Google Gears

Dion Almaer

code.google.com
ajaxian.com
Offline Web via Open Web

- Why just solve this problem for Google?
- Why not solve it for others?

*Solution:* Make it open source with a liberal license
  - New BSD
Why?

“How often are you on a plane?”

- Reliability
  - 1% of downtime can hurt at the wrong time

- Performance
  - Local acceleration

- Convenience
  - Not having to find a connection

- You are offline more than you think!
What is the philosophy?

• One application, one URL
• Seamless transitions between online and offline
• Ability to use local data, even when online
• Available to all users on all platforms

...and a pony
What is the philosophy?

Browser plugin: IE, Firefox, Safari (almost!)
What is the philosophy?
What is the philosophy?

*Do for offline what XMLHttpRequest did for web apps*

- **Ajax Libraries**
  - Dojo, jQuery, Prototype, GWT

- **Gears Libraries**
  - Dojo Offline, GWT

- **XMLHttpRequest**

- **Gears**

- **Open Web**
Ajax Architecture

Internet

Client

XHR

UI
Gears Architecture

- Read and write using local store
- Changes are queued for later synchronization
- Server communication is completely decoupled from UI actions, happens periodically whenever there is a connection
“Web 1.0” Traffic Pattern

[Diagram showing the flow of traffic between client and server over time]
Ajax Traffic Pattern
Gears Traffic Pattern

[Diagram showing the traffic pattern between Client, Gears, and Server over time.]

Dion Almaer — Google Gears
What are the pieces?

**LocalServer**
Cache and serve application resources (HTML, JavaScript, images, etc.) locally

**Database**
Store data locally in a fully-searchable relational database

**WorkerPool**
Make your web applications more responsive by performing resource-intensive operations asynchronously
Database

Embedded using SQLite
Contributed Full Text Search

```
var db = google.gears.factory.create('beta.database', '1.0');
db.open('database-demo');
db.execute('create table if not exists Demo (Phrase varchar(255), Timestamp int)');
db.execute('insert into Demo values (?, ?)', [phrase, currTime]);
var rs = db.execute('select * from Demo order by Timestamp desc');
```
db.insertRow('person', bob);
db.insertRow('person', bob, 'name = ?', ['Bob']);

db.selectAll('select * from person', null, function(person) {
  document.getElementById('selectAll').innerHTML += ' ' + person.name;
});

var person = db.selectRow('person', 'id = 1');

// update
person.name = 'Harry';
db.updateRow('person', person);
person = db.selectRow('person', 'id = 1');

// force
person.name = 'Sally';
db.forceRow('person', person);
person = db.selectRow('person', 'id = 1');

db.deleteRow('person', bob);
Are we going to get a GearsHibernate?

var Person = new GearsOrm.Model("Person", {
    firstName: GearsOrm.Fields.String({maxLength:25}),
    lastName: GearsOrm.Fields.String({maxLength:25})
});

GearsORM.Transaction(function() {
    new Person({name:"John"}).save();
    new Person({name:"Doe"}).save();
});

Person.select("firstName = 'Uriel'");
Person.count("lastName = ?",["Katz"])
GearShift

DB Migrations for Gears

Gearshift.rules[1] = {
    // create the demo table
    up: function() {
        return this.e("CREATE TABLE demo_table (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(30), movie VARCHAR(30))").success;
    },
    down: function() {
        return this.e("DROP TABLE demo_table").success;
    }
};
Database Tools
Alignment with AIR

Ext.data.SqlDB

“The APIs for AIR and Google Gears are nothing alike. In fact, AIR's SQLite database API is 100% asynchronous via events while Gears API is all synchronous with results coming immediately on execution.

So Ext.data.SqlDB was created to abstract both of these APIs into a single API to access both.”
The Hard Part

- From one DB
- To one DB per user
- Eek
- Wait, other apps handle this fine.
Local Server

A mini-web server that groks 200 and 304
ResourceStore

Manually Capturing

```javascript
var pageFiles = [
    location.pathname,
    'gears_base.js',
    '../scripts/gears_db.js',
    'foo.html'
];

try {
    localServer = google.gears.factory.create('beta.localserver', '1.0');
} catch (e) {
    alert('Could not create local server: ' + e.message);
    return;
}

var store = localServer.openStore(this.storeName) ||
    localServer createStore(this.storeName);

store.capture(pageFiles, function(url, success, captureId) {
    console.log(url + ' capture ' + (success ? 'succeeded' : 'failed'));
});
```
ManagedResourceStore:

* Capture entire applications

- List application resources in a separate manifest
- Gears captures and updates the list atomically
- Gears auto-updates automatically on each view (within reason)
- Supports multiple users per application
Sample Code

```javascript
var localserver = google.gears.factory.create('beta.localserver', '1.0');

var store = localserver.createManagedStore('mystore');

store.manifestUrl = 'http://myapp.com/offline-manifest.json';

store.checkForUpdates();
```
ManagedResourceStore

```json
{
    // version of the manifest file format
    "betaManifestVersion": 1,

    // version of the set of resources described in this manifest file
    "version": "my_version_string",

    // optional
    // If the store specifies a requiredCookie, when a request would hit
    // an entry contained in the manifest except the requiredCookie is
    // not present, the local server responds with a redirect to this URL.
    "redirectUrl": "login.html",

    // URLs to be cached (URLs are given relative to the manifest URL)
    "entries": [
        { "url": "main.html", "src": "main_offline.html" },
        { "url": ".", "redirect": "main.html" },
        { "url": "main.js" }
        { "url": "formHandler.html", "ignoreQuery": true }
    ]
}
```
json = Google::Gears::LocalServer::Manifest.new do |m|
  m.version = 'MyNewVer'
  m.add_entry({ :url => 'main.html', :src => 'foo.html' })
  m.add_extra_info :to => 'main.html', :redirect => 'foo_redirect.html'
  m.find_entries :in => '.', :ignore =>

  Google::Gears::LocalServer::Manifest::LEADING_PERIOD
end
"There's a concept of an **application cache**. An application cache is a group of resources, the group being identified by a URI (which typically happens to resolve to a manifest). Resources in a cache are either top-level or not; top-level resources are those that are HTML or XML and when parsed with scripting disabled have with the value of the attribute pointing to the same URI as identifies the cache.

When you visit a page you first check to see if you have that page in a cache as a known top-level page."
Worker Pool

*JavaScript needs threads after all? Brendan!*
Worker Pool

Run JavaScript in the background

- Provides thread-like functionality to JavaScript
- No more blocking the browser UI
- Communication is via IPC, no shared state or threading primitives
function nextPrime(n) {
    google.gears.workerPool.sendMessage(result);
}

var pool = google.gears.factory.create('beta.workerpool', '1.0');
pool.onmessage = function(message) {
    alert('next prime is: ' + message);
}

var worker = pool.createWorker(String(nextPrime) + '; nextPrime()');
Worker Pool Improved!

- Cross-origin API allows Gears apps from different sites to work together

- WorkerPool improvements:
  - `createWorkerFromUrl()`
  - `onerror` allows handling exceptions thrown by workers

- New HttpRequest module allows fetching from WorkerPools

- New Timer module allows timer events in WorkerPools
  - Implements the WhatWG Timer specification

```javascript
var timer = google.gears.factory.create("beta.timer", "1.0");
timer.setTimeout(function() { alert("Hello, from the future!"); }, 1000);
```
Why?

How about Encryption

dojo.sql("INSERT INTO CUSTOMERS VALUES (?,?, ENCRYPT(?)), "Neuberg", "Brad", "555-34-8962")
The Digg Oracle

Note:
This tool currently routes all Digg API requests through my personal web server. Due to the 'Digg Effect', the server may experience outages. If you have any issues, try back in a day or two.

What is it?
This tool will search Digg.com for your entire voting history, then download the details for each story. Once downloaded, stories are indexed in a local SQLite database via Google Gears. This allows you to perform search queries on your Digg history without having to connect to Digg's servers.

First Time
The first time you use it, you will need to do a one-time full sync. This may take some time, so please be patient. Modern computers can index 1,000 stories in about 15 minutes. Obviously, the time it takes to sync depends on how many stories you have Dugg.

Please type in your Digg user name:
[Input field for username] Go!
Gears added FTS2 to SQLite

Create the database
```
db.execute('CREATE VIRTUAL TABLE recipe
 USING fts2(dish, ingredients)');
```

Search the database
```
db.execute('SELECT dish FROM recipe
 WHERE recipe MATCH ?', ['tomatoes']);
```

Fun queries: `dish:stew tomatoes`

Find rows with 'stew' in the dish field, and 'tomatoes' in any field.
What didn’t you see here?

Hint: Sync, sync, sync
Syncing is hard

- Read only, small data
- Read write, small data
- Read only, huge data
- Read write, huge data

start simple, like Zoho Writer
Think about users

and don’t make them think
Offline UI

When to ask for user input?
Working with and without Gears

We aren’t that arrogant!

content = hasGears() ? new GearsBaseContent() : new CookieBaseContent();
Two Versions? Really?

Only in the extreme

```json
{
  'url': 'main.html',
  'src': 'main_offline.html'
}
```
Debugging is a Pain

On the web? Duh.

- Add Helper Code:
  - To clear out the DB
  - Remove captured files

- Disable the cache
- Use Firebug / Lite
Debugging is a Pain

On the web? Duh.

GearsBaseContent.prototype.clearServer = function() {
    if (this.localServer.openStore(this.storeName)) {
        this.localServer.removeStore(this.storeName);
        this.store = null;
    }
}

GearsBaseContent.prototype.clearTables = function() {
    if (this.db) {
        this.db.run('delete from BaseQueries');
        this.db.run('delete from BaseFeeds');
    }
    displayQueries();
○ An **offline widget** that you can easily embed in your web page with just a few lines of code, automatically providing the user with network feedback, sync messages, offline instructions, and more

○ A **sync framework** to help you store actions done while offline and sync them with a server once back on the network

○ **Automatic network and application-availability detection** to determine when your application is on- or off-line so that you can take appropriate action

○ A **slurp() method** that automatically scans the page and figures out all the resources that you need offline, including images, stylesheets, scripts, etc.; this is much easier than having to manually maintain which resources should be available offline, especially during development.

○ **Dojo Storage**, an easy to use hashtable abstraction for storing offline data for when you don’t need the heaviness of Google Gear’s SQL abstraction; under the covers Dojo Storage saves its data into Google Gears

○ **Dojo SQL**, an easy to use SQL layer that executes SQL statements and returns them as ordinary JavaScript objects

○ **New ENCRYPT() and DECRYPT() SQL keywords** that you can mix in when using Dojo SQL, to get transparent cryptography for columns of data. Cryptography is done on a Google Worker Pool thread, so that the browser UI is responsive.

○ Integration with the rest of Dojo, such as the Dojo Event system
GWT and Gears

```java
try {
    db = new Database("database-demo");
    db.execute("create table if not exists Demo (Phrase varchar(255), Timestamp int)");  
} catch (GearsException e) { ... }

button.addClickListener(new ClickListener() {
    public void onClick(Widget sender) {
        try {
            String phrase = input.getText();
            if (phrase.length() > 0) {
                db.execute("insert into Demo values (?, ?)", new String[] {
                    phrase, Integer.toString((int) System.currentTimeMillis())
                });
                displayRecentPhrases();
            }
        } catch (DatabaseException e) {
            Window.alert(e.toString());
        }
    }
});
```