

# Gears

Brian Love  
Webucator

# Why can't web apps be just as powerful as a desktop app?

*Gears exposes new modules and capabilities that provide general purpose building blocks for web applications to unlock the capabilities of the local machine in a safe manner.*

# Overview

- What Gears is all about
- 3 Main Modules
  - LocalServer
  - Database
  - WorkerPool
- Security
- Cool *\*new\** stuff

# Its all about

- Extending the Browser
- Open source
- Ubiquitous

# Getting Started

Get instance of desired module via factory

```
// gears_init.js defines google.gears.factory.
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
function init(){
  // Check whether Gears is installed.
  if (window.google && google.gears) {
    // Instantiate Gears objects
    var db = google.gears.factory.create('beta.database');
    db.open();
  }else{
    //get the user to install Gears!
  }
}
</script>
...
<body onload="init()">
```

# LocalServer

- Requests for URLs in the LocalServer's cache are intercepted and served locally from the user's disk.
- ResourceStore - assets such as PDFs and images are manually stored and updated
- ManagedResourceStore - specified list of URLs that are automatically updated automatically and periodically
- Is enabled or disabled by your application

# ResourceStore

## Example

```
var localServer = google.gears.factory.create('beta.localserver');
```

```
var store = localServer.createStore('name-of-store');
```

```
var storeThisArray = ['images/myimage.jpg', '/favicon.ico'];
```

```
store.capture(storeThisArray, function(url, success, id){ ... });
```

# ManagedResourceStore

## Example

```
// gears_init.js defines google.gears.factory.  
var localServer = google.gears.factory.create('beta.localserver');  
var store = localServer.createManagedStore('name-of-store');  
store.manifestUrl = 'manifest.json';  
store.checkForUpdate();
```

## manifest.json

```
{  
  "betaManifestVersion": 1,  
  "version": "site_version_string",  
  "entries": [  
    { "url": "index.cfm" },  
    { "url": "gears_init.js" }  
  ]  
}
```

# Database

- Data persistence layer
- Uses SQLite
- Includes full-text search extension fts2

# Database Example

Demo

# WorkerPool

- Collection of processes for JavaScript
- Don't distract the UI with large computations, background processes that required large amount of I/O
- Do not share execution state - no access to DOM objects such as *document* and *window* - communicate with each other through messages.

# WorkerPool Example

```
var workerPool = google.gears.factory.create('beta.workerpool');
```

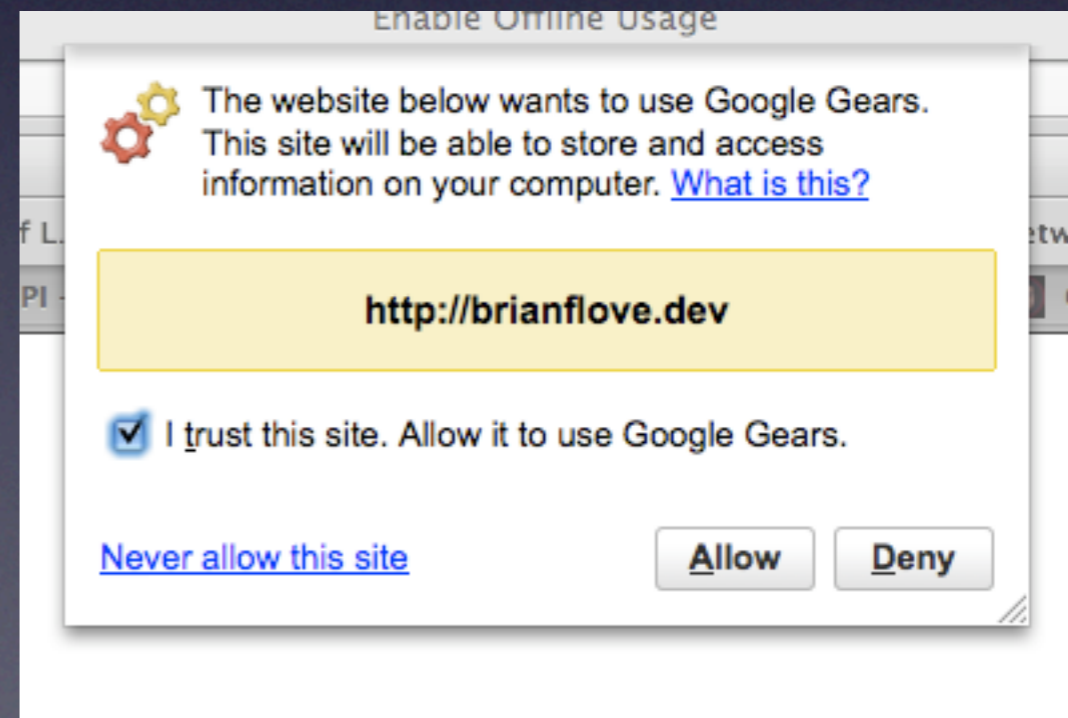
```
workerPool.onmessage = function(a, b, message) {  
  alert('Received message from worker ' + message.sender + ':\n' + message.body);  
};
```

```
var childWorkerId = workerPool.createWorkerFromUrl('worker.js');
```

```
workerPool.sendMessage(["3..2..", 1, {helloWorld: "Hello world!"}], childWorkerId);
```

# Security

- All aspects of Gears follows the same-origin policy (domain and port)
- Each origin must be given permission to use Gears.



# Cool \*new\* stuff

- Desktop shortcuts
- Notifications
- The BLOB
- Geolocation

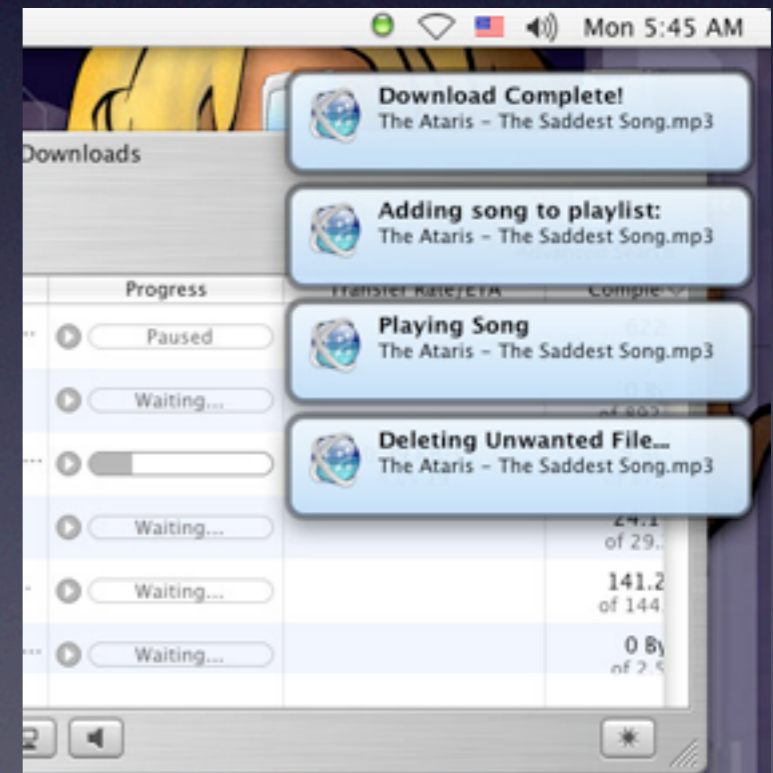
# Desktop

- Create shortcut to application with custom icon and name.

# Notifications

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

```
notifier.notify({  
  application: "My App",  
  title: 'warning',  
  description: 'some text',  
  priority: 2,  
  sticky: 'True',  
  password: 'Really Secure',  
});
```



# The BLOB

- Having a common object to represent and manipulate binary data across all APIs provides a lot of power with minimum footprint.

For example:

- Get a blob from a large file, chop it up into smaller blobs, and upload the pieces using XMLHttpRequest to get a large file uploader.
- Get a blob from a file while offline, store it in the database or application cache, and upload it later when connected.
- Get a blob from the network, send it into canvas, then get the modifications back out and re-upload them.

# Geolocation

- The Geolocation API allows web apps to retrieve the user's current position. The API should provide the following features:
  - One-shot position requests (e.g. for recommendations sites -- "where am I right now?")
  - Repeated position updates (e.g. for continuously updating one's location on a map)
  - Ability to get the last-known position cheaply before doing an expensive new request
  - Compatibility with future use as a singleton in the standard DOM (e.g. `window.geolocation`)

Questions?