Etendre l'interface de Mozilla avec XPFE

A fin d'assurer la portabilité de Mozilla, il était important de créer un *framework* indépendant du système d'exploitation. C'est le rôle rempli par XPFE. XPFE est le résultat de l'association entre XML, Javascript et CSS. De la même manière, nous pouvons voir le DHTML comme l'association entre HTML, JavaScript et CSS.

Si vous voulez ajouter une application dans Mozilla, inutile de devenir un cador du C, C++ ou Java. Dans cet article, nous allons voir qu'avec un peu de XML, de JavaScript et quelques CSS, il est très facile d'étendre l'interface de Mozilla.

Le XML est ici utilisé comme langage de définition d'interface ; il s'agit en fait de XUL (*XML User-interface Language*), JavaScript assure le côté "dynamique". Quant aux CSS, elles vont permettre de donner du style ;).

La création d'une application se fait donc aussi simplement que l'écriture d'une page Web standard. Le fichier XUL, servant de point d'entrée, sera interprété par le moteur de rendu : Gecko. Cela apporte tout un tas de possibilités, allant de la simple modification de l'interface de Mozilla jusqu'à la création d'applications "autonomes".

Bien entendu, nous pouvons être amenés à interagir avec l'environnement. Pour cela, nous allons utiliser les composants XPCOM (*Cross-Platform Component Object Model*) de Mozilla. Et pour que tout soit on ne peut plus simple, nous utiliserons cela dans du JavaScript, XPCOM bénéficie d'un "wrapper" pour ce dernier.

Dans la suite de cet article, je ne rentrerai pas dans les détails de XPCOM. Cela serait beaucoup trop long. Sachez cependant qu'il existe une abondante documentation sur le sujet [1][2]. Lorsque j'en aurai besoin, je vous fournirai les scripts nécessaires avec un minimum d'explications. Sinon, patientez, nous y reviendrons dans un prochain article.

Préparation de l'environnement

Nous allons créer une application complète pour Mozilla. Pour cela, je vous présenterai brièvement `XPInstall` qui permet de "packager" l'application en vue de permettre son installation sur n'importe quelle plateforme. Les applications écrites pour Mozilla sont placées dans le répertoire `chrome` (soit dans `/usr/lib/mozilla/chrome` pour moi qui utilise une Debian).

Il est très fastidieux et risqué de travailler directement dans ce répertoire. Nous allons donc commencer par paramétrer Apache de façon à pouvoir "exécuter" des applications depuis un serveur Web distant. Pour cela, ajoutez la ligne suivante dans votre fichier `httpd.conf` :

```
AddType application/vnd.mozilla.xul+xml .xul
```

(Si vous préférez, vous pouvez tout aussi bien ajouter cette ligne dans un fichier `.htaccess`).

Premier exemple

L'avantage de créer un premier exemple de type "Hello World" c'est que nous allons pouvoir aborder la création d'une "application" dans sa globalité. En effet, nous avons vu que XPFE est basé sur XML, JavaScript et CSS. Nous allons donc créer trois fichiers :

exemple1.xul :

```
1  <?xml version="1.0"?>
2  <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
3  <?xml-stylesheet href="exemple1.css" type="text/css"?>
4
5  <!DOCTYPE window [
6    <!ENTITY exemple:1:hello  "Hello...">
7    <!ENTITY exemple:1:world  "...World !">
8  ]>
9
10 <window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
   there.is.only.xul">
11
12  <script type="application/x-javascript" src="./exemple1.js"/>
13
14  <box align="center">
15   <button class="bnw" label="&exemple:1:hello;" onclick="butAction(
      '&exemple:1:world;' );" />
16  </box>
17
18 </window>
```

exemple1.css :

```
1  .bnw {
2      background-color: Red;
```

```
3       color: Yellow;
4   }
```

exemple1.js :

```
1   function butAction( txt ) {
2     alert( txt );
3   }
```

Dans `exemple1.xul`, nous créons une simple fenêtre dans laquelle nous plaçons un bouton. Un clic sur ce bouton appelle la fonction "`butAction`" définie dans `exemple1.js`.

Nous appliquons le style "`bnw`" à notre bouton. Ce style est défini dans `exemple1.css` (changé à la ligne 3 de `exemple1.xul`). Notez que nous avons utilisé une autre feuille de style (ligne 2). Cette feuille définit en fait le style par défaut. Vous pouvez ne pas l'utiliser, mais attendez-vous à des résultats "étranges". Pas exemple, si vous supprimez la ligne 2 dans `exemple1.xul`, toute la fenêtre sera en jaune sur rouge !

Dernière chose avant de clore le chapitre "Hello World". Dans notre exemple, nous avons défini des entités (`exemple:1:hello` et `exemple:1:world`). Ces définitions propres à une DTD peuvent être externalisées (dans un fichier `exemple1.dtd` par exemple et que l'on chargerait de la façon suivante) :

```
<!DOCTYPE window SYSTEM "http://localhost/exemple1.dtd">
```

XUL...

XUL, permettant de définir l'interface des applications, met à notre disposition tout un ensemble de *widgets*. Nous allons voir certains de ces widgets au travers de la création d'une petite application (que nous appellerons `freshzilla`), permettant de rechercher des projets sur le site Freshmeat.net.

Avant de commencer, il faut savoir que chaque élément (fenêtre, bouton, liste...) hérite des propriétés d'un élément XUL générique. Cet élément possède des attributs que nous retrouverons pour chacun de ses héritiers. Voici la liste (non exhaustive) des attributs les plus importants :

- `id` : permet d'attribuer un identifiant unique à l'élément ;
- `style` : permet de préciser le style (défini dans un CSS) ;
- `width` et `height` : positionnent la largeur et la hauteur de l'élément. Associés à ces attributs nous avons `minwidth`, `minheight`, `maxwidth` et `maxheight` ;
- `hidden` : précise si l'élément est visible (`false`) ou pas (`true`) ;
- `orient` : indique l'orientation (horizontal ou vertical).

De plus, nous ajoutons aux éléments racines l'attribut `xmlns` avec la valeur "`http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`". Ceci permet de déclarer l'espace de nommage de XUL.

...et consorts

En plus des fichiers XUL, nous avons vu que nous avons besoin de fichiers JS, DTD et CSS. Vous avez (j'espère) bien compris à quoi ils servent. Cependant, il y a un cinquième type de fichier que nous n'avons pas encore évoqué : les fichiers RDF.

RDF signifie *Resource Description Framework* ou, en français, Cadre de Description de Ressource. Bref, avec ça on n'est pas plus avancé et on ne voit toujours pas le rapport avec le reste. Disons simplement que le fichier RDF sert à faire reconnaître notre application par Mozilla. Retenons donc cette phrase tirée du document de spécification du W3C [3] : "Le but général de RDF est de définir un mécanisme pour décrire les ressources sans faire d'hypothèses sur un domaine particulier d'application [...]". Surtout, restez calme ;)

Il s'agit maintenant d'organiser ces fichiers de façon à pouvoir par la suite packager notre application. Nous allons créer les répertoires suivants :

```
freshzilla
|-- content
|   `-- freshzilla
|-- skin
|   `-- freshzilla
`-- locale
    `-- en-US
        `-- freshzilla
```

Dans `freshzilla/content/freshzilla/` nous placerons l'ensemble de nos fichiers XUL et JS ainsi qu'un fichier `contents.rdf`. Dans `freshzilla/skin/freshzilla/` nous mettrons les CSS (et un `contents.rdf`). Dans `freshzilla/locale/en-US/freshzilla/` nous mettrons les fichiers DTD (et un `contents.rdf`). Vous l'aurez compris, vous pouvez ajouter autant de sous-répertoires à `freshzilla/locale/` que vous le souhaitez, si vous voulez distribuer l'application traduite en d'autres langues (fr-FR par exemple). En effet, l'utilisation de DTD nous permet d'internationaliser notre travail. Je laisse cela à votre initiative.

Notre application va se composer d'une fenêtre principale contenant :

- Un champ de saisie dans lequel on indiquera ce que l'on recherche ;
- Un bouton pour lancer la recherche ;
- Une liste pour l'affichage des résultats avec un menu "popup" permettant d'accéder à la page Freshmeat du projet, à sa *home page* ou de voir les informations relatives au projet trouvé ;
- Une barre de progression indiquant que la recherche est en cours.

Pour cela, nous allons donc créer les fichiers `freshzilla.xul` et `freshzilla.js`. Dans un esprit "synthétique", nous créerons aussi les fichiers `freshzilla.css` et `freshzilla.dtd` communs à l'ensemble de l'application.

Dans notre fenêtre principale nous ajouterons un bouton ouvrant une fenêtre "About". Cette dernière contiendra un simple champ texte et un bouton pour fermer la fenêtre et sera définie dans le fichier `about.xul`.

De plus, afin de véritablement intégrer notre application, nous ajouterons une entrée dans menu "Window" de Mozilla, permettant de placer la fenêtre principale dans la Sidebar. Nous allons donc devoir véritablement étendre l'interface et aborder le principe du recouvrement (*overlay*). Pour cela, nous aurons les fichiers `freshzillaOverlay.xul` et `addpanel.js`.

Au final nous aurons donc les fichiers suivants :

```
freshzilla
|-- content
|   `-- freshzilla
|       |-- about.xul
|       |-- addpanel.js
|       |-- contents.rdf
|       |-- freshzilla.js
|       |-- freshzilla.xul
|       |-- freshzillaOverlay.xul
|       `-- server.js
|-- locale
|   `-- en-US
|       `-- freshzilla
|           |-- contents.rdf
|           `-- freshzilla.dtd
`-- skin
    `-- freshzilla
        |-- contents.rdf
        `-- freshzilla.css
```

Les fenêtres

La création d'une fenêtre se fait en utilisant le widget `<window>`. Nous pouvons fixer un certain nombre de paramètres à notre fenêtre en utilisant des attributs. En plus de ceux que nous avons déjà cités, nous utiliserons :

- `title` : pour préciser le titre ;
- `class` : pour indiquer le style à utiliser.

Comme nous l'avons vu lors du premier exemple, `<window>` est un élément racine, nous n'oublierons donc pas de lui ajouter l'attribut `xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"`.

freshzilla.xul :

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!-
 3 Desc: Fenetre principale
 4 ->
 5
 6 <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
 7 <?xml-stylesheet href="chrome://freshzilla/skin/freshzilla.css"
type="text/css"?>
 8
 9 <!DOCTYPE window SYSTEM "chrome://freshzilla/locale/freshzilla.dtd">
10
11 <window
12
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
13     id="fzmain"
14     class="dialog"
15     title="&freshzilla:label;"
16     width="640"
17     height="480"
18     orient="horizontal">
...
58 </window>
```

Vous remarquerez que nous appliquons le style "`dialog`" à notre fenêtre. Ce style est défini dans la feuille de style globale (chargé ligne 6). Le libellé du titre est externalisé dans le fichier `freshzilla.dtd` (chargé à la ligne 9).

freshzilla.dtd :

```
1 <!ENTITY freshzilla:label  "FreshZilla">
2 <!ENTITY freshzilla:version  "0.0.1.0">
3 <!ENTITY freshzilla:project:name "Project name">
4 <!ENTITY freshzilla:project:desc "Description">
```

freshzilla.css :

```
1  .bold {
2    font-weight: bold;
3  }
4
5  .bold-and-white {
6    background: #FFFFFF;
7    font-weight: bold;
8  }
```

Pour ajouter les boutons, champ texte, liste et barre de progression, nous allons utiliser les widgets `<label>`, `<textbox>`, `<button>`, `<tree>` et `<progressmeter>`. Cependant, il peut être nécessaire d'organiser tout cela de manière harmonieuse. Nous avons le choix entre l'utilisation de grilles (`<grid>` - équivalent à un tableau) ou de cadres (`<box>`, `<vbox>` et `<hbox>`). Dans notre cas, nous adopterons la seconde solution. Pourquoi ? Parce qu'il faut bien en choisir une ;)

La fenêtre va être découpée en trois dans le sens vertical. Chaque cadre (haut, milieu, bas) contenant des éléments affichés dans le sens horizontal :

freshzilla.xul :

```
...
20 <script type="application/x-javascript"
src="chrome://freshzilla/content/freshzilla.js"/>
21 <script type="application/x-javascript"
src="chrome://freshzilla/content/server.js"/>
...
32 <vbox flex="1">
33   <hbox>
34     <label value="Search for :" class="bold"/>
35     <textbox flex="1" id="prjfind"/>
36     <button id="findbut"
image="http://images.freshmeat.net/button.gif" label="go!" class="bold-and-
white" oncommand="search( )"/>
37   </hbox>
38   <hbox flex="1" context="clipmenu">
39     <tree id="projectTree" flex="1" onclick="treeSelect( event.but-
ton );">
40       <treecols>
```

```
41          <treecol id="prjname" label="&freshzilla:project:name;"
flex="2" primary="true" persist="width ordinal hidden"/>
42          <splitter class="tree-splitter"/>
43          <treecol id="prjdesc" label="&freshzilla:project:desc;"
flex="1" primary="false" persist="width ordinal hidden"/>
44        </treecols>
45
46        <treechildren id="root">
47        </treechildren>
48      </tree>
49    </hbox>
50    <hbox>
51      <progressmeter id="loadprogress" mode="determined" value="0"
flex="1"/>
52      <button id="about" label="About" class="bold-and-white"
53              oncommand="window.openDialog(
'chrome://freshzilla/content/about.xul',
'FreshZilla','chrome,centerscreen,modal' );"/>
54    </hbox>
55 </vbox>
...
```

Détaillons ! A la ligne 32, nous créons un cadre vertical (`vbox`). Cela implique que les éléments (parlons de fils, nous sommes dans du XML que diable !) seront placés les uns en dessous des autres, soit les trois cadres horizontaux (`<hbox>`), ligne 33, 38 et 50. L'attribut `flex` permet d'établir la flexibilité relative des widgets les uns par rapport aux autres. Ici le `flex="1"` appliqué à notre `<vbox>` indique qu'il doit s'étendre sur toute la place disponible. Nous appliquons la même valeur au second cadre horizontal (ligne 38), ce qui fait qu'il sera le seul à être étendu en cas de redimensionnement, au détriment des deux autres.

Dans le premier cadre horizontal, nous plaçons en premier une zone de texte (`<label>`) à laquelle nous appliquons le style `"bold"` défini dans `freshzilla.css`. Le texte affiché est précisé par l'attribut `"value"` (`value="Search for :"`). Nous ajoutons ensuite la `<textbox>` qui sera étendue (`flex="1"`), et enfin le `<button>` affublé d'un libellé (`label="go!"`) et d'une image chargée depuis le site Freshmeat.net (`image="http://images.freshmeat.net/button.gif"`). On précise la commande à appeler lors d'un clic sur le bouton grâce à l'attribut `"oncommand"` dont la valeur correspond à une fonction javascript (définie dans `freshzilla.js` chargé ligne 20).

Le second cadre horizontal contient la liste (`<tree>`) où seront affichés les résultats de la recherche. Il s'agit d'une liste multi-colonnes. Les en-têtes de colonnes sont définis dans `<treecols>` par `<treecol>`. L'attribut `"label"` permet d'indiquer le titre de le la colonne. Nous avons fixé `flex="2"` pour la première colonne et `flex="1"` pour la seconde afin de préciser que, par défaut, la colonne contenant le nom du projet sera deux fois plus large que celle contenant la description. L'attribut `primary` positionné à `true` empêche de masquer une colonne, contrairement au cas où il est à `false`. `"persist"` enfin est utilisé pour spécifier la persistance des attributs appliqués à l'élément, d'une session à l'autre. Entre les deux définitions de colonnes, nous avons placé un *tag* `<splitter>`. Celui-ci permet de redimensionner les colonnes les unes par rapport aux autres.

Le contenu de la liste sera ajouté comme fils de `<treechildren>`. Pour cela, nous devrons ajouter à notre document des items (`<treeitem>`) dans lesquels on précisera, pour chaque ligne (`<treerow>`), le contenu des cellules (`<treecell>`).

Cela prendra la forme suivante :

```
<treechildren id="root">
  <treeitem>
    <treerow>
      <treecell label="nom du projet"/>
      <treecell label="description du projet"/>
    </treerow>
  </treeitem>
</treechildren>
```

Il faudra donc ajouter dans l'arbre XML (XUL si vous préférez) autant de `<treeitem>` que de projets trouvés.

Dans le dernier cadre horizontal, nous plaçons la barre de progression pour la recherche (`<progressmeter>`) et le bouton pour l'affichage de la fenêtre `About`. L'action associée au bouton est très simple, il s'agit d'un simple appel à la fonction `window.openDialog`. Cette fonction, qui permet d'ouvrir une nouvelle fenêtre, prend en paramètre :

- Le chemin d'accès au script XUL décrivant le contenu de la nouvelle fenêtre ;
- Le nom de la fenêtre à ouvrir ;
- Des options à appliquer à la fenêtre.

Pour terminer sur la fenêtre principale, nous allons ajouter un menu popup (menu ouvert par un clic droit) dans la liste de résultats.

freshzilla.xul :

```
...
23 <popupset>
24   <popup id="clipmenu">
25     <menuitem label="View" oncommand="project_view( )"
class="bold"/>
26     <menuitem label="Home page" oncommand="project_home( )"/>
27     <menuseparator/>
28     <menuitem label="Info" oncommand="project_info( )"/>
29   </popup>
30 </popupset>
...
```

Pour créer notre menu popup, nous créons un *container* de popup avec `<popupset>`. Ce n'est absolument pas obligatoire, c'est pourquoi nous le faisons quand même. Le `<popup>` est placé comme élément enfant du container.

Il est très important de spécifier l'id du popup, car c'est grâce à lui que l'on pourra préciser à quel élément de la fenêtre il est assigné. Et c'est ainsi que ceux qui suivent comprennent à quoi correspond l'attribut `context` du cadre horizontal de la ligne 38 de notre code ;)Il ne reste plus qu'à ajouter les items (`<menuitem>`) et les séparateurs (`<menuseparator>`) de notre menu.

La fenêtre About est construite sur le même modèle que la fenêtre principale.

about.xul :

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!–
 3 Desc: Fenetre “About...”
 4 –>
 5
 6 <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
 7
 8 <!DOCTYPE window SYSTEM “chrome://freshzilla/locale/freshzilla.dtd”>
 9
10 <window
11
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
12     id="fzabout"
13     class="dialog"
14     title="&freshzilla:label;"
15     width="205"
16     height="200"
17     orient="horizontal">
18
19 <vbox autostretch="never" align="center">
20
21  <vbox autostretch="never" align="center" style="border: 3px black
inset; padding: 8px; background-color: white; color: black">
22    <image src="http://images.freshmeat.net/logo.gif"/>
23    <text value="&freshzilla:label; &freshzilla:version;" style="mar-
gin-bottom: 15px; font-size: larger; font-weight: bolder"/>
24    <text value="(c) 2003 Gregoire Lejeune"/>
25    <text value="greg@webtime-project.net"/>
26  </vbox>
27  <button label="OK" oncommand="window.close()"/>
28 </vbox>
29 </window>
```

Les seules choses notables ici sont :

- L'utilisation des widgets <image> et <text> ;
- Le fait que nous utilisions l'attribut “style” pour définir le style d'un widget. Bien entendu, on aurait pu utiliser un style défini dans le fichier freshmeat.css et utiliser l'attribut “class”.

Recouvrement

Pour ajouter une entrée dans le menu de Mozilla, nous allons devoir utiliser le principe du recouvrement. Pour cela, on utilise l'élément <overlay>. Le recouvrement est utilisé pour partager un bloc (un contenu) entre différentes fenêtres, ou pour appliquer des changements dans le contenu d'une fenêtre existante.

Chaque élément de recouvrement est inséré, dans la fenêtre parent, au niveau de l'élément déterminé par son attribut d'identification (id). Par exemple, si un élément de recouvrement a comme attribut id="location", alors, dans la fenêtre qui utilise le recouvrement, l'élément d'id de même valeur (“location”) sera modifié par le recouvrement. Les attributs déclarés dans le recouvrement sont ajoutés à l'élément ainsi que les enfants.

Dans notre exemple, nous modifions le menu de Mozilla. Il est donc nécessaire de savoir comment, et où, faire les changements. Cela implique de connaître les valeurs des attributs id des différents éléments composant la fenêtre de Mozilla.

Ne rêvez pas, je ne vais pas vous en donner la liste complète. D'abord, cela serait beaucoup trop long, et puis cela vous enlèverait votre plaisir de les rechercher.

freshzillaOverlay.xul :

```
 1 <?xml version="1.0"?>
 2
 3 <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
 4 <?xml-stylesheet href="chrome://freshzilla/skin/freshzilla.css"
type="text/css"?>
 5
 6 <!DOCTYPE window SYSTEM “chrome://freshzilla/locale/freshzilla.dtd”>
 7
 8 <overlay id="freshzillaOverlay" xmlns="http://www.mozilla.org/key-
master/gatekeeper/there.is.only.xul">
 9
10 <script type="application/x-javascript"
src="chrome://freshzilla/content/addpanel.js"/>
11
12 <menupopup id="windowPopup">
13   <menuitem label="&freshzilla:label;"
14             oncommand="addPanel( ‘chrome://freshzilla/content/
freshzilla.xul’, ‘FreshZilla’ );"
15             insertbefore="sep-window-list" accesskey="z"/>
16 </menupopup>
17
18 </overlay>
```

Pour comprendre ce que nous faisons dans freshzillaOverlay.xul, il peut être intéressant de voir comment on crée un barre de menu avec XUL. Tout commence avec la création de la barre proprement dite. Pour cela, nous utilisons <menubar> qui est le container du menu.

Chaque entrée dans le menu (Fichier, Edition...) est déclaré grâce au widget <menu> dont l'attribut label sert à spécifier le libellé. A chaque menu, nous allons ajouter un container <menupopup>, qui contiendra lui-même tous les items (<menuitem>). Pour plus de clarté, regardez l'exemple suivant :

menu.xul :

```
 1 <?xml version="1.0"?>
 2
 3 <?xml-stylesheet href="chrome://global/skin/"
type="text/css"?>
 4
 5 <window id="example-menu" title="Exemple de menu avec XUL"
 6         xmlns="http://www.mozilla.org/keymaster/gatekee-
per/there.is.only.xul">
 7
 8 <toolbox flex="1">
 9   <menubar id="menubar">
10     <menu id="file-menu" label="Fichier">
11       <menupopup id="file-popup">
```

```
12          <menuitem label="Nouveau"/>
13          <menuitem label="Ouvrir"/>
14          <menuitem label="Sauvegarder"/>
15          <menuseparator/>
16          <menuitem label="Quitter"/>
17        </menupopup>
18      </menu>
19      <menu id="edit-menu" label="Edition">
20        <menupopup id="edit-popup">
21          <menuitem label="Couper"/>
22          <menuitem label="Copier"/>
23          <menuitem label="Coller"/>
24        </menupopup>
25      </menu>
26    </menubar>
27 </toolbox>
28
29 </window>
```

Vous constaterez que la barre de menu est placée dans une `<toolbox>` qui n'est ni plus ni moins qu'un container de barre d'outil, ou, si vous préférez, ce machin que l'on peut masquer/afficher en cliquant sur la partie gauche dans Mozilla.

Mais revenons à notre recouvrement. Dans `freshzillaOverlay.xul`, à la ligne 12, nous déclarons un `menupopup` avec comme valeur d'id : `"windowPopup"`.

Dans le `<menupopup>` de l'entrée `"Window"` de la fenêtre de navigation, on a comme id `"windowPopup"` ! Il ne reste plus qu'à lui ajouter un item, c'est ce qui est fait aux lignes 13 à 15. En sélectionnant ce nouvel item dans le menu, on appelle la fonction `addPanel()` définie dans `addpanel.js`.

Mais où va donc être placé ce nouvel item ? La réponse est donnée par l'attribut `insertbefore`. En effet, vous ne le savez pas, mais le séparateur dans le menu `"Window"` a pour valeur d'id : `"sep-window-list"`.

Donc, notre item est ajouté avant ce séparateur. Si vous souhaitez placer le nouvel item après, utilisez `insertafter`.

JavaScript

Jusqu'à maintenant, nous nous sommes consacrés à la création de l'interface. Il est donc temps d'aborder la partie JavaScript.

De plus, nous n'avons toujours pas parlé de la manière de récupérer les données du site de Freshmeat. Je serai beaucoup plus concis. En effet, toute personne ayant déjà développé pour le Web sera à même de lire et comprendre les sources.

La récupération des données se fait en appelant le moteur de recherche XML de Freshmeat : **http://freshmeat.net/search-xml/?q=cible** où `cible` est remplacé par ce que l'on cherche. Essayez, dans votre navigateur, l'URL suivante : **http://freshmeat.net/search-xml/?q=xul**. Nous récupérons un flux XML qu'il n'y a qu'à "parser".

Un clic sur le bouton `"go!"` (`freshzilla.xul`, ligne 36 - eh oui, faut suivre ;)) appelle la fonction `search()`. Dans cette fonction, on va lancer la requête sur le site de Freshmeat. Le principe est très simple : on crée un objet `XMLHttpRequest` (`server.js`, ligne 14) et on fait une requête de type GET (`server.js`, ligne 27). Le résultat est passé à la fonction de *parsing* (`server.js`, ligne 73).

La fonction de parsing est déclarée dans le fichier `freshzilla.js`, ligne 36. Nous utilisons un objet `DOMParser` pour parser le flux XML reçu et ajouter le résultat dans la liste de la fenêtre principale (fonction `parseNode` dans `freshzilla.js`).

Pour ceux qui liront les scripts, remarquez qu'au début de la fonction `search()` on commence par supprimer de la liste les anciens résultats par un appel à la fonction `removeNode` définie ligne 119. De plus, on modifie la valeur de l'attribut `"mode"` de la barre de progression à `"undetermined"` pour la mettre en mouvement.

Lors de la création du widget `liste` contenant les résultats de la recherche, nous avons précisé qu'un clic sur la liste entraînait un appel à la fonction `treeSelect()` (`freshzilla.xul`, ligne 39). Cette fonction est définie dans `freshzilla.js` et prend en paramètre l'id du bouton de la souris (`event.button`). Dans `treeSelect`, on récupère l'id de la ligne de la liste sélectionnée lors du clic et, s'il s'agit d'un "clic gauche" (`mbut == 0`), on affiche la page Freshmeat du projet dans le fenêtre de navigation. Cela se fait en deux étapes :

- On récupère l'URL de la fenêtre de navigation ;
- On ouvre cette fenêtre avec l'URL correspondant au projet.

Je vous encourage à regarder le code des fonctions `openTopWin()`, `getTopWin()` et `getBrowserURL()` : vous y découvrirez comment on utilise les classes XPCOM. La syntaxe de XPCOM est quelque peu déroutante pour qui ne l'a jamais utilisée, c'est pourquoi un coup d'œil sur la documentation[1][2] devrait vous aider à ne pas vous sentir perdu.

Pour terminer sur le sujet JavaScript, et pour aller plus loin avec XPCOM, je vous laisse le soin de découvrir le contenu du fichier `addpanel.js` qui contient l'ensemble du code utilisé pour ajouter la fenêtre principale à la Sidebar.

content.rdf

La "partie RDF" est certainement la plus obscure quand on crée une application Mozilla. Les puristes vont donc très certainement me trouver très imprécis et je m'en excuse par avance. Comme je l'ai dit un peu plus haut, nous avons besoin de fichiers RDF pour faire "reconnaître" notre application par Mozilla.

En effet, si vous essayez de packager et installer notre application "en l'état", vous n'avez aucune chance de la voir fonctionner. Les fichiers RDF vont nous permettre d'indiquer à Mozilla ce que nous voulons ajouter, et comment.

Pour cela, nous devons créer des fichiers `content.rdf` dans les répertoires `freshzilla/content/freshzilla/`, `freshzilla/locale/en-US/freshzilla/` et `freshzilla/skin/freshzilla/`. Examinons ces fichiers dans l'ordre.

freshzilla/content/freshzilla/content.rdf :

```
 1 <?xml version="1.0"?>
 2
 3 <RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 4          xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
 5
 6   <RDF:Seq about="urn:mozilla:package:root">
 7     <RDF:li resource="urn:mozilla:package:freshzilla"/>
 8   </RDF:Seq>
 9
10   <RDF:Description about="urn:mozilla:package:freshzilla"
11         chrome:displayName="FreshZilla"
12         chrome:author="Gregoire Lejeune"
13         chrome:name="freshzilla">
14   </RDF:Description>
15
16   <RDF:Seq about="urn:mozilla:overlays">
17     <RDF:li
resource="chrome://communicator/content/tasksOverlay.xul"/>
18   </RDF:Seq>
19
20   <RDF:Seq about="chrome://communicator/content/tasksOverlay.xul">
21     <RDF:li>chrome://freshzilla/content/freshzillaOverlay.xul</
RDF:li>
22   </RDF:Seq>
23
24 </RDF:RDF>
```

RDF étant du XML, il est facile à comprendre. Le plus important dans ce fichier, outre les lignes 6 à 14 qui permettent de décrire notre application, sont les lignes 16 à 22. En effet, les lignes 16 à 18 indiquent que l'on va faire du recouvrement sur le fichier `chrome://communicator/content/tasksOverlay.xul`.

C'est ce fichier qui contient la partie du script XUL qui définit (entre autres) le menu "Window" de Mozilla. Si vous voulez en savoir plus sur ce fichier, vous pouvez recopier le fichier `comm.jar` depuis le répertoire `chrome` de Mozilla et extraire son contenu (avec `jar` ou `unzip`). Le fichier se trouve dans l'archive : `content/communicator/tasksOverlay.xul`. Mais revenons à notre RDF. Lignes 20 à 22, on précise que le recouvrement est défini dans le fichier `freshzillaOverlay.xul`.

Vous remarquerez que quand on fait référence à un fichier de l'application, on supprime systématiquement du chemin d'accès le nom du package qui précède le nom du fichier. Regardez les lignes 20 et 21 de `freshzilla.xul` par exemple, ou même les lignes 17 et 21 du fichier `content.rdf` que nous venons d'écrire. En effet, s'agissant du nom de package, et ce dernier étant enregistré au niveau de `chrome`, il ne faut pas l'ajouter à l'URL. Pour vous en convaincre, vous pouvez faire le petit test suivant : modifiez la ligne 53 de `freshzilla.xul` en remplaçant l'appel au fichier `about.xul` par `'chrome://freshzilla/content/freshzilla/about.xul'` et essayez de cliquer sur le bouton `about`. Vous verrez plus loin ce qui se cache exactement derrière les termes "enregistré au niveau de chrome".

freshzilla/locale/en-US/freshzilla/contents.rdf :

```
 1 <?xml version="1.0"?>
 2
 3 <RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 4          xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
 5
 6   <RDF:Seq about="urn:mozilla:locale:root">
 7     <RDF:li resource="urn:mozilla:locale:en-US"/>
 8   </RDF:Seq>
 9
10   <RDF:Description about="urn:mozilla:locale:en-US">
11     <chrome:packages>
12       <RDF:Seq about="urn:mozilla:locale:en-US:packages">
13         <RDF:li resource="urn:mozilla:locale:en-US:freshzilla"/>
14       </RDF:Seq>
15     </chrome:packages>
16   </RDF:Description>
17
18 </RDF:RDF>
```

Dans ce fichier, on donne les indications concernant la localisation. Vous l'aurez peut-être remarqué, mais quand on lie un script XUL avec une DTD, nous omettons, là aussi, une partie du chemin.

Ligne 9 dans `freshzilla.xul`, nous avons écrit :
`<!DOCTYPE window SYSTEM "chrome://freshzilla/locale/freshzilla.dtd">`

Or, notre DTD se trouve dans `freshzilla/locale/en-US/freshzilla/freshzilla.dtd`. Vous savez déjà pourquoi le `freshzilla` a été supprimé.

Pour ce qui est du `en-US`, cela s'explique par le fait que l'existence de fichiers d'internationalisation `en-US` pour la package `freshzilla` est déclarée au niveau de `chrome`. Vous l'aurez donc certainement compris, grâce à cela, les fichiers DTD seront chargés en fonction de la langue choisie dans les préférences de Mozilla (menu Edit->Préférences... puis onglet Appearance->Languages/Content).

Si vous voulez supporter d'autres langues, il suffit de recopier ce fichier dans le sous-répertoire `"locale"` correspondant et de le modifier en conséquence.

freshzilla/skin/freshzilla/contents.rdf :

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

```
 2
 3 <RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 4          xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
 5
 6   <RDF:Seq about="urn:mozilla:skin:root">
 7     <RDF:li resource="urn:mozilla:skin:classic/1.0"/>
 8   </RDF:Seq>
 9
10   <RDF:Description about="urn:mozilla:skin:classic/1.0">
11     <chrome:packages>
12       <RDF:Seq about="urn:mozilla:skin:classic/1.0:packages">
13         <RDF:li resource="urn:mozilla:skin:classic/1.0:freshzilla"/>
14       </RDF:Seq>
15     </chrome:packages>
16   </RDF:Description>
17
18 </RDF:RDF>
```

Ici on déclare l'existence d'un *skin* de notre application. Ce skin est considéré comme faisant partie du skin "classic" de Mozilla.

Cela vous permet de personnaliser notre application pour chaque skin existant pour Mozilla (Voir menu Edit->Préférences... puis onglet Appearance->Themes).

XPInstall

Enfin, notre application est prête. Il faut maintenant s'empresser de partager le fruit de notre travail avec le reste du monde ;).

Pour cela, nous allons packager le tout. Pour ce faire, nous allons utiliser XPInstall (Cross-Platform Install) [4].

Le principe de XPI est très simple. Il s'agit de créer un fichier `.xpi` contenant l'ensemble des fichiers de notre application, plus un script (`install.js`) réalisant l'enregistrement de notre application au niveau de `chrome`.

Pour bien comprendre ce que nous allons faire, je vous propose de faire à la main ce que devra faire notre script après avoir – toujours manuellement – préparé le contenu de l'archive `xpi`.

Une application est enregistrée au niveau de `chrome` quand elle est "référencée" dans le fichier `installed-chrome.txt` du répertoire `chrome` de Mozilla. Si vous éditez ce fichier (qui est un simple fichier texte), vous remarquerez trois types de lignes : celles concernant les skins, celles concernant les locales et celles de contenus.

En fait, pour chaque application (ou morceau d'application) référencée au niveau de `chrome`, il faut préciser où se trouve chaque composant.

Loguez-vous sous le compte `root` et copiez le répertoire `freshzilla` contenant les sources de l'application dans le répertoire `chrome`.

Ajoutez ensuite les lignes suivantes à la fin du fichier `installed-chrome.txt` :

```
content,install,url,resource:/chrome/freshzilla/content/freshzilla/
locale,install,url,resource:/chrome/freshzilla/locale/en-US/freshzilla/
skin,install,url,resource:/chrome/freshzilla/skin/freshzilla/
```

Chacune de ces lignes indique le chemin d'accès aux composants de notre application. Pour ce qui concerne l'internationalisation, n'oubliez pas de rajouter les lignes adéquates si vous supportez d'autres langues.

Et c'est ici que vous comprenez ce que je voulais dire par "enregistré au niveau de chrome" dans la section précédente.

Vous pouvez maintenant redémarrer votre Mozilla et utiliser FreshZilla. Cependant, vous avez certainement remarqué que dans le répertoire `chrome`, les fichiers sont regroupés dans des archives `jar`.

Qu'à cela ne tienne, faisons la même chose. Nous allons donc nous placer dans le répertoire `freshzilla` et taper la commande suivante :

```
# jar cfM freshzilla.jar content/* locale/* skin/*
```

Nous avons maintenant un fichier `freshzilla.jar` que vous pouvez recopier dans le répertoire `chrome`.

Glossaire

- **XML-based User-interface Language (XUL) :** Langage utilisé pour créer la structure et le contenu d'une application.
- **Cascading Style Sheets (CSS) :** Utilisé pour créer le "look and feel" d'une application.
- **JavaScript :** Langage assurant le côté dynamique. Notez que d'autres langages peuvent être utilisés (Python, Perl, PHP...).
- **Cross-Platform Install (XPInstall) :** Utilisé pour packager une application en vue de permettre son installation sur n'importe quelle plateforme.
- **eXtensible Binding Language (XBL) :** Utilisé pour créer des widgets réutilisables.
- **XPCOM/XPConnect :** Permet d'accéder et utiliser des bibliothèques C et C++ avec le langage de script choisi (JavaScript ou autre).
- **Resource Description Framework (RDF) :** Utilisé pour stocker et transmettre des informations à une application.
- **Document Type Definition (DTD) :** Utilisé pour la localisation et l'internationalisation.

Glossaire

Il suffit de modifier le fichier installed-chrome.txt de la façon suivante :

```
content,install,url,jar:resource:/chrome/freshzilla.jar!/content/freshzilla/
locale,install,url,jar:resource:/chrome/freshzilla.jar!/locale/en-US/freshzilla/
skin,install,url,jar:resource:/chrome/freshzilla.jar!/skin/freshzilla/
```

Nous déclarons maintenant avoir une ressource de type jar (jar:resource:/chrome/freshzilla.jar), le chemin d'accès (dans cette archive jar) du composant étant précisé après le point d'exclamation (!).

Relancez votre Mozilla... Ça fonctionne toujours !

Pour pouvoir créer notre xpi, il ne nous reste plus qu'à créer le script (install.js) qui va automatiser la mise à jour du fichier installed-chrome.txt. Le fichier .xpi n'est rien de plus qu'un fichier de type jar contenant freshzilla.jar et install.js, et que l'on créera donc via la commande suivante :

```
# jar cfM freshzilla.xpi freshzilla.jar install.js
```

XPI met à notre disposition une API complète pour la création de scripts d'installation. Cet article étant déjà assez long, je vais me contenter de vous en présenter les bases en détaillant le contenu de install.js.

Par la suite, en espérant que cet article vous en aura donné l'envie, lors de vos propres créations, vous aurez peut-être besoin de scripts d'installation beaucoup plus complexes que celui que nous allons écrire ici. Consultez donc la documentation sur l'API XPInstall [5].

install.js :

```
 1 initInstall('FreshZilla','/Gregoire Lejeune/FreshZilla','0.0.1.0');
 2
 3 var fldr = getFolder('Chrome');
 4 setPackageFolder(fldr);
 5
 6 addFile('freshzilla.jar');
 7
 8 var regfldr;
 9 regfldr = getFolder(fldr,'freshzilla.jar');
10 registerChrome(Install.CONTENT |
Install.DELAYED_CHROME,regfldr,'content/freshzilla/');
11 registerChrome(Install.SKIN |
Install.DELAYED_CHROME,regfldr,'skin/freshzilla/');
12 registerChrome(Install.LOCALE |
Install.DELAYED_CHROME,regfldr,'locale/en-US/freshzilla/');
13
14 var err=getLastError();
15
16 if (err == SUCCESS){
17   logComment('Installing FreshZilla.');
18   performInstall();
19 } else {
20   logComment('Error installing FreshZilla: ' + err);
21   cancelInstall();
22 }
```

A la ligne 1, initInstall permet d'initialiser l'installation de l'application.

La syntaxe que nous retiendrons est :

```
int initInstall (
   String displayName,
   String package,
   String version);
```

displayName est le nom de l'application : c'est ce nom qui sera affiché par l'installeur lors de l'installation. package contient le chemin (*path*) de l'application dans le registre local du client.

Il ne s'agit absolument pas du répertoire d'installation, mais le chemin d'accès aux informations de l'application. version enfin est le numéro de version représenté par au moins quatre entiers séparés par des points.

Sachant que nous installons notre application dans le répertoire chrome de Mozilla, et étant donné que nous ne pouvons pas connaître, à priori, où se trouve ce répertoire sur la machine du client, nous utilisons la fonction getFolder.

Cette fonction renvoie un objet FileSpecObject contenant les informations sur le répertoire. Il suffit ensuite d'invoquer la fonction setPackageFolder, en lui passant en paramètre l'objet FileSpecObject, pour positionner le répertoire d'installation.

Pour copier le fichier jar contenu dans notre xpi, nous appelons la fonction addFile en lui passant en paramètre le nom du fichier à copier (ligne 6).

La mise à jour du fichier installed-chrome.txt se fait en utilisant la fonction registerChrome.

Dans cette fonction, le premier argument est un ensemble de *flags* indiquant ce que l'on enregistre (CONTENT, SKIN ou LOCALE). Le flag DELAYED_CHROME indique que l'enregistrement se fait après un redémarrage.

Le second argument, dont nous avons récupéré la valeur ligne 9, est un objet de type FileSpecObject représentant la source (le fichier freshzilla.jar). Le dernier paramètre indique le chemin d'accès au fichier contents.rdf dans l'archive jar.

Pour terminer, on vérifie qu'il n'y a pas eu d'erreur et on termine l'installation avec un appel à performInstall, ou on l'annule avec cancelInstall.

Le script étant terminé, vous pouvez le copier dans le répertoire freshzilla. Il ne reste plus qu'à écrire un Makefile pour faciliter la création de notre package.

Makefile :

```
1 SOURCE = \
2     content/* \
3     locale/* \
4     skin/*
```

```
 5
 6 CHROME_FILE = /usr/lib/mozilla/chrome/installed-chrome.txt
 7
 8 all: install.js freshzilla.jar
 9     jar cfM freshzilla.xpi install.js freshzilla.jar
10
11 freshzilla.jar: $(SOURCE)
12     jar cfM freshzilla.jar $(SOURCE)
13
14 install: freshzilla.jar
15     cp freshzilla.jar $(MOZILLA)/chrome/freshzilla.jar
16     echo
"content,install,url,jar:resource:/chrome/freshzilla.jar!/content/fresh-
zilla/" >> $(CHROME_FILE)
17     echo
"locale,install,url,jar:resource:/chrome/freshzilla.jar!/locale/en-US/fresh-
zilla/" >> $(CHROME_FILE)
18     echo
"skin,install,url,jar:resource:/chrome/freshzilla.jar!/skin/freshzilla/" >>
$(CHROME_FILE)
19
20 clean:
21     rm -rf freshzilla.jar freshzilla.xpi
```

L'arborescence des sources est donc finalement la suivante :

```
freshzilla
|-- Makefile
|-- content
|   `-- freshzilla
|       |-- about.xul
|       |-- addpanel.js
|       |-- contents.rdf
|       |-- freshzilla.js
|       |-- freshzilla.xul
|       |-- freshzillaOverlay.xul
|       `-- server.js
|-- install.js
|-- locale
|   `-- en-US
|       `-- freshzilla
|           |-- contents.rdf
|           `-- freshzilla.dtd
`-- skin
    `-- freshzilla
        |-- contents.rdf
        `-- freshzilla.css
```

Allez dans le répertoire `freshzilla` et lancez la commande `make`. Vous obtenez un fichier `freshzilla.xpi` que vous n'avez plus qu'à mettre à disposition sur votre site Web.

Si vous souhaitez que vos visiteurs puissent installer le package directement en cliquant sur un lien, ajoutez le bout de code suivante dans votre page Web :

```
...
<script language="javascript">
<!-
function triggerURL(url) {
  if (!InstallTrigger.updateEnabled()) {
    alert( "Sorry, a can't install FreshZilla. Try to download it
and open the xpi file from Mozilla (File -> Open)." );
    return false;
  } else
    InstallTrigger.startSoftwareUpdate(url);
}
//->
</script>
...
<a href="http://monsite.com/freshzilla.xpi"
   onclick="triggerURL('http://monsite.com/freshzilla.xpi');">fresh-
zilla.xpi</a>
...
```

Conclusion

Cet article ne fait que survoler les possibilités offertes par XPFE. Beaucoup de sujets ont été passés sous silence ou traités très rapidement. Mais vous avez maintenant en main les bases du sujet, et en cherchant sur le Net, à partir des sites de Mozilla ou XulPlanet, vous devriez rapidement trouver ce qui vous manque pour réaliser des applications bien plus intéressantes et importantes.

Si vous cherchez encore un intérêt à tout cela, je vous invite à consulter la page du projet Zoolmark [6] de votre serviteur ;)

Grégoire Lejeune
greg@webtime-project.net

Références

[1] http://mozilla.org/projects/xpcom/

[2] http://www.xulplanet.com/references/xpcomref/

[3] http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ ou (français) : http://www.la-grange.net/w3c/REC-rdf-syntax/

[4] http://www.mozilla.org/projects/xpinstall/

[5] http://devedge.netscape.com/library/manuals/2001/xpinstall/1.0/

[6] http://www.zoolmark.org

Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0

Creative Commons n'est pas un cabinet d'avocats et ne fournit pas de services de conseil juridique. La distribution de la présente version de ce contrat ne crée aucune relation juridique entre les parties au contrat présenté ci-après et Creative Commons. Creative Commons fournit cette offre de contrat-type en l'état, à seule fin d'information. Creative Commons ne saurait être tenu responsable des éventuels préjudices résultant du contenu ou de l'utilisation de ce contrat.

Contrat

L'Oeuvre (telle que définie ci-dessous) est mise à disposition selon les termes du présent contrat appelé Contrat Public Creative Commons (dénommé ici « CPCC » ou « Contrat »). L'Oeuvre est protégée par le droit de la propriété littéraire et artistique (droit d'auteur, droits voisins, droits des producteurs de bases de données) ou toute autre loi applicable. Toute utilisation de l'Oeuvre autrement qu'explicitement autorisée selon ce Contrat ou le droit applicable est interdite.

L'exercice sur l'Oeuvre de tout droit proposé par le présent contrat vaut acceptation de celui-ci. Selon les termes et les obligations du présent contrat, la partie Offrante propose à la partie Acceptante l'exercice de certains droits présentés ci-après, et l'Acceptant en approuve les termes et conditions d'utilisation.

1. Définitions

a. **« Oeuvre »** : oeuvre de l'esprit protégeable par le droit de la propriété littéraire et artistique ou toute loi applicable et qui est mise à disposition selon les termes du présent Contrat.
b. **« Oeuvre dite Collective »** : une oeuvre dans laquelle l'oeuvre, dans sa forme intégrale et non modifiée, est assemblée en un ensemble collectif avec d'autres contributions qui constituent en elles-mêmes des oeuvres séparées et indépendantes. Constituent notamment des Oeuvres dites Collectives les publications périodiques, les anthologies ou les encyclopédies. Aux termes de la présente autorisation, une oeuvre qui constitue une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée (telle que définie ci-après).
c. **« Oeuvre dite Dérivée »** : une oeuvre créée soit à partir de l'Oeuvre seule, soit à partir de l'Oeuvre et d'autres oeuvres préexistantes. Constituent notamment des Oeuvres dites Dérivées les traductions, les arrangements musicaux, les adaptations théâtrales, littéraires ou cinématographiques, les enregistrements sonores, les reproductions par un art ou un procédé quelconque, les résumés, ou toute autre forme sous laquelle l'Oeuvre puisse être remaniée, modifiée, transformée ou adaptée, à l'exception d'une oeuvre qui constitue une Oeuvre dite Collective. Une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée aux termes du présent Contrat. Dans le cas où l'Oeuvre serait une composition musicale ou un enregistrement sonore, la synchronisation de l'oeuvre avec une image animée sera considérée comme une Oeuvre dite Dérivée pour les propos de ce Contrat.
d. **« Auteur original »** : la ou les personnes physiques qui ont créé l'Oeuvre.
e. **« Offrant »** : la ou les personne(s) physique(s) ou morale(s) qui proposent la mise à disposition de l'Oeuvre selon les termes du présent Contrat.
f. **« Acceptant »** : la personne physique ou morale qui accepte le présent contrat et exerce des droits sans en avoir violé les termes au préalable ou qui a reçu l'autorisation expresse de l'Offrant d'exercer des droits dans le cadre du présent contrat malgré une précédente violation de ce contrat.

2. Exceptions aux droits exclusifs. Aucune disposition de ce contrat n'a pour intention de réduire, limiter ou restreindre les prérogatives issues des exceptions aux droits, de l'épuisement des droits ou d'autres limitations aux droits exclusifs des ayants droit selon le droit de la propriété littéraire et artistique ou les autres lois applicables.

3. Autorisation. Soumis aux termes et conditions définis dans cette autorisation, et ceci pendant toute la durée de protection de l'Oeuvre par le droit de la propriété littéraire et artistique ou le droit applicable, l'Offrant accorde à l'Acceptant l'autorisation mondiale d'exercer à titre gratuit et non exclusif les droits suivants :

a. reproduire l'Oeuvre, incorporer l'Oeuvre dans une ou plusieurs Oeuvres dites Collectives et reproduire l'Oeuvre telle qu'incorporée dans lesdites Oeuvres dites Collectives;
b. distribuer des exemplaires ou enregistrements, présenter, représenter ou communiquer l'Oeuvre au public par tout procédé technique, y compris incorporée dans des Oeuvres Collectives;
c. lorsque l'Oeuvre est une base de données, extraire et réutiliser des parties substantielles de l'Oeuvre.

Les droits mentionnés ci-dessus peuvent être exercés sur tous les supports, médias, procédés techniques et formats. Les droits ci-dessus incluent le droit d'effectuer les modifications nécessaires techniquement à l'exercice des droits dans d'autres formats et procédés techniques. L'exercice de tous les droits qui ne sont pas expressément autorisés par l'Offrant ou dont il n'aurait pas la gestion demeure réservé, notamment les mécanismes de gestion collective obligatoire applicables décrits à l'article 4(d).

4. Restrictions. L'autorisation accordée par l'article 3 est expressément assujettie et limitée par le respect des restrictions suivantes :

a. L'Acceptant peut reproduire, distribuer, représenter ou communiquer au public l'Oeuvre y compris par voie numérique uniquement selon les termes de ce Contrat. L'Acceptant doit inclure une copie ou l'adresse Internet (Identifiant Uniforme de Ressource) du présent Contrat à toute reproduction ou enregistrement de l'Oeuvre que l'Acceptant distribue, représente ou communique au public y compris par voie numérique. L'Acceptant ne peut pas offrir ou imposer de conditions d'utilisation de l'Oeuvre qui altèrent ou restreignent les termes du présent Contrat ou l'exercice des droits qui y sont accordés au bénéficiaire. L'Acceptant ne peut pas céder de droits sur l'Oeuvre. L'Acceptant doit conserver intactes toutes les informations qui renvoient à ce Contrat et à l'exonération de responsabilité. L'Acceptant ne peut pas reproduire, distribuer, représenter ou communiquer au public l'Oeuvre, y compris par voie numérique, en utilisant une mesure technique de contrôle d'accès ou de contrôle d'utilisation qui serait contradictoire avec les termes de cet Accord contractuel. Les mentions ci-dessus s'appliquent à l'Oeuvre telle qu'incorporée dans une Oeuvre dite Collective, mais, en dehors de l'Oeuvre en elle-même, ne soumettent pas l'Oeuvre dite Collective, aux termes du présent Contrat. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Offrant, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Offrant, comme demandé. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Auteur, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Auteur, comme demandé.

b. L'Acceptant ne peut exercer aucun des droits conférés par l'article 3 avec l'intention ou l'objectif d'obtenir un profit commercial ou une compensation financière personnelle. L'échange de l'Oeuvre avec d'autres Oeuvres protégées par le droit de la propriété littéraire et artistique par le partage électronique de fichiers, ou par tout autre moyen, n'est pas considéré comme un échange avec l'intention ou l'objectif d'un profit commercial ou d'une compensation financière personnelle, dans la mesure où aucun paiement ou compensation financière n'intervient en relation avec l'échange d'Oeuvres protégées.
c. Si l'Acceptant reproduit, distribue, représente ou communique l'Oeuvre au public, y compris par voie numérique, il doit conserver intactes toutes les informations sur le régime des droits et en attribuer la paternité à l'Auteur Original, de manière raisonnable au regard au médium ou au moyen utilisé. Il doit communiquer le nom de l'Auteur Original ou son éventuel pseudonyme s'il est indiqué ; le titre de l'Oeuvre Originale s'il est indiqué ; dans la mesure du possible, l'adresse Internet ou Identifiant Uniforme de Ressource (URI), s'il existe, spécifié par l'Offrant comme associé à l'Oeuvre, à moins que cette adresse ne renvoie pas aux informations légales (paternité et conditions d'utilisation de l'Oeuvre). Ces obligations d'attribution de paternité doivent être exécutées de manière raisonnable. Cependant, dans le cas d'une Oeuvre dite Collective, ces informations doivent, au minimum, apparaître à la place et de manière aussi visible que celles à laquelle apparaissent les informations de même nature.
d. Dans le cas où une utilisation de l'Oeuvre serait soumise à un régime légal de gestion collective obligatoire, l'Offrant se réserve le droit exclusif de collecter ces redevances par l'intermédiaire de la société de perception et de répartition des droits compétente. Sont notamment concernés la radiodiffusion et la communication dans un lieu public de phonogrammes publiés à des fins de commerce, certains cas de retransmission par câble et satellite, la copie privée d'Oeuvres fixées sur phonogrammes ou vidéogrammes, la reproduction par reprographie.

5. Garantie et exonération de responsabilité

a. En mettant l'Oeuvre à la disposition du public selon les termes de ce Contrat, l'Offrant déclare de bonne foi qu'à sa connaissance et dans les limites d'une enquête raisonnable :
    i. L'Offrant a obtenu tous les droits sur l'Oeuvre nécessaires pour pouvoir autoriser l'exercice des droits accordés par le présent Contrat, et permettre la jouissance paisible et l'exercice licite de ces droits, ceci sans que l'Acceptant n'ait aucune obligation de verser de rémunération ou tout autre paiement ou droits, dans la limite des mécanismes de gestion collective obligatoire applicables décrits à l'article 4(e);
b. L'Oeuvre n'est constitutive ni d'une violation des droits de tiers, notamment du droit de la propriété littéraire et artistique, du droit des marques, du droit de l'information, du droit civil ou de tout autre droit, ni de diffamation, de violation de la vie privée ou de tout autre préjudice délictuel à l'égard de toute tierce partie.
c. A l'exception des situations expressément mentionnées dans le présent Contrat ou dans un autre accord écrit, ou exigées par la loi applicable, l'Oeuvre est mise à disposition en l'état sans garantie d'aucune sorte, qu'elle soit expresse ou tacite, y compris à l'égard du contenu ou de l'exactitude de l'Oeuvre.

6. Limitation de responsabilité. A l'exception des garanties d'ordre public imposées par la loi applicable et des réparations imposées par le régime de la responsabilité vis-à-vis d'un tiers en raison de la violation des garanties prévues par l'article 5 du présent contrat, l'Offrant ne sera en aucun cas tenu responsable vis-à-vis de l'Acceptant, sur la base d'aucune théorie légale ni en raison d'aucun préjudice direct, indirect, matériel ou moral, résultant de l'exécution du présent Contrat ou de l'utilisation de l'Oeuvre, y compris dans l'hypothèse où l'Offrant avait connaissance de la possible existence d'un tel préjudice.

7. Résiliation

a. Tout manquement aux termes du contrat par l'Acceptant entraîne la résiliation automatique du Contrat et la fin des droits qui en découlent. Cependant, le contrat conserve ses effets envers les personnes physiques ou morales qui ont reçu de la part de l'Acceptant, en exécution du présent contrat, la mise à disposition d'Oeuvres dites Dérivées, ou d'Oeuvres dites Collectives, ceci tant qu'elles respectent pleinement leurs obligations. Les sections 1, 2, 5, 6 et 7 du contrat continuent à s'appliquer après la résiliation de celui-ci.
b. Dans les limites indiquées ci-dessus, le présent Contrat s'applique pendant toute la durée de protection de l'Oeuvre selon le droit applicable. Néanmoins, l'Offrant se réserve à tout moment le droit d'exploiter l'Oeuvre sous des conditions contractuelles différentes, ou d'en cesser la diffusion; cependant, le recours à cette option ne doit pas conduire à retirer les effets du présent Contrat (ou de tout contrat qui a été ou doit être accordé selon les termes de ce Contrat), et ce Contrat continuera à s'appliquer dans tous ses effets jusqu'à ce que sa résiliation intervienne dans les conditions décrites ci-dessus.

8. Divers

a. A chaque reproduction ou communication au public par voie numérique de l'Oeuvre ou d'une Oeuvre dite Collective par l'Acceptant, l'Offrant propose au bénéficiaire une offre de mise à disposition de l'Oeuvre dans des termes et conditions identiques à ceux accordés à la partie Acceptante dans le présent Contrat.
b. La nullité ou l'inapplicabilité d'une quelconque disposition de ce Contrat au regard de la loi applicable n'affecte pas celle des autres dispositions qui resteront pleinement valides et applicables. Sans action additionnelle par les parties à cet accord, lesdites dispositions devront être interprétées dans la mesure minimum nécessaire à leur validité et leur applicabilité.
c. Aucune limite, renonciation ou modification des termes ou dispositions du présent Contrat ne pourra être acceptée sans le consentement écrit et signé de la partie compétente.
d. Ce Contrat constitue le seul accord entre les parties à propos de l'Oeuvre mise ici à disposition. Il n'existe aucun élément annexe, accord supplémentaire ou mandat portant sur cette Oeuvre en dehors des éléments mentionnés ici. L'Offrant ne sera tenu par aucune disposition supplémentaire qui pourrait apparaître dans une quelconque communication en provenance de l'Acceptant. Ce Contrat ne peut être modifié sans l'accord mutuel écrit de l'Offrant et de l'Acceptant.
e. Le droit applicable est le droit français.

Creative Commons n'est pas partie à ce Contrat et n'offre aucune forme de garantie relative à l'Oeuvre. Creative Commons décline toute responsabilité à l'égard de l'Acceptant ou de toute autre partie, quel que soit le fondement légal de cette responsabilité et quel que soit le préjudice subi, direct, indirect, matériel ou moral, qui surviendrait en rapport avec le présent Contrat. Cependant, si Creative Commons s'est expressément identifié comme Offrant pour mettre une Oeuvre à disposition selon les termes de ce Contrat, Creative Commons jouira de tous les droits et obligations d'un Offrant.

A l'exception des fins limitées à informer le public que l'Oeuvre est mise à disposition sous CPCC, aucune des parties n'utilisera la marque « Creative Commons » ou toute autre indication ou logo afférent sans le consentement préalable écrit de Creative Commons. Toute utilisation autorisée devra être effectuée en conformité avec les lignes directrices de Creative Commons à jour au moment de l'utilisation, telles qu'elles sont disponibles sur son site Internet ou sur simple demande.

Creative Commons peut être contacté à http://creativecommons.org/.