

Introduction à gestion d'un site en mode offline avec Google Gears



par Mourad Lafer ([Edis-Consulting](#)) Maxime Alexandre ([Page d'Accueil](#))

Date de publication : 02/10/2007

Google Gears est un des produits Google les plus récents. Il propose de créer des applications web fonctionnant également en mode deconnecté.

Cet article nous présente le principe générale du fonctionnement de Google Gears.

Introduction

- 1 - Présentation de Google Gears
- 2 - Google Gears ouvre de nouvelles perspectives aux applications web
- 3 - Google Gears : un framework au sommet des technologies Web
 - 3.1 - Initialisation du kit de développement
 - 3.2 - Création de la factory
 - 3.3 - Zoom sur les différents modules proposés
 - 3.3.1 - Database
 - 3.3.2 - HttpRequest
 - 3.3.3 - LocalServer
 - 3.3.4 - Timer
 - 3.3.5 - WorkerPool

4 - Conclusion

Liens

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. Le meilleur exemple est la connectivité au réseau Internet. Que se passe t-il lorsqu'on perd sa connexion Internet ? Que peut-on faire pour continuer à travailler ou utiliser partiellement sa solution applicative en mode déconnecté ? Comme à son habitude, Google se démarque par son dynamisme et son innovation en proposant une extension aux navigateurs afin de gérer cette problématique : Google Gears.

1 - Présentation de Google Gears

Sur le blog dédié à ce framework, nous pouvons lire que " *Google Gears est une amélioration incrémentale au Web d'aujourd'hui. Il apporte juste le nécessaire en fonctionnalité AJAX de sorte à permettre aux applications Web actuelles de fonctionner en mode offline* " .

Google Gears répond à un besoin qui est " malheureusement " aujourd'hui peu commun : la connectivité permanente au plus grand réseau mondial. Que se passe-t-il quand on ne peut pas se passer d'un besoin ? On le simule#

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 (0.1.56.0).

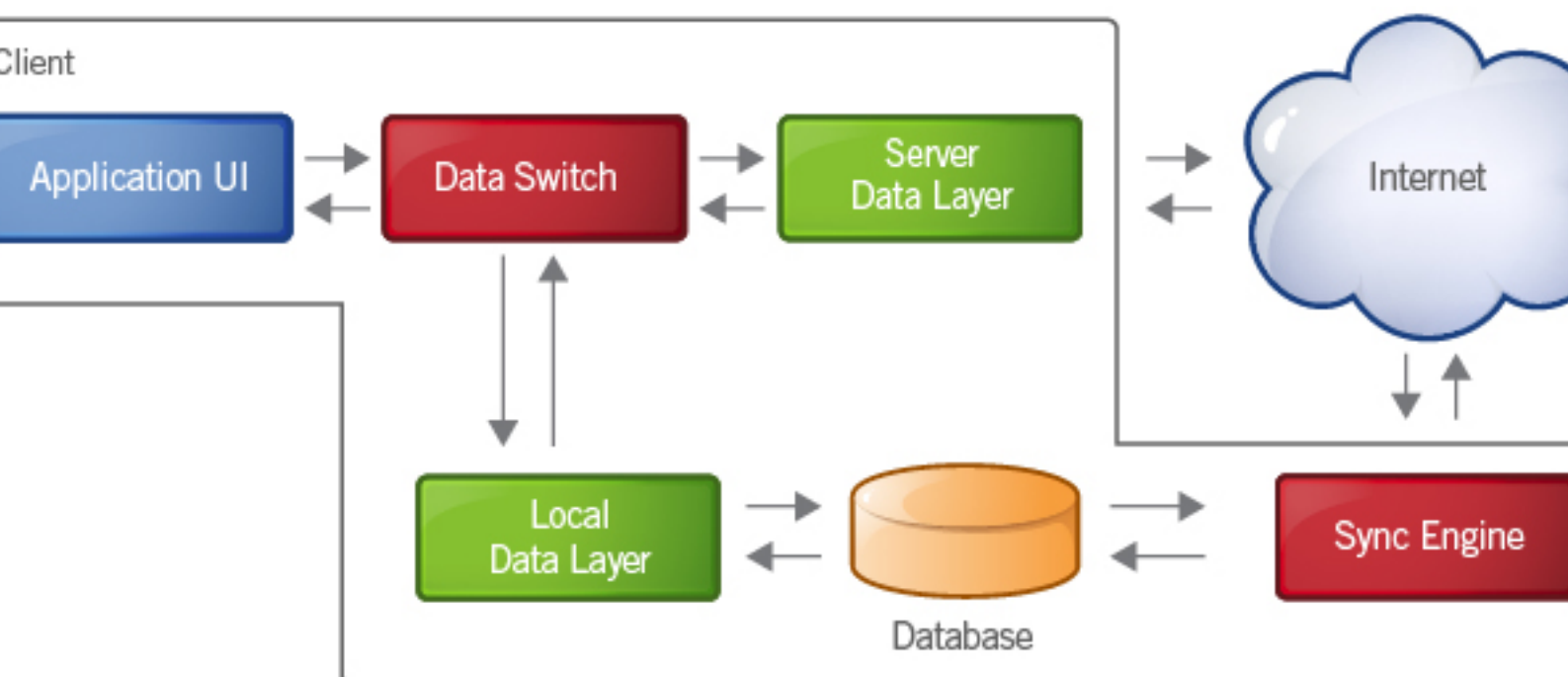
Autre bonne nouvelle : Google a choisi d'ouvrir le code de l'application sous la licence Open Source BSD [\[1\]](#) . Cela signifie que tout le monde peut consulter le code source, participer aux nouvelles fonctionnalités, envoyer des correctifs# Ce mode de travail est avantageux pour tous les utilisateurs qui peuvent se reposer sur la communauté existante autour du projet.

2 - Google Gears ouvre de nouvelles perspectives aux applications web

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 localement 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.

Un court schéma vaut mieux qu'un long discours, voici comment s'effectuent les transactions entre les modes " connecté " et " déconnecté " :



Architecture des composants Google Gears

Ici nous voyons qu'il existe une détection de la connectivité réseau qui va choisir entre afficher les données provenant d'Internet et afficher les informations stockées localement, ou les deux.

Le navigateur " Application UI " se connecte au module " Data Switch " pour savoir ce qu'il peut afficher. Ce dernier :

- si la connexion réseau est présente, aller chercher les données sur le serveur " Server Data Layer "
- si des données locales sont présentes, appeler le module " Local Data Layer " pour les récupérer.

La dernière brique à utiliser est le " Sync Engine " qui permet de choisir quand synchroniser les données locales sur le serveur en ligne.

Voyons maintenant ce que propose Google Gears comme API pour arriver à ce résultat.

3 - 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 juste nécessaire sur le serveur
- SQLite : base de données légère et performante pour le stockage des données locales
- JSON : format Javascript d'échange de données entre les différents mondes

Ces technologies sont regroupées sous plusieurs modules distincts :

- Localserver : un mini serveur web local ayant le rôle de cache de contenu qui intercepte et gère les accès aux ressources (html, images, css, #) en mode sans connexion au serveur.
- Database : une base de données relationnelle SQL locale (moteur SQLite) accessible par le navigateur qui stocke les données générées par l'utilisateur.
- Worker pool : un outil de synchronisation qui assure la communication entre les deux premiers composants et la mise à jour des données une fois la connexion au serveur distant rétablie. En outre, il permet l'exécution du JavaScript en tâche de fond. Ces scripts s'exécutant en mode asynchrone, ne bloquent pas l'interface utilisateur ce qui améliore considérablement les performances de l'application.

3.1 - Initialisation du kit de développement

Pour commencer, il faut déclarer dans les pages du site que l'on veut utiliser les fonctionnalités de Google Gears. Cela s'opère en incluant le fichier gears_init.js dans la page HTML. Nous allons directement utiliser le fichier fourni, disponible sur le site de Google. Ce fichier permet d'initialiser Google Gears afin de pouvoir l'utiliser depuis Javascript.

```
<script type="text/javascript"
src="http://code.google.com/apis/gears/tools/gears_init.js"></script>
```

Une fois ce fichier chargé, nous pouvons utiliser les API du framework. Il est néanmoins conseillé de tester la présence du " plugin " Google Gears sur le navigateur client. Nous le faisons lors de l'appel à la fonction Javascript " onload " qui est appelée une fois la page chargée :

```
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%99accueil
&return=http://www.monsite.com", "_blank");
return;
}
```

Ici nous testons la présence des variables principales du framework, en choisissant de proposer à l'utilisateur d'installer Google Gears sur son navigateur préféré s'il ne l'a pas déjà fait. Grâce à ce test, nous allons prévoir de désactiver les fonctionnalités spécifiques à Google Gears pour ceux qui n'ont pas ou ne veulent pas l'installer.

3.2 - 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

Voici quelques exemples utilisant la " factory " dans le code Javascript :

```
// pour instancier la base de données
var db = google.gears.factory.create('beta.database', '1.0');

// pour instancier le serveur de cache local
var localServer = google.gears.factory.create('beta.localserver', '1.1');
```

Maintenant que nous savons comment instancier les différents représentants des composants du framework, nous allons nous intéresser un peu plus en détail à chacun d'eux.

Pour que le site ou service en ligne tire partie de la stratégie " offline " de Google Gears, les développeurs disposent de primitives de haut niveau entièrement en JavaScript. Elles permettent d'accéder aux trois composants présentés ci-dessus.

Dans la documentation API Developer's Guide [2], sont détaillées les classes du framework.

3.3 - Zoom sur les différents modules proposés

3.3.1 - Database

Ce composant met à la disposition du code JavaScript une base de données de type relationnel, provenant du projet Open Source SQLite [3]. 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('twitygears');
db.execute('create table if not exists twitmsg (data TEXT, timestamp INT)');
```

La méthode **execute** permet des requêtes paramétrées ce qui évite l'injection d'attaques SQL. Ce type d'attaque, fréquent dans les applications Web, correspond à une erreur dans la protection des données d'entrées de la requête. Cela "permet" à celui qui connaît cette faille d'exécuter n'importe quelle requête sur la base de données (imaginez les conséquences d'un " DELETE * FROM# " sur les données).

```
// insertion
db.execute('INSERT INTO twitmsg VALUES (?, ?)', [message, currTime]);
```

```
// sélection
var rs = db.execute('SELECT * FROM twitmsg ORDER BY timestamp');

while (rs.isValidRow()) {
    alert(rs.field(0) + " correspond à " + rs.fieldName(0));
    rs.next();
}
rs.close();
```

Le résultat renvoyé par une requête type SELECT est de type **ResultSet** . Nous pouvons itérer dessus et lire les données de chaque ligne. Les méthodes **field()** et **fieldName()** nous permettent de lire la valeur et le nom du champ indexé.

Comme nous le connaissons de certaines bases de données, Google Gears rajoute lui aussi une colonne **ROWID** à chaque table. Ainsi après l'ajout d'une nouvelle ligne de données, l'attribut **lastInsertRowid** contenant la valeur du dernier ROWID est incrémenté automatiquement.

Il est à noter que SQLite peut supporter le mode transactionnel afin de garder un système de données cohérent.

Pour plus d'informations sur l'api database: [\[4\]](#)

3.3.2 - HttpRequest

Note : Cette fonctionnalité est uniquement disponible dans la version 0.2.

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

```
var request = google.gears.factory.create('beta.httprequest', '1.0');
request.open('GET', '/index.html');
request.onreadystatechange = function() {
    if (request.readyState == 4) {
        console.write(request.responseText);
    }
};
request.send();
```

Pour plus d'informations sur l'api httprequest: [\[5\]](#)

3.3.3 - LocalServer

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.localserver', '1.1');
```

L'interaction avec le composant LocalServer s'effectue par le biais de conteneur à URL de type **ResourceStore** . Ce conteneur d'URL peut contenir un nombre quelconque d'URL et une application peut créer un nombre quelconque de containers. Vous aurez compris que vous pouvez mettre en cache toutes les ressources de votre site.

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

Une fois le conteneur disponible, nous pouvons y ajouter les URL et leur assigner une méthode de type " callback " qui sera invoquée à chaque fois que ces URL sont sollicitées.

```
var localServer = google.gears.factory.create('beta.localserver', '1.1');
var store = localServer.createStore('stockage');

var filesToCapture = [
  location.pathname,
  'index.js',
  'style.css'
];

function captureCallback(url, success, captureId) {
  alert(url + 'captured: ' + success ? 'success' : 'failure');
}

int captureId = store.capture(filesToCapture, captureCallback);
```

La signature de la fonction " captureCallback " comporte trois paramètres :

- 1 le premier est l'URL appelée
- 2 le second paramètre de type booléen indique le statut de l'opération de chargement
- 3 le troisième paramètre joue le rôle de jeton et est passé par la suite à la méthode **abortCapture** du container afin de stopper la tâche de fond asynchrone de capture.

Lorsque le nombre d'URL est trop important et qu'on ne peut les gérer individuellement, on utilise l'objet **ManagedStore** . La manipulation de cet objet s'effectue à l'identique de son homologue **ResourceStore** . Cependant, la déclaration est faite à l'aide d'un fichier **manifest** au format générique **JSON** .

```
{
// version du format du fichier manifest
"betaManifestVersion": 1,

// version des ressources contenues dans ce manifest
"version": "v1.0",

// les url à mettre dans le cache
"entries": [
  { "url": "managed_store.html",
    "src": "managed_store_v1.html" },
```

```
{ "url": "managed_store.js",  
  "src": "managed_store_v1.js" },  
{ "url": "managed_store_demo_utils.js" },  
{ "url": "sample.css", "src": "../styles.css" },  
{ "url": "sample.js", "src": "../index.js" },  
{ "url": "gears_init.js", "src": "../gears_init.js" } ]  
}
```

Pour plus d'informations sur l'api localserver: [\[6\]](#)

3.3.4 - Timer

Note : Cette fonctionnalité est uniquement disponible dans la version 0.2.

Google Gears propose dans ce module une classe "Timer" regroupant la gestion du temps et d'appel à des fonctions Javascript une fois un temps écoulé. 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.

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

Prenons l'exemple d'une méthode "checkEmails()" qui appellerait un webservice pour récupérer des messages. Pour appeler cette méthode toutes les 42 secondes, nous pouvons utiliser la méthode bien connue "setTimeout" :

```
var timer = google.gears.factory.create("beta.timer", "1.0");  
timer.setTimeout(checkEmails, 42000);
```

Pour plus d'informations sur l'api timer: [\[7\]](#)

3.3.5 - WorkerPool

Le module WorkerPool permet d'exécuter du code JavaScript en tâche de fond ce qui permet de lancer certains traitements sans pour autant bloquer l'exécution et l'affichage de la page principale.

```
// instantiation de la classe workerpool  
var wp = google.gears.factory.create('beta.workerpool', '1.1') ;
```

Une fois la classe initialisée, on affecte à l'attribut **onmessage** la fonction qui sera invoquée lorsque la file d'attente reçoit un message. Les échanges se font à travers des messages textes ; des objets peuvent donc être échangés en utilisant la sérialisation JSON.

```
var wp = google.gears.factory.create('beta.workerpool', '1.1') ;  
  
// fonction appelée de la réception d'un message dans la file d'attente  
wp.onmessage = function(a, b, message) {  
  alert('Message reçu ' + message.sender + ': ' + message.text);  
}
```

```
function evalHandler(msg, sender) {
  var result = eval(msg);
  google.gears.workerPool.sendMessage(String(result), sender)
}
// creation du 'producteur'
var workerCode = String(evalHandler) + 'google.gears.workerPool.onmessage = evalHandler;';
var workerId = wp.createWorker(workerCode);

// envoi un message
wp.sendMessage('salut', workerId);
```

Les processus **worker** sont créés par le processus parent à l'aide de la méthode **createWorker** disponible au sein de l'objet **WorkerPool** qui retourne un **handle** spécifique. Cette méthode de création ne prend qu'un seul argument, contenant le code que le processus doit exécuter. L'exemple ci-dessus présente les différentes étapes à suivre. Il faut remarquer que le **worker** ne peut faire de manipulation directe sur le DOM de la page principale. Il est obligé de renvoyer le résultat de son traitement dans le cas où l'on désire l'afficher.

Pour plus d'informations sur l'api wokerpool: [\[8\]](#)

4 - 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.







Nous pouvons encore aller plus loin en regardant les prémices des futures applications de type RIA (Rich Internet Application) intégrant complètement Google Gears dans leur système :

- Google Widget Toolkit : le framework de développement Java de Google intègre directement l'API de Google Gears
- Mozilla Xulrunner : la couche basse de Firefox permettant de développer ses propres applications web directement en tant qu'application de type client lourd est de facto utilisable avec Google Gears [9] .

Google Gears devient incontournable pour beaucoup de sites Internet désirant toujours être de plus en plus disponibles pour les utilisateurs. Nous pouvons aussi nous demander l'intérêt d'une telle technologie quand on sait que presque tout le monde dispose d'une connexion permanente à Internet. 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 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 cliente qui affiche des contenus Web ? Voilà le réel enjeu de Google Gears. Alors n'hésitez pas, participez et communiquez vos idées à la communauté.

Liens

- [1]  <http://www.opensource.org/licenses/bsd-license.php>
- [2]  <http://code.google.com/apis/gears/>
- [3]  <http://www.sqlite.org/>
- [4]  http://code.google.com/apis/gears/api_database.html
- [5]  http://code.google.com/apis/gears/api_httprequest.html
- [6]  http://code.google.com/apis/gears/api_localserver.html
- [7]  http://code.google.com/apis/gears/api_timer.html
- [8]  http://code.google.com/apis/gears/api_workerpool.html
- [9]  <http://www.iosart.com/blog/2007/06/05/install-google-gears-in-a-xulrunner-app-in-3-quick-steps/>

Mourad Lafer est ingénieur d'étude senior, architecte en développement pour EDIS Consulting.

Maxime Alexandre est chef de projet et ingénieur d'étude spécialisé dans les technologies Web.



 EDIS Consulting

