Mozilla Composer fournit des icônes pour les boutons ouvrir et sauvegarder, nous les utiliserons pour gagner du temps. Nous pouvons choisir l'image d'un bouton en utilisant la propriété CSS `list-style-image`.

```
#opensearch {
  list-style-image: url("chrome://editor/skin/icons/btn1.gif");
  -moz-image-region: rect(48px 16px 64px 0);
  -moz-box-orient: vertical;
}

#saveas {
  list-style-image: url("chrome://editor/skin/icons/btn1.gif");
  -moz-image-region: rect(80px 16px 96px 0);
  -moz-box-orient: vertical;
}
```

Mozilla met à disposition une propriété de style spécifique, `-moz-image-region`, qui permet à un élément d'utiliser une partie d'une image. Vous pouvez vous la représenter comme un découpage de l'image. Vous mettez comme valeur de la propriété une position et une taille comprise dans une image et le bouton n'affichera que cette section de l'image. Cette technique vous permet d'utiliser la même image pour plusieurs boutons et de mettre une portion différente pour chacun. Quand vous avez beaucoup de boutons, avec chacun d'eux des états pour hover, active et disabled, elle fait gagner de l'espace qui serait sinon occupé par plusieurs images. Dans le code ci-dessus, nous utilisons la même image pour tous les boutons, mais nous mettons une portion différente de l'image pour chacun. Si vous regardez cette image (btn1.gif), vous remarquerez qu'elle contient une grille d'images plus petites, de 16 fois 16 pixels chacune.

Note du traducteur : étant donné que l'image en question fait partie de Mozilla Composer, si vous ouvrez l'exemple avec Firefox, vous ne verrez pas les images des boutons, car Mozilla Composer n'est livré qu'avec la suite Mozilla.



La propriété `-moz-box-orient` est utilisée pour orienter le bouton verticalement,

pour que l'image apparaisse au-dessus de son libellé. Cette propriété a la même signification que l'attribut `orient`. Elle est pratique comme le thème ne peut pas modifier le code XUL. La plupart des attributs de l'élément `box` ont des propriétés CSS qui leur correspondent.

Nous allons maintenant faire quelques changements aux boutons situés en bas de la boîte de dialogue, en réutilisant à nouveau des icônes de Mozilla pour gagner du temps. Si vous créez votre propre thème, vous aurez besoin de créer de nouvelles icônes ou de copier les icônes dans de nouveaux fichiers. Si vous suivez l'exemple de cette section, copiez juste les fichiers vers votre nouveau thème et modifiez les URL en conséquence.

```
#find-button {
    list-style-image: url("chrome://global/skin/checkbox/images/cbox-check.jpg");
    font-weight: bold;
}

#cancel-button {
    list-style-image: url("chrome://global/skin/icons/images/close-button.jpg");
}

button:hover {
    color: #000066;
}
```



Nous ajoutons des images aux boutons et mettons le texte du bouton

Rechercher en gras pour indiquer que c'est le bouton par défaut. La dernière règle s'applique aux boutons quand la souris les survole (état hover). Nous mettons une couleur de texte bleu foncé dans ce cas.

Finalement, quelques changements mineurs de l'espacement autour des items, par l'utilisation de marges :

```
tabbox {
    margin: 4px;
}

toolbarbutton {
    margin-left: 3px;
    margin-right: 3px;
}
```

Après ces changements, la boîte de dialogue de recherche de fichiers apparaît maintenant comme ceci :



Comme vous pouvez le voir, de simples changements des règles de styles apportent une apparence assez différente à la fenêtre de recherche de fichiers. Nous pourrions continuer en modifiant les menus, les poignées sur la barre d'outils et les éléments textbox et checkbox.

## Création d'un thème global

Le thème créé ci-dessus est simple et ne s'applique qu'à la boîte de dialogue de recherche de fichiers. Certaines des modifications du thème pourraient être placées dans les feuilles de styles globales (situées dans le répertoire global du thème) pour s'appliquer à toutes les applications. Par exemple, il serait peu cohérent d'avoir des images différentes entre les cases à cocher de la fenêtre de recherche de fichiers et celles d'autres fenêtres. Ce changement devrait vraiment être intégré dans la feuille de styles globale.

Essayez de déplacer les styles CSS de `findfile.css` dans `global.css` puis regardez les fenêtres de dialogue de Mozilla (Le visualisateur de cookie est un bon exemple). Vous remarquerez qu'elles ont adopté les règles que nous avons ajoutées. Certaines des règles entrent en conflit avec celles déjà dans les feuilles de styles globales. Par exemple, des règles sont déjà définies pour les boutons, les onglets, etc., et nous avons défini des règles additionnelles pour ces éléments. Quand vous modifiez le thème global, vous devriez fusionner les changements avec les règles existantes.

Pour une meilleure adaptation du thème graphique, il vaut mieux déclarer les règles de styles liées à l'apparence dans le répertoire global plutôt que dans des fichiers de styles individuels. Les couleurs, les polices de caractère et l'apparence générale des composants graphiques doivent y être inclus. Si vous modifiez la couleur d'un élément dans un fichier de thème local (comme `findfile.css`), la fenêtre de dialogue peut paraître bizarre si l'utilisateur change son style global. N'espérez pas que l'utilisateur utilise le thème par défaut.

La section suivante explique comment rendre une application XUL localisable.

L'exemple du dialogue de recherche de fichier avec ce thème :

## 10.5 Localisation

Écrit par Neil Deakin. Traduit par **BrainBooster** (22/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/locale.html>

XUL et XML fournissent des entités qui sont une solution permettant la localisation.

## Entités

De nombreuses applications sont construites de telle sorte que la traduction de l'interface en différentes langues soit le plus simple possible. En général, une table de chaînes de caractères est créée pour chaque langue. Au lieu de la coder directement dans l'application, chaque partie de texte est seulement une référence dans la table de chaînes. XML fournit des entités qui peuvent être utilisées dans un but similaire.

Vous devriez déjà être familier avec les entités si vous avez déjà écrit en HTML. Les codes `&lt;` et `&gt;` sont des exemples d'entités qui peuvent être utilisées pour placer les signes "supérieur" et "inférieur" dans le texte. XML a une syntaxe qui autorise la déclaration de vos propres entités. Vous pouvez les utiliser de manière à ce que l'entité soit remplacée par sa valeur qui peut être une chaîne de caractères. Des entités peuvent être employées toutes les fois où du texte est utilisé, y compris pour les valeurs des attributs. L'exemple ci-dessous décrit l'utilisation d'une entité dans un bouton.

```
<button label="&findLabel;" />
```

Le texte qui apparaîtra sur le libellé sera la valeur de l'entité `&findlabel;`. Un fichier contenant les déclarations d'entités pour chaque langue supportée est créé. En français, on affectera probablement la valeur de texte *Rechercher* à l'entité `&findlabel;`.

## Les fichiers DTD

Les entités sont déclarées dans des fichiers DTD (Document Type Declaration). Ces types de fichiers sont en général utilisés pour déclarer la syntaxe et la sémantique d'un fichier XML particulier, mais ils autorisent aussi la déclaration d'entités. Dans le système chrome de Mozilla, vous trouverez les fichiers DTD dans le sous-répertoire locales. Vous devriez normalement avoir un fichier DTD (avec une extension *dtd*) par fichier XUL.

Si vous regardez dans le répertoire chrome, vous devriez voir une archive pour votre langue (fr-FR.jar par défaut pour le français). Vous pouvez avoir les fichiers de locales dans des langues multiples, par exemple, Anglais US (en-US) et Danois (dk). Dans ces archives vous trouverez les fichiers qui contiennent les traductions pour chaque fenêtre. La structure de l'archive est très similaire à la structure des répertoires utilisées pour les thèmes.

Dans les archives, vous placerez les fichiers DTD, dans lesquels vous déclarez les entités. Normalement vous aurez un fichier DTD par fichier XUL, en général avec le même nom de fichier excepté qu'il aura une extension *.dtd*. Donc pour la fenêtre de dialogue de recherche de fichiers, vous aurez besoin d'un fichier nommé *findfile.dtd*.

Pour les fichiers chromes non installés, vous pouvez juste mettre le fichier DTD dans le même répertoire que le fichier XUL.

Une fois que vous avez créé le fichier DTD pour votre fichier XUL, vous aurez besoin d'ajouter une ligne dans le fichier XUL qui indiquera que vous voulez utiliser le fichier DTD. Sinon, des erreurs seront générées car il ne sera pas capable de trouver les entités. Il vous suffit d'ajouter une ligne de la forme suivante vers le début du fichier XUL :

```
<!DOCTYPE window SYSTEM "chrome://findfile/locale/findfile.dtd">
```

Cette ligne spécifie que l'URL indiquée est à utiliser en tant que fichier DTD. Dans ce cas, nous avons déclaré que nous voulons utiliser le fichier DTD *findfile.dtd*. Cette ligne est en général placée juste avant l'élément window.

## Déclarer les entités

Les entités sont déclarées en utilisant une syntaxe simple vue ci-dessous :

```
<!ENTITY findLabel "Rechercher">
```

Cet exemple crée une entité avec le nom *findLabel* et la valeur *Rechercher*. Elle signifie que quelque soit l'endroit où le texte *&findLabel;* apparaîtra dans le fichier XUL, il sera remplacé par le texte *Rechercher*. Dans le fichier DTD d'une autre langue, le texte pour cette langue sera utilisé à la place.

Notez que les déclarations d'entités n'ont pas de slash terminal.

Par exemple, le texte suivant :

```
<description value="&findLabel;" />
```

est converti en :

```
<description value="Rechercher" />
```

Vous devrez déclarer une entité pour chaque libellé ou chaîne de caractères que vous utiliserez dans votre interface. Vous ne devriez plus avoir de texte affiché directement dans le fichier XUL.

En plus d'utiliser les entités pour les libellés, vous devriez les utiliser pour chaque valeur qui pourrait être différente selon la langue. Les touches d'accès et les raccourcis claviers par exemple.

```
<menuitem label="&undo.label;" accesskey="&undo.key;" />
<!ENTITY undo.label "Annuler">
<!ENTITY undo.key "u">
```

L'exemple ci-dessus utilise deux entités, une pour le libellé de l'élément de menu Annuler et une seconde pour la touche d'accès.

## Modification de la boîte de dialogue de recherche de fichiers

Jetons un oeil sur la manière dont nous pourrions utiliser tout ce que nous avons appris en modifiant la boîte de dialogue de recherche de fichiers de manière à ce qu'elle utilise un fichier DTD pour toutes ses chaînes de caractères. La totalité du fichier XUL est décrite ci-dessous avec les changements décrits en rouge.

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="findfile.css" type="text/css"?>

<!DOCTYPE window SYSTEM "chrome://findfile/locale/findfile.dtd">

<window
  id="findfile-window"
  title="&findWindow.title;"
  persist="screenX screenY width height"
  orient="horizontal"
  onload="initSearchList()"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script src="findfile.js"/>

<popupset>
  <popup id="editpopup">
    <menuitem label="Cut" accesskey="&cutCmd.accesskey;" />
```

```

    <menuitem label="Copy" accesskey="&copyCmd.accesskey;" />
    <menuitem label="Paste" accesskey="&pasteCmd.accesskey;" disabled="true"/>
</popup>
</popupset>

<keyset>
    <key id="cut_cmd" modifiers="accel" key="&cutCmd.commandkey;" />
    <key id="copy_cmd" modifiers="accel" key="&copyCmd.commandkey;" />
    <key id="paste_cmd" modifiers="accel" key="&pasteCmd.commandkey;" />
    <key id="close_cmd" keycode="VK_ESCAPE" oncommand="window.close();" />
</keyset>

<vbox flex="1">

    <toolbox>

        <menubar id="findfiles-menubar">
            <menu id="file-menu" label="&fileMenu.label;"
                accesskey="&fileMenu.accesskey;">
                <menupopup id="file-popup">
                    <menuitem label="&openCmd.label;"
                        accesskey="&openCmd.accesskey;" />
                    <menuitem label="&saveCmd.label;"
                        accesskey="&saveCmd.accesskey;" />
                    <menuseparator/>
                    <menuitem label="&closeCmd.label;"
                        accesskey="&closeCmd.accesskey;" key="close_cmd" oncommand="window.close();" />
                </menupopup>
            </menu>
            <menu id="edit-menu" label="&editMenu.label;"
                accesskey="&editMenu.accesskey;">
                <menupopup id="edit-popup">
                    <menuitem label="&cutCmd.label;"
                        accesskey="&cutCmd.accesskey;" key="cut_cmd" />
                    <menuitem label="&copyCmd.label;"
                        accesskey="&copyCmd.accesskey;" key="copy_cmd" />
                    <menuitem label="&pasteCmd.label;"
                        accesskey="&pasteCmd.accesskey;" key="paste_cmd" disabled="true" />
                </menupopup>
            </menu>
        </menubar>

        <toolbar id="findfiles-toolbar">
            <toolbarbutton id="opensearch" label="&openCmdToolbar.label;" />
            <toolbarbutton id="savesearch" label="&saveCmdToolbar.label;" />
        </toolbar>
    </toolbox>

    <tabbox>
        <tabs>
            <tab label="&searchTab;" selected="true" />
            <tab label="&optionsTab;" />
        </tabs>

        <tabpanel>
            <description>
                &findDescription;
            </description>

            <spacer class="titlespace" />

            <groupbox orient="horizontal">
                <caption label="&findCriteria;" />

```

```

<menulist id="searchtype">
  <menupopup>
    <menuitem label="&type.name;" />
    <menuitem label="&type.size;" />
    <menuitem label="&type.date;" />
  </menupopup>
</menulist>
<spacer class="springspace" />
<menulist id="searchmode">
  <menupopup>
    <menuitem label="&mode.is;" />
    <menuitem label="&mode.isnot;" />
  </menupopup>
</menulist>
<spacer class="springspace" />

<menulist id="find-text" flex="1"
  editable="true"
  datasources="file:///mozilla/recents.rdf"
  ref="http://www.xulplanet.com/rdf/recent/all">
  <template>
    <menupopup>
      <menuitem label="rdf:http://www.xulplanet.com/rdf/recent#Label" uri="rdf:*" />
    </menupopup>
  </template>
</menulist>

</groupbox>

</tabpanel>

<tabpanel id="optionspanel" orient="vertical">
  <checkbox id="casecheck" label="&casesensitive;" />
  <checkbox id="wordcheck" label="&matchfilename;" />
</tabpanel>

</tabpanel>
</tabbox>

<tree id="results" style="display: none;" flex="1">
  <treecols>
    <treecol id="name" label="&results.filename;" flex="1" />
    <treecol id="location" label="&results.location;" flex="2" />
    <treecol id="size" label="&results.size;" flex="1" />
  </treecols>

  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="mozilla" />
        <treecell label="/usr/local" />
        <treecell label="&bytes.before;2520&bytes.after;" />
      </treerow>
    </treeitem>
  </treechildren>
</tree>

<splitter id="splitbar" resizeafter="grow" style="display: none;" />

<spacer class="titlespace" />

<hbox>
  <progressmeter id="progmeter" value="50%" style="display: none;" />
  <spacer flex="1" />
  <button id="find-button" label="&button.find;"

```

```

        oncommand="doFind()" />
<button id="cancel-button" label=" &button.cancel;"
        oncommand="window.close();" />
</hbox>
</vbox>

</window>

```

Chaque chaîne de caractères a été remplacée par une référence à une entité. Un fichier DTD a été inclu au début du fichier XUL. Chaque entité qui a été ajoutée doit être déclarée dans le fichier DTD. La fenêtre ne sera pas affichée si une entité non déclarée est trouvée dans le fichier XUL.

Notez que le nom de l'entité n'est pas important. Dans l'exemple ci-dessus, les mots dans les entités ont été séparés par des points. Vous n'avez pas à faire ça. Les noms des entités ici suivent des conventions similaires au reste du code de Mozilla.

Vous pourriez noter que le texte *2520 octets* a été remplacé par deux entités. En fait, la structure de la phrase pourrait être différente dans une autre langue. Par exemple, le nombre pourrait apparaître après l'équivalent du mot 'octets' au lieu d'avant. Bien sûr, il est beaucoup plus compliqué de vouloir l'affichage des Ko ou des Mo selon les besoins.

Les touches d'accès et les raccourcis claviers ont aussi été traduits dans les entités car ils seront peut être différents dans une autre langue.

Voici le fichier DTD (findfile.dtd) :

```

<!ENTITY findWindow.title "Recherche de fichiers">
<!ENTITY fileMenu.label "Fichier">
<!ENTITY editMenu.label "Edition">
<!ENTITY fileMenu.accesskey "f">
<!ENTITY editMenu.accesskey "e">
<!ENTITY openCmd.label "Ouvrir une recherche...">
<!ENTITY saveCmd.label "Sauvegarder une recherche...">
<!ENTITY closeCmd.label "Fermer">
<!ENTITY openCmd.accesskey "o">
<!ENTITY saveCmd.accesskey "s">
<!ENTITY closeCmd.accesskey "f">
<!ENTITY cutCmd.label "Couper">
<!ENTITY copyCmd.label "Copier">
<!ENTITY pasteCmd.label "Coller">
<!ENTITY cutCmd.accesskey "p">
<!ENTITY copyCmd.accesskey "c">
<!ENTITY pasteCmd.accesskey "l">
<!ENTITY cutCmd.commandkey "X">
<!ENTITY copyCmd.commandkey "C">
<!ENTITY pasteCmd.commandkey "V">
<!ENTITY openCmdToolbar.label "Ouvrir">
<!ENTITY saveCmdToolbar.label "Sauvegarder">
<!ENTITY searchTab "Rechercher">
<!ENTITY optionsTab "Options">
<!ENTITY findDescription "Entrez votre critère de recherche ci-dessous et appuyer sur le bouton Rechercher">
<!ENTITY findCriteria "Critère de recherche">
<!ENTITY type.name "Nom">
<!ENTITY type.size "Taille">
<!ENTITY type.date "Date de modification">
<!ENTITY mode.is "Est">
<!ENTITY mode.isnot "N'est pas">
<!ENTITY casesensitive "Recherche sensible à la casse">
<!ENTITY matchfilename "Rechercher un nom entier">
<!ENTITY results.filename "Nom de fichier">
<!ENTITY results.location "Emplacement">
<!ENTITY results.size "Taille">

```



```
<!ENTITY bytes.before ">
<!ENTITY bytes.after "octets">
<!ENTITY button.find "Rechercher">
<!ENTITY button.cancel "Annuler">
```

Maintenant pour changer de langue, tout ce que vous avez à faire est de créer un nouveau fichier DTD. En utilisant le système chrome pour ajouter le fichier DTD dans une langue différente, le même fichier XUL peut être utilisé pour toutes les langues.

---

Dans la prochaine section, nous regarderons les fichiers de propriétés.

## 10.6 Les fichiers de propriétés

Écrit par Neil Deakin. Traduit par *Sylvain Costard* (21/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/locprops.html>

Dans un script, les entités ne peuvent être utilisées. On utilise alors plutôt les fichiers de propriétés.

### Propriétés

Les fichiers DTD peuvent convenir lorsque vous avez du texte dans un fichier XUL. Néanmoins, dans un script, les entités ne sont pas analysées pour être remplacées. De plus, vous pourriez souhaiter afficher un message généré par un script, sans par exemple connaître à l'avance son contenu. Les fichiers de propriétés peuvent être utilisés dans ce but.

Un fichier de propriétés contient une suite de chaînes de caractères. Vous trouverez les fichiers de propriétés aux côtés des fichiers DTD avec l'extension `.properties`. Les propriétés dans un de ces fichiers sont déclarées selon une structure `nom=valeur`. Un exemple est décrit ci-dessous :

```
notFoundAlert=Aucun fichier trouvé correspondant aux critères.
deleteAlert=Cliquez sur OK pour effacer tous vos fichiers.
```

Ici, le fichier de propriétés contient deux propriétés. Elles pourront être lues par un script et affichées. Vous pouvez écrire le code de lecture des propriétés vous-même, néanmoins XUL fournit l'élément `stringbundle` qui le fait pour vous. Cet élément dispose de plusieurs fonctions pouvant être utilisées pour récupérer les chaînes de caractères des fichiers de propriétés et d'autres informations de localisation. Cet élément lit le contenu des fichiers de propriétés et construit une liste de ces propriétés pour vous. Vous pouvez donc ensuite y accéder par leur nom.

```
<stringbundle id="chaines" src="chaines.properties"/>
```

L'inclusion de cet élément permettra de lire les propriétés via le fichier `chaines.properties` dans le même répertoire que le fichier XUL. Utilisez une URL chrome pour lire un fichier de localisation.

L'élément `stringbundle` a plusieurs propriétés. La première est `getString` pouvant être utilisée dans un script pour lire une chaîne de caractères de la collection.

```
var strbundle=document.getElementById("chaines");
var nofilesfound=strbundle.getString("notFoundAlert");

alert(nofilesfound);
```

Cet exemple récupère tout d'abord une référence sur la collection en utilisant son id. Puis il recherche la chaîne de caractères `notFoundAlert` dans le fichier de propriétés. La fonction `getString` retourne la valeur de la chaîne ou `null` si la chaîne n'existe pas. Enfin, la chaîne de caractères est affichée dans une boîte d'alerte.

Dans la prochaine section, nous découvrirons XBL qui peut être utilisé pour définir le comportement d'un élément.

# 11. Bindings

## 11.1 Introduction à XBL

Écrit par Neil Deakin. Traduit par *Nadine Henry* (24/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/introxbl.html>

XUL a un langage qui lui est parent, XBL (eXtensible Bindings Language). Ce langage est utilisé pour déclarer le comportement des éléments graphiques de XUL.

### Liaisons

Vous pouvez utiliser XUL pour définir la mise en page de l'interface utilisateur d'une application. Vous pouvez adapter l'apparence des éléments en leur appliquant des styles. Vous pouvez aussi créer de nouveaux thèmes en modifiant les styles. L'apparence basique de tous les éléments, comme les barres de défilement et les cases à cocher pourrait être modifiée en ajustant le style ou en déclarant des attributs à l'élément. Cependant, XUL ne fournit aucun moyen de vous permettre de changer le fonctionnement d'un élément. Par exemple, vous pourriez vouloir changer le fonctionnement des composants d'une barre de défilement. Pour cela, vous avez besoin de XBL.

Un fichier XBL contient un ensemble de liaisons. Chaque liaison décrit le comportement d'un élément graphique de XUL. Par exemple, une liaison pourrait être attachée à une barre de défilement. Le comportement décrit les propriétés et méthodes de la barre de défilement en plus de décrire les éléments de XUL qui la composent.

Comme XUL, XBL est un langage XML, ainsi il a des règles syntaxiques similaires. L'exemple suivant montre le squelette basique d'un fichier XBL :

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl">
  <binding id="liaison1">
    <!-- le contenu, la propriété, la méthode et les descriptions d'évènements viennent ici -->
  </binding>
  <binding id="liaison2">
    <!-- le contenu, la propriété, la méthode et les descriptions d'évènements viennent ici -->
  </binding>
</bindings>
```

L'élément **bindings** est l'élément racine d'un fichier XBL et contient un ou plusieurs élément **binding**. Chaque élément **binding** déclare une liaison simple. L'attribut **id** peut être utilisé pour identifier la liaison, comme dans l'exemple ci-dessus. Le modèle a deux liaisons, l'une appelée *liaison1* et l'autre appelée *liaison2*. L'une pourrait être attachée à une barre de défilement et l'autre à un menu. Une liaison peut être attachée à n'importe quel élément de XUL. Si vous utilisez des classes CSS, vous pouvez utiliser autant de liaisons que vous avez besoin. Notez l'espace de nommage de l'élément **bindings** dans le modèle ci-dessus. Il déclare que nous sommes en train d'utiliser une syntaxe XBL.

Vous assignez une liaison à un élément en déclarant la propriété CSS `-moz-binding` avec l'URL des fichiers de liaisons. Par exemple :

```
scrollbar {
  -moz-binding: url('chrome://findfile/content/findfile.xml#binding1');
}
```

L'URL pointe vers la liaison avec l'id *binding1* dans le fichier 'chrome://findfile/content/findfile.xml'. La syntaxe '#binding1' est utilisée pour pointer vers une liaison spécifique, de la même façon que si vous

pointiez vers une ancre dans un fichier HTML. Vous mettrez habituellement toutes vos liaisons dans un seul fichier. Le résultat dans cet exemple, est que toutes les barres de défilement auront leur comportement décrit par la liaison "binding1". Si vous n'utilisez pas d'ancres dans l'URL `-moz-binding`, la première liaison dans le fichier XBL sera utilisée.

Une liaison déclare cinq types de choses :

1. Le contenu : les éléments fils qui sont ajoutés à l'élément auquel la liaison est attachée.
2. Les propriétés : les propriétés ajoutées à l'élément. On peut y accéder par un script.
3. Les méthodes : les méthodes ajoutées à l'élément. Elles peuvent être appelées à partir d'un script.
4. Les événements : les événements, comme les clics de souris et les appuis sur les touches auxquels l'élément répondra. La liaison peut ajouter des scripts pour fournir la manipulation par défaut. En plus de cela, de nouveaux événements peuvent être définis.
5. Le style : adapte les propriétés de style que l'élément XBL possède.

## Exemple de liaison

L'élément `box` est suffisamment générique pour que vous l'utilisiez afin de créer des éléments graphiques personnalisés (bien que vous puissiez utiliser d'autres éléments, même un élément composé par vous-même). En assignant un attribut `class` à une balise `box`, vous ne pouvez associer une liaison qu'aux boîtes qui appartiennent à cette classe. L'exemple suivant le démontre :

**XUL (example.xul):**

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="chrome://example/skin/example.css" type="text/css"?>

<window
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <box class="okcancelbuttons"/>
</window>
```

**CSS (example.css):**

```
box.okcancelbuttons {
  -moz-binding: url('chrome://example/skin/example.xml#okcancel');
}
```

**XBL (example.xml):**

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
  xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <binding id="okcancel">
    <content>
      <xul:button label="OK"/>
      <xul:button label="Annuler"/>
    </content>
  </binding>
</bindings>
```

Cet exemple crée une fenêtre avec une seule boîte. La boîte a été déclarée pour avoir un attribut `class` de valeur `okcancelbuttons`. La feuille de styles associée au fichier indique que les boîtes avec les classes `okcancelbuttons` ont une liaison spécialisée, définie dans le fichier XBL. Vous pouvez employer d'autres éléments derrière l'élément `box`, même pour votre propre balise adaptée.

Nous verrons plus de détails concernant la partie XBL dans la section suivante. Cependant, pour récapituler, cet exemple entraîne l'ajout automatique de deux boutons dans la boîte, un bouton *Ok* et un autre *Annuler*.

Dans la prochaine section, nous verrons comment créer un contenu avec XBL.

## 11.2 Contenu anonyme

Écrit par Neil Deakin. Traduit par *Nadine Henry* (27/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblcontent.html>

Dans cette section, nous allons voir comment créer un contenu avec XBL.

### Le contenu XBL

XBL peut être utilisé pour ajouter automatiquement un ensemble d'éléments à l'intérieur d'un autre élément. Le fichier XUL a uniquement besoin de spécifier l'élément externe tandis que l'élément interne est décrit dans un fichier XBL. C'est utile pour créer un élément graphique simple qui est construit à partir d'un ensemble d'autres éléments, mais qui peut être référencé comme un seul élément graphique. Des mécanismes sont fournis pour ajouter des attributs aux éléments internes qui étaient spécifiés dans l'élément externe.

L'exemple ci-dessous montre comment une barre de défilement pourrait être déclarée (il a été un peu simplifié par rapport à la réalité) :

```
<bindings xmlns="http://www.mozilla.org/xbl"
           xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <binding id="scrollbarBinding">
    <content>
      <xul:scrollbarbutton type="decrement"/>
      <xul:slider flex="1">
        <xul:thumb/>
      </xul:slider>
      <xul:scrollbarbutton type="increment"/>
    </content>
  </binding>
</bindings>
```

Ce fichier contient une seule liaison, déclarée avec l'élément binding. L'attribut `id` doit être déclaré avec l'identifiant de la liaison. De cette façon il y est fait référence au travers de la propriété CSS `-moz-binding`.

La balise content est utilisée pour déclarer le contenu anonyme qui sera ajouté à la barre de défilement. Tous les éléments à l'intérieur de la balise content seront ajoutés au sein de l'élément auquel la liaison est liée. Cette liaison sera vraisemblablement liée à une barre de défilement, bien qu'elle puisse ne pas l'être. Chaque élément dont la propriété CSS `-moz-binding` est déclarée avec l'URL de la liaison, va l'utiliser.

L'utilisation de la liaison ci-dessus a pour résultat que la ligne de XUL ci-dessous soit interprétée comme suit, en supposant que la barre de défilement est liée au XBL ci-dessus :

```
<scrollbar>

devient :

<scrollbar>
  <xul:scrollbarbutton type="decrement"/>
  <xul:slider flex="1">
    <xul:thumb/>
  </xul:slider>
  <xul:scrollbarbutton type="increment"/>
</scrollbar>
```

Les éléments au sein de la balise content sont ajoutés anonymement à la barre de défilement. Bien que le contenu anonyme soit affiché à l'écran, vous ne pouvez pas y accéder à l'aide d'un script par la voie normale. Dans XUL, c'est comme s'il n'y avait qu'un seul élément, bien qu'il se compose en réalité de plusieurs éléments.

Si vous observez une barre de défilement dans une fenêtre Mozilla, vous verrez qu'elle est composée d'un bouton en forme de flèche, d'une zone de coulissement, d'une poignée à l'intérieur et d'un second bouton en forme de flèche à la fin, ce sont les éléments qui apparaissent dans le contenu XBL ci-dessus. Ces éléments pourraient à leur tour être liés à d'autres liaisons qui utilisent les éléments XUL de base. Notez que les éléments de contenu ont besoin de l'espace de nommage de XUL (ils apparaissent précédés de *xul:*), parce que ce sont des éléments de XUL et qu'ils ne sont pas valides dans XBL. Cet espace de nommage a été déclaré dans la balise bindings. Si vous n'utilisez pas l'espace de nommage sur les éléments de XUL, Mozilla supposera que ce sont des éléments XBL et il ne les comprendra pas, et vos éléments ne fonctionneront pas correctement.

Un autre exemple, cette fois pour un champ dans lequel vous pourriez entrer un nom de fichier :

```
<binding id="fileentry">
  <content>
    <textbox/>
    <button label="Parcourir..." />
  </content>
</binding>
```

L'attachement de cette liaison sur un élément lui fera contenir un champ de saisie de texte, suivi d'un bouton *Parcourir....* Ce contenu interne est créé anonymement et ne peut être vu en utilisant le DOM.

Le contenu anonyme est créé automatiquement chaque fois qu'une liaison est attachée à un élément. Si vous placez des éléments fils à l'intérieur du contenu XUL, ils vont écraser les éléments fournis par la liaison. Par exemple, prenez cet extrait XUL, en supposant qu'il soit lié à la barre de défilement XBL de tout à l'heure :

```
<scrollbar/>

<scrollbar>
  <button label="Écraser" />
</scrollbar>
```

Puisque la première barre de défilement n'a pas de contenu qui lui est propre, celui-ci sera généré à partir de la définition de la liaison du fichier XBL. La seconde barre de défilement a son propre contenu donc elle l'utilisera à la place du contenu XBL, ce qui a pour résultat quelque chose qui ne ressemble plus à une barre de défilement. Notez que beaucoup d'éléments natifs de construction, comme les barres de défilement, ont leur définition XBL stockée dans des fichiers situés dans le répertoire "bindings" du paquetage toolkit.jar.

Ce mécanisme s'applique seulement aux éléments définis à l'intérieur de la balise content. Les propriétés, les méthodes et d'autres aspects d'XBL restent disponibles, que le contenu provienne d'XBL ou que XUL fournisse son propre contenu.

Il peut y avoir des fois où vous souhaitez que soient montrés à la fois le contenu XBL et celui fournit par le fichier XUL. Il vous suffit d'utiliser l'élément children. Les éléments fils dans XUL sont ajoutés en lieu et place de l'élément children. C'est pratique pour créer des éléments graphiques de menu personnalisés. Par exemple, une version simplifiée d'un élément menulist éditable, pourrait être créée comme ceci :

**XUL :**

```
<menu class="dropbox">
  <menupopup>
    <menuitem label="1000" />
```

```

    <menuitem label="2000"/>
  </menupopup>
</menu>

```

**CSS :**

```

menu.dropbox {
  -moz-binding: url('chrome://example/skin/example.xml#dropbox');
}

```

**XBL:**

```

<binding id="dropbox">
  <content>
    <children/>
    <xul:textbox flex="1"/>
    <xul:button src="chrome://global/skin/images/dropbox.jpg"/>
  </content>
</binding>

```

Cet exemple crée un champ de saisie suivi d'un bouton. Le menupopup sera ajouté au contenu à l'emplacement spécifié par l'élément children. Notez que pour les fonctions du DOM, le contenu apparaîtra comme s'il était dans le fichier XUL, ainsi le menupopup sera un fils du menu. Le contenu XBL est caché ainsi le développeur d'une application sous XUL n'a même pas besoin de savoir qu'il est là.

Le contenu résultant serait :

```

<menu class="dropbox">
  <menupopup>
    <menuitem label="1000"/>
    <menuitem label="2000"/>
  </menupopup>
  <textbox flex="1"/>
  <button src="chrome://global/skin/images/dropbox.jpg"/>
</menu>

```

Dans certains cas, vous souhaitez n'inclure que des types de contenus spécifiques et non d'autres. Ou bien, vous souhaitez placer différents types de contenus à différents endroits. L'attribut `includes` peut être utilisé pour n'autoriser que certains éléments à apparaître dans le contenu. Sa valeur doit être déclarée pour un simple nom de balise, ou pour une liste de balises séparées par des barres verticales (Le symbole `|`).

```

<children includes="button">

```

Cette ligne va ajouter tous les boutons qui sont fils de l'élément lié en lieu et place de la balise children. Les autres éléments ne correspondront pas avec cette balise. Vous pouvez placer plusieurs éléments children dans une liaison pour placer différents types de contenus à différents endroits. Si un élément dans XUL ne correspond pas aux éléments children, cet élément (et les autres n'y correspondant pas) sera utilisé à la place du contenu lié.

Voici un autre exemple. Disons que nous voulions créer un élément graphique qui affiche une image avec un bouton de zoom "agrandir" et un bouton de zoom "diminuer" de part et d'autre. Il pourrait être créé avec une boîte qui contiendrait l'image et deux boutons. L'élément image doit être placé à l'extérieur du fichier XBL car il sera différent selon l'usage.

**XUL :**

```

<box class="zoombox">
  <image src="images/happy.jpg"/>
  <image src="images/angry.jpg"/>
</box>

```

**XBL :**

```
<binding id="zoombox">
  <content>
    <xul:box flex="1">
      <xul:button label="Zoom In"/>
      <xul:box flex="1" style="border: 1px solid black">
        <children includes="image"/>
      </xul:box>
      <xul:button label="Zoom Out"/>
    </xul:box>
  </content>
</binding>
```

Les fils explicites dans le fichier XUL seront placés à l'endroit où se trouve la balise children. Il y a deux images, ainsi toutes les deux seront ajoutées à côté de l'une et l'autre. Cela a pour conséquence un affichage équivalent au code suivant :

```
<binding id="zoombox">
  <content>
    <xul:box flex="1">
      <xul:button label="Zoom in"/>
      <xul:box flex="1" style="border: 1px solid black">
        <image src="images/happy.jpg"/>
        <image src="images/angry.jpg"/>
      </xul:box>
      <xul:button label="Zoom Out"/>
    </xul:box>
  </content>
</binding>
```

D'un point de vue du DOM, les éléments fils sont toujours à leur emplacement original. En effet, la boîte XUL externe a deux fils qui sont les deux images. La boîte interne avec la bordure a un fils, la balise children. C'est une distinction importante lorsqu'on utilise le DOM avec XBL. Elle s'applique également aux règles du sélecteur CSS.

## Les éléments fils multiples

Vous pouvez également utiliser plusieurs éléments children et avoir certains éléments placés à un endroit et d'autres éléments placés à un autre. En ajoutant l'attribut `includes` contenant une liste de nom de balises séparés par des barres verticales, vous pouvez placer uniquement les éléments correspondants à cet endroit. Par exemple, le fichier XBL suivant va faire apparaître des libellés et des boutons à un endroit différent des autres éléments :

### Exemple 11.2.1 :

```
<binding id="navbox">
  <content>
    <xul:vbox>
      <xul:label value="Libellés et boutons"/>
      <children includes="label|button"/>
    </xul:vbox>
    <xul:vbox>
      <xul:label value="Autres éléments"/>
      <children/>
    </xul:vbox>
  </content>
</binding>
```



Le premier élément `children` n'inclut que les éléments `label` et `button`, comme indiqué dans son attribut `includes`. Le second élément `children`, parce qu'il n'a pas d'attribut `includes`, ajoute tous les éléments restants.

Dans la prochaine section, nous verrons comment des attributs peuvent être hérités dans le contenu anonyme.

## 11.3 Héritage d'attributs XBL

Écrit par Neil Deakin. Traduit par *Cyril Cheneson* (15/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblatin.html>

Dans cette section, nous verrons comment les attributs peuvent être hérités.

### L'héritage d'attributs

XBL nous permet de construire des éléments graphiques composites tout en cachant leur implémentation réelle. Cependant, avec les fonctionnalités mentionnées jusque là, le contenu anonyme est toujours créé de la même façon. Il serait utile de pouvoir ajouter des attributs aux éléments extérieurs pour modifier les éléments intérieurs. Par exemple :

**XUL :**

```
<searchbox/>
```

**XBL :**

```
<binding id="searchBinding">
  <content>
    <xul:textbox/>
    <xul:button label="Rechercher"/>
  </content>
</binding>
```

Dans cet exemple, l'attribut `label` est placé directement sur l'élément `button`. Le problème avec cette technique est que le libellé sera le même à chaque fois que la liaison sera utilisée. Dans ce cas, il serait préférable que l'attribut soit plutôt défini sur la balise `searchbox`. XBL fournit un attribut `inherits` pouvant être utilisé pour hériter des attributs de l'élément extérieur. Il devra être placé sur l'élément qui héritera de ces attributs, dans notre cas sur le bouton. Sa valeur devra être initialisée par une liste des noms des attributs à hériter, séparés par des virgules.

```
<xul:textbox xbl:inherits="flex"/>
<xul:button xbl:inherits="label"/>
```

Lorsque le contenu est généré, `textbox` obtient l'attribut `flex` à partir de `searchbox` et `button` obtient l'attribut `label` à partir de `searchbox`. Ils permettent à la flexibilité du champ de saisie et au libellé du bouton d'être différents à chaque utilisation de la liaison. En plus, changer la valeur des attributs de la balise `searchbox` avec un script mettra aussi à jour la balise `textbox` et la balise `button`. Vous pouvez ajouter un attribut `inherits` sur autant d'éléments que vous le souhaitez, pour hériter de n'importe quel nombre d'attributs.

Remarquez comment l'attribut `inherits` a été placé dans l'espace de nommage XBL, en utilisant le préfixe `xbl:`. L'espace de nommage devrait être déclaré quelque part avant, généralement dans l'élément `bindings`, comme dans l'exemple suivant :

```
<bindings xmlns:xbl="http://www.mozilla.org/xbl"
          xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

```
<xbl:binding id="buttonBinding">
  <xbl:content>
    <xul:button label="OK" xbl:inherits="label"/>
  </xbl:content>
</xbl:binding>
```

Dans cet exemple, le bouton hérite de l'attribut `label`, mais cet attribut a aussi une valeur assignée directement dans le XBL. Cette technique est utilisée pour définir une valeur par défaut si l'attribut n'est pas présent. Ce bouton héritera son attribut `label` de l'élément extérieur. Cependant, si aucun `label` n'est présent, la valeur *OK* par défaut lui sera donnée.

Il peut arriver que deux éléments générés aient besoin d'hériter d'un attribut qui a le même nom. Par exemple, pour créer un champ de saisie muni d'un libellé, contenant donc un élément label et un élément textbox, le libellé (`label`) aura besoin d'hériter son texte à partir de l'attribut `value` et le champ de saisie (`textbox`) aura aussi besoin d'hériter sa valeur par défaut également à partir de l'attribut `value`. Pour résoudre cela, nous aurons besoin d'utiliser un attribut différent et le faire pointer sur le bon. L'exemple suivant montre comment procéder :

**XUL :**

```
<box class="textboxlibelle" title="Entrer du texte:" value="OK"/>
```

**CSS :**

```
box.textboxlibelle {
  -moz-binding: url('chrome://example/skin/example.xml#labeledtextbox');
}
```

**XBL :**

```
<binding id="textboxlibelle ">
  <content>
    <xul:label xbl:inherits="value=title"/>
    <xul:textbox xbl:inherits="value"/>
  </content>
</binding>
```

L'élément textbox hérite directement de l'attribut `value`. Pour initialiser l'attribut `value` du libellé, nous aurons besoin d'utiliser un nom différent d'attribut et le faire pointer vers le vrai attribut. L'attribut `inherits` sur la balise label obtient son attribut `title` à partir de l'élément extérieur et le fait pointer vers l'attribut `value` de l'élément label. La syntaxe `<attribut intérieur>=<attribut extérieur>` est utilisée ici pour faire pointer un attribut vers un autre. Voici un autre exemple :

**XUL :**

```
<box class="okannuler" oktitle="OK" canceltitle="Annuler" image="happy.png"/>
```

**CSS :**

```
box.okannuler {
  -moz-binding: url('chrome://example/skin/example.xml#okcancel');
}
```

**XBL :**

```
<binding id="okannuler">
  <content>
    <xul:button xbl:inherits="label=oktitle,image"/>
    <xul:button xbl:inherits="label=canceltitle"/>
  </content>
</binding>
```

La valeur de l'attribut `oktitle` est projetée vers l'attribut `label` du premier bouton. L'attribut `canceltitle` est projeté vers l'attribut `label` du second bouton. Le premier bouton hérite aussi de l'attribut `image`. Le résultat est le suivant :

```
<box class="okannuler" oktitle="OK" canceltitle="Annuler" image="happy.png">
  <button label="OK" image="happy.png" />
  <button label="Annuler" />
</box>
```

Remarquez que les attributs sont dupliqués dans le contenu intérieur (anonyme). La modification des attributs de la boîte avec la classe *okannuler* affectera automatiquement les valeurs des boutons. Vous avez probablement aussi remarqué que nous avons créé nos propres noms d'attribut. Ceci est valide en XUL.

---

Dans la section suivante, nous regarderons l'ajout de propriétés, méthodes et événements à une liaison.

## 11.4 Ajout de propriétés

Écrit par Neil Deakin. Traduit par *Nadine Henry* (13/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblprops.html>

Nous allons voir comment ajouter des propriétés personnalisées aux éléments XBL.

### L'interface XBL

Javascript et le DOM fournissent un moyen pour obtenir et définir les propriétés des éléments. Avec XBL, vous pouvez définir vos propres propriétés pour les éléments que vous créez. Vous pouvez aussi ajouter des méthodes qui peuvent être appelées. De cette façon, tout ce dont vous avez besoin est d'obtenir une référence à l'élément (en utilisant `GetElementById` ou une fonction similaire) et ainsi obtenir ou modifier ses propriétés additionnelles et appeler ses méthodes.

Il y a trois types d'items que vous pouvez ajouter. Les *champs* sont utilisés pour contenir une valeur simple. Les *propriétés* peuvent aussi être utilisées pour contenir une valeur mais elles peuvent aussi contenir du code pouvant être exécuté lorsqu'une tentative est faite pour récupérer ou modifier la valeur. Les *méthodes* sont des fonctions qui peuvent être exécutées.

Chacun des trois est défini dans un élément implementation, qui doit être un fils de l'élément de liaison binding. À l'intérieur de la balise implementation, vous définirez pour chacun d'eux un élément field, un élément property et un élément method selon ce que vous voulez. La syntaxe générale est celle-ci :

```
<binding id="element-name">
  <content>
    -- le contenu vient ici --
  </content>
  <implementation>
    <field name="field-name-1"/>
    <field name="field-name-2"/>
    <field name="field-name-3"/>

    <property name="property-name-1"/>
    <property name="property-name-2"/>
    <property name="property-name-3"/>
    .
    .
    .
    <method name="method-name-1"/>
    -- le contenu vient ici --
  </method>
```

```

        .
        .
        .
    </implementation>
</binding>

```

## Les champs

Chaque champ est défini en utilisant l'élément `field`. Souvent, les champs correspondent à un attribut placé sur l'élément comme `label` ou `disabled`, mais ils ne le devraient pas.

L'attribut `name` de `field` est utilisé pour indiquer le nom du champ. Vous pouvez utiliser le nom dans un script pour obtenir et déterminer une valeur. L'exemple ci-dessous crée un bouton qui génère et stocke un nombre aléatoire. Vous pouvez rechercher ce même nombre plusieurs fois en obtenant la propriété `number` du bouton. Le plus gros du travail ici est fait par les gestionnaires `oncommand`. Plus tard, nous verrons comment transformer cela en XBL.

**XUL :**

```

<box id="random-box" class="randomizer"/>

<button label="Générer"
        oncommand="document.getElementById('random-box').number=Math.random();" />
<button label="Voir"
        oncommand="alert(document.getElementById('random-box').number)"/>

```

**XBL :**

```

<binding id="randomizer">
    <implementation>
        <field name="number"/>
    </implementation>
</binding>

```

Un champ `number` stockant le nombre aléatoire a été défini dans la liaison. Les deux boutons spéciaux définissent et obtiennent la valeur de ce champ. La syntaxe est très similaire pour obtenir et définir les propriétés des éléments HTML. Dans cet exemple, aucun contenu n'a été placé à l'intérieur que ce soit la boîte XUL ou sa définition dans XBL, ce qui est parfaitement valide.

Cet exemple n'est pas tout à fait correct car il n'y a pas de valeur par défaut assignée dans le champ. Pour en mettre une, ajoutez la valeur par défaut dans le contenu de la balise `field`. Par exemple :

```

<field name="number">
    25
</field>

```

Ici, la valeur **25** sera affectée comme valeur par défaut du champ "number". En fait, vous pourriez aussi insérer un script au sein de la balise `field` pour calculer la valeur par défaut. Cela pourrait être nécessaire si la valeur a besoin d'être calculée. Par exemple, le champ suivant donne une valeur par défaut égale à l'heure courante :

```

<field name="currentTime">
    new Date().getTime();
</field>

```

## Les propriétés

Parfois vous voulez valider la donnée qui est assignée à une propriété. Ou bien, pour souhaitez que la valeur soit calculée dynamiquement lorsqu'on le lui demande. Par exemple, si vous souhaitez une propriété qui

prenne en compte l'heure courante, vous voudrez que sa valeur soit générée au besoin. Dans ces cas là, vous avez besoin d'utiliser une balise property à la place de la balise field. Sa syntaxe est similaire mais comporte des particularités supplémentaires.

Vous pouvez utiliser les attributs `onget` et `onset` pour que le code soit exécuté lorsque la propriété est récupérée ou modifiée. Ajoutez les à l'élément property et définissez leur valeur dans un script qui, au choix, obtient ou déclare la valeur de la propriété.

Par exemple, vous pouvez assigner un script à la valeur de `onget` pour calculer le temps courant. Chaque fois qu'un script tente d'accéder à la valeur de la propriété, le script `onget` sera appelé pour fournir la valeur. Le script devra retourner la valeur qui devrait être traitée comme étant la valeur de la propriété.

Le gestionnaire `onset` est similaire mais est appelé chaque fois qu'un script tente d'assigner une nouvelle valeur à la propriété. Ce script devrait stocker la valeur quelque part, ou la valider. Par exemple, certaines propriétés pourraient juste être capables de stocker des nombres. Tenter d'assigner des caractères alphabétiques à ce genre de propriétés ne devrait pas fonctionner.

```
<property name="size"
  onget="return 77;"
  onset="alert('Modifié en :'+val); return val;"/>
```

Cette propriété retournera toujours `77` lorsqu'elle sera récupérée. Lorsqu'elle sera affectée, un message d'alerte s'affichera et indiquera la valeur à assigner à la propriété. La variable spéciale `val` contient cette valeur. Utilisez-la pour la valider ou la stocker. La propriété `onset` devrait aussi retourner la nouvelle valeur.

Ce qui suit décrit ce qui se passe dans un cas typique :

Il y a deux éléments, l'un appelé "banana" et l'autre "orange". Chacun d'eux a une propriété spécifique appelée 'size'. Lorsque la ligne de script suivante est exécutée :

```
banana.size = orange.size;
```

1. Le script `onget` est appelé pour la propriété "size" de "orange". Il calcule la valeur et la retourne.
2. Le gestionnaire `onset` de la propriété "size" de "banana" est appelé. Ce script utilise la valeur passée dans la variable `val` et l'assigne à la propriété "size" de "banana" de façon quelconque.

Notez que contrairement à un champ, une propriété ne contient pas de valeur. Tenter de définir une propriété qui n'a pas de gestionnaire `onset` provoquera une erreur. Vous utiliserez souvent un champ séparé pour mémoriser la valeur actuelle de la propriété. Il est aussi commun que les propriétés correspondent à un attribut dans l'élément défini XBL. L'exemple suivant fait correspondre une propriété à un attribut sur un élément.

```
<property name="size"
  onget="return this.getAttribute('size');"
  onset="return this.setAttribute('size',val);"
/>
```

Chaque fois qu'un script tente d'obtenir la valeur de la propriété, elle est récupérée d'un attribut de même nom de l'élément XUL. Chaque fois qu'un script tente de définir la valeur de la propriété, elle est affectée à l'attribut 'size' de l'élément. C'est pratique parce qu'ainsi vous pouvez modifier la propriété ou l'attribut et tous les deux auront la même valeur.

Vous pouvez utiliser une syntaxe alternative pour les attributs `onget` et `onset` ce qui est utile si les scripts sont plus longs. Vous pouvez remplacer l'attribut `onget` par l'élément fils nommé getter. De même, vous pouvez remplacer l'attribut `onset` par l'élément setter. L'exemple ci-dessous le montre :

```

<property name="number">
  <getter>
    return this.getAttribute('number');
  </getter>
  <setter>
    var v = parseInt(val);
    if (!isNaN(v)) return this.setAttribute('number',''+v);
    else return this.getAttribute('number');
  </setter>
</property>

```

La propriété dans cet exemple sera juste capable d'avoir des valeurs d'entiers. Si d'autres caractères sont entrés, ils sont supprimés. S'il n'y a aucun chiffre, la valeur n'est pas modifiée. Ces opérations sont effectuées dans le code au sein de l'élément setter. La valeur réelle de la propriété est stockée dans l'attribut `number`.

Vous pouvez utiliser l'une ou l'autre syntaxe pour créer des gestionnaires d'obtention et d'affectation.

Vous pouvez rendre un champ ou une propriété en lecture seule en ajoutant un attribut `readonly` à la balise field ou à la balise property, et en le déclarant à `true`. Toute tentative d'affecter une valeur à une propriété en lecture seule échouera.

La prochaine section montre comment ajouter des méthodes aux éléments définis en XBL.

## 11.5 Ajout de méthodes

Écrit par Neil Deakin. Traduit par *Nadine Henry* (18/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblmethods.html>

Nous allons voir comment ajouter des méthodes personnalisées aux éléments définis en XBL.

### Les méthodes

En plus d'ajouter des propriétés de script à l'élément défini en XBL, vous pouvez aussi y ajouter des méthodes. Ces méthodes peuvent être appelées à partir d'un script. Les méthodes sont des fonctions d'objets, comme `window.open()`. Vous pouvez définir des méthodes personnalisées pour vos éléments en utilisant l'élément method. La syntaxe générale des méthodes est définie comme suit :

```

<implementation>
  <method name="method-name">
    <parameter name="parameter-name1"/>
    <parameter name="parameter-name2"/>
    .
    .
    .
    <body>
      -- le script de la méthode vient ici --
    </body>
  </method>
</implementation>

```

Une déclaration de méthode se fait au travers de l'élément implementation, comme pour les champs et les propriétés. L'élément method contient deux types d'éléments fils, les éléments parameter qui décrivent les paramètres de la méthode et body qui contient le script de la méthode.

La valeur de l'attribut `name` devient le nom de la méthode. De même, les attributs `name` des éléments parameter deviennent les noms de chaque paramètre. Chaque élément parameter est utilisé pour

déclarer l'un des paramètres pour la méthode. Par exemple, si la méthode a trois paramètres, il y aura trois éléments parameter. Si vous n'avez pas besoin d'en avoir, la méthode n'aura pas de balise parameter.

L'élément body contient le script qui est exécuté lorsque la méthode est appelée. Les noms des paramètres sont définis comme des variables dans le script comme s'ils étaient passés en paramètre. Par exemple, la fonction JavaScript suivante serait écrite en tant que méthode XBL comme ceci :

```
function getMaximum(num1,num2)
{
  if (num1<=num2) return num2;
  else return num1;
}
```

**XBL :**

```
<method name="getMaximum">
  <parameter name="num1" />
  <parameter name="num2" />
  <body>
    if (num1<=num2) return num2;
    else return num1;
  </body>
</method>
```

Cette fonction, `getMaximum`, retourne la plus grande des valeurs passées chacune comme un paramètre dans la méthode. Notez que le symbole inférieur doit être un caractère d'échappement parce qu'autrement il ressemblerait au commencement d'une balise. Vous pouvez aussi utiliser une section CDATA pour échapper le bloc entier de code. Vous pouvez appeler la méthode en utilisant un code comme `element.getMaximum(5,10)` où `element` est une référence à un élément défini par l'élément XBL contenant la méthode `getMaximum`. (L'élément attaché.)

La balise parameter vous permet de définir des paramètres pour une méthode. Comme Mozilla utilise JavaScript pour son langage de script, et que JavaScript est un langage non typé, vous n'avez pas besoin de spécifier le type des paramètres. Cependant, dans le futur, d'autres langages devraient être utilisés avec XBL.

## Accéder au contenu anonyme

Il peut y avoir des fois où vous voulez modifier certains aspects des éléments définis dans l'élément content, que ce soit à partir d'une méthode body ou d'ailleurs. Ces éléments sont créés anonymement et ne sont pas accessibles à partir des fonctions habituelles du DOM. Elles sont cachées de telle sorte que les développeurs n'aient pas besoin de savoir comment l'élément est implémenté pour l'utiliser. Cependant, il existe un moyen spécial pour obtenir ce contenu anonyme.

Les éléments auxquels un comportement XBL est attaché ont une propriété spéciale qui contient un tableau des éléments fils anonymes. Chaque élément du tableau stocke chaque élément fils direct de l'élément XBL défini. Cette propriété spéciale ne peut pas être accessible directement. À la place, vous devez appeler la méthode `getAnonymousNodes` du document.

```
var value=document.getAnonymousNodes(element);
```

Ici, *element* devrait être une référence à l'élément dont vous voulez obtenir le contenu anonyme. La fonction retourne un tableau d'éléments qui est le contenu anonyme. Pour obtenir des éléments en-dessous de celui-ci, vous pouvez utiliser les fonctions habituelles du DOM car elles ne sont pas cachées. Notez qu'il est possible pour un élément XBL attaché d'être placé à l'intérieur d'un autre, auquel cas vous devrez utiliser une nouvelle fois la fonction `getAnonymousNodes`.

L'exemple suivant crée une rangée de boutons :

```
<binding id="buttonrow">
  <content>
    <button label="Oui" />
    <button label="Non" />
    <button label="Trier par" />
  </content>
</binding>
```

Pour vous référer à chaque bouton, vous pouvez utiliser la fonction `getAnonymousNodes`, en lui passant une référence à l'élément auquel la liaison est attachée, en tant que paramètre. Dans le tableau renvoyé, le premier bouton est stocké dans le premier item du tableau (`getAnonymousNodes(element)[0]`), le deuxième bouton est stocké dans le deuxième item du tableau et le troisième est stocké dans le troisième item du tableau. Pour coder à l'intérieur d'une méthode de liaison, vous pouvez passer `this` comme paramètre à `getAnonymousNodes`.

Le prochain exemple peut être utilisé pour créer un texte avec un libellé. La méthode `showTitle` peut être utilisée pour montrer ou cacher un libellé. Elle fonctionne en obtenant une référence à l'élément titre en utilisant le tableau anonyme et en changeant sa visibilité.

**XUL :**

```
<box id="num" class="labeledbutton" title="Plusieurs choses :" value="52"/>

<button label="Montrer" oncommand="document.getElementById('num').showTitle(true)"/>
<button label="Cacher" oncommand="document.getElementById('num').showTitle(false)"/>
```

**XBL :**

```
<binding id="labeledbutton">
  <content>
    <xul:label xbl:inherits="value=title" />
    <xul:label xbl:inherits="value" />
  </content>
  <implementation>
    <method name="showTitle">
      <parameter name="state" />
      <body>
        if (state) document.getAnonymousNodes(this)[0].
          setAttribute("style","visibility: visible");
        else document.getAnonymousNodes(this)[0].
          setAttribute("style","visibility: collapse");
      </body>
    </method>
  </implementation>
</binding>
```

Les deux boutons ajoutés dans le contenu XUL ont des gestionnaires `oncommand` qui sont utilisés pour changer la visibilité du libellé. Chacun d'eux appelle la méthode `showTitle`. Cette méthode vérifie le paramètre `state` pour voir si l'élément sera caché ou montré. Dans un cas comme dans l'autre, elle récupère le premier élément du tableau anonyme. Celui-ci se rapporte au premier fils de l'élément content qui ici est le premier élément label de l'élément graphique. La visibilité est changée en modifiant le style de l'élément.

Pour aller dans l'autre sens, et obtenir l'élément XUL liée depuis l'intérieur du contenu anonyme, utilisez la propriété `parentNode` du DOM. Elle permet d'obtenir l'élément parent d'un élément. Par exemple, nous pourrions déplacer les boutons *Montrer* et *Cacher* dans le fichier XBL et faire la chose suivante :

Exemple 11.5.1 :

```
<binding id="labeledbutton">
```



```

<content>
  <xul:label xbl:inherits="value=title"/>
  <xul:label xbl:inherits="value"/>
  <xul:button label="Montrer" oncommand="parentNode.showTitle(true);"/>
  <xul:button label="Cacher" oncommand="parentNode.showTitle(false);"/>
</content>
<implementation>
  <method name="showTitle">
    <parameter name="state"/>
    <body>
      if (state) document.getAnonymousNodes(this)[0].setAttribute("style","visibility: visible");
      else document.getAnonymousNodes(this)[0].setAttribute("style","visibility: collapse");
    </body>
  </method>
</implementation>
</binding>

```

Les gestionnaires `oncommand` obtiennent ici d'abord une référence à leur élément parent. Il n'agit pas de l'élément content mais de l'élément XUL auquel l'élément XBL est lié (Dans cet exemple, c'est la boîte avec la classe *labeledbutton*). Ainsi, la méthode `showTitle` est appelée, et fonctionne comme avant.

Les propriétés et méthodes personnalisées sont ajoutées seulement à l'élément XUL externe auquel l'élément XBL est lié. Aucun des éléments déclarés au sein de la balise content n'ont ces propriétés ou méthodes. C'est pourquoi nous devons obtenir l'élément parent d'abord.

Les fils d'un élément placés dans le fichier XUL peuvent être récupérés par la voie normale et ne bougent pas même si vous utilisez la balise children. Par exemple :

**XUL :**

```

<box id="outer" class="container">
  <button label="Un"/>
  <button label="Deux"/>
  <button label="Trois"/>
  <button label="Quatre"/>
</box>

```

**XBL :**

```

<binding id="labeledbutton">
  <content>
    <description value="Une pile :"/>
    <stack>
      <children/>
    </stack>
  </content>
</binding>

```

Si vous utilisez les fonctions du DOM telles que `childNodes` pour obtenir les fils d'un élément, vous verrez que la boîte XUL qui a l'id *outer*, a 4 fils. Ceux-ci correspondent à ses 4 boutons, même si ces boutons sont dessinés à l'intérieur de la pile (stack). La pile n'a qu'un seul fils, l'élément children lui-même. La longueur du tableau anonyme de la boîte externe est de deux, le premier élément étant l'élément description et le second étant l'élément stack.

## Constructeurs et Destructeurs

XBL supporte deux méthodes spéciales créées avec des balises séparées, constructor et destructor. Un constructeur est appelé chaque fois qu'une liaison est attachée à un élément. Il est utilisé pour initialiser le contenu tel que le chargement de préférences ou l'initialisation des valeurs par défaut de champs. Le destructeur est appelé lorsqu'une liaison est enlevée d'un élément. Il peut s'avérer utile pour sauvegarder des

informations.

Il y a deux points à savoir lorsqu'une liaison est attachée à un élément. Le premier se produit lorsqu'une fenêtre est affichée. Tous les constructeurs des éléments qui ont un contenu XBL attaché seront invoqués. L'ordre dans lequel ils sont appelés ne devrait pas être pris en compte, car ils sont chargés à partir divers fichiers. Le gestionnaire de chargement de la fenêtre (onload) n'est pas appelé tant que toutes les liaisons n'ont pas été attachées et leurs constructeurs exécutés. Le second point quand une liaison est attachée, est lorsque vous changez la propriété de style `-moz-binding` d'un élément. Après que son destructeur ait été appelé, la liaison existante sera enlevée. Ainsi, la nouvelle liaison sera ajoutée à sa place et son constructeur sera invoqué.

Le script pour un constructeur ou un destructeur devrait être placé directement à l'intérieur de la balise appropriée. Il ne doit y avoir qu'un seul de chaque par liaison et ils ne prennent aucun argument. Voici quelques exemples :

```
<constructor>
  if (this.childNodes[0].getAttribute("open") == "true"){
    this.loadChildren();
  }
</constructor>

<destructor action="saveMyself(this);" />
```

---

La prochaine section montre comment ajouter des gestionnaires d'évènements aux élément XBL définis.

## 11.6 Ajout de gestionnaire d'évènements

Écrit par Neil Deakin. Traduit par *Nadine Henry* (24/08/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblevents.html>

Nous allons voir comment ajouter des gestionnaires d'évènements aux éléments XBL.

### Les gestionnaires d'évènements

Comme vous pourriez le prévoir, des clics de souris, des touches pressées et d'autres évènements sont passés à chacun des éléments à l'intérieur du contenu. Cependant, vous voudrez maîtriser les évènements et les gérer à votre façon. Vous pouvez ajouter des gestionnaires d'évènements aux éléments au sein du contenu si besoin. Le dernier exemple de la section précédente en faisait une démonstration. Dans cet exemple, des gestionnaires `oncommand` avaient été ajoutés sur quelques boutons.

Touefois, vous pouvez vouloir ajouter un gestionnaire d'évènement général au contenu, ce qui représente tous les éléments définis au sein de la balise `content`. Cela pourrait être utile pour maîtriser les évènements de sélection (NdT : Focus) et de désélection. Pour définir un gestionnaire d'évènement, utilisez l'élément `handler`. Chacun de ces éléments décrit l'action d'un seul gestionnaire d'évènement. Vous pouvez utiliser plus d'un gestionnaire si nécessaire. Si un évènement ne correspond pas à l'un des évènements `handler`, il est passé simplement dans le contenu comme d'habitude.

La syntaxe générale du gestionnaire est la suivante :

```
<binding id="binding-name">
  <handlers>
    <handler event="event-name" action="script" />
  </handlers>
</binding>
```

Placez tous vos gestionnaires dans l'élément handlers. Chaque élément handler définit l'action prise pour un événement particulier grâce à son attribut `event`. Les événements valides sont ceux supportés par XUL et JavaScript, tels que *click* et *focus*. Utilisez le nom de l'évènement sans son préfixe 'on'.

La principale raison nécessitant la déclaration de gestionnaires est la modification des propriétés personnalisées lorsqu'un événement se produit. Par exemple, une case à cocher personnalisée pourrait avoir une propriété `checked` qui doit être modifiée lorsque l'utilisateur clique dans la case à cocher :

```
<handlers>
  <handler event="mouseup" action="this.checked=!this.checked"/>
</handlers>
```

Lorsque l'utilisateur clique et relâche le bouton de la souris au dessus de la case à cocher, l'évènement *mouseup* lui est envoyé, et le gestionnaire défini ici est appelé, entraînant le renversement de l'état de la propriété `checked`. De façon similaire, vous pouvez modifier une propriété lorsque l'élément est sélectionné. Vous devrez ajouter des gestionnaires pour ajuster les propriétés lors de sollicitations de la souris ou du clavier.

Pour des événements concernant la souris, vous pouvez utiliser l'attribut `button` pour que le gestionnaire n'accepte que les événements qui correspondent à un certain bouton. Sans cet attribut, le gestionnaire intercepterait tous les événements concernant la souris sans se soucier du bouton qui a été pressé. L'attribut `button` doit être défini à *0* pour le bouton gauche de la souris, *1* pour le bouton du milieu de la souris ou *2* pour le bouton droit de la souris.

```
<handlers>
  <handler event="click" button="0" action="alert('Le bouton gauche est pressé');"/>
  <handler event="mouseup" button="1" action="alert('Le bouton du milieu est pressé')"/>
  <handler event="click" button="2" action="alert('Le bouton droit est pressé');"/>
</handlers>
```

Pour les événements provenant des touches du clavier, vous pouvez utiliser plusieurs attributs similaires à ceux de l'élément key pour les faire correspondre à une touche spécifique et seulement lorsque certaines touches alternatives sont pressées. L'exemple précédent pourrait être complété de telle sorte que la propriété `checked` de la case à cocher soit modifiée lorsque la barre d'espacement est pressée.

```
<handlers>
  <handler event="keypress" key=" " action="this.checked=!checked"/>
</handlers>
```

Vous pouvez aussi utiliser l'attribut `keycode` pour vérifier les touches non imprimables. La section sur les raccourcis clavier fournit plus d'informations. Les touches alternatives peuvent être vérifiées en ajoutant un attribut `modifiers`. Il peut être défini avec l'une des valeurs suivante :

*alt*

L'utilisateur doit presser la touche **Alt**.

*control*

L'utilisateur doit presser la touche **Ctrl**.

*meta*

L'utilisateur doit presser la touche **Meta**.

*shift*

L'utilisateur doit presser la touche **Maj**.

*accel*

L'utilisateur doit presser la touche alternative spéciale qui est habituellement utilisée pour les raccourcis clavier sur sa plate-forme.

S'il est déclaré, le gestionnaire est appelé uniquement lorsque la touche alternative est pressée. Vous pouvez

combiner plusieurs touches modificatrices en les séparant par des espaces.

La syntaxe alternative suivante peut être utilisée lorsque le code dans un gestionnaire est plus complexe :

```
<binding id="binding-name">
  <handlers>
    <handler event="event-name">
      -- le code du gestionnaire vient ici --
    </handler>
  </handlers>
</binding>
```

## Exemple de gestionnaires

L'exemple suivant ajoute des gestionnaires de touches pour créer un presse-papiers local très primitif :

Exemple 11.6.1 :

```
<binding id="clipbox">
  <content>
    <xul:textbox/>
  </content>
  <implementation>
    <field name="clipboard"/>
  </implementation>
  <handlers>
    <handler event="keypress" key="x" modifiers="control"
      action="this.clipboard=document.getAnonymousNodes(this)[0].value;
document.getAnonymousNodes(this)[0].value='';"/>
    <handler event="keypress" key="c" modifiers="control"
      action="this.clipboard=document.getAnonymousNodes(this)[0].value;"/>
    <handler event="keypress" key="v" modifiers="control"
      action="document.getAnonymousNodes(this)[0].value=this.clipboard ? this.clipboard : '';"/>
  </handlers>
</binding>
```

Le contenu est un champ de saisie simple. Un champ `clipboard` lui a été ajouté pour stocker le contenu du presse-papiers. Il signifie que les opérations de presse-papiers sont limitées à ce seul champ de saisie. Le tampon sera propre à ce champ.

Trois gestionnaires ont été ajoutés, un pour couper, un pour copier et l'autre pour coller. Chacun d'eux a sa propre combinaison de touche appellante. Le premier gestionnaire est l'opération *Couper* et est appelé lorsque la touche **Ctrl** est pressée suivie de la touche **x**. Le script à l'intérieur de l'attribut `action` est utilisé pour couper le texte du champ de saisie et pour le mettre dans le champ du presse-papiers. Pour faire simple, le texte entier est coupé et pas seulement le texte sélectionné. Le code fonctionne comme suit :

```
1. this.clipboard=document.getAnonymousNodes(this)[0].value;
```

On récupère le premier élément du tableau de contenu anonyme qui donne une référence à l'élément `textbox` qui s'avère être le premier (et le seul) élément au sein de l'élément `content`. On récupère la propriété `value` qui fournira le texte du champ de saisie. Elle est ainsi assignée au champ du presse-papiers. Le résultat est que le texte qui se trouve dans la champ de saisie est copié dans ce presse-papiers spécial.

```
2. document.getAnonymousNodes(this)[0].value=''
```

On assigne ainsi au texte de l'élément `textbox` une valeur de type chaîne, vide. Elle efface le texte dans le champ de saisie.

Une opération de copie est similaire mais n'efface pas le texte. Le collage est l'opération inverse si ce n'est que la valeur du champ de saisie est assignée à partir de la valeur du champ du presse-papiers. Si nous étions en train de créer une réelle implémentation de ces raccourcis clavier de presse-papiers, nous utiliserions probablement l'interface réelle du presse-papiers et nous gérerions également la sélection courante.

---

Dans la prochaine section, nous verrons comment étendre les définitions XBL existantes.

## 11.7 Héritage XBL

Écrit par Neil Deakin. Traduit par *Nadine Henry* (17/09/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblinherit.html>

Dans cette section, nous verrons comment étendre des définitions XBL existantes.

### Héritage

Parfois vous pouvez vouloir créer un élément graphique XBL qui est similaire à un élément existant. Par exemple, disons que vous souhaitez créer un bouton XBL avec une fenêtre surgissante. Une manière de faire pour le créer est de dupliquer le code XBL existant des boutons. Cependant, il serait préférable de simplement étendre ce code.

Une liaison peut être étendue à une autre. La liaison fille peut ajouter des propriétés, des méthodes et des gestionnaires d'événements. La liaison fille aura toutes les caractéristiques qui la définissent en plus des caractéristiques de la liaison dont elle hérite (et celles dont cette liaison aura elle même hérité et ainsi de suite dans l'arbre).

Pour étendre une liaison existante, ajoutez un attribut `extends` à l'intérieur de la balise `binding`. Par exemple, la liaison suivante crée un champ de saisie qui ajoute le texte `http://www` au début de sa valeur lorsque la touche F4 est pressée.

Exemple 11.7.1 :

```
<binding id="textboxwithhttp"
        extends="chrome://global/content/bindings/textbox.xml#textbox">
  <handlers>
    <handler event="keypress" keycode="VK_F4">
      this.value="http://www"+value;
    </handler>
  </handlers>
</binding>
```

L'élément XBL étend ici les fonctionnalités du champ de saisie XUL `textbox`. L'URL donnée dans l'attribut `extends` ci-dessus est l'URL de la liaison de la balise `textbox`. Elle signifie que nous héritons de tous les contenus et comportements fournis par la liaison de `textbox`. En plus, nous ajoutons un gestionnaire qui répond à l'événement `keypress`.

### Champ de saisie semi-automatique

L'exemple ci-dessus est similaire au dispositif de saisie semi-automatique qui fonctionne sous Mozilla. Un champ de saisie qui supporte la saisie semi-automatique n'est qu'un champ de saisie basique étendu avec une liaison XBL.

Le champ de saisie semi-automatique ajoute une gestion spéciale d'événement de telle sorte que lorsqu'une URL est tapée, un menu va surgir proposant des suites de saisies possibles. Vous pouvez aussi l'utiliser dans

vos propres applications. Créez simplement un `textbox` avec deux attributs spéciaux.

```
<textbox type="autocomplete" searchSessions="history"/>
```

Déclarez l'attribut `type` à *autocomplete* pour ajouter un dispositif de saisie semi-automatique à un champ de saisie existant. Déclarez `searchSessions` pour indiquer le type de données à surveiller. Dans ce cas, la valeur *history* surveillant les URLs dans l'historique est utilisée (Vous pouvez aussi utiliser la valeur *addrbook* pour surveiller les adresses dans le carnet d'adresses.)

---

Dans la prochaine section, nous verrons un exemple d'élément graphique défini en XBL.

## 11.8 Exemple XBL

Écrit par Neil Deakin. Traduit par *Nadine Henry* (17/09/2004), mise à jour par Laurent Jouanneau (20/09/2004) .

Page originale : <http://www.xulplanet.com/tutorials/xultu/xblex.html>

Cette section va décrire un exemple d'élément XBL.

### Un élément de présentation

Construisons un exemple complet d'un élément XBL. Il s'agira de créer un élément graphique qui stocke un ensemble d'objets, en les affichant un par un. Des boutons de navigation situés sur le bas permettront à l'utilisateur d'afficher les objets les uns après les autres (NdT : comme si c'était des pages) tandis qu'un élément graphique textuel entre les boutons affichera le numéro de la page courante. Vous pourriez mettre n'importe quoi dans les pages, cependant, cet élément graphique pourrait être utile pour afficher une série d'images. Nous l'appellerons *élément de présentation* (NdT : 'slideshow').

Tout d'abord, déterminons quels sont les éléments qui ont besoin d'aller dans le contenu XBL. Puisque nous voulons un changement de page, un élément `deck` sera le plus approprié pour contenir les pages. Le contenu des pages sera spécifié dans le fichier XUL, et non dans XBL, mais nous aurons besoin de l'ajouter au sein du paquet (`deck`). La balise `children` devra être utilisée. En bas, nous aurons besoin d'un bouton pour aller à la page précédente, d'un élément graphique pour afficher le numéro de la page courante, et d'un bouton pour aller à la page suivante.

Exemple 11.8.1 :

```
<binding id="slideshow">
  <content>
    <xul:vbox flex="1">
      <xul:deck xbl:inherits="selectedIndex" selectedIndex="0" flex="1">
        <children/>
      </xul:deck>
      <xul:hbox>
        <xul:button xbl:inherits="label=previoustext"/>
        <xul:label flex="1"/>
        <xul:button xbl:inherits="label=nexttext"/>
      </xul:hbox>
    </xul:vbox>
  </content>
</binding>
```

Cette liaison crée la structure de la présentation que nous souhaitons. L'attribut `flex` a été ajouté à plusieurs éléments pour qu'ils s'étendent de la bonne manière. Les attributs `label` sur les deux boutons héritent des valeurs de l'élément qui leur est attaché. Ici, ils héritent de deux attributs personnalisés, `previoustext` et `nexttext`. Ils rendent le changement des libellés des boutons simple. Les fils de l'élément auquel l'élément

XBL est relié seront placés au sein de l'élément deck. L'attribut `selectedIndex` est hérité par le paquet, ainsi nous pouvons déclarer la page initiale dans XUL.

Le fichier XUL suivant produit le résultat affiché dans l'image.

```
<box class="slideshow" previoustext="Précédent" nexttext="Suivant"
flex="1">
  <button label="Bouton 1"/>
  <checkbox label="Case à cocher 2"/>
  <textbox/>
</box>
```

Le style CSS utilisé ici est :

```
.slideshow {
  -moz-binding: url("slideshow.xml#slideshow");
}
```



Le premier bouton, *Bouton 1* a été utilisé comme première page du

paquet. L'élément graphique label (NdT : celui du XBL) n'est pas apparu puisqu'aucun attribut `value` ne lui a été spécifié. Nous pourrions déclarer une valeur, mais elle sera plutôt calculée plus tard.

Ensuite, une propriété contenant le numéro de la page courante est ajoutée. Lorsqu'on obtient cette propriété personnalisée, il est nécessaire de rechercher la valeur de l'attribut `selectedIndex` du paquet qui contient le numéro de la page affichée. De façon similaire, lorsqu'on modifiera cette propriété, il sera nécessaire de changer l'attribut `selectedIndex` du paquet. De plus, l'élément graphique textuel devra être mis à jour pour afficher le numéro de la page courante.

```
<property name="page"
  onget="return
parseInt(document.getAnonymousNodes(this)[0].childNodes[0].getAttribute('selectedIndex'));"
  onset="return this.setPage(val);" />
```

La propriété `page` obtient sa valeur en observant le premier élément du tableau anonyme. Elle renvoie la boîte verticale, donc, pour obtenir le paquet, nous devons obtenir le premier noeud fils de la boîte. Le tableau anonyme n'est pas utilisé puisque le paquet n'est pas anonyme à partir de la boîte. Finalement, la valeur de l'attribut `selectedIndex` est récupérée. Pour spécifier la page, une méthode `setPage` qui sera définie plus tard est appelée.

Un gestionnaire `oncommand` devra être ajouté aux boutons *Précédent* et *Suivant* pour que la page soit changée lorsque les boutons sont pressés. Nous pouvons changer facilement la page en utilisant la propriété personnalisée `page` qui vient d'être ajoutée :

```
<xul:button xbl:inherits="label=previoustext"
  oncommand="parentNode.parentNode.parentNode.page--;" />
<xul:description flex="1"/>
<xul:button xbl:inherits="label=nexttext"
  oncommand="parentNode.parentNode.parentNode.page++;" />
```

Etant donné que la propriété `page` est dans l'élément XUL externe, nous devons utiliser la propriété `parentNode` pour l'obtenir. La première propriété `parentNode` retourne l'élément parent du bouton qui est la boîte horizontale, la seconde son parent, la boîte verticale, et la dernière son parent qui est la boîte externe. La propriété `page` est incrémentée ou décrétementée. Elle va appeler le script `onget` pour obtenir la valeur, incrémentera ou décrétera la valeur, et enfin appellera le gestionnaire `onset` pour enregistrer la valeur.

Définissons à présent la méthode `setPage`. Elle prendra un paramètre, le numéro de page qui sert à spécifier la page. Il sera nécessaire de vérifier que le numéro de page n'est pas en dehors des limites et ensuite modifier les attributs `selectedIndex` du paquet et l'attribut `label` de l'élément graphique textuel.

```
<method name="setPage">
  <parameter name="newidx"/>
  <body>
    <![CDATA[
      var thedeck=document.getAnonymousNodes(this)[0].childNodes[0];
      var totalpages=this.childNodes.length;

      if (newidx<0) return 0;
      if (newidx>=totalpages) return totalpages;
      thedeck.setAttribute("selectedIndex",newidx);
      document.getAnonymousNodes(this)[0].childNodes[1].childNodes[1]
        .setAttribute("value",(newidx+1)+" sur "+totalpages);
      return newidx;
    ]]>
  </body>
</method>
```

Cette fonction est appelée `setPage` et prend un paramètre `newidx`. Le corps de la méthode a été encapsulé entre `<![CDATA[` et `]]>`. C'est le mécanisme général dans tous les fichiers XML qui peut être utilisé pour échapper tout le texte à l'intérieur. De cette manière, vous n'avez pas besoin d'échapper tous les signes "inférieur" et "supérieur" à l'intérieur.

Décomposons le code morceau par morceau.

```
var thedeck=document.getAnonymousNodes(this)[0].childNodes[0];
  Récupère le premier élément du tableau de contenu anonyme qui sera la boîte verticale, puis obtient
  son premier fils qui sera le paquet deck.
var totalpages=this.childNodes.length;
  Obtient le nombre de fils que détient la boîte qui est liée. Cela donnera le nombre total de pages qui
  s'y trouve.
if (newidx<0) return 0;
  Si le nouvel index est avant la première page, ne pas changer la page et retourner 0. La page ne
  devrait pas donner une valeur plus petite que la première page.
if (newidx>=totalpages) return totalpages;
  Si le nouvel index est après la dernière page, ne pas changer la page et retourner le dernier index de
  page. La page ne devrait pas devenir celle qui est après la dernière.
thedeck.setAttribute("selectedIndex",newidx);
  Changer l'attribut selectedIndex du paquet. Cela entraîne l'affichage de la page demandée.
document.getAnonymousNodes(this)[0].childNodes[1].childNodes[1].setAttribute("value",
(newidx+1)+" sur "+totalpages);
  Cette ligne modifie l'élément label pour qu'il affiche l'index de la page courante. L'élément
label peut être récupéré en obtenant le premier élément du contenu anonyme (la boîte verticale), le
second fils de cet élément (la boîte horizontale), et enfin le second élément de cette boîte. L'attribut
value est modifié pour indiquer 1 sur 3 ou quelque chose de similaire. Notez que l'index est
incrémenté de un parce que les indices commence à 0.
```



Nous allons aussi avoir besoin d'un constructeur pour initialiser l'élément `label` afin qu'il s'affiche correctement la première fois que la présentation est affichée. Nous utilisons un code similaire à la méthode ci-dessus pour déclarer le numéro de page. La référence à 'this.page' va appeler le script onget de la propriété page qui à son tour va récupérer la page initiale à partir de l'attribut `selectedIndex`.

```
<constructor>
  var totalpages=this.childNodes.length;
  document.getAnonymousNodes(this)[0].childNodes[1].childNodes[1]
    .setAttribute("value",(this.page+1)+" sur "+totalpages);
</constructor>
```

Nous pouvons aussi ajouter quelques caractéristiques supplémentaires. Certains raccourcis claviers peuvent être utilisés pour les boutons *Précédent* et *Suivant*, (disons l'effacement arrière et la touche Entrée). Des boutons *Premier* et *Dernier* peuvent être ajoutés pour aller à la première et à la dernière page. L'élément `label` pourrait être transformé en un champ de saisie où l'utilisateur pourrait entrer la page à afficher, ou une fenêtre surgissante pourrait être ajoutée pour permettre la sélection de la page à partir d'un menu. Nous pourrions aussi ajouter une bordure autour de la boîte avec un style CSS pour la rendre plus jolie.

Le code final est le suivant :

Exemple 11.8.2 :

```
<binding id="slideshow">
  <content>
    <xul:vbox flex="1">
      <xul:deck xbl:inherits="selectedIndex" selectedIndex="0" flex="1">
        <children/>
      </xul:deck>
      <xul:hbox>
        <xul:button xbl:inherits="label=previoustext"
          oncommand="parentNode.parentNode.parentNode.page--;" />
        <xul:description flex="1"/>
        <xul:button xbl:inherits="label=nexttext"
          oncommand="parentNode.parentNode.parentNode.page++;" />
      </xul:hbox>
    </xul:vbox>
  </content>

  <implementation>

    <constructor>
      var totalpages=this.childNodes.length;
      document.getAnonymousNodes(this)[0].childNodes[1].childNodes[1]
        .setAttribute("value",(this.page+1)+" sur "+totalpages);
    </constructor>

    <property name="page"
      onget="return
parseInt(document.getAnonymousNodes(this)[0].childNodes[0].getAttribute('selectedIndex'));"
      onset="return this.setPage(val);"/>

    <method name="setPage">
      <parameter name="newidx"/>
      <body>
        <![CDATA[
          var thedeck=document.getAnonymousNodes(this)[0].childNodes[0];
          var totalpages=this.childNodes.length;

          if (newidx<0) return 0;
          if (newidx>=totalpages) return totalpages;
          thedeck.setAttribute("selectedIndex",newidx);
          document.getAnonymousNodes(this)[0].childNodes[1].childNodes[1]
```

```
        .setAttribute("value",(newidx+1)+" sur "+totalpages);
    return newidx;
  ]]>
</body>
</method>
</implementation>

</binding>
```

Tester dans une fenêtre : .

---

Nous allons voir ensuite quelques propriétés additionnelles d'une fenêtre.

## 12. Fenêtres spécialisées

### 12.1 Caractéristiques d'une fenêtre

Écrit par Neil Deakin. Traduit par *Julien Appert* (16/05/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/featwin.html>

Nous avons déjà vu quelques caractéristiques de fenêtres. Nous allons en voir quelques unes de plus dans cette section.

#### Créer une autre fenêtre

Vous pouvez créer une seconde fenêtre pour votre application de la même manière que vous avez créé la première. Il suffit de créer un second fichier XUL avec le code de la fenêtre à l'intérieur. Comme en HTML, vous pouvez utiliser la fonction `window.open` pour ouvrir la seconde fenêtre. Cette fonction retournera une référence à la fenêtre nouvellement ouverte. Vous pouvez utiliser cette référence pour appeler des fonctions de l'autre fenêtre.

La fonction `open` prend trois arguments. Le premier est l'URL du fichier que vous souhaitez ouvrir. Le second est le nom interne de la fenêtre. Le troisième est une liste de drapeaux de paramètres d'affichage. Le drapeau `chrome` est important pour ouvrir la fenêtre comme un fichier `chrome`. Si vous n'ajoutez pas le drapeau `chrome`, le fichier sera ouvert dans une nouvelle fenêtre du navigateur.

Par exemple :

```
window.open("chrome://findfile/content/findfile.xul", "findfile", "chrome");
```

#### Spécifier la largeur et la hauteur

Vous aurez noté que lorsque des éléments sont ajoutés à une fenêtre, la largeur de la fenêtre s'étend pour s'adapter aux nouveaux éléments. La fenêtre n'est en fait qu'une boîte flexible et qui prend par défaut une orientation verticale. Vous pouvez également spécifier la largeur et la hauteur directement dans la balise `window`. Cela, bien sûr, oblige la fenêtre à prendre une taille spécifique. Si vous ôtez ces spécifications, la taille sera déterminée par les éléments qui la constituent.

```
<window
  id="findfile-window"
  title="Recherche de fichiers"
  width="400"
  height="450"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Dans cet exemple, la fenêtre sera ouverte avec une largeur de 400 pixels et une hauteur de 450 pixels. Même s'il n'y a pas assez d'éléments pour lui faire prendre cette taille, la fenêtre s'ouvrira toujours à cette taille et il y aura un espace blanc dans l'espace restant. S'il y a trop d'éléments, la fenêtre ne s'adaptera pas pour eux. L'utilisateur devra redimensionner la boîte de dialogue. Vous devez faire attention, lorsque vous spécifiez une largeur et une hauteur, à ce que la fenêtre ne soit pas trop petite ou trop grande.

Notez que vous devez spécifier à la fois la largeur *et* la hauteur. Si vous ne spécifiez qu'une des deux valeurs, l'autre sera mise à 0. Pour que la fenêtre détermine sa taille automatiquement, omettez les toutes les deux.

La largeur et la hauteur déterminent seulement la taille initiale de la fenêtre. L'utilisateur peut toujours redimensionner la fenêtre, à condition que celle-ci soit redimensionnable.

## Autres propriétés de fenêtre

Les drapeaux ci-après peuvent être utilisés comme troisième argument de la fonction `window.open`. Votre système d'exploitation peut ne pas tous les supporter. Vous pouvez également utiliser un des drapeaux pré-existants que vous devriez trouver dans une référence Javascript. Vous pouvez désactiver une propriété en la réglant sur *no*, par exemple `dialog=no`.

*alwaysLowered*

La fenêtre apparaîtra toujours derrière les autres fenêtres.

*alwaysRaised*

La fenêtre apparaîtra toujours devant les autres fenêtres.

*centerscreen*

À son ouverture, la fenêtre sera centrée par rapport à l'écran.

*dependent*

La position de la fenêtre sera toujours relative à la fenêtre qui l'a ouverte. Si la position de la fenêtre initiale est modifiée, la deuxième fenêtre bougera automatiquement de même manière.

*dialog*

la fenêtre est une boîte de dialogue pouvant apparaître différemment.

*modal*

La boîte de dialogue est modale. La fenêtre qui a ouvert la fenêtre modale ne peut pas être utilisée tant que la fenêtre modale n'est pas fermée.

*resizable*

L'utilisateur peut redimensionner la fenêtre.

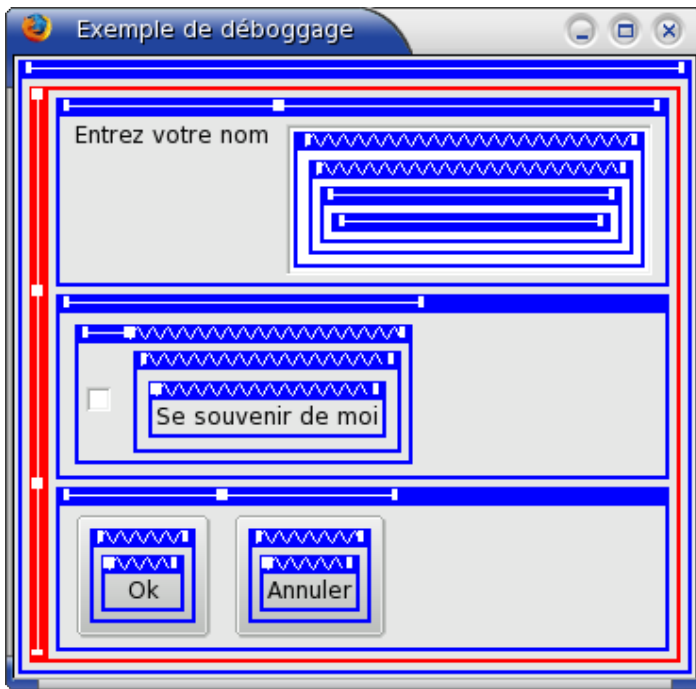
## Déboguer une fenêtre

Une autre propriété très utile lors des développements est l'activation du mode débogage sur une fenêtre. Pour l'activer, ajoutez un attribut `debug` à la fenêtre et régler le sur *true*. Il obligera la fenêtre à mettre en évidence les boîtes et les espacements pour que vous puissiez voir ce qui se passe. L'exemple ci-après montre comment l'utiliser.

Exemple 12.1.1 :

```
<window
  id="findfile-window"
  title="Recherche de fichiers"
  debug="true"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

L'image ci-dessous montre l'effet appliqué à une simple fenêtre.



- Les boîtes bleues indiquent les boîtes horizontales.
- Les boîtes rouges indiquent les boîtes verticales. Vous pouvez voir que la fenêtre est également une boîte verticale.
- Les zigzags montrent où sont les éléments flexibles. Dans ce cas, l'espacement est flexible donc un zigzag apparaît au dessus.
- Les lignes montrent où sont les éléments non flexibles, dans notre cas le texte, les champs de formulaire et les boutons.
- Les boîtes carrés blanches indiquent les limites des éléments.

Sur l'image, vous pouvez noter plusieurs boîtes supplémentaires. En fait, chaque élément XUL est lui-même constitué d'un certain nombre de boîtes, définies avec XBL. Vous pouvez normalement les ignorer. Vous pouvez définir l'attribut `debug` sur chaque boîte, pas seulement sur la fenêtre.

Nous allons voir ensuite comment ouvrir des boîtes de dialogue secondaires.

## 12.2 Créer des boîtes de dialogues

Écrit par Neil Deakin. Traduit par *Durandal* (06/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/dialogs.html>

Une application XUL aura souvent besoin d'afficher des boîtes de dialogues. Cette section décrit comment les construire.

### Créer une boîte de dialogue

La fonction `open` est utilisée pour ouvrir une fenêtre. Une fonction apparentée est `openDialog`. Cette fonction a plusieurs grandes différences. Elle affiche une boîte de dialogue au lieu d'une fenêtre, ce qui implique qu'elle demande quelque chose à l'utilisateur. Elle peut avoir des différences subtiles dans sa manière de fonctionner et d'apparaître à l'utilisateur. Ces différences varient selon chaque plate-forme.

De plus, la fonction `openDialog` peut prendre des arguments additionnels en plus des trois premiers. Ces arguments sont passés à la nouvelle boîte de dialogue et placés dans un tableau stocké dans la propriété `arguments` de la nouvelle fenêtre. Vous pouvez passer autant d'arguments que nécessaire. C'est un moyen

pratique de fournir des valeurs par défaut aux champs de la boîte de dialogue.

```
var somefile=document.getElementById('enterfile').value;

window.openDialog("chrome://findfile/content/showdetails.xul","showmore",
    "chrome",somefile);
```

Dans cet exemple, la boîte de dialogue *showdetails.xul* sera affichée. L'argument *somefile* provenant d'un élément d'id *enterfile* lui est transmis. Dans un script utilisé par la boîte de dialogue, nous pouvons alors faire référence à l'argument en utilisant la propriété *arguments* de la fenêtre. Par exemple :

```
var fl=window.arguments[0];

document.getElementById('thefile').value=fl;
```

C'est un moyen efficace de passer des valeurs à la nouvelle fenêtre. Vous pouvez renvoyer des valeurs de la fenêtre ouverte vers la fenêtre originelle de deux manières. La première, vous pouvez utiliser la propriété *window.opener* qui contient la fenêtre qui a ouvert la boîte de dialogue. La seconde, vous pouvez passer une fonction ou un objet en argument, puis appeler la fonction ou modifier l'objet dans la boîte de dialogue ouverte.

## L'élément dialog

L'élément dialog doit être utilisé à la place de l'élément window lors de la création d'une boîte de dialogue. Il offre la possibilité utile de construire jusqu'à quatre boutons au bas de la boîte de dialogue, pour OK, Annuler, etc. Vous n'avez pas besoin d'inclure le code XUL pour chaque bouton mais vous devez fournir le code à exécuter quand l'utilisateur presse chaque bouton. Ce mécanisme est nécessaire car les différentes plates-formes ont un ordre spécifique d'affichage des boutons.

Exemple 11.2.1 :

```
<?xml version="1.0" encoding="iso-8859-1"?>

<?xml-stylesheet href="chrome://global/skin/global.css" type="text/css"?>

<dialog id="donothing" title="Ne fait rien"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
    buttons="accept,cancel"
    ondialogaccept="return doOK();"
    ondialogcancel="return doCancel();">

<script>
function doOK()
{
    alert("Vous avez appuyé sur OK !");
    return true;
}

function doCancel()
{
    alert("Vous avez appuyé sur Annuler !");
    return true;
}
</script>

<description value="Veuillez sélectionner un bouton"/>

</dialog>
```

Vous pouvez placer tous les éléments que vous souhaitez dans une boîte de dialogue. L'élément `dialog` a des attributs additionnels que les fenêtres n'ont pas. L'attribut `buttons` est utilisé pour spécifier quels boutons doivent apparaître dans la boîte de dialogue. Les valeurs suivantes peuvent être utilisées en les séparant par des virgules :

- *accept* – un bouton OK
- *cancel* – un bouton Annuler
- *help* – un bouton Aide
- *disclosure* – un bouton d'information, utilisé pour montrer plus d'informations.

Vous pouvez faire que du code s'exécute lors de l'appui des boutons en utilisant les attributs `ondialogaccept`, `ondialogcancel`, `ondialoghelp` et `ondialogdisclosure`. Si vous essayez l'exemple ci-dessus, vous remarquerez que la fonction `doOK` est appelée quand le bouton *OK* est pressé et la fonction `doCancel` est appelée quand le bouton *Annuler* est pressé.

Les deux fonctions `doOK` et `doCancel` renvoient *true*, ce qui indique que la boîte de dialogue doit être fermée. Si la valeur *false* était renvoyée, la boîte de dialogue resterait ouverte. Ce fonctionnement pourrait être utilisé dans le cas où une valeur invalide serait entrée dans un champ de la boîte de dialogue.

Dans la prochaine section, nous allons voir comment ouvrir des boîtes de dialogue de sélection de fichiers.

## 12.3 Boîte de dialogue de fichiers

Écrit par Neil Deakin. Traduit par *Cyril Cheneson* (28/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/filedialog.html>

Un type commun de boîtes de dialogue avec lequel un utilisateur peut sélectionner un fichier à ouvrir ou à enregistrer.

### Sélecteurs de fichiers

Un sélecteur de fichiers est une boîte dialogue qui permet à l'utilisateur de sélectionner un fichier. Il est fréquemment utilisé pour les commandes de menu 'Ouvrir' ou 'Enregistrer Sous', mais vous pouvez l'utiliser n'importe où l'utilisateur a besoin de sélectionner un fichier. L'interface `XPCOM nsIFilePicker` est utilisée pour implémenter un sélecteur de fichiers.

Vous pouvez utiliser le sélecteur de fichiers dans l'un des trois modes :

- `Open` : Il est demandé à l'utilisateur de sélectionner un fichier à ouvrir.
- `GetFolder` : Il est demandé à l'utilisateur de sélectionner un répertoire.
- `Save` : Il est demandé à l'utilisateur de sélectionner le nom sous lequel sera sauvegardé le fichier.

L'apparence de la boîte de dialogue sera différente pour chaque type et variera selon la plate-forme. Une fois que l'utilisateur aura sélectionné le fichier ou le répertoire, celui-ci pourra être lu ou enregistré.

L'interface du sélecteur de fichiers `nsIFilePicker` est responsable de l'affichage de la boîte de dialogue dans l'un des trois modes. Vous pouvez définir plusieurs fonctionnalités de la boîte de dialogue en utilisant l'interface. Une fois que la boîte de dialogue est fermée, vous pouvez utiliser les fonctions de l'interface pour obtenir le fichier qui a été sélectionné.

Pour commencer, vous avez besoin de créer un composant du sélecteur de fichiers et l'initialiser.

```
var nsIFilePicker = Components.interfaces.nsIFilePicker;
var fp = Components.classes["@mozilla.org/filepicker;1"]
```

```
.createInstance(nsIFilePicker);
fp.init(window, "Sélectionner un fichier", nsIFilePicker.modeOpen);
```

Tout d'abord, un nouvel objet sélecteur de fichiers est créé et stocké dans une variable `fp`. La fonction `init` est utilisée pour initialiser le sélecteur de fichiers. Cette fonction prend trois arguments, la fenêtre qui ouvre la boîte de dialogue, le titre de la boîte de dialogue et le mode. Ici, le mode est *modeOpen* qui est utilisé pour une boîte de dialogue Ouvrir. Vous pouvez aussi utiliser *modeGetFolder* et *modeSave* pour les deux autres modes. Ces modes sont des constantes de l'interface `nsIFilePicker`.

Il y a deux fonctionnalités que vous pouvez définir pour la boîte de dialogue avant qu'elle ne soit affichée. La première est le répertoire par défaut qui est affiché quand la boîte de dialogue s'ouvre. La seconde est un filtre indiquant la liste des types de fichiers à afficher dans la boîte de dialogue. Elle pourrait être utilisée, par exemple, afin de tout cacher sauf les fichiers HTML.

Vous pouvez définir le répertoire par défaut en définissant la propriété `displayDirectory` de l'objet du sélecteur de fichiers. Le répertoire doit être un objet `nsILocalFile`. Si vous ne le définissez pas, un répertoire par défaut sera sélectionné pour vous. Pour ajouter des filtres, appelez la fonction `appendFilters` pour définir les types de fichiers que vous souhaitez voir afficher.

```
fp.appendFilters(nsIFilePicker.filterHTML | nsIFilePicker.filterImages);
fp.appendFilters(nsIFilePicker.filterText | nsIFilePicker.filterAll);
```

Le premier exemple ajoutera des filtres pour les fichiers HTML et les images. L'utilisateur ne pourra sélectionner que ces types de fichiers. La manière de procéder est spécifique à la plate-forme. Sur quelques plates-formes, chaque filtre sera séparé et l'utilisateur pourra choisir entre les fichiers HTML et les fichiers images. Le second exemple ajoutera des filtres pour les fichiers textes et pour tous les fichiers. L'utilisateur a ainsi une option pour n'afficher que les fichiers textes ou tous les fichiers.

Vous pouvez aussi utiliser *filterXML* et *filterXUL* pour filtrer les fichiers XML et XUL. Si vous voulez filtrer des fichiers personnalisés, vous pouvez utiliser la fonction `appendFilter` dans ce but :

```
fp.appendFilter("Fichiers Audio", "*.wav; *.mp3");
```

Cette ligne ajoutera un filtre pour les fichiers audio Wav et MP3. Le premier argument est le titre du type de fichier et le second est une liste de masques de fichiers séparés par un point virgule. Vous pouvez mettre autant de masques que vous le souhaitez. Vous pouvez appeler `appendFilter` autant de fois que nécessaire pour ajouter les filtres supplémentaires. L'ordre dans lequel vous les ajoutez détermine leur priorité. Habituellement, le premier ajouté est sélectionné par défaut.

Enfin, vous pouvez afficher la boîte de dialogue en appelant la fonction `show`. elle ne prend aucun argument mais retourne un code d'état qui indique ce que l'utilisateur a sélectionné. Notez que la fonction ne retourne aucune valeur tant que l'utilisateur n'a pas sélectionné un fichier. La fonction retourne une des trois constantes :

*returnOK* :

l'utilisateur a sélectionné un fichier et a pressé OK. Le fichier que l'utilisateur a sélectionné sera stocké dans la propriété `file` du sélecteur de fichiers.

*returnCancel* :

l'utilisateur a pressé Annuler.

*returnReplace* :

dans le mode enregistrement, cette valeur de retour signifie que l'utilisateur a sélectionné un fichier à remplacer (*returnOK* sera retournée lorsque l'utilisateur aura rentré le nom d'un nouveau fichier).

Vous devrez vérifier la valeur de retour et ensuite utiliser l'objet `file` du sélecteur de fichiers en utilisant la propriété `file`.



```
var res = fp.show();
if (res == nsIFilePicker.returnOK){
    var thefile = fp.file;
    // --- faire quelque chose avec le fichier ici ---
}
```

---

Dans la suite, nous verrons comment créer un assistant.

## 12.4 Creation d'un assistant

Écrit par Neil Deakin. Traduit par *Laurent Jouanneau* (15/11/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/wizard.html>

Beaucoup d'applications utilisent des assistants pour aider l'utilisateur dans des tâches complexes. XUL fournit un moyen de créer des assistants facilement.

### L'élément wizard

Un assistant est un type spécial de boîte de dialogue, contenant un certain nombre de pages. Des boutons de navigation apparaissent en bas de la boîte de dialogue pour passer d'une page à l'autre. Les assistants sont habituellement utilisés pour aider l'utilisateur à effectuer des tâches complexes. Chaque page contient une seule question ou un ensemble de question associées. À la dernière page, l'opération est effectuée.

XUL fournit un élément wizard qui peut être utilisé pour créer des assistants. Le contenu à l'intérieur de l'élément wizard inclut tout le contenu de chaque page. Les attributs placés sur l'élément wizard sont utilisés pour contrôler la navigation. Quand vous créez un assistant, utilisez la balise wizard à la place de la balise window.

Notez que pour le moment les assistants ne fonctionnent correctement qu'à partir d'URLs chrome.

Un assistant consiste en un ensemble de sections, bien que la mise en page exacte variera pour chaque plate-forme. L'aspect visuel de l'assistant s'adaptera à la plate-forme de l'utilisateur. Une mise en page type inclura un titre en haut, un ensemble de boutons de navigations en bas, et le contenu de la page entre les deux.

Le titre du haut est créé en utilisant l'attribut `title` comme on le ferait pour les fenêtres normales. Les boutons de navigation sont créés automatiquement. Les pages de l'assistant sont créées en utilisant l'élément wizardpage. Vous pouvez y placer tout contenu que vous voulez. Voici un exemple d'assistant :

Exemple 12.4.1 :

```
<?xml version="1.0" encoding="iso-8859-1"?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<wizard id="example-window" title="Assistant de sélection de chien"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

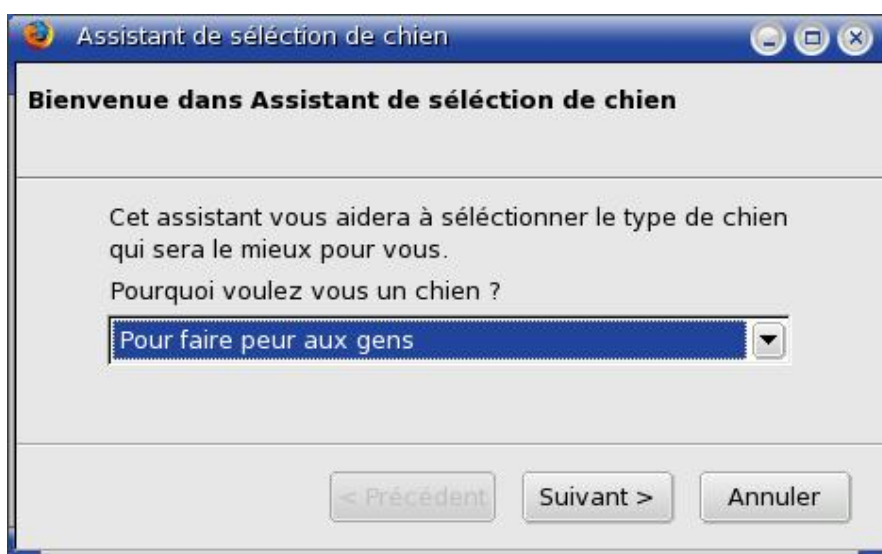
  <wizardpage>
    <description>
      Cet assistant vous aidera à sélectionner le type de chien qui sera le mieux pour vous.
    </description>
    <label value="Pourquoi voulez vous un chien ?"/>
    <menulist>
      <menupopup>
        <menuitem label="Pour faire peur aux gens"/>
        <menuitem label="Pour se débarrasser d'un chat"/>
        <menuitem label="J'ai besoin d'un meilleur ami"/>
      </menupopup>
    </menulist>
  </wizardpage>
</wizard>
```

```

    </menupopup>
</menulist>
</wizardpage>

<wizardpage description="Détails sur le chien">
  <label value="Fournissez plus de détails sur le chien que vous désirez :"/>
  <radiogroup>
    <caption label="Taille"/>
    <radio value="small" label="Petit"/>
    <radio value="large" label="Grand"/>
  </radiogroup>
  <radiogroup>
    <caption label="Sexe"/>
    <radio value="male" label="Male"/>
    <radio value="female" label="Femelle"/>
  </radiogroup>
</wizardpage>
</wizard>

```



L'assistant a deux pages, une qui a une liste déroulante et une autre qui a un ensemble de boutons radios. L'assistant sera formaté automatiquement, avec un titre en haut et un ensemble de boutons le long du bas de la fenêtre. L'utilisateur peut naviguer entre les pages de l'assistant avec les boutons *Précédent* et *Suivant* (NdT : "Back" et "Next" avec une version anglaise). Ces boutons s'activeront ou se désactiveront eux-mêmes aux moments appropriés. De plus, sur la dernière page, le bouton *Terminer* apparaîtra. Tout ceci est automatique, aussi, vous n'avez rien à faire pour manipuler les pages.

L'attribut `description` peut être éventuellement placé sur l'élément `wizardpage` pour fournir un sous-titre pour la page concernée. Dans l'exemple du dessus, il est placé sur la seconde page, mais pas sur la première.

## Gérer les changements de page

Généralement, vous souhaitez réaliser une action après que le bouton *Terminer* ait été pressé. Vous pouvez mettre un attribut `onwizardfinish` sur l'élément `wizard` pour accomplir cette tâche. Spécifiez-y un script qui exécutera ce que vous voulez et qui renverra *true*. Le script peut être utilisé pour sauvegarder les informations que l'utilisateur a saisi dans l'assistant.

Par exemple :

```
<wizard id="example-window" title="Assistant de sélection de chien">
```

```
onwizardfinish="return saveDogInfo();"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Quand l'utilisateur clique sur le bouton *Terminer*, la fonction `saveDogInfo` définie dans un script sera appelée pour sauvegarder les informations saisies. Si la fonction renvoie *true*, l'assistant se fermera. Si elle renvoie *false*, alors l'assistant ne se fermera pas, ce qui pourrait signifier par exemple que la fonction `saveDogInfo` a rencontré une saisie invalide.

Il existe également les attributs similaires `onwizardback`, `onwizardnext` et `onwizardcancel` qui sont appelés quand les boutons *Précédent*, *Suivant* et *Annuler* sont respectivement pressés. Ces fonctions sont appelées quelque soit la page en cours affichée.

Pour appeler un code différent en fonction de la page où vous êtes, utilisez les attributs `onpagerewound` ou `onpageadvanced` sur un élément `wizardpage`. Ils fonctionnent de manière similaire aux autres fonctions, excepté que vous pouvez utiliser un script différent pour chaque page. Cette méthode vous permet de valider les informations saisies sur chaque page avant que l'utilisateur ne continue.

Une troisième méthode est l'utilisation des attributs `onpagehide` et `onpageshow` sur l'élément `wizardpage`. Ils sont appelés lorsque la page est cachée ou affichée, indifféremment du bouton pressé (excepté quand le bouton *Annuler* est pressé ; vous devez utiliser `onwizardcancel` pour le vérifier).

Ces trois méthodes devraient fournir suffisamment de souplesse pour gérer la navigation comme vous le souhaitez. Ce qui suit est un résumé des fonctions d'attributs appelées quand l'utilisateur presse *Suivant*, dans l'ordre dans lequel elles sont vérifiées. Dès que l'une renvoie *false*, la navigation est annulée.

Attribut	Placé sur la balise	Quand est-il appelé
<code>pagehide</code>	<code>wizardpage</code>	Appelé sur la page que l'utilisateur quitte.
<code>pageadvanced</code>	<code>wizardpage</code>	Appelé sur la page que l'utilisateur quitte.
<code>wizardnext</code>	<code>wizard</code>	Appelé sur l'assistant.
<code>pageshow</code>	<code>wizardpage</code>	Appelé sur la page sur laquelle l'utilisateur entre.

Un processus similaire existe pour le bouton *Précédent*.

---

Dans la prochaine section, nous verrons des fonctionnalités supplémentaires sur les assistants.

## 12.5 Assistant avancé

Écrit par Neil Deakin. Traduit par *Damien Hardy* (25/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/advwiz.html>

Cette section décrit quelques fonctionnalités supplémentaires sur les assistants.

### Une Navigation plus complexe

Normalement, un assistant affiche chaque page dans l'ordre où vous les placez dans le fichier XUL. Dans certains cas cependant, vous pouvez vouloir que les pages de l'assistant apparaissent en fonction de ce que l'utilisateur a sélectionné sur les pages précédentes.

Dans ce cas, placez un attribut `pageid` sur chacune des pages. Il aura valeur d'identifiant pour chaque page. Ensuite, pour naviguer vers une page, utilisez une des deux méthodes suivantes :

1. Affectez à l'attribut `next` sur chaque page, la valeur de l'identifiant de page vers laquelle vous souhaitez aller. Vous pouvez changer ces attributs comme vous le souhaitez pour naviguer vers

d'autres pages.

2. Appelez la méthode `goTo` de l'assistant. Elle prend comme unique argument l'identifiant de la page suivante. Vous pouvez appeler cette méthode dans les gestionnaires `onpageadvanced` ou `onwizardnext`. Dans ce cas, n'oubliez pas de retourner la valeur *false*, car vous avez déjà changé la page par vous même.

Notez que la méthode `goTo` charge une nouvelle page, par conséquent les événements liés sont lancés. Vous devez donc vous assurer d'avoir prévu ce cas.

Par exemple, nous avons ici un ensemble de pages d'assistant (le contenu a été omis) :

```
<wizardpage pageid="type" next="font"/>
<wizardpage pageid="font" next="done"/>
<wizardpage pageid="color" next="done"/>
<wizardpage pageid="done"/>
```

L'assistant commence toujours à la première page, qui dans ce cas a l'identifiant de page *type*. La page suivante est celle qui a l'identifiant de page *font*, l'assistant affichera donc cette page juste après. Sur la page qui a l'identifiant *font*, nous pouvons voir que la page suivante est celle identifiée *done*, cette page sera alors affichée ensuite. La page identifiée *done* n'a pas d'attribut `next`, elle sera donc la dernière page. Un script ajustera la valeur de l'attribut `next` dès qu'il sera nécessaire d'aller sur la page identifiée *color*.

## Fonctions d'assistant

L'assistant fonctionne presque comme un panneau d'onglets, sauf que les onglets ne sont pas affichés et que l'utilisateur navigue entre les pages en utilisant les boutons de bas de page. Comme toutes les pages font parties d'un même fichier, toutes les valeurs des champs sur toutes les pages sont conservées. Par conséquent, il est inutile de sauvegarder ou charger ces informations entre les pages.

Cependant vous pourriez vouloir faire quelques validations sur chaque champ de chaque page. Pour ce faire, utilisez les gestionnaires décrits dans la section précédente. Si un champ est invalide, vous pouvez afficher un message d'alerte. Dans certains cas, il serait plus judicieux de désactiver le bouton *Suivant* jusqu'à ce qu'une valeur valide soit saisie.

L'assistant possède une propriété `canAdvance` qui, quand elle est mise à *true*, indique que le bouton *Suivant* doit être actif. Si elle a la valeur *false*, le bouton *Suivant* est désactivé. Vous pouvez changer la propriété quand des données valides ou invalides sont saisies.

Dans l'exemple suivant, l'utilisateur doit entrer un code secret dans le `textbox` sur la première page de l'assistant. La fonction `checkCode` est appelée dès que la première page est affichée comme indiquée par l'attribut `onpageshow`. Elle est aussi appelée dès qu'une touche est pressée dans le `textbox`, pour déterminer si le bouton *Suivant* doit être à nouveau actif.

Exemple 12.5.1 :

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<wizard id="theWizard" title="Code secret de l'assistant"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

  <script>
  function checkCode()
  {
    document.getElementById('theWizard').canAdvance=
      (document.getElementById('secretCode').value == "cabbage");
```

```

    }
</script>

<wizardpage onpageshow="checkCode();">
    <label value="Saisir le code secret:"/>
    <textbox id="secretCode" onkeyup="checkCode();" />
</wizardpage>

<wizardpage>
    <label value="Le code est correct."/>
</wizardpage>

</wizard>

```

Il existe aussi une propriété correspondante `canRewind` que vous pouvez utiliser pour activer ou désactiver le bouton *Précédent*. Les deux propriétés sont ajustées automatiquement dès que vous changez de page. Par conséquent, le bouton *Précédent* sera désactivé sur la première page, vous n'avez donc pas à le faire.

Une autre propriété utile de l'assistant est `currentPage` qui donne une référence à la page wizardpage actuellement affichée. Vous pouvez aussi changer la page courante en modifiant cette propriété. Si vous modifiez sa valeur, les différents événements de changement de page seront invoqués.

Ensuite, nous allons voir comment utiliser l'overlay pour gérer du contenu.

## 12.6 Overlays

Écrit par Neil Deakin. Traduit par **Alain B.** (24/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/overlay.html>

Cette section décrit les overlays qui peuvent être utilisés pour partager du contenu commun.

### Utilisation des Overlays

Dans une simple application ne comportant qu'une seule fenêtre, vous aurez généralement qu'un seul fichier XUL, associé avec un fichier de scripts, une feuille de styles, un fichier d'entités DTD et peut être quelques images. Certaines applications contiennent également plusieurs boîtes de dialogue. Celles ci sont placées dans des fichiers XUL séparés. Des applications plus sophistiquées peuvent contenir de nombreuses fenêtres et boîtes de dialogues.

Une application contenant de nombreuses fenêtres aura certains éléments ou parties d'interface utilisateur communs entre chacune d'elles. Par exemple, chaque composant de Mozilla partage des éléments communs. Certains menus sont similaires, comme les menus Outils et Aide, la barre latérale est similaire, et chaque fenêtre partage des raccourcis claviers globaux communs.

Il est bien entendu possible de ré-implémenter des éléments ou des fonctions similaires dans chaque fichiers nécessaires. Toutefois, la maintenance devient difficile. Si vous décidez de modifier quelque chose, vous devrez le faire à différents endroits. En revanche, il est plus intéressant d'utiliser un mécanisme permettant de séparer les éléments communs et de les partager entre les fenêtres. Vous pouvez le faire avec des overlays.

Vous pouvez placer dans un overlay des éléments qui seront partagés par toutes les fenêtres utilisant cet overlay. Ces éléments sont ajoutés à l'intérieur des fenêtres aux emplacements déterminés par leurs identifiants `id`.

Par exemple, disons que vous voulez créer un menu Aide partagé par plusieurs fenêtres. Le menu Aide sera placé dans un overlay, en utilisant la syntaxe XUL habituelle. Un attribut `id` sera affecté au menu pour l'identifier. Chaque fenêtre importera l'overlay en utilisant une directive qui sera décrite dans un moment.

Afin de pouvoir utiliser le menu Aide tel que définit dans l'overlay, il vous suffit d'ajouter un simple élément de menu ayant le même attribut `id` que celui utilisé dans l'overlay. Ce menu n'a pas besoin de contenir tous les éléments enfants comme ceux placés dans l'overlay.

Lorsqu'une fenêtre contenant un overlay est ouverte, les éléments de la fenêtre et ceux de l'overlay ayant le même identifiant `id` sont combinés entre eux. Les enfants des éléments correspondants sont ajoutés à la fin de l'ensemble des enfants des éléments de la fenêtre. Les attributs présents sur les éléments de l'overlay sont appliqués à ceux de la fenêtre. Ces détails seront expliqués ci après.

Pour importer un overlay dans une fenêtre, utilisez la syntaxe ci dessous. Ajoutons cette ligne vers le haut de notre exemple XUL de recherche de fichiers.

```
<?xul-overlay href="chrome://findfile/content/helpoverlay.xul"?>
```

Cette ligne doit être ajoutée quelque part en haut du fichier, habituellement juste avant les déclarations d'entités DTD. Dans l'exemple précédent, la fenêtre importera un overlay contenu dans le fichier *helpoverlay.xul*.

L'overlay lui même est un fichier XUL contenant un élément overlay au lieu d'un élément window. À part cette différence, ils sont similaires. Il est possible d'importer des overlays à l'intérieur d'autres overlays. Les overlays peuvent aussi disposer de leurs propres feuilles de styles, fichiers d'entités DTD et scripts. L'exemple ci dessous montre un simple menu Aide placé dans un overlay.

Exemple 12.6.1 :

```
<?xml version="1.0"?>

<!DOCTYPE overlay SYSTEM "chrome://findfile/locale/findfile.dtd">

<overlay id="toverlay"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<menu id="help-menu">
  <menupopup id="help-popup">
    <menuitem id="help-contents" label="&contentsCmd.label;"
      accesskey="&contentsCmd.accesskey;" />
    <menuitem id="help-index" label="&indexCmd.label;"
      accesskey="&indexCmd.accesskey;" />
    <menuitem id="help-about" label="&aboutCmd.label;"
      accesskey="&aboutCmd.accesskey;" />
  </menupopup>
</menu>

</overlay>
```

L'élément overlay entoure le contenu de l'overlay. Les mêmes espaces de nommage sont utilisés qu'avec les fichiers de fenêtres XUL. Un simple menu comprenant trois items a été défini dans l'overlay. L'attribut `id` de ce menu est *help-menu*. Il signifie que son contenu sera ajouté à une fenêtre dans laquelle un élément similaire existe avec la même valeur d'attribut `id`. Si un tel élément n'existe pas, cette partie de l'overlay sera ignoré. L'overlay peut contenir autant d'éléments que nécessaire. Notez que le fichier d'entités DTD doit être inclus dans l'overlay. Nous avons utilisé ici le même que celui de la fenêtre principale, mais normalement vous devez créer des fichiers DTD séparés pour chaque overlay.

Ensuite, nous allons ajouter le menu Aide à notre boîte de dialogue de recherche de fichiers. Ajoutez simplement un menu avec le même attribut `id` au bon endroit. Il trouvera sa juste place après le menu Edition.

```
<menu id="edit-menu" label="Edition" accesskey="e">
```

```

<menupopup id="edit-popup">
  <menuitem label="&cutCmd.label;" accesskey="&cutCmd.accesskey;"
    key="cut_cmd"/>
  <menuitem label="&copyCmd.label;" accesskey="&copyCmd.accesskey;"
    key="copy_cmd"/>
  <menuitem label="&pasteCmd.label;" accesskey="&pasteCmd.accesskey;"
    key="paste_cmd" disabled="true"/>
</menupopup>
</menu>
<menu id="help-menu" label="&helpCmd.label;"
  accesskey="&helpCmd.accesskey;" />
</menubar>

```

Ici, l'élément du menu Aide n'a aucun contenu. Les items du menu proviennent de l'overlay car leurs attributs id correspondent. Nous pouvons importer l'overlay dans d'autres fenêtres et n'avoir défini le contenu du menu Aide qu'à un seul endroit. Nous devons aussi ajouter quelques lignes au fichier d'entités DTD :

```

<!ENTITY helpCmd.label "Aide">
<!ENTITY helpCmd.accesskey "a">
<!ENTITY contentsCmd.label "Contenus">
<!ENTITY indexCmd.label "Index">
<!ENTITY aboutCmd.label "A propos de...">
<!ENTITY contentsCmd.accesskey "c">
<!ENTITY indexCmd.accesskey "i">
<!ENTITY aboutCmd.accesskey "a">
<!ENTITY findfilehelpCmd.label "Aide sur la recherche de fichiers">
<!ENTITY findfilehelpCmd.accesskey "f">

```

Nous utiliserons les deux dernières entités dans un moment.



Nous pouvons réduire encore la quantité de code dans la fenêtre en mettant les attributs du menu Aide directement dans l'overlay (label et accesskey dans cet exemple). Ces attributs seront hérités par l'élément. Si l'élément et celui de la fenêtre partagent le même attribut, la valeur dans l'overlay aura priorité sur celle de l'élément.

Changeons notre menu Aide de cette manière :

**findfile.xul :**

```
<menu id="help-menu"/>
```

**helpoverlay.xul :**

```

<menu id="help-menu" label="&helpCmd.label;"
  accesskey="&helpCmd.accesskey;">

```

Si la fenêtre XUL et l'overlay ont du contenu, celui de la fenêtre sera utilisé comme tel et celui de l'overlay sera ajouté à la fin. L'exemple suivant en est une illustration :

Source :

**stopandgo.xul :**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<?xml-stylesheet href="chrome://global/skin/global.css" type="text/css"?>

<window title="Marche - Arrêt" id="test-window"
      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

  <?xul-overlay href="toverlay.xul"?>

  <box id="singlebox">
    <button id="gobutton" label="Marche"/>
    <button id="stopbutton" label="Arrêt"/>
  </box>

</window>
```

**toverlay.xul :**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<overlay id="toverlay"
      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

  <box id="singlebox">
    <button id="backbutton" label="Précédent"/>
    <button id="forwardbutton" label="Suivant"/>
  </box>

</overlay>
```



Dans cet exemple, la boîte avec l'identifiant *singlebox* dispose de son propre contenu. Les éléments sont combinés et les deux boutons de l'overlay sont ajoutés à la fin de cette boîte.

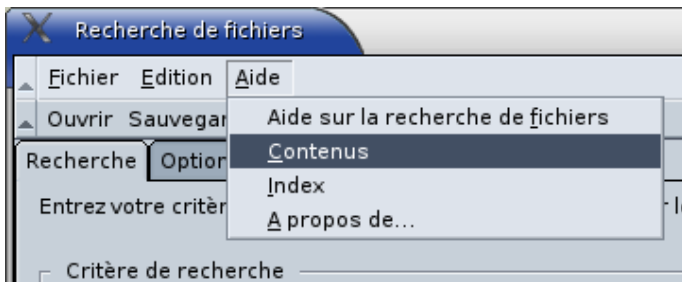
Nous pouvons aussi utiliser cette technique dans notre boîte de dialogue de recherche de fichiers.

findfile.xul :

```
<menu id="help-menu">
  <menupopup id="help-popup">
    <menuitem id="help-findfiles" label="&findfilehelpCmd.label;"
      accesskey="&findfilehelpCmd.accesskey;"/>
  </menupopup>
</menu>
</menubar>
```

L'attribut `id` de l'élément `menupopup` correspond à celui de l'overlay. Il en résulte l'ajout des items dans le même menu déroulant. Les overlays vont réunir les items de même `id` même s'ils sont situés à l'intérieur d'autres éléments.





## Placement des éléments d'un overlay

Dans l'exemple précédent, nous aurions voulu que les items du menu de l'overlay soient placés au début du menu plutôt qu'à la fin. XUL offre un mécanisme permettant de ne pas seulement placer les éléments en tête, mais d'en placer certains en premier et le reste à la fin (ou n'importe où entre). Il vous permet de positionner exactement où vous le souhaitez les overlays de menus, barres d'outils et autres éléments graphiques.

Pour cela, utilisez l'attribut `insertbefore` sur les éléments `menuitem`. Leurs valeurs doivent être l'identifiant `id` de l'élément avant lequel insérer les éléments. De même, vous pouvez utiliser l'attribut `insertafter` pour indiquer quels éléments insérer après. Seul l'élément contenant ces attributs est affecté. Si un élément est marqué par un attribut `insertbefore`, les autres continueront à être placés à la fin. Si vous voulez que tous les éléments soient placés avant, vous devez mettre l'attribut `insertbefore` sur tous les éléments.

De plus, vous pouvez utiliser l'attribut `position` pour spécifier une position d'index, numérotée à partir de 1.

Disons que nous souhaitons que les items de menu *Contenus* et *Index* de l'exemple précédent apparaissent avant l'item *Aide sur la recherche de fichiers*, et que l'item *À propos de* apparaisse ensuite. Il suffit d'ajouter l'attribut `insertbefore` sur les deux items du menu *Contenus* et *Index*. Pour être complet, vous devriez aussi ajouter un attribut `insertafter` sur le menu *A propos de*, mais ce n'est pas nécessaire car le menu apparaîtra par défaut à la fin.

Dans cet exemple ci dessus, l'identifiant `id` de l'item du menu est *help-findfiles*. De ce fait, nous devons associer les attributs `insertbefore` à cet `id`. voici les changements :

```
<menupopup id="help-popup">
  <menuitem id="help-contents" label="Contenus" insertbefore="help-findfiles"/>
  <menuitem id="help-index" label="Index" insertbefore="help-findfiles"/>
  <menuitem id="help-about" label="A propos de..." />
</menupopup>
```

Maintenant, lorsqu'une fenêtre utilisant l'overlay Aide (comme notre exemple de recherche de fichiers) est ouverte, les événements suivants se produisent :

1. Pour tous les éléments de l'overlay, qui sont tous les enfants de l'élément `overlay`, un élément de la fenêtre principale est recherché avec le même identifiant `id`. S'il n'est pas trouvé, cet élément est ignoré. Dans cet exemple, les éléments avec les identifiants *help-menu* et *help-popup* sont trouvés.
2. Si un élément est trouvé, les attributs de l'élément de l'overlay lui sont ajoutés.
3. Les enfants de l'élément `overlay`, ici chaque élément `menuitem`, sont insérés comme enfants de l'élément de la fenêtre.
  - ♦ Si l'élément de l'overlay contient un attribut `insertafter`, il est ajouté juste après celui de la fenêtre principale ayant le même attribut `id`.
  - ♦ Si l'élément de l'overlay contient un attribut `insertbefore`, il est ajouté juste avant celui de la fenêtre principale ayant le même attribut `id`.

- ◆ Si l'élément de l'overlay contient un attribut `position`, il est ajouté à son emplacement indexé par la valeur de l'attribut compté à partir de `1`.
- ◆ Dans les autres cas, l'élément est ajouté comme le dernier enfant.

En réalité, `insertbefore` et `insertafter` peuvent contenir des listes de valeurs séparées par des virgules, où le premier identifiant trouvé dans la liste est utilisé pour déterminer la position dans la fenêtre.

---

Nous allons voir ensuite comment appliquer des overlays à des fenêtres dans différents paquetages.

## 12.7 Overlays inter-paquetage

Écrit par Neil Deakin. Traduit par *Alain B.* (25/07/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/crosspov.html>

Cette section décrit comment appliquer des overlays à des fichiers qui ne les importent pas.

### Application d'overlays à d'autres paquetages

Les overlays ont d'autres fonctionnalités très utiles. Dans les exemples de la section précédente, les overlays étaient importés par la fenêtre. Vous pouvez aussi utiliser une autre méthode en indiquant aux overlays pour quelles fenêtres ils seront appliqués. Il vous suffit de modifier le fichier *contents.rdf* de votre paquetage. Cette méthode est très utile car elle permet à un overlay de modifier l'interface utilisateur d'un autre paquetage sans pour cela modifier celui-ci. Par exemple, vous pouvez ajouter des items de menu ou des barres d'outils à la fenêtre du navigateur Mozilla.

Nous utiliserons cette fonctionnalité pour ajouter une barre d'outils dans la fenêtre du navigateur Mozilla. Le client courrier de Mozilla utilise les overlays pour ajouter du contenu à la fenêtre du navigateur. Par exemple, si le client n'est pas installé, il n'y a pas de commande pour de nouveaux messages. Toutefois, si le client est installé, un overlay sera appliqué au menu pour ajouter une commande de nouveaux messages. Ci dessous, nous ajouterons une barre d'outils de recherche de fichiers au navigateur. Cette fonctionnalité n'a aucune utilité, mais nous l'intégreront quand même.

Mozilla vous permet d'ajouter une liste d'overlays dans le fichier *contents.rdf* utilisé pour spécifier les paquetages chrome, les thèmes graphiques et les localisations. Dès que vous avez créé un overlay, vous pouvez l'ajouter au fichier *contents.rdf*. Il vous suffit d'ajouter les informations pour chacune des fenêtres où vous voulez appliquer l'overlay.

Tout d'abord, créons un simple overlay. Il contiendra quelques champs de saisie pour la recherche d'un nom de fichier et d'un répertoire. Chargez le fichier *foverlay.xul* et ajoutez le au répertoire de notre exemple de recherche de fichiers à côté de *findfile.xul*.

Exemple 12.7.1 :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<overlay
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<toolbox id="navigator-toolbox">
  <toolbar id="findfile_toolbar">
    <label control="findfile_filename" value="Recherche des fichiers nommés:"/>
    <textbox id="findfile_filename"/>
    <label control="findfile_dir" value="Répertoire:"/>
    <textbox id="findfile_dir"/>
    <button label="Parcourir..."/>
  </toolbar>
</toolbox>
```

```
</toolbox>
```

```
</overlay>
```

Vous pouvez visualiser cet exemple en changeant l'élément overlay par window. La seule chose un peu spécifique est l'attribut `id` utilisé sur l'élément toolbox. Cette valeur (*navigator-toolbox*) est la même que l'identifiant de la boîte d'outils de la fenêtre du navigateur (*navigator.xul*). Elle signifie que cet overlay sera appliqué à la boîte d'outils du navigateur et que son contenu sera ajouté comme une barre d'outils supplémentaire.

Pour ajouter cet overlay au fichier manifest, vous devez ajouter deux ressources. Premièrement, nous en ajoutons une pour chaque fenêtre où l'overlay sera appliqué. Le code suivant doit être placé dans le fichier *contents.rdf* juste avant la balise fermante de RDF.

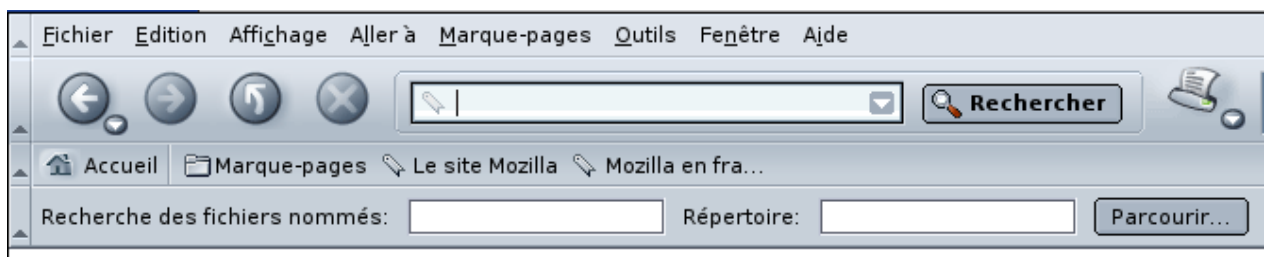
```
<RDF:Seq RDF:about="urn:mozilla:overlays">
  <RDF:li RDF:resource="chrome://navigator/content/navigator.xul"/>
</RDF:Seq>
```

Cette déclaration précise l'ajout d'une fenêtre overlay au système de gestion des overlays (*urn:mozilla:overlays*). Vous pouvez ajouter des noeuds supplémentaires pour chaque fenêtre à modifier en ajoutant des balises `li` supplémentaires.

Ensuite, nous ajoutons un noeud pour chaque overlay s'appliquant à la fenêtre. Dans ce cas, vous n'en avez qu'un, mais d'autres peuvent être appliqués. Ajoutez ces lignes à la suite des précédentes :

```
<RDF:Seq RDF:about="chrome://navigator/content/navigator.xul">
  <RDF:li>chrome://findfile/content/foverlay.xul</RDF:li>
</RDF:Seq>
```

Mozilla lit cette information et construit une liste d'overlays appliqués à d'autres fenêtres. Il enregistre cette information dans un répertoire *chrome/overlayinfo*. Il n'est pas nécessaire que vous modifiez manuellement les fichiers de ce répertoire. Ils sont générés automatiquement et modifiés au premier lancement de Mozilla ou lorsque de nouveaux paquets sont installés. Toutefois, vous pouvez forcer la reconstruction de leurs données en effaçant ce répertoire et le fichier *chrome.rdf*.



Note complémentaire : vous pouvez utiliser cette même technique pour appliquer des feuilles de styles supplémentaires. L'exemple suivant vous montre comment :

```
<RDF:Seq RDF:about="urn:mozilla:stylesheets">
  <RDF:li RDF:resource="chrome://messenger/content/messenger.xul"/>
</RDF:Seq>
```

```
<RDF:Seq RDF:about="chrome://messenger/content/messenger.xul">
  <RDF:li>chrome://blueswayedshoes/skin/myskinfile.css</RDF:li>
</RDF:Seq>
```

Nous verrons ensuite comment créer un installeur pour une application XUL.

# 13. Installation

## 13.1 Création d'un programme d'installation

Écrit par Neil Deakin. Traduit par *Alain B.* (01/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xpinstall.html>

NdT : Attention, cette section décrit le mécanisme XPInstall propre à la suite Mozilla et à des versions anciennes de Mozilla Firefox. Pour les versions récentes de Mozilla Firefox, ce mode d'installation n'est plus le même, mais il n'est pas encore décrit dans ce tutoriel. Voir [comment faire des extensions pour firefox](#) sur xulfr.org.

Cette section va décrire le packaging d'une application XUL dans un programme d'installation.

### Paquetages d'installation XPI

Mozilla propose un mécanisme qui peut être utilisé pour emballer des fenêtres XUL, des scripts, des thèmes graphiques et d'autres fichiers dans un seul installateur. Il suffit de placer le fichier d'installation quelque part où les utilisateurs pourront le télécharger. Un simple script peut être utilisé pour assurer le téléchargement et l'installation du paquetage. Ce mécanisme est appelé XPInstall (Cross Platform Install).

Les installateurs XPI sont emballés dans des fichiers JAR. À l'intérieur d'un fichier JAR, vous pouvez ajouter toutes sortes de fichiers que vous voulez voir installés. De plus, les installateurs doivent contenir un script (un fichier nommé `install.js`) qui décrit le processus d'installation. Ce script a accès à quelques fonctions d'installation qui peuvent être employées pour installer des fichiers et des composants.

Les fichiers d'installation JAR ont typiquement l'extension `.xpi` (prononcez zippy) pour les distinguer des autres fichiers d'archives. Les installateurs seront habituellement utilisés pour installer des composants Mozilla tels que des thèmes graphiques, des extensions et d'autres paquetages.

Il y a plusieurs étapes pour démarrer et installer les composants. Elles sont décrites pas à pas ci dessous :

1. Créer une page Web à partir de laquelle l'utilisateur peut charger l'application à installer. La page doit contenir un déclencheur d'installation qui est un petit script lançant l'installation.
2. Une boîte de dialogue présente à l'utilisateur le paquetage qui doit être installé. Il est possible pour le déclencheur de lancer de multiples installations. Dans ce cas, elles seront présentées dans une liste. L'utilisateur doit choisir de continuer ou d'annuler.
3. Si l'utilisateur choisit de continuer, le fichier d'installation XPI est téléchargé. Une barre de progression est affichée durant le processus.
4. Le fichier `install.js` est extrait de l'archive et exécuté. Le script va appeler les fonctions d'installation qui indiqueront quels fichiers de l'archive doivent être installés.
5. Une fois le script terminé, le nouveau paquetage a été installé. Si de multiples paquetages doivent être installés, leurs scripts se lanceront l'un après l'autre.

### Déclencheurs d'installation

Comme indiqué ci-dessus, le processus d'installation est lancé par un déclencheur d'installation. Il requiert l'utilisation de l'objet global spécial `InstallTrigger`. Il contient un certain nombre de méthodes qui peuvent être utilisées pour démarrer une installation. Vous pouvez utiliser cet objet dans un contenu local ou à distance, ce qui signifie qu'il est adapté pour un téléchargement à partir d'un site Web.

Créons un exemple de déclencheur d'installation. Il nécessite l'utilisation de la fonction `InstallTrigger.install`. Cette fonction a deux arguments, le premier est la liste des paquetages à

installer, et le second est la référence à une fonction de rappel qui sera appelée lorsque l'installation est terminée. Voici un exemple :

```
function doneFn ( name , result ){
    alert("Le paquetage" + name + " a été installé avec un résultat de " + result);
}

var xpi = new Object();
xpi["Calendar"] = "calendar.xpi";
InstallTrigger.install(xpi,doneFn);
```

Premièrement, nous définissons la fonction `doneFn` qui sera appelée lorsque l'installation est terminée. Bien entendu, vous pouvez nommer cette fonction comme vous le souhaitez. Cette fonction a deux arguments. Le premier argument est le nom du paquetage qui vient juste d'être installé. Celui ci est important si vous installez de multiples composants. Le second argument est un code de résultat. Un code de valeur `0` signifie que l'installation s'est terminée avec succès. Si le code de résultat n'est pas nul, une erreur a eu lieu et la valeur représente un code d'erreur. Ici, la fonction `doneFn` affiche simplement une boîte d'alerte à l'utilisateur.

Ensuite, nous créons un tableau `xpi` qui contient le nom (*Calendar*) et l'URL (*calendar.xpi*) du programme d'installation. Vous pouvez ajouter une ligne similaire pour chaque paquetage que vous souhaitez installer. Finalement, nous appelons la fonction d'installation.

Lorsque cette portion de script sera exécutée, le fichier `calendar.xpi` sera installé.

Essayons ce script avec notre exemple de recherche de fichiers.

```
function doneFn ( name , result ){
    if (result) alert("L'erreur suivante a eu lieu:" + result);
}

var xpi = new Object();
xpi["Find Files"] = "findfile.xpi";
InstallTrigger.install(xpi,doneFn);
```

## L'archive XPI

Le fichier d'installation XPI doit obligatoirement contenir au minimum un fichier appelé `install.js` qui est un fichier javascript exécuté lors de l'installation. Les autres fichiers sont les fichiers à installer. Ces derniers sont typiquement placés dans des répertoires de l'archive mais ils n'ont pas lieu de l'être. Pour des fichiers chrome, ils devraient être structurés comme le répertoire chrome.

Souvent, les seuls fichiers trouvés dans une archive XPI sont le script d'installation (`install.js`) et un fichier JAR. Ce fichier JAR contient tous les fichiers utilisés par votre application. Les composants de Mozilla sont installés de cette manière.

Parce que les fichier XPI ne sont rien d'autres que des fichiers ZIP, vous pouvez les créer en utilisant un utilitaire zip (NdT : les fichiers JAR sont également des fichiers ZIP).

Pour notre exemple de recherche de fichiers, nous créerons une structure dans l'archive comme ce qui suit :

```
install.js
findfile
  content
    contents.rdf
    findfile.xul
    findfile.js
  skin
```

```

contents.rdf
findfile.css
locale
contents.rdf
findfile.dtd

```

Un répertoire a été ajouté pour chaque partie du paquetage, pour le contenu, pour le thème graphique et pour la localisation. Des fichiers *contents.rdf* ont également été ajoutés car ils sont nécessaires pour l'enregistrement des fichiers chrome.

---

Dans la section suivante, nous aborderons le script d'installation.

## 13.2 Les scripts d'installation

Écrit par Neil Deakin. Traduit par *Alain B.* (14/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xpiscrpt.html>

NdT : Attention, cette section décrit le mécanisme XPInstall propre à la suite Mozilla et à des versions anciennes de Mozilla Firefox. Pour les versions récentes de Mozilla Firefox, ce mode d'installation n'est plus le même, mais il n'est pas encore décrit dans ce tutoriel. Voir [comment faire des extensions pour firefox](#) sur xulfr.org.

Cette section décrit le script d'installation.

### Création d'un script d'installation

Vous voulez généralement avoir une forme de contrôle sur vos processus d'installation. Par exemple, vous pouvez souhaiter vérifier les versions des fichiers existants et n'installer que des mises à jour, ou peut être souhaiteriez-vous simplement appliquer des corrections. Le script d'installation est même assez souple pour vous permettre de désinstaller des fichiers. Pour ces raisons, les programmes d'installation incluent un script propre à cette tâche.

Le script d'installation doit s'appeler *install.js* et doit être placé à la racine de l'archive de l'installateur. Ce script contient du code JavaScript qui appelle un certain nombre de fonctions d'installation.

Dans un document HTML ou un document XUL, l'objet window est l'objet global racine. Il signifie que vous pouvez appeler les méthodes de l'objet window sans les faire précéder de leur qualificateur, ainsi `window.open(...)` peut simplement être écrit `open(...)`. Dans un script d'installation, il n'y a pas de fenêtre associée, toutefois l'objet global sera l'objet `Install` qui contient un certain nombre de fonctions pour personnaliser le processus d'installation. Certaines fonctions de l'objet global `Install` seront décrites ci dessous.

Le script d'installation doit suivre les étapes suivantes :

1. Initialiser l'installation en précisant le paquetage et sa version.
2. Utiliser les fonctions d'installation pour spécifier les fichiers et les répertoires qui doivent être installés. Vous pouvez aussi spécifier les fichiers à déplacer ou à effacer.
3. Démarrer le processus qui installe les fichiers nécessaires.

Il est important de signaler que pendant l'étape numéro deux, vous n'indiquez seulement quels sont les fichiers qui seront installés et quelles autres opérations vous souhaitez réaliser. Aucun fichier ne sera copié avant l'étape trois. En procédant de la sorte, vous pouvez facilement définir plusieurs fichiers à installer, et en cas d'erreurs, vous pouvez annuler tout le processus d'installation sans modifier le système de l'utilisateur.

## Le registre d'extensions

Mozilla tient à jour un fichier qui est un registre de toutes les extensions actuellement installées. Les extensions incluent les nouveaux paquetages chrome, les thèmes graphiques et les plugins. Lorsqu'une nouvelle extension est installée, le registre est mis à jour. Le registre stocke aussi l'ensemble des informations des fichiers et de leurs versions sur les extensions installées. De cette manière, il est aisé de vérifier si une version de votre extension est déjà présente et de la mettre à jour seulement si nécessaire.

Le registre d'extensions fonctionne presque comme la base de registre de Windows. Il consiste en une série hiérarchisée de clefs et de valeurs. Vous n'avez pas besoin d'en savoir plus à son sujet pour créer des applications XUL à moins que vous ne créiez vos propres composants XPCOM.

Ce que vous devez savoir pour une installation est que le registre stocke une série d'informations sur votre application, tels que la liste des fichiers et leurs versions. Toutes ces informations sont stockées dans une clef (et à l'intérieur, des sous clefs) que vous fournissez dans le script d'installation (dans l'étape 1 mentionnée ci dessus).

Cette clef est structurée comme une arborescence de répertoire comme ceci :

```
/Auteur/Nom du Paquetage
```

Remplacez le mot *Auteur* par votre nom et remplacez le *Nom du Paquetage* avec le nom de votre paquetage que vous installez. Par exemple :

```
/Xulplanet/Find Files
```

```
/Netscape/Personal Security Manager
```

Le premier exemple est celui utilisé pour notre exemple de boîte de dialogue de recherche de fichiers. Le second est la clef utilisée pour le gestionnaire de données privées.

## Initialisation de l'installation

L'objet `Install` a une fonction, `initInstall`, qui sert à initialiser l'installation. Elle doit être appelée au lancement de votre script d'installation. La syntaxe de cette fonction est la suivante :

```
initInstall( packageName , regPackage , version );
```

### **Exemple :**

```
initInstall("Find Files", "/Xulplanet/Find Files", "0.5.0.0");
```

Le premier argument est le nom du paquetage sous une forme lisible pour l'utilisateur. Le second argument est la clef du registre utilisée pour mémoriser l'information du paquetage comme décrit ci avant. Le troisième argument est la version du paquetage à installer.

Ensuite, nous devons indiquer le répertoire où seront installés les fichiers. Il y a deux façons de le faire. La méthode simple est d'assigner un répertoire d'installation et d'y copier tous les fichiers. La seconde méthode vous permet d'assigner une destination à chaque fichier (ou répertoire). La première méthode est décrite ci dessous.

La fonction `setPackageFolder` assigne un répertoire d'installation. Pour l'exemple de recherche de fichiers, nous installerons les fichiers dans le répertoire chrome (nous pourrions aussi bien les mettre autre part). Cette fonction `setPackageFolder` ne requiert qu'un argument, le répertoire d'installation. Pour une compatibilité maximale, vous ne devez pas spécifier un répertoire absolu. Au lieu de cela, vous utiliserez un identifiant d'un répertoire connu et pointerez sur un de ses sous répertoires. Ainsi, si votre application a

besoin d'installer quelques librairies systèmes, vous n'avez pas besoin de connaître le nom de ces répertoires.

Les identifiants de sélection de répertoires sont expliqués sur la page de [référence](#). Pour le répertoire chrome, l'identifiant est **Chrome**. La fonction `getFolder` peut être utilisée pour récupérer un de ces répertoires spéciaux. Cette fonction prend deux arguments, le premier étant l'identifiant et le second étant un sous répertoire. Par exemple :

```
findDir = getFolder("Chrome", "findfile");
setPackageFolder(findDir);
```

Ici, nous récupérons l'emplacement du sous répertoire *findfile* dans répertoire Chrome et nous le passons directement à la fonction `setPackageFolder`. Le second argument de la fonction `getFolder` est le sous répertoire qui servira à l'installation de l'exemple et qui n'a pas besoin d'avoir été créé d'abord. Vous pouvez ignorer cet argument si vous n'en avez pas besoin.

## Marquage des fichiers d'installation

Ensuite, vous devez indiquer quels seront les fichiers à installer. Deux fonctions doivent être employées pour cela, `addDirectory` et `addFile`. La fonction `addDirectory` précise à l'installateur un répertoire de l'archive XPI (et tout son contenu) qui devra être installé à un emplacement particulier. La fonction `addFile` est similaire mais seulement pour un fichier.

Les deux fonctions `addDirectory` et `addFile` ont plusieurs paramètres. Le plus simple ne prend qu'un seul argument qui est le répertoire servant à l'installation.

```
addDirectory ( dir );
addFile ( dir );
```

### Exemple :

```
addDirectory("findfile");
```

L'exemple ci dessus spécifie que le répertoire *findfile* de l'archive d'installation est à installer. Nous pouvons appeler ces fonctions autant de fois que nécessaire pour les autres fichiers.

Ensuite, nous voulons enregistrer les fichiers de notre exemple dans le registre chrome afin de pouvoir les appeler par une URL chrome. La fonction `registerChrome` est utilisée pour cela. Elle prend deux arguments, le premier étant le type d'enregistrement chrome (*content* pour du contenu, *skin* pour du thème graphique, ou *locale* pour la localisation), le second pointant vers l'emplacement du fichier manifest *contents.rdf* à enregistrer. Comme notre exemple de recherche de fichiers contient les trois types, la fonction `registerChrome` devra être appelée trois fois.

```
registerChrome(Install.CONTENT | Install.DELAYED_CHROME, getFolder(findDir, "content"));
registerChrome(Install.SKIN | Install.DELAYED_CHROME, getFolder(findDir, "skin"));
registerChrome(Install.LOCALE | Install.DELAYED_CHROME, getFolder(findDir, "locale"));
```

L'indicateur `DELAYED_CHROME` sert à indiquer que le chrome devra être installé au prochain lancement de Mozilla.

## Finalisation de l'installation

Les fonctions `addDirectory` et `addFile` ne copient aucun fichier. Elles ne servent qu'à pointer quels fichiers devront être installés. De la même manière, la fonction `registerChrome` ne fait que pointer quel chrome devra être enregistré. Pour achever le processus et commencer la copie des fichiers, appelez la fonction `performInstall` sans argument.



Le script final pour installer notre exemple de recherche de fichiers est le suivant :

Exemple 13.2.1 :

```
initInstall("Find Files", "/Xulplanet/Find Files", "0.5.0.0");

findDir = getFolder("Chrome", "findfile");
setPackageFolder(findDir);

addDirectory("findfile");

registerChrome(Install.CONTENT | Install.DELAYED_CHROME, getFolder(findDir, "content"));
registerChrome(Install.SKIN | Install.DELAYED_CHROME, getFolder(findDir, "skin"));
registerChrome(Install.LOCALE | Install.DELAYED_CHROME, getFolder(findDir, "locale"));

performInstall();
```

---

Dans la section suivantes, nous verrons quelques fonctions supplémentaires pour l'installation.

## 13.3 Fonctions additionnelles d'installation

Écrit par Neil Deakin. Traduit par *Alain B.* (17/03/2004).

Page originale : <http://www.xulplanet.com/tutorials/xultu/xpiadv.html>

NdT : Attention, cette section décrit le mécanisme XPInstall propre à la suite Mozilla et à des versions anciennes de Mozilla Firefox. Pour les versions récentes de Mozilla Firefox, ce mode d'installation n'est plus le même, mais il n'est pas encore décrit dans ce tutoriel. Voir [comment faire des extensions pour firefox](#) sur xulfr.org.

Cette section décrit quelques spécificités supplémentaires des programmes d'installation.

### Manipulation de fichiers lors de l'installation

La section précédente décrivait un programme d'installation simple. Vous pouvez souhaiter réaliser quelques opérations plus élaborées pendant l'installation. Par exemple, vous voulez installer un paquetage seulement si certaines conditions sont réunies, comme d'avoir une librairie particulière installée.

En complément de l'objet `Install`, l'objet `File` est également disponible pendant le script d'installation. Il fournit quelques fonctions qui peuvent être employées pour examiner et modifier des fichiers sur le disque. Vous pouvez les utiliser pour déplacer, copier ou effacer des fichiers avant ou après que les fichiers du paquetage soient installés. Par exemple, vous voulez peut être faire une sauvegarde de quelques fichiers d'abord.

Le code ci dessous fera une copie du fichier `"/bin/grep"` dans le répertoire `"/main"`.

```
var binFolder=getFolder("file:///","bin");
var grep=getFolder(binFolder,"grep");

var mainFolder=getFolder("file:///","main");

File.copy(grep,mainFolder);
```

La première ligne va renseigner un pointeur sur le répertoire `/bin`. Le texte `file:///` est une chaîne de caractères spéciale qui correspond à la racine du système de fichiers. À partir de ce pointeur, nous récupérons le fichier `grep` qui se trouve dans le répertoire `bin`. Si ce fichier n'existe pas, une erreur sera signalée au processus d'installation. Ensuite, nous pointons sur le répertoire `main`, toujours à partir de la racine du système de fichiers. Finalement, nous appelons la fonction `File.copy` qui copie le fichier source

vers sa destination.

Des fonctions existent également pour déplacer, renommer et exécuter des fichiers. Ainsi, vous pouvez déplacer des fichiers qui peuvent se trouver en conflit avec votre paquetage.

## Interception des erreurs

Vous voulez certainement intercepter d'éventuelles erreurs proprement. Elles peuvent se produire si un fichier ou un répertoire ne peut pas être trouvé, si la capacité du disque n'est pas suffisante ou pour toutes autres raisons.

Il vous suffit d'appeler la fonction `getLastError` pour déterminer si une erreur a été rencontrée. Si elle renvoie **SUCCESS**, aucune erreur ne s'est produite. Autrement, elle renvoie un nombre qui indique le code d'erreur. Vous pouvez appeler cette fonction en tout point de votre script d'installation pour déterminer si une erreur est survenue lors de la dernière opération effectuée.

Si une erreur se produit, vous voulez sûrement interrompre l'installation. Vous pouvez également vouloir afficher un message d'erreur pour l'utilisateur. Par exemple, vous pourriez mettre le script suivant à la fin de votre script d'installation :

```
if (getLastError() == SUCCESS){
    performInstall();
}
else {
    cancelInstall();
}
```

Les codes d'erreur susceptibles d'être renvoyés par la fonction `getLastError` sont listés dans le fichier source de Mozilla [nsInstall.h](#). Pendant l'installation, un suivi d'événements contenant les opérations réalisées est créé. Il contiendra également toutes les erreurs qui se sont produites. Ces événements peuvent être trouvés dans le fichier *install.log* dans le répertoire d'installation de Mozilla. Un bloc de texte sera ajouté à ce fichier à chaque installation effectuée.

La fonction `logComment` peut être utilisée pour écrire un texte dans ce fichier d'événements. Elle nécessite un seul argument qui est le contenu du texte.

---

Le Tutoriel XUL est terminé.