Ceci est un extrait électronique d'une publication de Diamond Editions :

http://www.ed-diamond.com

Retrouvez sur le site tous les anciens numéros en vente par correspondance ainsi que les tarifs d'abonnement.

Pour vous tenir au courant de l'actualité du magazine, visitez :

http://www.gnulinuxmag.com

Ainsi que :

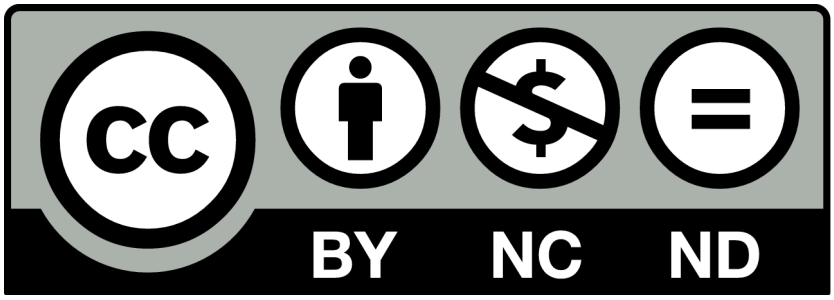
http://www.linux-pratique.com

et

http://www.miscmag.com

Ceci est un extrait électronique d'une publication de Diamond Editions

http://www.ed-diamond.com



Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification

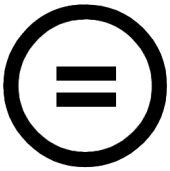
2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public.

Paternité. Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'oeuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'oeuvre).

Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.



Pas de Modification. Vous n'avez pas le droit de modifier, de transformer ou d'adapter cette création.

A chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition.

- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.
- Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Ceci est le Résumé Explicatif du Code Juridique. La version intégrale du contrat est attachée en fin de document et disponible sur :

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/legalcode

Créez des **composants XPCOM** pour **Mozilla**

XPCOM

Nous avons déjà vu brièvement comment utiliser des composants XPCOM. Nous allons maintenant en créer, ce qui devrait vous aider à mieux cerner certaines parties du code JavaScript de Freshzilla (Voir l'article "Etendre l'interface de Mozilla avec XPFE"). Mais avant de passer à la pratique, il est important de savoir ce qu'il en est vraiment et ce qui se cache derrière XPCOM.

Lors du précédent article, je vous ai présenté la solution permettant d'étendre l'interface de Mozilla. Cette fois-ci, nous allons voir comment ajouter des fonctionnalités en créant des composants. Pour cela, exit XML, bonjours C++.

A l'origine, XPCOM était dévolu à Mozilla, il faisait pleinement partie du projet. Mais une bonne idée étant faite pour être réutilisée, on retrouve XPCOM dans de nombreux projets, à commencer par Netscape bien entendu. (Notez, on serait en droit de se demander quelle différence il existe entre Netscape et Mozilla.)

XPCOM [1] (acronyme de *Cross Platform Component Object Model*) est un *framework*, semblable au COM de Microsoft, permettant d'écrire des applications modulaires et multiplateformes. En fait, il existe des différences importantes entre l'implémentation de Microsoft et celle de Mozilla. Les composants XPCOM présentent l'avantage de pouvoir être compilés et utilisés sur une grande variété de plateformes. COM quant à lui est "réservé" à Windows, mais il autorise l'utilisation de composants répartis sur différentes machines grâce à un mécanisme dit de *proxy*. Un tel mécanisme n'existe pas nativement dans XPCOM. De fait, il ne faut pas se méprendre, il n'y a pas de réelle compatibilité entre ces deux modèles, seule l'idée sous-jacente et la méthode d'implémentation sont communes. On pourrait dire que XPCOM est un COM léger mais multiplateforme. Enfin, je rajouterai à l'attention de ceux qui voudraient déjà laisser tomber que la complexité de COM (et surtout de DCOM) est sans commune mesure avec la simplicité de XPCOM ;)

Généralement, les composants XPCOM sont écrits en JavaScript, C ou C++, mais il est tout à fait possible de choisir un autre langage pour faire ce travail, tel que Java, Python [2], Perl [3] ou Ruby [4]. Toute plateforme possédant un compilateur C++, un interpréteur Perl et le panel classique des outils GNU peut permettre de développer des composants XPCOM. Et quel système digne de ce nom ne possède pas encore tout cela ?

Les composants XPCOM

Mozilla offre un certain nombre de services (affichage de page Web, envoi et réception de mails...). Tous ces services utilisent des "fonctionnalités" plus bas niveau (réseau, *parsing*...), généralement développées en C++ et compilées nativement pour la plateforme cible. Entre les deux, on trouve des *widgets* XBL qui sont (pour simplifier) des widgets accessibles pour XUL et ayant leurs propres propriétés, fonctions, événements et contenus (j'y reviendrai dans un prochain article). Les "fonctionnalités" bas niveau sont organisées en modules réutilisables, on les appelle donc composants XPCOM. L'ensemble du code au niveau applicatif étant du JavaScript, il est nécessaire de faire le lien entre ce dernier et le code C++. C'est le rôle tenu par XPConnect. Pour faire simple, disons que nous avons une bibliothèque en C++, nous créons un API avec XPCOM pour la rendre accessible en JavaScript, via XPConnect, à des widgets XBL permettant de créer des applications avec XPFE. Simple, non?

Vous l'avez sans doute déjà compris, un composant est un bout de code réutilisable qui est développé indépendamment du reste de l'application qui va l'utiliser. La principale fonction des composants est de mettre à disposition de nouvelles fonctionnalités en définissant une ensemble d'APIs permettant d'accéder, par le biais d'interfaces publiques, à des bibliothèques externes.

Ainsi, vous pourrez trouver sur **mozdev.org** le composant `mysqlxpcom` accompagné du `mysqlexplorer`, une interface XUL permettant d'administrer une base MySQL. Bien entendu, dans Mozilla, la majeure partie des composants est chargée d'accéder à Gecko ou d'étendre les fonctionnalités de ce dernier.

Grâce à XPCOM, de gros logiciels (comme Mozilla) sont en fait une simple association de petits composants. Ceci présente bien évidemment de gros avantages :

- Chaque composant peut être utilisé par différentes applications.
- La modification d'un des composants se fait sans entraîner la recompilation de l'ensemble du projet.
- Il est possible de charger les composants à la demande, en fonction des besoins, ce qui permet d'alléger l'application et de la rendre plus performante.
- Les défauts d'un composant n'ont pas d'impact sur les autres et ne remettent pas en jeu la stabilité globale de l'application.
- La maintenance est facilitée et l'identification des problèmes se fait plus facilement car le code à examiner est aisément identifiable.

Créer et utiliser des composants

La méthode la plus simple pour créer des composants pouvant facilement s'assembler les uns aux autres consiste à mettre en place un "canal de communication" entre ces composants à la manière d'un ORB. Donc, la première chose que nous avons à faire pour écrire notre composant est d'en spécifier l'interface en utilisant un langage indépendant du langage de programmation de l'application, l'IDL (Langage de Définition d'Interface). La suite, qui se base sur un compromis entre vos aspirations et goûts personnels et les possibilités offertes par le langage choisi, consiste à développer le composant en utilisant au choix du JavaScript, du C/C++, du Java, du Python, du Perl, du Ruby. Bref, n'importe quel langage ayant un *wrapper* XPCOM.

Une fois le composant et son interface créés, l'utilisation se fait en récupérant le composant et en l'associant avec l'interface *ad hoc*. Dans l'exemple JavaScript suivant, on charge le composant "parser DOM" (`@mozilla.org/xmlextras/domparser`) en lui associant l'interface correspondante (`nsIDOMParser`). Par la suite, il ne reste plus qu'à instancier l'objet (`domParser`) ainsi récupéré pour pouvoir jouir de l'utilisation d'un parseur DOM ;)

```
var domParser = new
Components.Constructor("@mozilla.org/xmlextras/domparser;1",
"nsIDOMParser");
...
```

La même chose peut bien évidemment être faite en C++ :

```
nsCOMPtr<nsIDOMParser> domParser;
nsresult rv = servManager-
>GetServiceByContractID("@mozilla.org/xmlextra/domparser;1",
NS_GET_IID(nsIDOMParser), getter_AddRefs(domParser));
...
```

Pour ceux qui se posent la question, "@mozilla.org/xmlextra/domparser;1" est le "Contract ID". Il s'agit d'une simple chaîne permettant d'identifier le composant. La forme est la suivante :

```
@domaine/module/nom;version.
```

- `domaine` sera toujours `mozilla.org`. A moins bien entendu que vous soyez en train de créer un composant pour une toute autre application.
- `module`, qui est purement facultatif, est le nom du module dans lequel se place votre composant. Vous pouvez bien entendu avoir des sous, des sous-sous, etc. modules.
- Quant à `nom` et `version`, j'hésite à vous en parler ;)

Un composant en JavaScript

Nous savons déjà qu'il existe un composant DOM. Imaginons maintenant qu'à partir de cela nous souhaitions accéder à des informations contenues dans un document XML en utilisant une syntaxe (pseudo) XPath [5]. Pas de problème, nous allons développer pour cela un composant en JavaScript.

Avant cela, il faut définir précisément les limites que nous allons mettre dans notre parser. En effet, je n'ai pas du tout l'intention de réimplémenter XPath en totalité. Nous allons donc nous borner à mettre en place quelque chose permettant de récupérer les valeurs des nœuds de type texte ou les valeurs des attributs. La syntaxe générique sera donc la suivante :

```
/node1[numéro|@attribut="valeur"][@attribut]
```

Si nous avons la source XML suivante :

```
<TEST>
  Exemple
  <CHILD ID="1" ZONE="fr">
    valeur du noeud #1
  </CHILD>
  avec
  <CHILD ID="2" ZONE="en">
    value of child #2
  </CHILD>
  notre
  <CHILD ID="3" ZONE="es">
    valor de nodo #3
  </CHILD>
  pseudo XPath
</TEST>
```

`/TEST` donnera comme résultat : "Exemple avec notre pseudo XPath".

`/TEST/CHILD[1]` donnera "valeur du noeud #1".

`/TEST/CHILD[@ID="3"]@ZONE` donnera "es".

Bien entendu, nous n'allons pas travailler n'importe où. Dans la suite, je partirai du principe que vous avez compilé votre propre Mozilla. Placez-vous dans le répertoire racine des sources (chez moi : `~/mozilla`). Si vous listez le contenu de ce répertoire, vous devriez trouver un sous-répertoire `xpcom`. C'est là que nous allons travailler. En fait, une fois dans `xpcom`, nous allons créer un répertoire pour chacun de nos composants :

```
uranium$ cd ~/mozilla/xpcom
uranium$ mkdir xpath
uranium$ cd xpath
uranium$ pwd
/home/greg/mozilla/xpcom/xpath
uranium$
```

Pour notre `pseudo XPath`, du point de vue de l'interface, nous avons besoin de deux méthodes :

- Une méthode (`load_xml`) permettant de charger en mémoire la chaîne XML et de construire la structure nécessaire pour la recherche.
- Une méthode (`find`) donnant le résultat d'une recherche.

Créons donc le fichier IDL suivant :

nsIXPath.idl
```
1  #include "nsISupports.idl"
2  [scriptable, uuid(01c4fe50-8c0c-46d0-995a-b2e2506b07a5)]
3  interface nsIXPath : nsISupports
4  {
5      void load_xml( in string xml );
6      string find( in string path );
7  };
```

A la ligne 1, nous incluons le fichier `nsISupports.idl`. Ce fichier définit l'interface "de base" dont hérite tout composant XPCOM. La ligne 2 permet d'associer un IDentifiant Universel Unique (UUID) à notre interface. C'est ce nombre de 128 bits qui permet d'identifier de manière unique chaque interface de composant. Ce nombre est obtenu grâce à la commande `uuidgen`.

```
uranium$ uuidgen
01c4fe50-8c0c-46d0-995a-b2e2506b07a5
uranium$
```

Vient ensuite la définition proprement dite de notre interface dans laquelle nous retrouvons nos deux méthodes :

- `load_xml` prend en entrée (*in*) la chaîne (*string*) XML à parser et ne renvoie rien.
- `find` prend en entrée (*in*) la chaîne (*string*) de recherche (*path*) et renvoie la chaîne (*string*) résultat.

C'est tout pour l'interface. Mais vous vous en doutez bien, on ne va pas l'utiliser telle quelle. Il faut générer un fichier d'interface binaire. Pour cela, nous allons utiliser la commande `xpidl` :

```
uranium$ ../../xpcom/typelib/xpidl/xpidl -m typelib -w -v -I
../../xpcom/base -o nsIXPath nsIXPath.idl
uranium$ ls -l
-rw-r--r--   1 greg  greg       191 Oct 24 01:23 nsIXPath.idl
-rw-r--r--   1 greg  greg       153 Oct 24 01:23 nsIXPath.xpt
uranium$
```

Les options passées sont les suivantes :

`-m` : précise le type de sortie, ici `typelib`, c'est-à-dire un fichier d'interface binaire (`XPConnect typelib`). Mais, comme nous le verrons plus loin, il est aussi possible d'avoir en sortie un fichier d'en-tête C++ (`header`), la documentation HTML (`doc`) ou une interface Java (`java`).

`-w` : pour avoir les *warnings*.

`-v` : pour le mode verbeux.

`-I` : permet de spécifier les répertoires ou chercher les *includes*.

`-o` : pour spécifier le nom de base (donc sans extension) à donner au fichier de sortie.

Reste maintenant à faire l'implémentation de notre composant en JavaScript. Notre code va se décomposer en trois parties.

Tout d'abord, la création de la classe comprenant l'ensemble des méthodes et attributs dont nous avons besoin :

nsIXPath.js
```
1  function XPathComponent(){}
2
3  /*
4   * ELEMENT_NODE : The node is a Element.
5   * ATTRIBUTE_NODE : The node is an Attr.
6   * TEXT_NODE : The node is a Text node.
7   * CDATA_SECTION_NODE : The node is a CDATASection.
8   * ENTITY_REFERENCE_NODE : The node is an EntityReference.
9   * ENTITY_NODE : The node is an Entity.
10  * PROCESSING_INSTRUCTION_NODE : The node is a
ProcessingInstruction.
11  * COMMENT_NODE : The node is a Comment.
12  * DOCUMENT_NODE : The node is a Document.
13  * DOCUMENT_TYPE_NODE : The node is a DocumentType.
14  * DOCUMENT_FRAGMENT_NODE : The node is a DocumentFragment.
15  * NOTATION_NODE : The node is a Notation.
16  */
17 var ELEMENT_NODE                =  1;
18 var ATTRIBUTE_NODE              =  2;
19 var TEXT_NODE                   =  3;
20 var CDATA_SECTION_NODE          =  4;
21 var ENTITY_REFERENCE_NODE       =  5;
22 var ENTITY_NODE                 =  6;
23 var PROCESSING_INSTRUCTION_NODE =  7;
24 var COMMENT_NODE                =  8;
25 var DOCUMENT_NODE               =  9;
26 var DOCUMENT_TYPE_NODE          = 10;
27 var DOCUMENT_FRAGMENT_NODE      = 11;
28 var NOTATION_NODE               = 12;
29
30 XPathComponent.prototype = {
31
32     load_xml: function ( xml ) {
33       var hvResult = new Array( );
34
35       var DOMParser=new Components.Constructor("@mozilla.org/xmlex-
tras/domparser;1", "nsIDOMParser");
36       var DOM = new DOMParser().parseFromString( xml, 'text/xml' );
37       oRootNode = DOM.documentElement;
38       xNodeName = oRootNode.nodeName;
39
40       hvResult[xNodeName] = this.getChilds( oRootNode );
41       if( hvResult[xNodeName] == null ) {
42         hvResult = null;
43       }
44
45       this.hvDataStr = hvResult;
46     },
47     getChilds: function ( oNode ) {
48       var i;
49       var hvNode = new Array( );
50       hvNode["#data"] = "";
51       hvNode["#attrs"] = new Array( );
52       hvNode["#childs"] = new Array( );
53
54       var xName = oNode.nodeName;
55
56       // Recuperation des attributs
57       var lvAttribs = oNode.attributes;
```

```
58        for( i=0; i<lvAttribs.length; i++ ) {
59          if( lvAttribs.item(i) ) {
60            hvNode["#attrs"][lvAttribs.item(i).nodeName] =
lvAttribs.item(i).nodeValue;
61          }
62        }
63
64        // Parcour des noeuds fils
65        var oChildNodes = oNode.childNodes;
66
67        for( i=0; i<oChildNodes.length; i++ ) {
68          var oChild = oChildNodes.item(i);
69          switch( oChild.nodeType ) {
70
71            case ELEMENT_NODE:
72              if( !hvNode["#childs"][oChild.nodeName] ) {
73                hvNode["#childs"][oChild.nodeName] = new Array( );
74              }
75              hvNode["#childs"][oChild.nodeName].push(
this.getChilds( oChild ) );
76              break;
77
78            case TEXT_NODE:
79              xText = oChild.nodeValue;
80              xTmp1 = xText.replace( /^(\n|\r|\ )*/, "" );
81              xTmp2 = xTmp1.replace( /(\n|\r|\ )*$/, "" );
82              xTmp3 = xTmp2.replace( /\n/g, " " );
83              xText = xTmp3.replace( /\ {2,}/g, " " );
84              if( xText.length != 0 ) {
85                if( hvNode["#data"] != "" ) {
86                  hvNode["#data"] += " " + xText;
87                } else {
88                  hvNode["#data"] = xText;
89                }
90              }
91              break;
92
93            case ATTRIBUTE_NODE:
94              break;
95            case CDATA_SECTION_NODE:
96              break;
97            case ENTITY_REFERENCE_NODE:
98              break;
99            case ENTITY_NODE:
100             break;
101           case PROCESSING_INSTRUCTION_NODE:
102             break;
103           case COMMENT_NODE:
104             break;
105           case DOCUMENT_NODE:
106             break;
107           case DOCUMENT_TYPE_NODE:
108             break;
109           case DOCUMENT_FRAGMENT_NODE:
110             break;
111           case NOTATION_NODE:
112             break;
113           default:
114             hvNode = null;
115             return( hvNode );
116         }
117       }
118
119       return( hvNode );
120     },
121     find: function ( path ) {
122       var i;
123       var hvNode = this.hvDataStr;
124       var xNode;
125       var xCond;
126       var xAttr;
127       var RCod = null;
128
129       xXPath = path.replace( /^\//, "" );
130       lxLevels = xXPath.split( "/" );
131       for( i=0; i<lxLevels.length; i++ ) {
132         var rNodeTt = /([^\[]*)\[([^\]]*)\]@(.*)/;
133         var rNodeNt = /([^\[]*)\[([^\]]*)\]/;
134         var rNodePt = /([^@]*)@(.*)/;
135         var rCond =
/^\s*@([^(\s*=)]*)\s*=\s*["']([^("|')]*)["']\s*/
136
137         xNode = null;
138         xCond = null;
139         xAttr = null;
140
141         if( rNodeTt.test( lxLevels[i] ) ) {
142           var lvNodeElmts = rNodeTt.exec( lxLevels[i] );
143           xNode = lvNodeElmts[1];
144           xCond = lvNodeElmts[2];
145           xAttr = lvNodeElmts[3];
146         } else if( rNodeNt.test( lxLevels[i] ) ) {
147           var lvNodeElmts = rNodeNt.exec( lxLevels[i] );
148           xNode = lvNodeElmts[1];
149           xCond = lvNodeElmts[2];
150         } else if( rNodePt.test( lxLevels[i] ) ) {
151           var lvNodeElmts = rNodePt.exec( lxLevels[i] );
152           xNode = lvNodeElmts[1];
153           xAttr = lvNodeElmts[2];
154         } else {
155           xNode = lxLevels[i];
156         }
157
158         if( i == 0 ) {
159           hvNode = hvNode[xNode];
160         } else {
161           if( xCond ) {
162             if( rCond.test( xCond ) ) {
163               var j, k;
164               var lxCondData = rCond.exec( xCond );
165               for( j = 0; j<hvNode["#childs"][xNode].length; j++ )
{
166                 if(
hvNode["#childs"][xNode][j]["#attrs"][lxCondData[1]] &&
hvNode["#childs"][xNode][j]["#attrs"][lxCondData[1]] == lxCondData[2] ) {
167                   k = j;
168                   break;
169                 }
170               }
171               if( k ) {
172                 hvNode = hvNode["#childs"][xNode][k];
173               } else {
174                 return( null );
175               }
176             } else if( /^[0-9]*$/.test( xCond ) ) {
177               var iIdx = xCond - 1;
178               if( xCond > hvNode["#childs"][xNode].length ) {
179                 iIdx = hvNode["#childs"][xNode].length - 1
180               }
181               if( iIdx < 0 ) {
182                 iIdx = 0;
183               }
184               hvNode = hvNode["#childs"][xNode][iIdx];
185             } else {
```

```
186                 return( null );
187               }
188             } else {
189               hvNode = hvNode["#childs"][xNode][0];
190             }
191           }
192         }
193
194         if( xAttr ) {
195           RCod = hvNode["#attrs"][xAttr];
196         } else {
197           RCod = hvNode["#data"];
198         }
199
200         return( RCod );
201       },
202       hvDataStr: null,
203
204       QueryInterface: function (iid) {
205           if (!iid.equals(Components.interfaces.nsIXPath)
206               && !iid.equals(Components.interfaces.nsISupports)) {
207               throw Components.results.NS_ERROR_NO_INTERFACE;
208           }
209           return this;
210       }
211   }
```

Je ne détaillerai pas cette partie, elle est relativement simple et ne comporte pas de surprise. Il faut cependant noter la présence de la méthode `QueryInterface`. Cette méthode, définie dans `nsISupports`, est appelée lors du chargement du composant pour valider qu'on l'associe avec la bonne interface (ici `nsIXPath`). Cela se fait en vérifiant que le IID passé "matche" bien avec le UUID de l'interface `nsIXpath`. Le IID n'est autre que le UUID de l'interface que l'on associe à notre composant au chargement. Si le *matching* est bon, on récupère une instance de classe de l'objet `XPathComponent`, sinon on lève une exception.

Dans la seconde partie de notre fichier, nous créons un objet "Module" :

```
nsIXPath.js (suite)
212
213   var Module = {
214       firstTime: true,
215
216       registerSelf: function (compMgr, fileSpec, location, type) {
217           if (this.firstTime) {
218               dump("*** Deferring registration of XPath JS compo-
nents\n");
219               this.firstTime = false;
220               throw
Components.results.NS_ERROR_FACTORY_REGISTER_AGAIN;
221           }
222           debug("*** Registering XPath JS components\n");
223           compMgr =
compMgr.QueryInterface(Components.interfaces.nsIComponentRegistrar);
224           compMgr.registerFactoryLocation(this.myCID,
225                                           "XPath JS Component",
226                                           this.myProgID,
227                                           fileSpec,
228                                           location,
229                                           type);
230       },
231
232       getClassObject : function (compMgr, cid, iid) {
233           if (!cid.equals(this.myCID))
234           throw Components.results.NS_ERROR_NO_INTERFACE
235           if (!iid.equals(Components.interfaces.nsIFactory))
236           throw Components.results.NS_ERROR_NOT_IMPLEMENTED;
237           return this.myFactory;
238       },
239
240       myCID: Components.ID("{280500ed-112d-41dc-89e2-6c24a0caaa13}"),
241
242       myProgID: "@mozilla.org/js_xpath;1",
243
244       myFactory: {
245           createInstance: function (outer, iid) {
246               dump("CI: " + iid + "\n");
247               if (outer != null)
248               throw Components.results.NS_ERROR_NO_AGGREGATION;
249               return (new XPathComponent()).QueryInterface(iid);
250           }
251       },
252
253       canUnload: function(compMgr) {
254           dump("****** Unloading: XPath JS component! ****** \n");
255           return true;
256       }
257   }; // END Module
258
```

Dans ce module, notons avant tout la définition du Contract ID (ici noté `myProgID`) et d'un IDentifiant unique pour le Composant (CID, noté ici `myCID`). Ce CID est au composant ce que l'UUID est à l'interface. La méthode `registerSelf` permet de vérifier si l'enregistrement de notre composant a déjà été fait et si tel n'est pas le cas, de passer les informations nécessaires à cet enregistrement au "Component Manager". Disons, pour rester simple, que le "Component Manager" a pour fonction d'enregistrer les composants, c'est-à-dire de valider leur prise en compte lors du démarrage de Mozilla, et donc d'assurer que l'on pourra les charger à la demande.

Lorsque l'on souhaite "charger" notre composant en vue de l'utiliser, c'est la méthode `getClassObject` qui est appelée. Cette méthode prend en paramètre une instance du "Component Manager", le CID et l'UUID d'une interface `nsIFactory`. Si les paramètres passés sont valides, cette méthode retourne une instance du composant `factory`, qui est en fait une instance de notre classe sans son interface. Par la suite, vous l'aurez sans doute compris en lisant le code des lignes 244 à 251, on pourra, en appelant la méthode `createInstance` de notre `factory`, récupérer une instance de notre classe `XPathComponent`, interface comprise. Et la boucle est bouclée.

Terminons sur une évidence, la méthode `canUnload` est appelée lors du désenregistrement, c'est-à-dire quand on ferme Mozilla.

Dans la dernière partie du script, nous créons la fonction `NSGetModule` qui est appelée par le "Component Manager" pour initialiser les méthodes et objets XPCOM nécessaires pour l'utilisation de notre composant.

nsIXPath.js (suite et fin)
```
   259  function NSGetModule(compMgr, fileSpec) { return Module; }
```

Tout ceci étant fait, nous pouvons passer à des choses beaucoup plus terre à terre mais nécessaires : créer un `Makefile` et un script de test.

Makefile
```
     1  TOP_SRC    = ../..
     2  INST_DIR   = $(TOP_SRC)/dist/bin/components
     3  XPIDL      = $(TOP_SRC)/xpcom/typelib/xpidl
     4  XPIDL_INC  = $(TOP_SRC)/xpcom/base
     5  FLAGS      = -m typelib -w -v -I $(XPIDL_INC) -o
     6  COMP_NAME  = nsIXPath
     7  TEST       = XPathTest.js
     8
     9  all:
    10          $(XPIDL)/xpidl $(FLAGS) $(COMP_NAME) $(COMP_NAME).idl
    11  install:
    12          cp $(COMP_NAME).xpt $(COMP_NAME).js $(INST_DIR)
    13  clean:
    14          rm -rf *.xpt
    15  uninstall:
    16          rm -f $(INST_DIR)/$(COMP_NAME).xpt
    17          rm -f $(INST_DIR)/$(COMP_NAME).js
    18  test:
    19          $(TOP_SRC)/dist/bin/run-mozilla.sh
$(TOP_SRC)/dist/bin/xpcshell -f $(TEST)
```

Dans le `Makefile`, nous retrouvons l'appel à `xpidl` pour la création de l'interface binaire. Le script de test est utilisé en lançant le shell JavaScript (`xpcshell`) fourni avec Mozilla.

XPathTest.js
```
     1  var XPath = new Components.Constructor("@mozilla.org/js_xpath;1",
"nsIXPath");
     2  var oXPath = new XPath( );
     3
     4  for(var list in oXPath)
     5    print( list );
     6
     7  var xml = '<TEST>\n<CHILD ID="1" ZONE="fr">\nvaleur du noeud
#1\n</CHILD>\n<CHILD ID="2" ZONE="en">\nvalue of child #2\n</CHILD>\n<CHILD
ID="3" ZONE="es">\nvalor de nodo #3\n</CHILD>\n</TEST>';
     8
     9  oXPath.load_xml( xml );
    10
    11  var RC = oXPath.find( "/TEST/CHILD" );
    12  print( "/TEST/CHILD = "+RC );
    13
    14  RC = oXPath.find( "/TEST/CHILD[1]@ZONE" );
    15  print( "/TEST/CHILD[1]@ZONE = "+RC );
    16
    17  RC = oXPath.find( "/TEST/CHILD[@ID='3']" );
    18  print( "/TEST/CHILD[@ID='3'] = "+RC );
    19
    20  RC = oXPath.find( "/TEST/CHILD[@ZONE='en']@ID" );
    21  print( "/TEST/CHILD[@ZONE='en']@ID = "+RC );
```

Dans le script de test, avant de charger le `xml` et d'appeler la méthode `find`, nous demandons à obtenir la liste des méthodes accessibles de notre composant (lignes 4, 5). Vous constaterez que nous ne voyons que les méthodes (publiques) définies dans `nsIXPath.idl` et `nsISupports.idl` (`QueryInterface`).

```
helium$ make
helium$ make install
helium$ make test
../../dist/bin/run-mozilla.sh ../../dist/bin/xpcshell -f test.js
Type Manifest File: /home/greg/mozilla/dist/bin/components/xpti.dat
+++ JavaScript debugging hooks installed.
nsNativeComponentLoader: autoregistering begins.
nsNativeComponentLoader: autoregistering succeeded
nNCL: registering deferred (0)
*** Deferring registration of XPath JS components
QueryInterface
load_xml
find
/TEST/CHILD = valeur du noeud #1
/TEST/CHILD[1]@ZONE = fr
/TEST/CHILD[@ID='3'] = valor de nodo #3
/TEST/CHILD[@ZONE='en']@ID = 2
****** Unloading: XPath JS component! ******
WARNING: nsExceptionService ignoring thread destruction after shut-
down, file nsExceptionService.cpp, line 189
+++ JavaScript debugging hooks removed.
helium$
```

Ouf, ça fonctionne !

Avant de passer à la suite, je voudrais souligner un élément important pour ceux qui auraient tendance à douter de l'intérêt de développer des composants en JavaScript.

L'avantage de ce genre de composants, outre la relative simplicité qu'il y a à les écrire, les modifier et donc de les déboguer, est qu'il n'y a absolument pas besoin de faire une recompilation pour les utiliser sur une autre plateforme.

Si vous en doutez, copiez les fichiers `nsIXPath.js` et `nsIXPath.xpt` dans le répertoire `mozilla/components` sur un système Windows ou Mac, vous verrez que votre composant sera immédiatement accessible... Essayez !

Un composant en C++

La création de composants en C++ n'est pas beaucoup plus compliquée qu'en JavaScript, pour peu que l'on ait quelques bases de C++. Cette méthode aurait même (selon moi) l'avantage de dévoiler certaines choses "cachées" par la méthode JavaScript et d'en simplifier d'autres.

Il existe un grand nombre de macros qui vont nous faciliter la tâche. Je me contenterai de vous les présenter quand cela sera nécessaire sans les "décoder". Je vous laisse le plaisir de fouiller les sources de XPCOM pour voir ce qui se cache derrière.

Dans l'exemple suivant, nous allons créer un composant recevant en entrée le contenu d'un fichier signé par GPG. L'objectif étant de nous dire si la signature est valide, et si tel est le cas, qui en est l'auteur. Attention, je ne vais pas vous faire un cours sur GPG. Si vous voulez vous rafraîchir la mémoire, je vous conseille l'excellent *tutorial* de Loïc Bernable [6].

De plus, mais vous vous en doutez sans doute déjà, nous allons utiliser la bibliothèque GPGME. Je pars donc du principe que vous n'êtes pas démunis de connaissances sur le sujet.

Enfin, le composant que nous allons créer ici est un exemple, il est donc (très) loin d'être parfait et est même susceptible de (très) mal se comporter dans certains cas...

Avant de commencer, n'oubliez pas de vous placer dans un nouveau répertoire.

```
uranium$ cd ~/mozilla/xpcom
uranium$ mkdir gpg
uranium$ cd gpg
uranium$ pwd
/home/greg/mozilla/xpcom/gpg
uranium$
```

Nous pouvons ensuite créer le fichier IDL.

nsIGpg.idl

```
1  #include "nsISupports.idl"
2  [scriptable, uuid(1049ef09-11e7-4785-9d21-778b38d36073)]
3  interface nsIGpg : nsISupports
4  {
5      attribute string source;
6      string getGPGSig( );
7      string getGPGVersion( );
8  };
```

Notre interface comprend donc :

- Un attribut (*source*) de type chaîne (*string*), permettant de positionner le contenu du fichier dont on souhaite vérifier la signature.
- Une méthode `getGPGSig` ne prenant aucun paramètre d'entrée mais renvoyant les informations sur la signature du fichier sous forme de chaîne (*string*).
- Une méthode `getGPGVersion` qui est là pour le *fun* et qui renvoie une chaîne (*string*) donnant la version de GPGME utilisée.

Nous ne devons pas générer de version binaire de notre interface (en tout cas, pas pour le moment). Par contre, il peut être très intéressant de générer le fichier d'en-tête C++ correspondant. Ce fichier va nous donner toutes les informations nécessaires pour l'implémentation de notre composant. Vous connaissez déjà la méthode :

```
helium$ ../../../dist/bin/xpidl -m header -w -v -I
~/mozilla/xpcom/base -o nsIGpg nsIGpg.idl
```

Vous devez obtenir un fichier `nsIGpg.h`. Editons ce fichier et commentons son contenu.

nsIGpg.h

```
1  /*
2   * DO NOT EDIT.  THIS FILE IS GENERATED FROM nsIGpg.idl
3   */
4
```

oups ;)

nsIGpg.h (suite)

```
5  #ifndef __gen_nsIGpg_h__
6  #define __gen_nsIGpg_h__
7
8
9  #ifndef __gen_nsISupports_h__
10 #include "nsISupports.h"
11 #endif
12
```

Comme on aurait pu s'y attendre, on charge le fichier d'en-tête de `nsISupports`.

```
13 /* For IDL files that don't want to include root IDL files. */
14 #ifndef NS_NO_VTABLE
15 #define NS_NO_VTABLE
16 #endif
17
18 /* starting interface:    nsIGpg */
19 #define NS_IGPG_IID_STR "1049ef09-11e7-4785-9d21-778b38d36073"
20
21 #define NS_IGPG_IID \
22   {0x1049ef09, 0x11e7, 0x4785, \
23     { 0x9d, 0x21, 0x77, 0x8b, 0x38, 0xd3, 0x60, 0x73 }}
24
```

Nous retrouvons notre UUID. Vous constatez qu'il a été "découpé" (`NS_IGPG_IID`). En effet, l'UUID est utilisé sous forme d'une structure :

```
struct nsID {
  PRUint32 m0;
  PRUint16 m1, m2;
  PRUint8 m3[8];
};
```

nsIGpg.h (suite)

```
25 class NS_NO_VTABLE nsIGpg : public nsISupports {
26  public:
27
28   NS_DEFINE_STATIC_IID_ACCESSOR(NS_IGPG_IID)
29
30   /* attribute string source; */
31   NS_IMETHOD GetSource(char * *aSource) = 0;
32   NS_IMETHOD SetSource(const char * aSource) = 0;
33
34   /* string getGPGSig (); */
35   NS_IMETHOD GetGPGSig(char **_retval) = 0;
36
37   /* string getGPGVersion (); */
38   NS_IMETHOD GetGPGVersion(char **_retval) = 0;
39
40 };
41
```

Et voici la partie la plus intéressante. Nous avons ici la déclaration de notre classe d'interface.Toutes les méthodes sont de type `NS_IMETHOD`. Cela implique donc que la valeur de retour (s'il y en a une) sera placée dans une variable dont l'adresse est passée en paramètre. Ainsi, par exemple, si nous avions déclaré une fonction `truc` prenant en paramètre une chaîne de caractères et un entier et retournant un booléen, la version IDL serait la suivante :

```
boolean truc(in string xChaine, in long lNombre);
```

ce qui donnerait en C++ :

```
NS_IMETHOD Truc(const char *xChaine, PRInt32 lNombre, PRBool *_retval);
```

Vous trouverez ci-dessous un tableau vous donnant les équivalences entre les types IDL et C++. Sachez aussi qu'il est tout à fait possible d'utiliser des types natifs.

Pour plus d'information sur le sujet et sur le langage XPIDL en général (oui, je devrais dire IDL et non pas XPIDL, mais vous aviez compris ;)), vous pouvez consulter [7].

Tableau des types IDL et leurs équivalents C++.

IDL	C++
void	void
boolean	PRBool
octet	PRUint8
short	PRInt16
long	PRInt32
long long	PRInt64
unsigned short	PRUint16
unsigned long	PRUint32
unsigned long long	PRUint64
float	float
double	double
char	char
wchar	PRUnichar
string	char*
wstring	PRUnichar*

Revenons à notre classe d'interface. L'attribut "source" dans l'IDL a abouti à la déclaration de deux méthodes : `GetSource` et `SetSource`. C'est grâce à ces méthodes que nous pourrons positionner et récupérer la valeur de cet attribut. Un accès en lecture de notre attribut se fait par un appel à la méthode `GetSource`, alors qu'un accès en écriture se fait par `SetSource`. En plus de ces deux (nouvelles) méthodes, nous retrouvons les méthodes `GetGPGSig` et `GetGPGVersion`.

Grâce au fichier `nsIGpg.h`, nous connaissons la structure de la classe d'interface. Nous savons donc que nous avons quatre méthodes à écrire dans la classe d'implémentation du composant et nous avons les prototypes précis.

```
nsIGpg.h (suite)
    42  /* Use this macro when declaring classes that implement this inter-
face. */
    43  #define NS_DECL_NSIGPG \
    44    NS_IMETHOD GetSource(char * *aSource); \
    45    NS_IMETHOD SetSource(const char * aSource); \
    46    NS_IMETHOD GetGPGSig(char **_retval); \
    47    NS_IMETHOD GetGPGVersion(char **_retval);
    48
    ...
```

A la ligne 43, nous avons une déclaration de macro qui va nous intéresser par la suite. Elle reprend l'ensemble des prototypes des méthodes que nous devons écrire dans notre composant.

Le reste du fichier ne présentant pas d'intérêt pour nous, je le passe sous silence.

Il faut maintenant écrire le code de notre composant et le "module" servant à son chargement. Pour le composant, il n'y a pas de surprise. Nous allons utiliser les mêmes principes que ceux que nous avons déjà vus.

```
nsGpg.h
     1  #include <gpgme.h>
     2  #include "nsIGpg.h"
     3
     4  // fb8cd7c6-46b4-41fe-8d8c-37ae2157410c
     5  #define NS_GPG_CID \
     6  { 0xfb8cd7c6, 0x46b4, 0x41fe, { 0x8d, 0x8c, 0x37, 0xae, 0x21, 0x57,
0x41, 0x0c } }
     7  #define NS_GPG_CONTRACTID "@mozilla.org/cpp_gpg;1"
     8  class nsGpgImpl : public nsIGpg
     9  {
    10  public:
    11      nsGpgImpl( );
    12      virtual ~nsGpgImpl( );
    13      // nsISupports interface
    14      NS_DECL_ISUPPORTS
    15      NS_DECL_NSIGPG
    16  private:
    17      char* mSource;
    18      const char * status_string( GpgmeSigStat );
    19      const char * get_sig( GpgmeCtx, GpgmeSigStat );
    20  };
```

Comme nous faisons du C++ et que nous voulons un code propre, nous créons un fichier d'en-tête (`nsGpg.h`) pour la classe d'implémentation de notre composant. Nous devons y définir le CID (lignes 5-6) et le Contract ID (ligne 7). Vient ensuite la déclaration de la classe d'implémentation proprement dite (`nsGpgImpl`). Nous y mettons l'ensemble des méthodes classiques, à savoir un constructeur, un destructeur. Les déclarations des méthodes `GetSource`, `SetSource`, `GetGPGSig` et `GetGPGVersion` se font en utilisant la macro `NS_DECL_NSIGPG`. `NS_DECL_ISUPPORTS` permet d'ajouter en plus les déclarations des méthodes héritées de `nsISupport` que nous nous garderons bien de ne pas oublier. Pour mémoire, il s'agit (entre autres) de la méthode `QueryInterface`.

Nous ajoutons à tout cela un attribut privé (`mSource`) qui servira à stocker en interne la valeur de l'attribut source et deux méthodes utilisées pour la vérification de la signature GPG.

Nous pouvons maintenant écrire le fichier `cpp`. Comme je l'ai déjà dit, ce qui nous intéresse ici concerne la création du composant et non pas l'utilisation de GPGME ; il est donc inutile de crier en regardant certaines parties du code ;)

```
nsGpg.cpp
     1  #include <stdio.h>
     2  #include <string.h>
     3
     4  #include "nsGpg.h"
     5  #include "nsMemory.h"
     6  #include "nsEmbedString.h"
```

L'important ici concerne l'*include* `nsMemory.h`. La classe `nsMemory` contient un ensemble de méthodes permettant de gérer la mémoire. Ainsi, les fonctions d'allocation et réallocation (`malloc` et `realloc`) seront respectivement remplacées par `nsMemory::Alloc` et `nsMemory::Realloc`, les appels à `free` se transformeront en appels à `nsMemory::Free` et nous utilisons aussi `nsMemory::Clone` pour "cloner" un pointeur.

```
nsGpg.cpp (suite)
     7
     8  #define BEGIN_PGP_SIGNATURE "—-BEGIN PGP SIGNATURE—-\n"
     9  #define END_PGP_SIGNATURE   "—-END PGP SIGNATURE—-\n"
    10
```

```
11  // c++ constructor
12  nsGpgImpl::nsGpgImpl( ) : mSource(nsnull) {
13      NS_INIT_ISUPPORTS();
14      mSource = (char*)nsMemory::Clone("", 1);
15  }
16
17  // c++ destructor
18  nsGpgImpl::~nsGpgImpl( ) {
19      if( mSource ) { nsMemory::Free(mSource); }
20  }
21
```

Vous avez là deux superbes exemples d'utilisation de `nsMemory`. Concernant l'appel à `NS_INIT_ISUPPORTS`, il s'agit d'une résurgence des versions antérieures à la 1.3 de Gesko. Vous pouvez l'oublier, voire même le supprimer ; de mon côté, je le garde, on ne sait jamais sur qui on peut tomber !

nsGpg.cpp (suite)

```
22  const char * nsGpgImpl::status_string (GpgmeSigStat status) {
23      const char *s = "?";
24
25      switch ( status ) {
26        case GPGME_SIG_STAT_NONE:
27          s = "None";
28          break;
29        case GPGME_SIG_STAT_NOSIG:
30          s = "No Signature";
31          break;
32        case GPGME_SIG_STAT_GOOD:
33          s = "Good";
34          break;
35        case GPGME_SIG_STAT_GOOD_EXP:
36          s = "Good but expired";
37          break;
38        case GPGME_SIG_STAT_GOOD_EXPKEY:
39          s = "Good but key exipired";
40          break;
41        case GPGME_SIG_STAT_BAD:
42          s = "Bad";
43          break;
44        case GPGME_SIG_STAT_NOKEY:
45          s = "No Key";
46          break;
47        case GPGME_SIG_STAT_ERROR:
48          s = "Error";
49          break;
50        case GPGME_SIG_STAT_DIFF:
51          s = "More than one signature";
52          break;
53      }
54      return s;
55  }
56
57  const char * nsGpgImpl::get_sig( GpgmeCtx tGMEContext, GpgmeSigStat
tGMESigStatus ) {
58      int         iCpt;
59      const char *xFingerprint = NULL;
60      time_t      tTimeSigCreate;
61      char        *xXMLKey = NULL;
62      GpgmeKey    tGMEKey;
63
64      for( iCpt = 0; ( xFingerprint = gpgme_get_sig_status(
tGMEContext, iCpt, &tGMESigStatus, &tTimeSigCreate ) ); iCpt++ ) {
65          if( !gpgme_get_sig_key( tGMEContext, iCpt, &tGMEKey ) ) {
66              xXMLKey = gpgme_key_get_as_xml( tGMEKey );
67              gpgme_key_release( tGMEKey );
68          }
69      }
70
71      return( xXMLKey );
72  }
73
74  // This macro implements the nsISupports interface methods
75  // QueryInterface, AddRef and Release
76  NS_IMPL_ISUPPORTS1_CI(nsGpgImpl, nsIGpg);
```

Les méthodes `status_string` et `get_sig` sont utilisées pour donner des informations concernant la signature GPG. Elles sont appelées par la méthode `GetGPGSig`. Je n'en dirai donc pas grand-chose, juste que vous noterez que les informations concernant le signataire du fichier seront renvoyées en XML grâce à la fonction `gpgme_key_get_as_xml` (ligne 66).

Plus important dans le bout de code précédent : l'utilisation de `NS_IMPL_ISUPPORTS1_CI`. Comme le signale le commentaire, cette macro nous abolit le devoir de créer la méthode `QueryInterface` comme nous l'avons fait lors de la création du composant en JavaScript. Les plus à l'aise en anglais me demanderont ce que sont les méthodes `AddRef` et `Release` ? Je leur répondrai qu'`AddRef` est une méthode permettant d'incrémenter le nombre de références d'interfaces associées à notre composant et que `Release` est là pour décharger celles qui ne sont pas/plus utilisées. Et logiquement, on se demande alors, mais de quoi il parle ?

Eh bien, en fait, un composant peut avoir plusieurs interfaces, en fait autant qu'il y a de demandes d'accès au composant. Donc, à chaque fois que l'on associe une interface à un composant, il faut bien le noter quelque part. Le rôle d'`AddRef` est de compter ces interfaces. `Release` quant à lui sert à décharger les interfaces qui ne sont pas ou plus utilisées. Bref, ici on s'en f... un peu, c'est `NS_IMPL_ISUPPORTS1_CI` qui fait le boulot.

nsGpg.cpp (suite)

```
77
78  NS_IMETHODIMP
79  nsGpgImpl::GetSource( char** aSource ) {
80      NS_PRECONDITION( aSource != nsnull, "null ptr" );
81
82      if( !aSource )
83          return NS_ERROR_NULL_POINTER;
84
85      if( aSource ) {
86          *aSource = (char*)nsMemory::Clone( mSource, strlen( mSource ) +
1 );
87          if (! *aSource)
88              return NS_ERROR_NULL_POINTER;
89      } else {
90          *aSource = nsnull;
91      }
92      return NS_OK;
93  }
94
95  NS_IMETHODIMP
96  nsGpgImpl::SetSource( const char* aSource ) {
97      NS_PRECONDITION( aSource != nsnull, "null ptr" );
98
99      if( !aSource )
```

```
100          return NS_ERROR_NULL_POINTER;
101
102      if( mSource ) {
103          nsMemory::Free( mSource );
104      }
105
106      mSource = (char*)nsMemory::Clone( aSource, strlen( aSource ) + 1 );
107      return NS_OK;
108  }
109
```

Les méthodes `GetSource` et `SetSource` sont relativement simples mais elles sont l'occasion pour nous d'aborder deux choses importantes :

- Les macros de débogage. Ces macros, telles que `NS_PRECONDITION`, sont définies dans le fichier `nsDebug.h`. Vous y trouverez, entre autres, des choses comme `NS_POSTCONDITION`, `NS_WARNING`, `NS_ERROR`, `NS_ASSERTION` ou `NS_ABORT`.
- Les codes retour des méthodes. En cas de succès, une méthode renverra toujours `NS_OK`. Dans le cas contraire, vous devez choisir un code retour dans le tableau ci-dessous.

```
129    GpgmeError   tGMEError;
130    GpgmeData    tGMEDataText;
131    GpgmeData    tGMEDataSignature;
132    GpgmeCtx     tGMEContext;
133    GpgmeSigStat tGMESigStatus;
134
135    char *data;
136    char *xSigBegin, *xSigEnd, *xSignature;
137    int  iSigLeng;
138
139    if( !_retval )
140        return NS_ERROR_NULL_POINTER;
141
142    if( _retval ) {
143        data = (char*)nsMemory::Clone( mSource, strlen( mSource ) + 1
);
144
145        /** Act: Recherche de la signature */
146        xSigBegin  = strstr( data, BEGIN_PGP_SIGNATURE );
147        xSigEnd    = strstr( data, END_PGP_SIGNATURE );
148        iSigLeng   = (int)xSigEnd - (int)xSigBegin + strlen(
END_PGP_SIGNATURE );
149        xSignature = (char*) nsMemory::Alloc( sizeof(char) *
(iSigLeng + 1) );
150
151        memset(xSignature, '\0', (iSigLeng+1));
```

Code retour

Valeur	Remarque
NS_ERROR_NOT_INITIALIZED	Indique qu'il y a eu une demande d'utilisation d'un objet non initialisé.
NS_ERROR_ALREADY_INITIALIZED	Indique qu'il y a eu demande d'initialisation d'un objet déjà initialisé.
NS_ERROR_NOT_IMPLEMENTED	Indique que la méthode n'est pas implémentée.
NS_ERROR_NO_INTERFACE	Indique que l'interface n'est pas supportée.
NS_ERROR_NULL_POINTER	Indique qu'un pointeur est `null` alors qu'il ne devrait pas.
NS_ERROR_FAILURE	Indique qu'il y a eu une erreur dans la méthode.
NS_ERROR_UNEXPECTED	Indique qu'une erreur inattendue s'est produite.
NS_ERROR_OUT_OF_MEMORY	Indique qu'il n'y a pas assez de mémoire.
NS_ERROR_FACTORY_NOT_REGISTERED	Indique que la classe demandée n'est pas enregistrée auprès du "Component Manager"

nsGpg.cpp (suite et fin)

```
110  NS_IMETHODIMP
111  nsGpgImpl::GetGPGVersion( char **_retval ) {
112    NS_PRECONDITION( _retval != nsnull, "null ptr" );
113
114    if( !_retval )
115        return NS_ERROR_NULL_POINTER;
116
117    if( _retval ) {
118            *_retval = (char*)nsMemory::Clone( gpgme_check_version(
NULL ), strlen( gpgme_check_version( NULL ) ) + 1 );
119    } else {
120        *_retval = nsnull;
121    }
122    return NS_OK;
123  }
124
125  NS_IMETHODIMP
126  nsGpgImpl::GetGPGSig( char **_retval ) {
127    NS_PRECONDITION( _retval != nsnull, "null ptr" );
128
```

```
152        strncpy( xSignature, xSigBegin, iSigLeng );
153
154        /** Act: Init GPGME context */
155        tGMEError = gpgme_new( &tGMEContext );
156        if( tGMEError != GPGME_No_Error ) {
157            // printf( "Error #%d : %s\n", (int)tGMEError,
gpgme_strerror( tGMEError ) );
158            *_retval = nsnull;
159            return NS_OK;
160        }
161
162        /** Act: GPGME Data text */
163        tGMEError = gpgme_data_new_from_mem (&tGMEDataText, mSource,
strlen (mSource), 0);
164        if( tGMEError != GPGME_No_Error ) {
165            // printf( "Error #%d : %s\n", (int)tGMEError,
gpgme_strerror( tGMEError ) );
166            *_retval = nsnull;
167            return NS_OK;
168        }
169
```

```
170        /** Act: GPGME Data signature */
171        tGMEError = gpgme_data_new_from_mem (&tGMEDataSignature,
xSignature, strlen (xSignature), 0);
172        if( tGMEError != GPGME_No_Error ) {
173            // printf( "Error #%d : %s\n", (int)tGMEError,
gpgme_strerror( tGMEError ) );
174            *_retval = nsnull;
175            return NS_OK;
176        }
177
178        /** Act: Verification */
179        tGMEError = gpgme_op_verify (tGMEContext, tGMEDataSignature,
tGMEDataText, &tGMESigStatus );
180
181        if( tGMEError == GPGME_No_Error ) {
182            const char *XML = get_sig( tGMEContext, tGMESigStatus );
183            // printf( "Verification Status: %s\n", status_string(
tGMESigStatus ) );
184            // printf( "%s\n", XML );
185            *_retval = (char*)nsMemory::Clone( XML, strlen( XML ) +
1 );
186        } else {
187            // printf( "Error #%d : %s\n", (int)tGMEError,
gpgme_strerror( tGMEError ) );
188            *_retval = nsnull;
189            return NS_OK;
190        }
191
192        /** Act: Liberation memoire */
193        gpgme_data_release( tGMEDataText );
194        gpgme_data_release( tGMEDataSignature );
195        gpgme_release( tGMEContext );
196            nsMemory::Free( xSignature );
197        nsMemory::Free( data );
198    } else {
199        *_retval = nsnull;
200    }
201    return NS_OK;
202  }
```

Il n'y a pas grand-chose à dire concernant les méthodes `GetGPGSig` et `GetGPGVersion` si ce n'est que le code est très certainement perfectible, mais bon...

La création du "module" pour l'enregistrement du composant auprès du "Component Manager" se fait dans un nouveau fichier :

nsGpgModule.cpp

```
1  #include "nsIGenericFactory.h"
2  #include "nsGpg.h"
3
4  NS_GENERIC_FACTORY_CONSTRUCTOR(nsGpgImpl)
5  static NS_METHOD nsGpgRegistrationProc(nsIComponentManager
*aCompMgr,
6                                         nsIFile *aPath,
7                                         const char
*registryLocation,
8                                         const char
*componentType,
9                                         const
nsModuleComponentInfo *info)
10 {
11     return NS_OK;
12 }
13
14 static NS_METHOD
nsGpgUnregistrationProc(nsIComponentManager *aCompMgr,
15                                         nsIFile
*aPath,
16                                         const char
*registryLocation,
17                                         const
nsModuleComponentInfo *info)
18 {
19     return NS_OK;
20 }
```

Nous avons déjà vu lors de la création du composant `XPath` qu'il faut créer pour le module une classe `factory`. En C++, tout ceci est caché par l'utilisation de la macro `NS_GENERIC_FACTORY_CONSTRUCTOR` à laquelle on passe le nom de la classe d'implémentation.

La seule chose que nous devons savoir est que la fonction permettant de récupérer une instance de notre composant (notre classe d'implémentation) via le `factory` est `nsGpgImplConstructor`.

Il faut aussi créer les deux méthodes utilisées pour l'enregistrement (`nsGpgRegistrationProc`) et le désenregistrement (`nsGpgUnregistrationProc`). Ici, nous n'avons rien à faire de particulier, ce qui rend le corps de ces fonctions réduit au strict minimum.

nsGpgModule.cpp

```
21
22 // For each class that wishes to support nsIClassInfo, add a line
like this
23 NS_DECL_CLASSINFO(nsGpgImpl)
24 static nsModuleComponentInfo components[ ] =
25 {
26   { "A Gpg Component",
27     NS_GPG_CID,
28     NS_GPG_CONTRACTID,
29     nsGpgImplConstructor,
30     nsGpgRegistrationProc,
31     nsGpgUnregistrationProc,
32     NULL,
33     NS_CI_INTERFACE_GETTER_NAME(nsGpgImpl),
34     NULL,
35     &NS_CLASSINFO_NAME(nsGpgImpl)
36   }
37 };
38 NS_IMPL_NSGETMODULE(nsGpgModule, components)
```

Pour terminer, on va placer tout ceci dans une structure de type `nsModuleComponentInfo`. De cette structure nous retiendrons particulièrement les champs :

1. contenant le texte du message qui sera affiché lors du chargement du composant (en mode `debug`);

2. contenant l'identifiant de la classe d'implémentation (le CID);

3. pour le "Contract ID";

Les champs 4 à 7 contiennent tous des pointeurs de

fonction, respectivement pour accéder au constructeur de la classe d'implémentation, pour l'enregistrement et le désenregistrement, et enfin pour le destructeur de la classe `factory`.

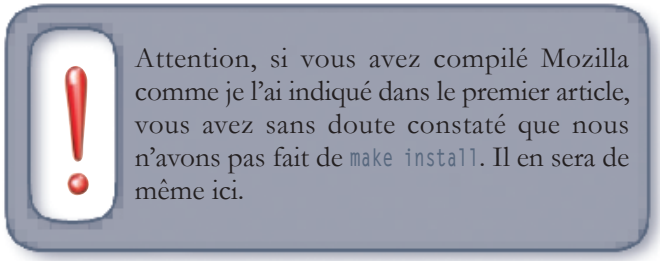
Il faut savoir que seuls les quatre premiers champs sont obligatoires, le reste est optionnel. D'ailleurs, si vous éditez le fichier `nsIGenericFactory.h`, vous retrouverez la déclaration de cette structure et constaterez qu'elle comprend plus de champs que nous en avons ici.

C'est fini pour le code, nous n'avons plus qu'à compiler tout cela et vérifier que ça fonctionne. Commençons par le `Makefile`.

```
Makefile
     1  DEPTH         = ../..
     2  topsrcdir     = ../..
     3  srcdir        = .
     4  VPATH         = .
     5
     6  include $(DEPTH)/config/autoconf.mk
     7
     8  MODULE        = xpcom
     9  XPIDL_MODULE  = gpg
    10  LIBRARY_NAME  = gpg
    11  IS_COMPONENT  = 1
    12  MODULE_NAME   = nsGpgModule
    13  REQUIRES      = string \
    14                  embedstring \
    15                  xpcom \
    16                  $(NULL)
    17
    18  CPPSRCS = \
    19          nsGpg.cpp \
    20          nsGpgModule.cpp \
    21          $(NULL)
    22
    23  XPIDLSRCS = nsIGpg.idl
    24
    25  TEST = test.js
    26
    27  include $(topsrcdir)/config/config.mk
    28
    29  LIBS += \
    30          $(XPCOM_LIBS) \
    31          $(NSPR_LIBS)  \
    32          $(NULL)
    33
    34  include $(topsrcdir)/config/rules.mk
    35
    36  EXTRA_DSO_LDOPTS += $(MOZ_COMPONENT_LIBS) -lgpgme
    37
    38  install:: $(TARGETS)
    39
    40  test:: $(TEST)
    41          $(topsrcdir)/dist/bin/run-mozilla.sh
$(topsrcdir)/dist/bin/xpcshell -f $(TEST)
```

Ce `Makefile` est écrit de façon à intégrer pleinement notre composant aux sources de Mozilla :

- Nous incluons donc les données de configuration (ligne 6) de Mozilla. On entend ici par configuration les informations concernant la compilation de manière générale. En effet, c'est dans `autoconf.mk` que sont déclarées les variables `prefix`, `exec_prefix`...
- Nous précisons que c'est un module de type XPCOM (ligne 8).
- Nous donnons le nom du module XPIDL et de la bibliothèque C++ (lignes 9 et 10).
- Il s'agit d'un composant (ligne 11).
- Et on précise le nom du module (ligne 12).
- A la ligne 27, nous incluons le fichier `config.mk`. C'est dans ce fichier que sont définies les variables `INCS`, `CFLAGS` et autre `LFLAGS`.
- Les règles de compilation sont définies dans le fichier `rules.mk` (ligne 34).
- Notre composant utilisant la bibliothèque `gpgme`, il faut bien le préciser quelque part. C'est fait ligne 36.
- Nous terminons en rajoutant deux règles, l'une pour l'installation et l'autre pour les tests.



Nous nous contenterons donc de faire un simple `make` suivi d'un `make test` pour vérifier que tout fonctionne comme prévu. En effet, si vous faites un `ls` dans le répertoire `components` après le `make`, vous devez retrouver un lien symbolique vers les fichiers générés lors de la compilation :

```
uranium$ ls -l ../../dist/bin/components/*gpg*
lrwxrwxrwx    1 greg  greg        36 Oct 24 01:23 ../../dist/bin/compo-
nents/gpg.xpt -> ../../../xpcom/gpg/_xpidlgen/gpg.xpt
lrwxrwxrwx    1 greg  greg        28 Oct 24 01:23 ../../dist/bin/compo-
nents/libgpg.so -> ../../../xpcom/gpg/libgpg.so
uranium$
```

Bon, ceci étant dit, nous pouvons donc tester :

```
uranium$ make test
../../dist/bin/run-mozilla.sh ../../dist/bin/xpcshell -f test.js
Type Manifest File: /home/greg/mozilla/dist/bin/components/xpti.dat
+++ JavaScript debugging hooks installed.
nsNativeComponentLoader: autoregistering begins.
nsNativeComponentLoader: autoregistering succeeded
nNCL: registering deferred (0)
QueryInterface
source
getGPGSig
getGPGVersion
```

```
gpg version 0.3.15
<GnupgKeyblock>
  <mainkey>
    <keyid>948B5992CC8701C5</keyid>
    <fpr>0730BEE9AC466F096ED55821948B5992CC8701C5</fpr>
    <algo>17</algo>
    <len>1024</len>
    <created>1065552159</created>
    <otrust>?</otrust>
  </mainkey>
  <userid>
    <raw>Gregoire Lejeune <greg@webtime-project.net></raw>
    <name>Gregoire Lejeune</name>
    <email>greg@webtime-project.net</email>
  </userid>
  <userid>
    <raw>Gregoire Lejeune <gregoire.lejeune@free.fr></raw>
    <name>Gregoire Lejeune</name>
    <email>gregoire.lejeune@free.fr</email>
  </userid>
  <subkey>
    <keyid>3709CD0692CE8D6B</keyid>
    <algo>16</algo>
    <len>1024</len>
    <created>1065552162</created>
  </subkey>
</GnupgKeyblock>

WARNING: nsExceptionService ignoring thread destruction after shutdown, file
nsExceptionService.cpp, line 189
+++ JavaScript debugging hooks removed.
uranium$
```

Pour cela, nous avons bien entendu créé le fichier de test suivant :

test.js

```
 1  /*
 2   * gpg –keyserver pgp.mit.edu –recv-key 0xCC8701C5
 3   */
 4  var Gpg = new Components.Constructor("@mozilla.org/cpp_gpg;1",
"nsIGpg");
 5  var gpg = new Gpg( );
 6  for( var list in gpg )
 7  print( list );
 8  var ver = gpg.getGPGVersion( );
 9  print( "gpg version "+ver );
10  gpg.source = "—-BEGIN PGP SIGNED MESSAGE—-\nHash:
SHA1\n\nhello\nca va bien ?\n—-BEGIN PGP SIGNATURE—-\nVersion: GnuPG
v1.2.3
(GNU/Linux)\n\niD8DBQE/kSNPlItZksyHAcURAnOHAKCKSKzboGoB9WyYtKqJNkz2z08p6ACghE
30\n3fs+VLIatWPeScWh+3LRFDo=\n=CFbG\n—-END PGP SIGNATURE—-\n";
11  var sig = gpg.getGPGSig( );
12  print( sig );
```

Nous avons vu le fonctionnement de nos deux composants par le biais de scripts de tests. Comme je ne suis pas ingrat, je vous propose ci-dessous un exemple directement dans une page HTML. Bien entendu, ce qui est fait ici peut parfaitement être reproduit dans une application développée avec XPFE.

test.html

```
 1 <HTML>
 2 <HEAD>
 3 <SCRIPT LANGUAGE="Javascript1.2">
 4 /*
 5  * gpg –keyserver pgp.mit.edu –recv-key 0xCC8701C5
 6  */
 7 function verify( text ) {
 8
netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
 9   var cGpg =
Components.classes["@mozilla.org/cpp_gpg;1"].createInstance();
10   var gpg = cGpg.QueryInterface(Components.interfaces.nsIGpg);
11
12   var ver = gpg.getGPGVersion( );
13   alert( "gpgme version "+ver );
14
15   gpg.source = text;
16   var sig = gpg.getGPGSig( );
17   alert( sig );
18 }
19 </SCRIPT>
20 </HEAD>
21 <BODY>
22
23 <FORM NAME="test">
24 <TEXTAREA ROWS="15" COLS="70" NAME="data">
25 —-BEGIN PGP SIGNED MESSAGE—-
26 Hash: SHA1
27
28 hello
29 ca va bien ?
30 -----BEGIN PGP SIGNATURE-----
31 Version: GnuPG v1.2.3 (GNU/Linux)
32
33 iD8DBQE/kSNPlItZksyHAcURAnOHAKCKSKzboGoB9WyYtKqJNkz2z08p6ACghE30
34 3fs+VLIatWPeScWh+3LRFDo=
35 =CFbG
36 -----END PGP SIGNATURE-----
37
38 </TEXTAREA>
39 </FORM>
40 <BR>
41 <A HREF="javascript:verify( document.test.data.value );">Verify</A>
42 </BODY>
43 </HTML>
```

J'en vois qui font des gros yeux ! En effet, la syntaxe utilisée dans la fonction `verify` est "déroutante" et certains préféreront peut-être la solution suivante, plus facile à comprendre, mais moins proche de ce que vous pourrez trouver dans les sources de Mozilla. De plus, la méthode utilisée ci-dessus a l'avantage de bien distinguer la partie "chargement du composant" (ligne 9) et "chargement/association de l'interface" (ligne 10) – et c'est là que vous comprenez enfin ce que je voulais dire avec mon délire sur le "Component Manager" et le `factory :p`.

Quoi qu'il en soit, il peut être bon de se familiariser avec les deux méthodes.

```
 7 function verify( text ) {
 8
netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
 9   var Gpg = new Components.Constructor("@mozilla.org/cpp_gpg;1",
"nsIGpg");
10   var gpg = new Gpg( );
11
12   var ver = gpg.getGPGVersion( );
13   alert( "gpgme version "+ver );
14
15   gpg.source = text;
16   var sig = gpg.getGPGSig( );
17   alert( sig );
18 }
```

Enfin, explicitons l'instruction qui se trouve à la ligne 8. En effet, ce petit bout de code a pour effet d'ouvrir une boîte de dialogue prévenant l'utilisateur qu'un script a demandé des privilèges de sécurité étendus et de lui demander s'il autorise (cette fois, chaque fois, ou pas du tout) cette extension de privilèges. Mais pourquoi faire cela ? Tout simplement parce que notre composant n'est pas signé. Je ne vous en dirai pas plus sur le sujet ; s'il me reste encore de l'encre, je ferai un petit papier sur le sujet plus tard. En attendant, vous pouvez jeter un œil sur [10].

Les composants de Mozilla

Bien entendu, il serait trop bête de réinventer la roue. Mozilla est une véritable mine de composants que vous pouvez (parce que maintenant vous savez ;)) utiliser pour vos propres créations. Le tout est de savoir où chercher.

La première solution consiste à installer le "XPCOM Component Viewer" [9]. Il s'agit d'une application écrite en XUL permettant de lister l'ensemble des composants et interfaces associés accessibles.

Vous pouvez aussi parcourir la page "Linux Cross Reference" sur le site de Mozilla où vous retrouverez l'ensemble des sources (`idl`, `cpp`, `js`) des composants fournis en standard. Enfin, sachez que sur le site **xulplanet.org** se trouve toute la documentation de référence de XPCOM [8].

Et Windows ?

Si vous achetez Linux Mag pour faire joli sur la table du salon, mais qu'en fait vous ne pouvez pas lâcher votre Windows, rassurez-vous, vous pouvez très bien créer des composants XPCOM aussi ;)

Pour cela, vous aurez besoin de plusieurs packages :

- Les bibliothèques `libIDL` et `glib` que vous pourrez trouver ici : **http://ftp.mozilla.org/pub/mozilla/libraries/win32/**
- Le SDK que vous trouverez sur le FTP de Mozilla : **http://ftp.mozilla.org/pub/mozilla/releases/mozilla1.4/**

Pour le reste, c'est une histoire d'amour entre vous et visual C++, et je vous laisse lire le chapitre correspondant de l'excellent ouvrage "Creating XPCOM Components" [11].

Conclusion

Nous avons maintenant deux nouveaux composants et nous savons comment les utiliser... Je vous laisse terminer. En effet, vous aurez certainement compris qu'il ne manque plus qu'un peu de XUL pour que Mozilla soit capable de vous informer sur d'éventuelles signatures GPG de pages HTML. Et ça tombe plutôt bien que nous ayons un composant qui renvoie du XML et l'autre capable de le parser ;). Il ne vous reste plus qu'à vous lancer pour faire de Mozilla ce que vous souhaitez. Dans le pire des cas, et le meilleur des mondes, vous ne bouderez plus lors de vos prochains plantages et trouverez le courage de contribuer au projet sans avoir peur de rechercher (et corriger) les bogues.

Gregoire Lejeune
greg@webtime-project.net

Références

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

[2] http://aspn.activestate.com/ASPN/Downloads/Komodo/PyXPCOM/

[3] http://plxpcom.mozdev.org/

[4] http://rbxpcom.mozdev.org/

[5] http://www.w3.org/TR/xpath

[6] http://www.vilya.org/gpg/gpg-intro.html

[7] http://www.mozilla.org/scriptable/xpidl/idl-authors-guide/

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

[9] http://www.hacksrus.com/~ginda/cview/

[10] http://www.mozilla.org/projects/security/components/signed-scripts.html

[11] http://www.mozilla.org/projects/xpcom/book/cxc/html/appA.html

Les sources de cet article sont disponibles ici : http://greg.webtime-project.net/articles.php (et sur le CD du magazine)

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/.