



Technologies Web

Intégration du plugin Google Gears dans une application desktop xulrunner

Introduction



Les applications Web ont énormément évolué ces deux dernières années, tout particulièrement depuis l'avènement du concept Web 2.0. Nous disposons de plus en plus d'applications Web dynamiques, élégantes et riches en fonctionnalités se rapprochant de leurs homologues Desktop. Néanmoins, il reste

encore beaucoup d'applications qui ne peuvent avoir leur équivalent en version web car les navigateurs sont encore limités par leur nature même. Comme à son habitude, Google se démarque par son dynamisme et son innovation en proposant une extension aux navigateurs afin de gérer ces nouvelles problématiques : Google Gears.

Notre article va plus loin en proposant l'intégration des mécanismes Gears dans une RDA¹ (Rich Desktop Application) développée sur le framework de Mozilla : xulrunner. Le but de vous initier au développement d'application lourdes bénéficiant du meilleur des innovations web : les technologies du présent n'ont qu'à bien se tenir.

Google Gears ouvre de nouvelles perspectives aux applications web

Gears se présente comme un module permettant d'étendre les fonctionnalités du navigateur Internet « bridé » dans son mode client serveur. L'idée derrière Gears est de proposer tout type de nouvelles fonctionnalités, en commençant par :

- L'installation d'une base de donnée locale pour gérer les données dynamiques
- Un serveur web interne de gestion de ressources statiques
- Des méthodes de conception d'applications AJAX standardisées

Les gains apportés par l'extension sont directs :

- Gestion de la perte de connexion réseau avec continuité des services proposés
- Nouvelles perspectives de développements d'applications Web
- Amélioration des performances car la boule client serveur distant est brisée

1 <http://www.fredcavazza.net/2006/10/15/apres-les-ria-les-rda/>

Ce framework prend la forme d'une extension pour les navigateurs Mozilla Firefox (version 1.5 ou ultérieur) et Internet Explorer (version 6.0 ou ultérieur). À l'heure actuelle, plusieurs versions de Google Gears sont disponibles. La plus récente à l'écriture de ce document est la version 0.2 (0.2.2.0).

Autre bonne nouvelle : Google a choisi d'ouvrir le code de l'application sous la licence Open Source BSD². Cela signifie que tout le monde peut consulter le code source, participer aux nouvelles fonctionnalités, envoyer des correctifs... Ce mode de travail est avantageux pour tous les utilisateurs qui peuvent se reposer sur la communauté existante autour du projet.

■ ■ ■ Google Gears : un framework au sommet des technologies Web

La nouvelle « killer application » de Google est avant tout un mélange savamment composé de plusieurs technologies puissantes :

- Ajax : pour mettre à jour les interfaces en temps réel en allant chercher l'information nécessaire sur le serveur
- SQLite : base de données légère pour le stockage des données locales
- JSON : format JavaScript d'échange de données entre les différents mondes applicatifs

■ ■ ■ La gestion d'un site en mode déconnecté

Les sites Web d'aujourd'hui proposent des fonctionnalités uniques, intimement liées à la connexion permanente à l'Internet. Sans connexion, le site, et donc ses fonctionnalités, est tout simplement inaccessible.

L'extension Google Gears, installée sur le navigateur du client, permet de proposer à l'utilisateur une continuité de certains de ses services. Il va pouvoir continuer à naviguer et utiliser tout ou partie du site qui sera copié sur son disque dur local. Cette continuité de services est à la fois intéressante pour l'utilisateur, car il peut continuer à utiliser le site en mode hors-ligne, et pour les propriétaires du site, car ces utilisateurs continuent à utiliser leur site et non pas un outil de type client lourd. Voici comment s'effectuent les transactions entre les modes « connecté » et « déconnecté » :

² <http://www.opensource.org/licenses/bsd-license.php>

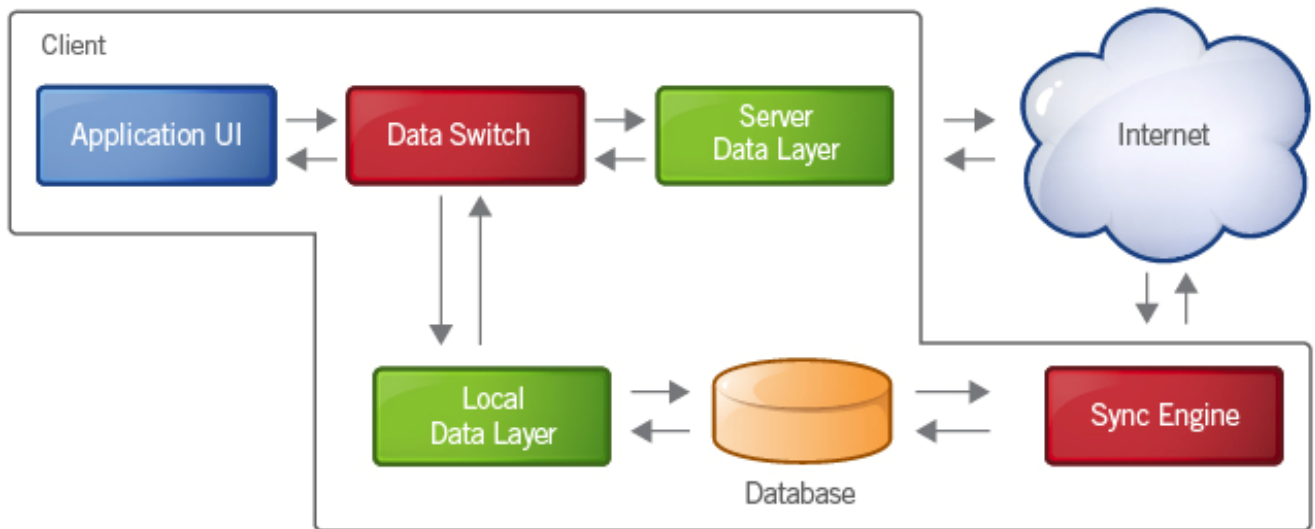


Figure 1 : Architecture des composants Google Gears.

Démonstration de l'utilisation de Google Gears dans une application concrète

Nous allons, au fil de l'article, construire ensemble une application Internet mettant en évidence la puissance de Google Gears dans une application web locale. Afin de démontrer l'utilité de la gestion du manque de connectivité, nous allons travailler sur une application de type RDA. Ce type d'application allie la robustesse d'une application cliente lourde, tout en se reposant sur les technologies Internet. Le framework utilisé est celui de Mozilla : XPFE (Cross Platform Front End)³, avec l'utilisation du projet xulrunner.

L'application test est développée en XUL, qui est le langage utilisé par Mozilla pour mettre en forme les interfaces de ses applications. Soyez rassuré si vous ne connaissez pas encore le XUL, il s'agit d'un langage compatible XML largement ressemblant au XHTML.

Les fichiers sont hébergés sur un serveur web distant. L'application xulrunner va consulter ce site en mode connecté. Nous allons voir comment ces mêmes pages peuvent être affichées et utilisées lorsque la connexion au site n'existe plus.

Vous pouvez télécharger le code source de l'application ICI, fonctionnant dans un environnement Windows.

³ <http://www.journaldunet.com/developpeur/tutoriel/dht/070725-xpfe-mozilla-alexandre/0.shtml>

Installation du module Gears

Pour faire fonctionner Google Gears dans un environnement xulrunner, nous avons plusieurs étapes à réaliser : (les instructions sont aussi disponibles ici⁴)

1. Installation de Gears dans le framework Mozilla :

Comme cité dans l'introduction, vous pouvez télécharger la dernière version⁵ (v0.2) de Gears. Fermez tous les navigateurs, et lancez l'installation. Sous Windows, les fichiers sont copiés dans « C:\Program Files\Google\Google Gears », ce répertoire contenant l'extension Firefox qui sera elle-même utilisée dans xulrunner.

2. Mise à disposition de l'extension Gears pour son application xulrunner :

Il faut éditer la base de registre pour ajouter une clé identifiant l'extension par rapport à l'application cible (« GearsRunner » dans l'exemple).

```
[HKEY_LOCAL_MACHINE\SOFTWARE\GearsRunner]
[HKEY_LOCAL_MACHINE\SOFTWARE\GearsRunner\Extensions]
"{000a9d1c-beef-4f90-9363-039d445309b8}"="C:\\Program Files\\Google\\Google
Gears\\Firefox\\"
```

3. Enregistrement dans Gears de l'application cible :

L'extension Google Gears est initialement développée pour IE et Firefox. Ce dernier étant basé sur les mêmes technologies que celles de xulrunner, il suffit de rajouter une nouvelle application cible dans le code l'extension Gears (fichier « C:\Program Files\Google\Google Gears\Firefox\install.rdf ») en concaténant le texte suivant :

```
<em:targetApplication>
  <Description>
    <em:id>gearsrunner@edis-consulting.com</em:id>
    <em:minVersion>0.1</em:minVersion>
    <em:maxVersion>1.0</em:maxVersion>
  </Description>
</em:targetApplication>
```

4. Activer le système de gestionnaire d'extension dans xulrunner : (tâche déjà réalisée dans les sources à télécharger)

Rajoutez dans le fichier application.ini de l'application xulrunner :

```
[XRE]
```

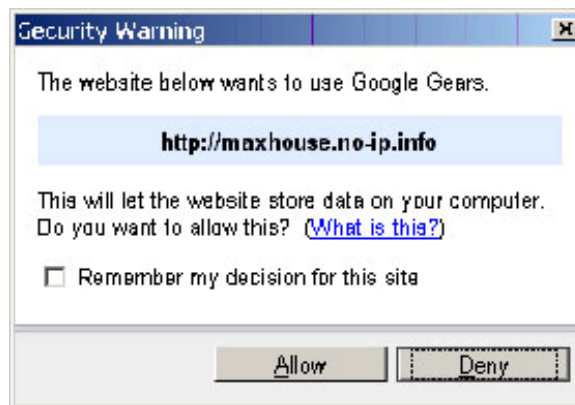
⁴ <http://www.iosart.com/blog/2007/06/05/install-google-gears-in-a-xulrunner-app-in-3-quick-steps/>

⁵ <http://code.google.com/p/google-gears/downloads/list>

```
EnableExtensionManager=1
```

Si malgré tout cela ne fonctionne pas, essayer de supprimer les fichiers de cache de xulrunner dans C:\Documents and Settings\[votre_utilisateur]\Application Data\GearsRunner

L'application GearsRunner se lance en double-cliquant sur le fichier « demo-gears.exe ». Au premier lancement, une fenêtre de sécurité va demander si l'on veut permettre l'accès à ces fichiers :



Acceptez et ouvrez bien les yeux.

Initialisation du kit de développement Gears dans notre application Desktop

A partir de ce moment, nous allons développer dans le fichier principal de l'application : gears.xul. Pour commencer, il faut déclarer que l'on veut utiliser les fonctionnalités de Google Gears. Cela s'opère en incluant le fichier ***gears_init.js***⁶ dans la page XUL. Ce fichier permet d'initialiser Google Gears afin de pouvoir l'utiliser depuis JavaScript.

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

Note : ce fichier devrait à terme disparaître des applications utilisant Gears.

Une fois ce fichier chargé, nous pouvons utiliser les API du framework. Il est néanmoins conseillé de tester l'intégration du « plugin » Google Gears avec notre application. Nous le faisons lors de l'appel à la fonction JavaScript « onload » qui est appelée une fois la page chargée :

gears.xul :

```
<window ...  
  onload="onLoad()"
```

⁶ http://code.google.com/apis/gears/resources/gears_init.js

```
...>
```

index.js :

```
var isGearsLoaded = false;

function onLoad() { ...
if (!window.google || !google.gears) {
    alert("merci d'installer l'extension google gears");
    window.open("http://gears.google.com/?action=install&message=Mon+message+d%E2%80%99ac
    cueil >&return=http://www.edis-consulting.com", "_blank");
    return;
}
isGearsLoaded = true;
...}
```

Grâce à ce test, nous allons prévoir de désactiver les fonctionnalités spécifiques à Google Gears du reste de l'application quand Gears n'est correctement chargé (variable `isGearsLoaded`).

Création de la « factory »

Le framework étant disponible et initialisé, nous avons alors accès aux différents composants. Afin d'instancier les objets du modèle, il faut passer par un point central de création qui est l'objet usine ***google.gears.factory*** et sa méthode ***create***.

Les différentes briques du framework sont représentées dans ce tableau :

Module	Nom de la classe	Présent dans la version
Database	beta.database	0.1
HttpRequest	beta.httprequest	0.2
LocalServer	beta.localserver	0.1
Timer	beta.timer	0.2
WorkerPool	beta.workerpool	0.1

Zoom sur les différents modules proposés

Database

Présentation

Ce composant met à la disposition du code JavaScript une base de données de type relationnel, provenant du projet Open Source SQLite⁷. L'intérêt est de pouvoir continuer à utiliser des données dynamiques quand bien même l'utilisateur est connecté ou déconnecté d'Internet.

Afin d'obtenir une instance de cet objet on procède comme précédemment :

```
var db = google.gears.factory.create('beta.database', '1.1');
```

Nous pouvons maintenant établir une connexion à la base de données et exécuter des requêtes SQL :

```
db.open('demogears');  
db.execute('create table if not exists twitmsg (data TEXT, timestamp INT)');
```

Utilisation

Pour l'exemple d'implémentation, nous allons faire saisir des données à l'utilisateur qu'il va falloir synchroniser ou non en fonction de l'état de connectivité réseau.

Voici le code XUL permettant d'afficher un champ texte et un bouton :

gears.xul :

```
<textbox id="message" multiline="true" value=""/>  
<button label="envoyer" onclick="onSubmit()" />
```

Dans le code JavaScript maintenant, nous allons étudier ce qu'il se passe quand on clique sur le bouton :

mygears.js :

```
function onSubmit() {  
    if (!google.gears.factory || !db) {  
        alert("Google Gears n'est pas chargé.");  
        return;  
    }  
  
    var elm = document.getElementById('message');  
    var message = elm.value;  
    var currTime = new Date().getTime();  
    db.execute('INSERT INTO twitmsg VALUES (?, ?)', [message, currTime]);  
  
    // Reset the text message  
    elm.value = '';  
    showQueue();  
}
```

⁷ <http://www.sqlite.org/>

```
}
```

On vérifie tout d'abord que Gears est installé, puis on récupère le contenu du champ texte pour l'insérer dans la base de données locale. Nous pourrions améliorer cette fonctionnalité en détectant la présence de la connexion en temps réel et le cas échéant manipuler la base de données locale ou distante.

HttpRequest

Présentation

Ce module fournit au développeur la classe « HttpRequest » supportant le standard W3C. Cette dernière permet d'appeler des pages sur Internet sans avoir à recharger toute la page. Cette méthode est la base du fonctionnement AJAX. Elle a été ajoutée dans Google Gears afin d'unifier son utilisation pour le développeur.

Utilisation

Ce module de Google Gears va nous permettre de développer une nouvelle fonctionnalité : détecter en temps réel la présence de la connectivité réseau.

Le principe retenu est de faire un appel AJAX sur notre serveur toutes les X secondes, et regarder s'il y a eu une erreur ou si le contenu nous est retourné. Afin de ne pas « trop » surcharger le serveur, nous allons faire des requêtes HTTP de type « HEAD ».

La fonction JavaScript ci-dessous va être appelée périodiquement (on verra dans la suite de l'article que Gears nous propose une solution à ce besoin)

```
function detect() {
request.open('HEAD', 'http://maxhouse.no-ip.info/index.html?id=' + Math.random(9999999));
  request.onreadystatechange = function() {
    try {
      if (request.readyState == 4) {
        if (request.status == 200) {
          timer.setTimeout(onInternetOk, 0);
        }
        else {
          timer.setTimeout(onInternetKo, 0);
        }
      }
    }
    catch (ex) {
      timer.setTimeout(onInternetKo, 0);
    }
  };
  request.send();
}
```

Ici nous appelons une certaine adresse, jugée unique pour chaque appel via la gestion d'un nombre aléatoire dans l'URL (sinon nous avons le système de cache qui prend le dessus et nous trompe). En cas de succès de la réponse du serveur, nous appelons la méthode globale « onInternetOk » ; cette dernière affiche une image de couleur verte. Dans le cas contraire, nous appelons la méthode globale « onInternetKo » qui affiche une image de couleur rouge.

Nous savons à partir de maintenant si l'utilisateur est connecté au serveur web distant, et nous pouvons mettre à jour les comportements de l'application corrélés à chacun de ces états.

LocalServer

Présentation

Le module « LocalServer » est la brique de Google Gears permettant à l'utilisateur de pouvoir continuer à naviguer sur le site une fois la connexion Internet perdue. Il s'agit tout simplement d'un système de cache sur les protocoles et HTTP et HTTPS, permettant de fournir localement des ressources statiques du site (pages HTML, JavaScript, feuilles de styles, images...)

L'instanciation de l'objet correspondant se fait comme suit :

```
// instanciation de la classe localserveur
var localSrv = google.gears.factory.create('beta.localeserver', '1.1') ;

// création d'un conteneur
var store = localServer.createStore('stockage');
```

Utilisation

Cette partie est la pièce maîtresse pour gérer le contenu hors ligne de l'application. Nous allons traiter le contenu de façon manuelle, via l'ajout de boutons :

```
<hbox>
  <button oncommand="createStore()" label="Capture" />
  <button oncommand="removeStore()" label="Erase" />
  <image src="" id="connectivity" width="50" height="50" />
</hbox>

function createStore() {
  if (!window.google || !google.gears) {
    alert("merci d'installer l'extension google gears");
    return;
  }

  store.manifestUrl = "manifest.json";
  store.checkForUpdate();

  var timerId = window.setInterval(function() {
    if (store.currentVersion) {
      window.clearInterval(timerId);
    }
  }, 1000);
}
```

```
        } else if (store.updateStatus == 3) {
            textOut("Erreur: " + store.lastErrorMessage);
        }
    }, 500);
}

function removeStore() {
    if (!window.google || !google.gears) {
        alert("You must install Google Gears first.");
        return;
    }
}

localServer.removeManagedStore(STORE_NAME);
}
```

Comme vous pouvez le remarquer, le fichier « manifest.json » contient la liste des fichiers à mettre dans le cache. Notez qu'à chaque changement d'au moins un des fichiers du cache, il faut régénérer le nom de la version du cache.

Timer

Présentation

Google Gears propose dans ce module une classe « Timer » regroupant la gestion du temps. Il est à noter que ce module est identique à ce que propose par défaut votre navigateur, mis à part que l'implémentation respecte les spécifications du HTML version 5.

Utilisation

```
// instantiation de la classe timer
var timer = google.gears.factory.create("beta.timer", "1.0");
```

Nous utilisons le module de « timer » pour appeler le test de la connectivité vu plus haut. Il faut le relancer toutes les X secondes. Nous utilisons ce code au chargement de l'application :

```
function detectInternetConnectivity() {
    timer.setInterval("detect()", 5000);
}
```

■■■ Conclusion

Google fait un cadeau extraordinaire à tous les développeurs d'un framework léger et lève une des limitations critiques des applications Web. Bien que disponible en version beta, cette boîte à outil bénéficie déjà d'une certaine maturité. Son avenir est très prometteur de part les potentialités offertes comme nous avons pu le constater dans les exemples présentés.

L'intégration des fonctionnalités Gears avec les technologies RDA dépassent les limites connues d'Internet depuis sa naissance. Il s'agit d'une nouvelle révolution pour les applications lourdes, et le framework de Mozilla bénéficie d'un nouvel avantage dans la guerre des technologies RDA.

Si vous développez des sites Internet, Google Gears devient incontournable pour ceux qui désirent toujours être de plus en plus disponibles pour ses utilisateurs. Les applications qui trouvent le plus de potentiel dans la gestion du mode « offline » ne seraient-elles pas dans les machines à haute mobilité ? Quelque chose me dit que le nombre de sites et d'applications supportant Google Gears va vite augmenter dans les prochains mois. Êtes-vous prêts de votre côté à intégrer Google Gears dans les fonctionnalités que vous proposez sur votre site ?

Pour le mot de la fin, je vous poserais la question aventureuse : qu'aimeriez-vous que votre navigateur préféré puisse faire, sortant du cadre strict de l'application client serveur qui affiche des contenus Web ? Voilà le réel enjeu de Google Gears. Alors n'hésitez pas, participez et communiquez vos idées à la communauté.

Maxime ALEXANDRE, EDIS Consulting