Web Development

Dr. Dobb's Journal

June 2012

HTML5

Using the localStorage API to improve your websites

Next

ALSO INSIDE

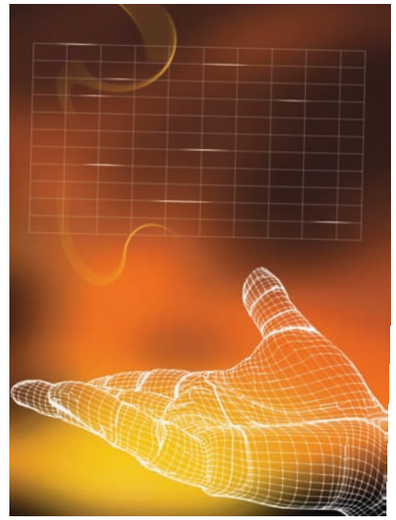
UBM TechWeb

www.drdobbs.com

Dr. Dobb's Journal

CONTENTS

June 2012



COVER STORY

More on DrDobbs.com

The New Native Languages

D and Go are at the forefront of a new generation of native languages emerging in the space between C and C++.

http://www.drdobbs.com/architecture-and-design/232901652

The New C Standard Explored

Tom Plum continues his series on the new C standard, discussing how C11 specifies many security features that require minimal changes to existing code. They greatly reduce unexpected behavior and prevent many kinds of common attacks.

http://www.drdobbs.com/cpp/232901670

Java Concurrency: The Executor Service

Eric Bruno explores the Java Executor Service, which is useful for managing pools of threads, as well as the scheduling of future events.

http://www.drdobbs.com/blogs/jvm/240000161

A Language-Design Puzzle in Operator Overloading

According to Andrew Koenig, resolving an overloaded function call involves finding a single possibility that is strictly better than all the others.

http://www.drdobbs.com/blogs/cpp/240000124

Jolt Awards: The Best Testing Tools

The advent of numerous testing options has made it possible for rank and file developers to turn out far more reliable code than at any other point in history. While this is great news for end users, it does confront developers and QA teams with the need to choose the appropriate types of tests for their products and the correct tools to run them accurately.

http://www.drdobbs.com/joltawards/232901286

Mailbag

Oracle editorial draws comment from computing luminaries, and readers weigh in on Software Engineering accreditation

Copyrights and APIs

In response to an editorial regarding the possible grave effects on APIs of Oracle's lawsuit versus Google (http://www.drdobbs.com/jvm/232901227), we got lots of mail and a record 38 comments on the article page.

I agree with you on the point that it should be impossible to copyright an API. I campaigned against user interface copyright through the League for Programming Freedom, which made an amicus brief, and I will campaign against API copyright in the appeals that I expect will follow this case. (The only way there might be no appeal is if the jury's findings on facts result in total defeat for Oracle.)

Because the conditions for fair use are not clearly defined, it will be hard for one decision to have much effect as a precedent for other cases. I fear therefore that the kind of ruling you proposed as a kind of solution would be no solution at all. It would leave all developers in doubt and in danger; even if it might spare Google, it would harm the whole field all the same.

It is unfortunate that the article used the term "intellectual property" to refer to copyrights. That lumped copyrights gratuitously together with many other unrelated kinds of legal privileges, all of which work differently. Thus, it presents an image of deeper understanding, which really is solely confusion (see http://www.gnu.org/philosophy/not-ipr.html). To say "the acquirers of Netscape's copyrights" would have made things clear.

— Richard Stallman
President, Free Software Foundation

The authors of the 386BSD operating system took Oracle to task for not living up to the terms it seeks to enforce against Oracle:

If you search for "open solaris jolitz" on an Oracle page, you find our copyright notices embedded for 386BSD code: "Copyright 1989-1998

William F. Jolitz." Note that it omits the mention of 386BSD and is edited, which violates the Berkeley copyright. So I guess it's OK if Oracle/Sun screws up, but not OK if Google does? Everyone said 386BSD would be sued out of existence and there would be eternal liability for those who used it. It never happened. And they used it.

— Lynne Jolitz

Note: 386BSD was first released on the world in 1992 in a series of articles (http://www.drdobbs.com/architecture-and-design/184408764) in *Dr. Dobb's Journal* by William and Lynne Jolitz.

Accreditation for Software Engineers

On the editorial regarding the lack of formal accreditation for software engineers, we received numerous emails, most of them decrying the absence of both such requirements and of proper programming, rather than computer science, education.

Excellent article! I just finished the process of hiring a summer intern in software engineering. Of the several candidates I spoke with, only one had heard of Agile practices (an MSCS student with a couple of years of experience) and none were aware of TDD or pair programming. The guy I hired will probably learn more about software engineering this summer than in all his coursework. I (of course?) am entirely self-taught; my degree is in physics.

— Name withheld

I enjoyed the article. Whether attending user group meetings or large conferences, it is what the person has accomplished that matters. I can't think of the last time a speaker mentioned where he/she went to school and no one asks.

— Judy Calla
Penn National Insurance

While I agree that a serious name change and requirements are sorely needed by this field, the fact is Scott Thompson, the ex-CEO of Yahoo, faked a degree. He outright claimed he had a degree he didn't have. It wasn't that he had a degree in "Computer Science" and claimed he had one in "Computer Engineering" or "Software Engineering," he didn't have any type of Computer Science degree whatsoever. Accounting — yes, computer science — no. If he had said he had a software engineering degree, he still would have been faking it given what he had.

I'm also heavily involved in the QA field, and my reputation is the only thing I have at times. If I outright lie, then my employer can't trust me anymore and I get fired (and rightly so). If I lie, people could get hurt or killed, so even if I do put in a bug that causes a mess, I own up to it and work on getting it fixed.

— Bruce Hartley
SPX Corp.

Have a correction or a thoughtful opinion on *Dr. Dobb's* content? Let us know! Write to Andrew Binstock at alb@drdobbs.com. Letters chosen for publication may be edited for clarity and brevity. All letters become property of *Dr. Dobb's*.

ASP.NET and Testability: An Uneasy Relationship

Making ASP.NET apps testable ranges from the easy to the very difficult. Knowing how to tweak the framework for testability simplifies the process

by Dino Esposito

Testability has been recognized as one of the fundamental attributes of software since 1991, according to ISO/IEC 9126, a standard that you'll find well summarized on Wikipedia (http://is.gd/Bpa8Y1). However, a decade ago when the ASP.NET framework made its debut, the whole theme of testability was not as popular as it is today, at least among developers focusing on Microsoft technologies. One of the key takeaways of ASP.NET Web Forms was "focus on the business logic and let the framework deal with what browsers expect." Gaining abstraction over typical elements of a Web solution such as HTML, JavaScript, and CSS was a specific goal of ASP.NET Web Forms.

Developers were given server controls to quickly and effectively arrange views, and code-behind classes to serve incoming requests and produce appropriate responses. A code-behind class is a piece of code that is tightly bound to the ASP.NET infrastructure. Any code placed in a code-behind class has direct access to the HTTP context and can read and write cookies, posted data, query string parameters, session state, HTTP headers, and so forth. Code-behind classes serve two main purposes: For HTTP GET requests, they set up the page for display; for HTTP POST requests, they grab posted data, prepare a call to the application's back end, receive a response, and prepare the next view for the user.

Sounds like a simple workflow, right? But in fact, it is not necessarily simple. The real complexity of such a workflow is determined by the inherent complexity of the call made to the application's back end. It is one thing to invoke a stored procedure from the code-behind class and display its results; it is quite another to invoke a stored procedure that is only one step in a far more complex piece of business logic. As an example, consider what it means for an e-commerce application to place an order. There might be extremely simple scenarios in which all that is required is adding a few records in a couple of tables; say, one record in the `OrderDetails` table for each item ordered, and one record for the order itself in the `Orders` table. In such a case, it might be acceptable that developers place all the code — a single call to the Data Access Layer — right in the code-behind class.

What if, instead, placing an order involves several (sequential and conditional) steps such as checking the credit status of the customer, checking the availability of goods, refilling stock by placing orders to suppliers, interacting with the back end systems of the shipping company, updating the company's accounting systems, and maybe more? You might still need to add a bunch of records to the `Orders` and `OrderDetails` tables, but there's now a whole workflow to be orchestrated, which significantly increases the level of complexity of the code you need to trigger from the user interface. Can this orchestration code be managed from within code-behind classes?

Over the past few years, scenarios where the code required to serve a Web request is fairly complex have become ubiquitous. As a result, having fat methods in code-behind classes shifted from being an acceptable practice to being a bad practice best avoided for the sake of maintainability and testability.

More complexity in the logic expressed via software inevitably means a stricter need to test the code to ensure that changes and new development don't break existing features.

How Easy Is It To Test Code Written for the ASP.NET Framework?

Design for Testability is a methodology aimed at making software easier to test automatically through other software. Design for Testability is centered on three pillars: visibility, control, and simplicity. Code that is easy to test is code that exposes the most relevant parts of the internal state so that appropriate assertions can be written and checked. Likewise, for tests to be relevant and reliable, testers should be able to force ad hoc input values on methods and exercise control over their invocation. Finally, the simpler the code, the more reliable response you get from the test.

The main impact of testability on software development comes through unit tests and integration tests. Unit tests are automatic tests done for the most part in isolation on code that has no dependencies whatsoever on the surrounding environment. Integration tests instead check whether interconnected parts of the system work well together.

When you try to apply Design for Testability to ASP.NET Web Forms, you find that you can gain a good level of visibility over the state of the infrastructure (that is, ASP.NET intrinsic objects), but you can hardly force test values without spinning up the entire ASP.NET runtime in the test environment. In fact, in ASP.NET Web Forms, you can't just simulate a fake HTTP context with test values in session state, a response stream that saves to memory, or an ad hoc query string. Because code-behind classes are tightly bound to the ASP.NET HTTP runtime, and the ASP.NET HTTP runtime doesn't allow mocking, covering ASP.NET Web Forms applications with unit tests is pretty hard. Integration tests then seem to be the only affordable option you have for testing ASP.NET Web Forms pages. But integration tests, by nature, are slower to run, thus developers tend to run them less often.

What Can You Do About It?

Unit testing is quite problematic in ASP.NET Web Forms if developers place all of the use-case logic into code-behind classes. An alternative option would be introducing a new layer of code that bridges code-behind classes (still part of the presentation layer) to the back end of the system. Such intermediate classes would receive HTTP context-specific data as an argument and operate in total isolation from the surrounding HTTP context. As a result, you extract all code that really expresses application logic from the presentation layer. The resulting design is therefore cleaner and much more testable.

So the bottom line is that while ASP.NET Web Forms certainly was not laid out with testability in mind (it was designed at a time when testability was not a primary concern for most developers), by intensively applying good principles of software design (Separation of Concerns and layers) you can easily write ASP.NET code that is unit testable.

On a final note, let's consider the twin ASP.NET framework — the increasingly popular ASP.NET MVC. Being a newer framework, ASP.NET MVC was designed for helping developers to write more testable code. And those good old principles of software design applied to ASP.NET MVC can make not just your code but also your unit tests far simpler.

— Dino Esposito is a frequent contributor to **Dr. Dobb's.** *His most recent book, Programming Microsoft ASP.NET MVC (http://is.gd/DTIw3U) was published by Microsoft Press.*

Comment

The localStorage API

To use the localStorage API, developers rely on a wide choice of libraries, each with their own strengths and weaknesses. Pamela Fox examines the major contenders and explains where their features can best be used

By Pamela Fox

In the first article of this series, "Understanding Client-Side Storage in Web Apps" (http://www.drdobbs.com/web-development/232900805), I reviewed the HTML5 storage options and concluded that the localStorage API is the most viable option. Now, in this second and final part of this series, I'll dive deep into that API and show ways to use it to improve your websites.

The localStorage API is very simple — it's basically an associative array (hash map) with wrapper methods to set and get items, iterate through all the stored items, and throw exceptions that notify the developer when something goes wrong, such as going over quota.

The table at right shows the methods and attributes, based on the official WebStorage specification.

The API is subject to the "same origin" policy, with which most developers are familiar thanks to `XMLHttpRequest`. That policy dictates which websites can see what data based on matching the subdomains and ports of a URL.

method/attribute	args	returns
setItem	String key, String value	
getItem	String key	String value
removeItem	String key	
clear		
key	int index	String key
length		int length

For example, given the URL "http://store.company.com/dir/page.html," here's a summary of who could view locally stored data:

http://store.company.com/dir2/other.html	Yes
http://store.company.com/dir/inner/another.html	Yes
https://store.company.com/secure.html	No (Different protocol)
http://store.company.com:81/dir/etc.html	No (Different port)
http://news.company.com/dir/other.html	No (Different host)

From a security standpoint, this means that you need to be careful not to use localStorage on a shared domain (because every page on that domain will have access to the data), and to be aware of security exploits like DNS spoofing. You should also know that your locally stored data will not be shared across subdomains unless you do a bit of magic with document.domain. You can read more about the same-origin policy on Mozilla's Developer Network (http://is.gd/Ix98Xq).

Abstraction Libraries

The localStorage API is quite easy to use, and you could use it directly via the API. But, as with many JavaScript Web APIs, there are several reasons why you might want to use a localStorage abstraction library; and there are an increasing number of libraries available.

Most of them include a feature-detection check to see whether the browser actually supports the API and whether it's currently enabled (as browsers let users disable it, just like with cookies). In the case that the feature check fails, some libraries fallback to using other techniques (such as cookies or browser-specific hacks). Most libraries handle serialization of objects using JSON, and some libraries are designed for specific use cases. Let's take a look at a few of them.

store.js

store.js (https://github.com/marcuswestin/store.js) is probably the most popular localStorage library, and it's also a great example of what to expect from a library. After it checks for browser support, it uses fallbacks to `globalStorage` for older Firefox versions and `userData` for older IE versions — two browser-specific client-side storage APIs that never made it into the standards world.

In the example code below, we store an object representing a song in localStorage (which is then serialized by store.js), retrieve and display it, and then remove it.

```
store.set('song', {artist: 'Roxette',
                   title: 'Listen to your heart'})
var song = store.get('song');
document.getElementById('song').innerHTML =
    song.artist + ' sings "' + song.title + '"';
store.remove('song');
```

Lawnchair

Another popular wrapper library is Lawnchair (http://brian.io/lawnchair/). It's similar in capabilities but its `get/set` methods are all asynchronous, so that it can be used with both synchronous and asynchro-

nous APIs. It features "adapters," which let you use it with various client-side storage options, including older options such as `userData` but also uber modern options like `IndexedDB`. It also includes a plugin API and plugins for common tasks like aggregation, pagination, and queries.

In the example code below, we store an object representing a band, and then we retrieve and display it inside a callback function. When Lawnchair is used with localStorage, the callback is called pretty much instantly, since it's a synchronous API. If you used Lawnchair with something like `IndexedDB`, it may actually take a few seconds before the callback is called.

```
var bands = Lawnchair(function() {
   this.save({key:'abba', name: 'ABBA', hometown: 'Stock-
holm'});

  this.get('abba', function(band) {
    document.getElementById('band').innerHTML =
        band.name + ' is from ' + band.hometown;
  });
});
```

lscache

The lscache library (https://github.com/pamelafox/lscache) is written with a particular use case in mind — in this case, expiration. The library lets you set expiration times when storing items in localStorage, so that you can kick items out after a particular amount of time. Its API mimics the memcache API, a popular server-side caching API, and its use cases are similar. Later, I'll explore when you would actually want to use localStorage in this way.

In the example code that follows, I store an array of objects representing songs (such as we might retrieve from an API), set those songs to expire in 24 hours, and display them on the page.

Full disclosure: This is the library I wrote, and is the one I use most frequently in my web apps. It's certainly not the best API for every use case, however, especially because it does not yet include a fallback for older browsers.

```
var songs =
[{artist: 'Ansiktet', title: 'Ackligt'},

{artist: 'Norlie & Kkv', title: 'Trojan Du Hatar'},

{artist: 'Michel Telo', title: 'Ai Se Que Te Pego'},

{artist: 'Avicii', title: 'Levels'},
{artist: 'Flo Rida & Sia', title: 'Wild Ones'},

{artist: 'Moa Lignell', title: 'When I Held Ya'},
{artist: 'David Guetta & Sia', title: 'Titanium'},

{artist: 'Timbuktu', title: 'Flickan '},
{artist: 'Rihanna & Calvin Harris', title: 'We Found Love'},

{artist: 'Takida', title: 'You Learn'}];

 lscache.set('swedentop10', songs, 60*24);

 songs = lscache.get('swedentop10');

 for (var i = 0; i < songs.length; i++) {
   var song = songs[i];
   document.getElementById('songs').innerHTML +=
   song.artist + ': ' + song.title;
}
```

What Can We Use localStorage For?

We can use localStorage for some of the things we used to use cookies for. We can also use it for a wider range of uses cases to help us improve the user experience for our websites — which is presumably what we're all striving for. With it, we can remember user data, retain appli-

cation state, remember form input, improve performance, and help the app work offline.

Remembering User Data

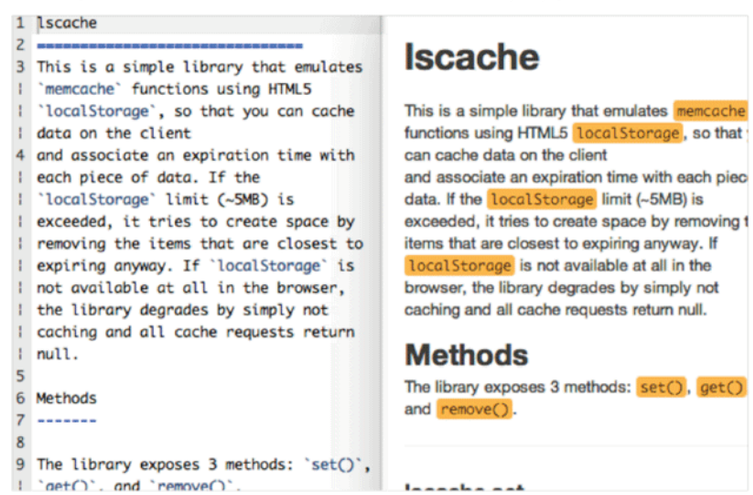
One reason to use localStorage is to remember user data. While we still want to remember most user data on the server, there are some cases where you might want to store data on the client instead:

- If you want to build an app without a server (maybe for prototyping),
- If you want to give users a customized experience before forcing them to create an account, or
- If you're building an app that inherently lives only in a client (like browser extensions or mobile apps).

Example: dillinger.io

dillinger.io is a Markdown editor that uses localStorage to remember user preferences and the most recently typed text, so that you can use it without ever signing in. It gives users the option to sign in with Github, and ideally it would migrate the user data from the client to the server once you signed in, so that it would follow you across platforms.

Here's an example of what they store in localStorage:

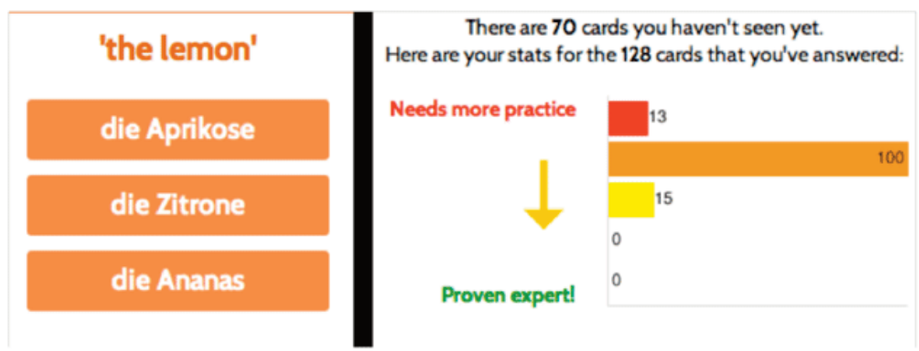


profile	{"theme":"ace/theme/textmate","showPaper":false, "currentMd":"lscache\nThis is a simple library that emulates 'memcache' functions using HTML5 'localStorage',...

Note that in this and the following examples, the key values are often truncated for readability. To see everything they store, visit the site themselves and inspect their localStorage using the resources panel in Chrome Developer Tools (http://is.gd/tOhLeX).

Example: QuizCards

QuizCards is a Chrome extension that I wrote for learning a language through interactive flash cards. Once installed, the extension lives only in your browser, via an icon on the toolbar. As you answer the flash



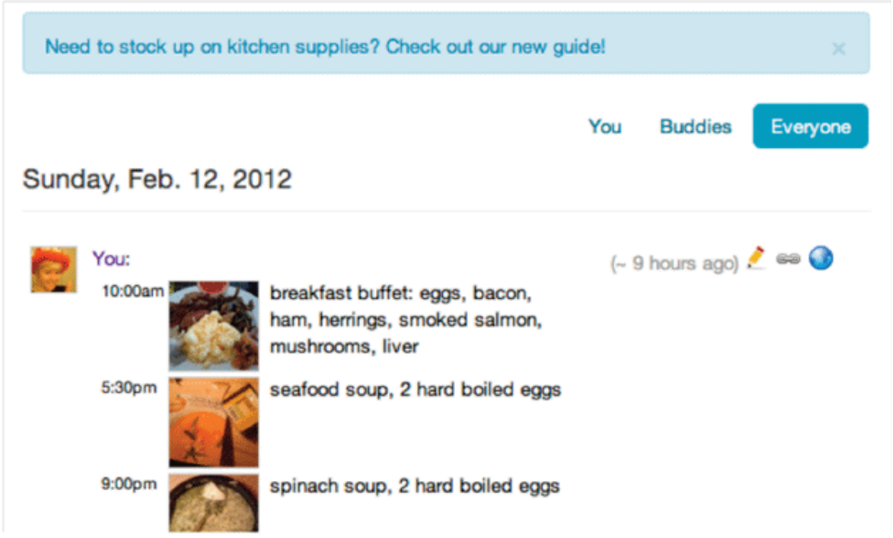
cards, the extension uses localStorage to remember which cards you answered correctly and how many times, so that it can be smart about which ones to show you next.

Here's an example of what it stores in localStorage:

german-answer-mode	multiple-choice
german-ham-Schinken	{"lastAsked":1322691448489, "timesAsked":1, "timesCorrect":1, "timesIncorrect":0}
german-bucket1	[{"id":"arm-Arm","lastAsked":0}, {"id":"cheek-Backe","lastAsked":0}, {"id":"belly-Bauch","lastAsked":0},...

Even if you are using a server to store user preferences and data, there are still some types of data that are better to store on the client, because they are actually about the client — like the current "state" of the website. Some websites are more like static documents, and don't really have a state beyond where the users have scrolled to, but other websites (such as gmail) are more like applications, with many different sections that can be opened, closed, and moved around, all under the same URL. Users doesn't always expect application state to follow them across clients, and it can be overkill to store the data on your server. If you realize that you do need to store the data there, you can always migrate it later.

Example: EatDifferent



EatDifferent is the website I'm working on now, and it's more of an app than a document. It's a nutrition tracking tool for users to log their updates and view others updates, and features the stream of latest updates on the home screen. I use localStorage to remember which stream filter the user most recently clicked, and also to remember which UI messages the user has seen and dismissed.

Here's an example of what it stores in localStorage:

cache-promo-supplies-jan30	hidden
lscache-stream-filter-users	everyone

Example: jshint

JSHint is a tool for checking the quality of JavaScript code, and since JavaScript style is so variable, it includes many options for evaluating the quality. The JSHint website features an interactive tool that lets you paste in code and report problems with it, and it uses localStorage to remember your enabled options for the next time you visit.

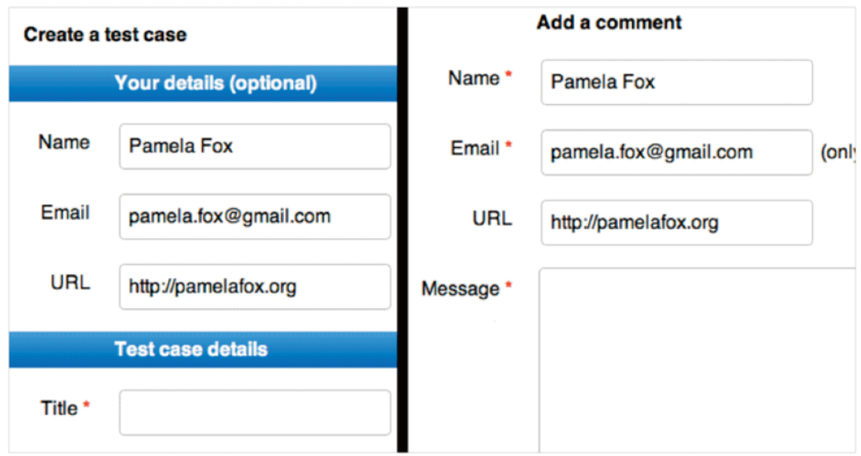
It stores all the options as a serialized object in one key:

opts	{"debug":true, "forin":true, "eqnull":false, "noarg":true, "noempty":false, "eqeqeq":true, "boss":false, "loopfunc":true, "evil":true, "laxbreak":true, "bitwise":true, "strict":true, "undef":true, "curly":true, "nonew":true, "browser":true, "devel":false, "jquery":false, "es5":false, "node":false}

Remembering Form Input

You can also use localStorage to remember user-entered form input, like their login username, commonly filled options in forms (like location and phone number), and long text input that they may accidentally forget to save. It's a great use of localStorage because form input doesn't need to be remembered — but when it is, it can bring a lot of joy to the user — almost like magic. But if for some reason localStorage isn't supported or the user changes computers, they won't actively notice their form input hasn't followed them. If you decide to use localStorage to remember form input, don't just remember everything — carefully consider which parts of a form should be remembered, and for how long. There are a few libraries specially designed for remembering forms, notably autoStorage and savify.

jsperf.com



The jsperf.com website lets developers create tests to compare performance of JavaScript snippets, and it starts with a form for creating new test cases. The website uses localStorage to remember author information, since that's the same across tests, and then it uses that same author information to auto-populate the commenting form.

It stores the author information in 3 keys in localStorage:

author-email	pamela.fox@fake-email.com
author	Pamela Fox
author-url	http://pamelafox.org

Example: Disqus

Disqus is a popular embedded commenting system for blogs and websites [*and used on Dr. Dobb's –Ed.*]. It adds a feature that uses localStorage to remember any text a user typed in a comment box but didn't post — so that when users leave a Web page and return, they'll see their draft still there, waiting to be posted. The Twitter web client also does this, and they also save an expiration for the draft, so that they don't show it after a certain point.

Disqus stores the drafts in one localStorage key, and keys it by thread name and website:

disqus.drafts	[{"thread:delayed_image_loading_on_long_pages": "I'm going to write a really long and epic comment, and I will be really mad if I accidentally move away from the page and lose this comment, mmm kay?"}]

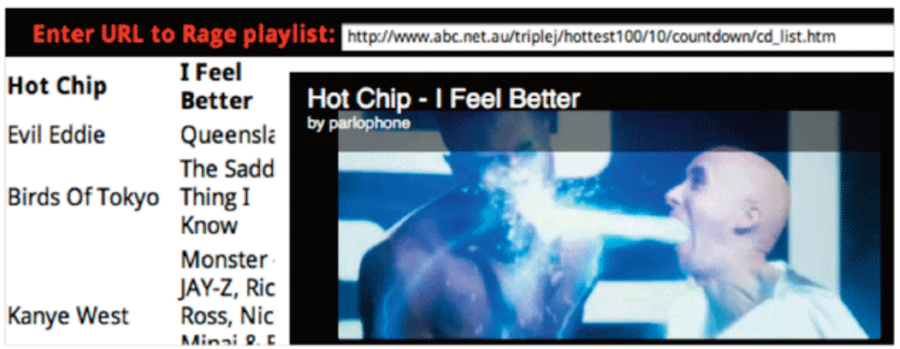
Improving Performance

Websites are increasingly reliant on AJAX and API requests, and in many cases, the results of these requests can be cached so that users do not have to wait as long. The traditional way of caching server-side requests is to set cache headers on the server so the browser serves them out of its own cache, but there are times when it's better to use localStorage. First, you have more control when you can do the caching yourself — you can decide when to invalidate resources and cache requests for different amounts of time in different areas on your site. You can also decide to first load from your cache and then refresh the cache, improving the *apparent* loading speed for a page. If you're developing for mobile browsers, you can significantly improve your performance there by using localStorage instead of relying on their browser cache, because many mobile browsers have smaller caches and can do less HTTP requests than desktop browsers.

There are several libraries customized for caching – such as lscache, which we discussed earlier (and an lscache jQuery plugin), YQL local-

Cache for caching YQL API results, jQuery offline for storing AJAX results, and Inject and basket.js for caching JS files.

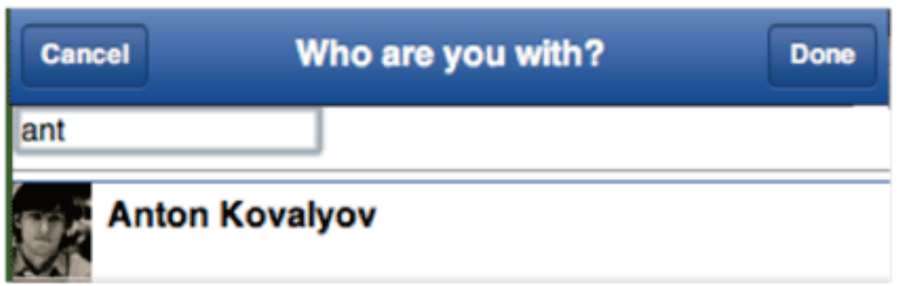
Example: RageTube



RageTube is a website that I made for watching music videos based on online playlists. It uses a server-side XMLHR to scrape and parse the HTML playlists, and it uses the YouTube API to find the top result for each song on YouTube. I use localStorage (via my lscache library) to store the results of the playlist scraping indefinitely, and to store the API results for a few days (because video searches don't change often). When a user watches the same playlist often, the app won't have to make so many XMLHRs and API requests, making it both faster for the user and cheaper for me, the developer.

For each request, there are two keys in localStorage, one to store the actual data, and one to remember the expiration date:

YouTube: Hot Chip I Feel Better	[{"id":"5GOZjlwIwfk","uploaded":"2010-03-17T17:53:17.000Z","category":"Music","title":"Hot Chip — I Feel Better",...
YouTube: Hot Chip I Feel Better-expiration	22153109
parser: http://www.a...	{"songs":[{"artist":"Angus & Julia Stone","title":"Big Jet Plane","id":"angusandamp; julia stone-big jet plane"},...



Example: Facebook Mobile

The mobile-friendly version of the Facebook website (http://m.facebook.com/) lets users find friends when posting status updates or searching for profiles, and gives users a fast autocomplete experience by caching the friends information in localStorage. Users don't add new friends that often, so Facebook can be OK with the data being a little bit stale and refreshing it when it has the chance. The Twitter web client does something similar, to give a fast autocomplete when mentioning usernames in a status update.

Facebook stores all the autocomplete data in a single "typeahead" key:

typeahead	{"time": 1329151694363, "value": {"friends": [{"path": "/anton.kovalyov", "photo": "http://profile.ak.fbcdn.net/hprofile-ak-snc4/186412_506803098_992151709_q.jpg", "text": "Anton Kovalyov", "uid": 506803098,},...

Example: Google Mobile

Both Google Mobile and Bing Mobile use localStorage to cache their HTML, CSS, and JS, to reduce the download size of their search pages. To accomplish this, they assign IDs to the various parts of their Web page, store the Web page parts in localStorage, and remember IDs and expirations in a cookie. When their server sees that cookie, it decides not to resend the stored parts. If for some reason the cookie is there but the data isn't, then the cookie is cleared, the page is reloaded, and the server sends the whole page.

Google and Bing are two of the most popular websites in the world, so it makes sense for them to optimize their website delivery; but this is definitely an advanced technique, and this level of optimization is probably not needed by most sites.

Here's a snippet of Google's JS code for checking what it stores in localStorage, followed by the storage example:

```
var c=localStorage.getItem("mres."+a);
if (c) {
    document.write(c);
    localStorage.setItem("mres:time."+a,Date.now());
} else {
     window._clearCookie("MRES");
     document.location.reload(!0);
}
```

mres.-8Y5Dw_nSfQztyYx	<style>a{color: #11c} a:visited{color: #551a8b} body{margin:0;pad...
mres.-Kx7q38gfNkQMtpx	<script> //<![CDATA[var Zn={},bo=function(a,b){b&&Zn[b]||(ne...
mres:time.-8Y5Dw_nSfQztyYx	1301368541872
mres:time.-Kx7q38gfNkQMtpx	1301368542755

Example: ESRI MAPs

Despite the fact that localStorage can only store strings, it can be used to cache images by converting them from binary data into data URIs. If your Web app uses a large number of images, you may want to do this. For example, the ESRI Maps API includes an option

for caching map tile images in localStorage, by using a key for each tile URL.

http://server.../tile/12/1409/2075	/9j/4AAQSkZJRgABAQEAYABgAAD/2wB...
http://server.../tile/12/1410/2077	/9j/4AAQSkZJRgABAQEAYABgAAD/2wB...

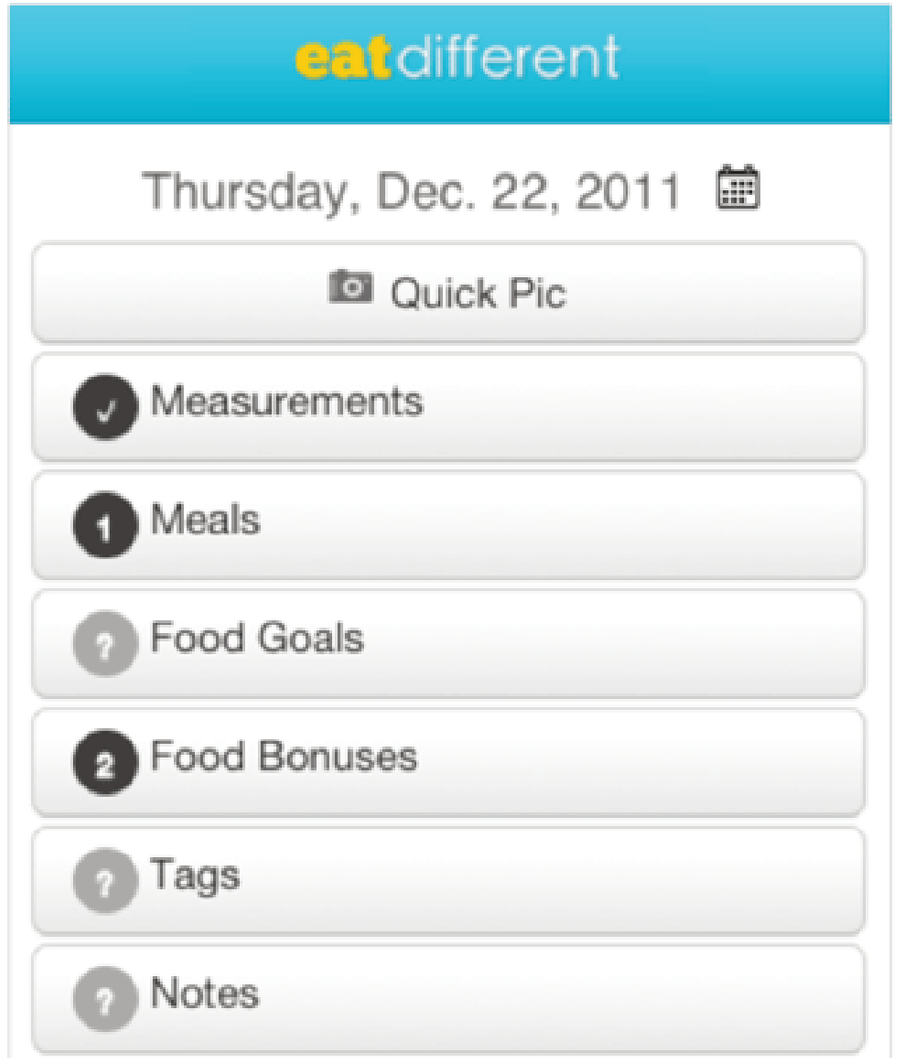
Making Your App Work Offline

Using the same techniques for improving website performance, localStorage can be used for enabling a website to work offline. However, since websites can be used offline for indefinite amounts of time (like when a user is on vacation on a desert island), you need to think carefully about what you cache and for how long. It's fine to have stale data for a few minutes on the Web, but what about for a few hours or a whole day on a mobile device? In addition, if you want a website to work offline, you have to cache *all* of the necessary data, not just the data that make the app perform better.

(Note: HTML5 also offers the cache manifest API for storing resources offline, so you should try to use that in addition to localStorage when designing an offline website.)

Example: EatDifferent

The mobile app for EatDifferent is actually written in HTML5, thanks to using PhoneGap as a wrapper API to access native APIs and present it as a native app. That means that I can use localStorage to help it continue working when it's offline. The app remembers the profile information for the last logged in user, as well as previously fetched logs. Once it successfully connects to the Internet, it tries to re-fetch both the user profile and log data, and refreshes the UI when there's new data.



It stores the user profile in one key, and stores each log in a key based on the user profile and log date:

lscache-user	{"first_name":"Testy", "last_name":"McTesterFace", "id", 166, ...
lscache-166:log02/14/2012	{"measurements":{"weight": {"units":"","value":"150"}, "meals": [{"what":"sausage","when":"12:00pm",...

Best Practices

Now that you've seen everything that you can do with client-side storage, and localStorage in particular, here are some best practices to keep in mind:

- Pick good key names: Your localStorage key names are like global variables across your entire domain — and as we know, it's easy for global variables to clash with each other, particularly when a site grows in size or number of developers. You can use pseudo namespaces, however, by prefixing the key name with the section of the site it's used in. As a general rule, it never hurts to go with a longer, more descriptive key names.
- Guard private data: You should be careful about storing sensitive information like credit-card details. It's still possible for hackers to use DNS spoofing to access the localStorage of a website, and we may find out in the future that there are other ways hackers can access localStorage (it took a few years to discover all the cookie exploits, after all).
- Be quota aware: Whenever you use any client-side storage option, you should be aware that quotas exist. Use libraries that handle the over-quota exception, and code in a way that doesn't assume stored data will be stored forever.
- Use a library: You can use the raw localStorage API, but the available abstraction libraries make your use of localStorage more robust — and they can make it easier to switch to more appropriate client-side storage once there are viable options.

Conclusion

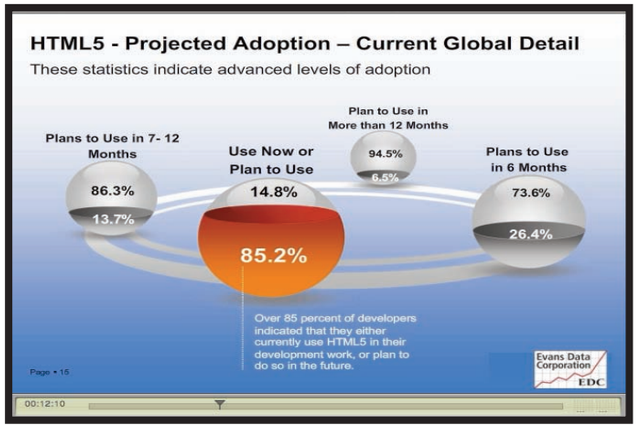
Web developers have made do with cookies and hacks for a long time, but we're now at a time where we can start seriously considering other options for client-side storage. We have localStorage in all the modern browsers, and it looks like we are close to having IndexedDB and the File API in them soon. You can start using localStorage now to improve your websites using many third-party libraries. We can look forward to a future in which we can pick the absolute best tool for the job.

— Pamela Fox spent five years at Google helping developers use the Maps and Wave APIs in their apps, and is now working on her own web apps using a mix of Python, JavaScript, and HTML5.

Comment

HTML5 Virtual Event

[VIDEO LINKS]



Ben Hanley, Senior Project Manager at Evans Data Corporation gives you a structural overview of HTML5 adoption. Click screen to see how quickly HTML5 use is gaining traction.

Click screen to see Aditya Bansod, Senior Director of Product Management at Sencha, offer an introduction to HTML5 application development for mobile environments.

Browser Fragmentation? Feature Detection? Polyfills? Click screen to see Microsoft Developer Evangelist Doris Chen discuss practical HTML5 development and how you can get started today!

Click screen to see Pamela Fox discuss Client-Side Storage and explain how you can use new APIs to manipulate powerful clients and improve both performance and user experience.

Developing in HTML5: The What and How

A member of the W3C discusses what HTML5 is exactly and points to useful resources for developers wanting to stay close to the evolving standards.

By Mike Smith

We are in the midst of a generational change to the Web platform. HTML5 is the shorthand term often used to describe this change, although the new technologies driving this transformation are not limited to the features defined in the HTML5 specification.

I use the phrase "Web platform" to refer to the broad set of protocols, formats, and APIs that are natively supported in Web browsers. Developers use these features to build stylish applications that support social interaction and take full advantage of device capabilities such as cameras, microphones, and GPS. HTTP, TLS, and WebSockets are some of the protocols that are part of the Web platform; and HTML, CSS, SVG, and JavaScript are examples of some formats. APIs include the Geolocation API and the 2D drawing API for the `<canvas>` element, among others.

Ultimately, a feature becomes part of the platform once it is broadly implemented and developers use it regularly to deliver a consistent user experience. The World Wide Web Consortium (W3C) plays a role in achieving consistent interoperability by bringing a range of stakeholders to the table, developing standards that can be implemented without royalties, and creating supporting materials such as test suites and validators.

A standard represents a shared agreement and is very useful as such. But the Web platform consists of a great many technologies at different levels of maturity, not all of which are uniformly standardized. Prac-

tical questions arise when leading-edge developers start to use them, including:

- How do I minimize the amount of special-case code?
- What features are stable enough for me to use now?
- How do I reach users that do not have modern browsers?

The answers change over time, so developers have devised practical solutions such as fallbacks and "polyfills" (which will be further explained later). These can make content useable in both older browsers and in the latest browsers. In this article, rather than focus on the state of HTML5 standardization, I explore some resources that can help developers start using Open Web Platform technology today. These resources provide the "what and how" for using HTML5 now, complement the standards, and ultimately help accelerate deployment.

"How does a busy developer keep up with current feature support, and which fallbacks and polyfills (if any) are available?"

platform.html5.org

How does a busy developer keep up with current feature support, and which fallbacks and polyfills (if any) are available? I put together a site, platform.html5.org, that acts as a dashboard for keeping you up-to-date about the set of technologies that constitute the Web platform. (It's backed by a github repository (http://is.gd/AjgszC), and I encourage you to help maintain it by forking the repo (http://is.gd/otOaIF) and sending pull requests with proposed updates.)

The site groups technologies into categories such as graphics and typography, media, and storage. Icons on the site indicate the level of maturity of each feature. A green flag indicates that a feature is ready to use. A yellow lighting bolt means "use with caution." However, these are minimal indicators; to get a true picture of the readiness of a feature, you need to follow the helpful links that are provided. Each technology links to the specification that defines it as well as other helpful links to sites such as:

- HTML5 Please
- When Can I Use…
- MDN (Mozilla Developer Network)
- other sites that provide a test suite for the feature

The HTML5 Please site (http://html5please.com/), in particular, is worthy of further comment. It was created by the H5BP developer collective (the same group of developers behind HTML5 Boilerplate; http://html5boilerplate.com/). It lists features by name and provides a quick way to get a high-level overview of the maturity of each feature. It shows an expandable panel with status information (use/caution/avoid). The "caution" and "use" keywords are in some cases further qualified with the phrases "with fallback" or "with polyfill." In the cases where "with fallback" appears, you can expand the feature for that panel to view information about exactly how to provide fallback for that feature.

A polyfill (http://is.gd/GoEbKW) is a piece of JavaScript code that acts as a kind of shim for a feature; that is, it mimics a future API providing fallback functionality to older browsers.

The site operators for HTML5 Please make it easy for anyone to contribute by providing an "Edit this info" link in each panel, enabling you to create a copy of its contents, which you can then submit back to the site. (The mechanism is backed by a github repository (https://github.com/h5bp/html5please/) and your contributions are sent as pull requests for that repo.) The maintainers review and merge contributions.

Sometimes you'll want more detailed status information than what HTML5 Please provides. For example, you might want to know exactly which browsers support a particular feature, and which versions of each browser support it, or what level of support it currently has in mobile browsers, and which OS versions of those browsers. In those cases, the site you'll want to turn to for that information is When Can I Use...

When Can I Use (http://caniuse.com/) is maintained by Alexis Deveria. He follows the readiness status of a large set of features and updates the site as new versions of browsers are released. And if there is a feature you want to know about, but that is not yet listed on When Can I Use, you can contribute feature suggestions (http://is.gd/fPVTXW).

Each feature currently tracked at When Can I Use has a table with a column for each major desktop browser (IE, Firefox, Chrome, Safari, and Opera); for each major mobile browser (iOS Safari, Opera Mini, Opera Mobile, and the Android Browser); and rows with version numbers for each browser. As in other sites, color codes indicate the level of support for a feature in each particular browser version (support/partial, support/no, support/unknown).

For example, consider support as of today for CSS Counters (http://caniuse.com/#feat=css-counters). A table full of green entries makes clear that the feature is well supported. Other features that are not well supported are shown with red highlights.

Each feature table at When Can I Use provides "see-also" links to tables for related features, as well as links to third-party sites that provide how-to information about actually developing content with that feature.

MDN (the Mozilla Developer Network) is one of the sites that both platform.html5.org and When Can I Use link to. Think of MDN as a "how can I use" feature guide. If, for example, you want to implement Web Workers, and need code examples and links to resources with detailed information, MDN has a page on using Web Workers (https://developer.mozilla.org/En/Using_web_workers) that's a great place to get started.

As with HTML5 Please, When Can I Use, and platform.html5.org, MDN welcomes contributions and actually makes contributing even quicker and easier than the other sites do: The entire site is a wiki, so once you create an account there, you can edit any page.

Test Suites

There is no better means to evaluate the maturity of a particular feature than to have a complete test suite for that feature, run the test cases for it yourself, and analyze the results. There is no single central repository for all test suites for the Web platform, and no single place to view the results for all of them, but the W3C has started work on a shared test framework site (http://w3c-test.org/) and the W3C CSS Working Group has a CSS-specific test framework site (http://test.csswg.org/harness/).

These sites allow you to view per-browser and per-browser-version results for a number of test suites. You can also run the test cases in your own browser and submit your results to be integrated back into the framework results database. See, for example, the results data for the CSS Multi-column Layout Module test suite (http://test.csswg.org/harness/results/CSS3-MULTICOL_DEV/), or the start page that allows you to run the test suite in your own browser (http://test.csswg.org/harness/suite/CSS3-COLOR_DEV/).

Conclusion

The aforementioned sites can help you stay current on the emerging Web platform. I expect even more helpful sites to appear as the platform continues to mature and as people share their code and their experience. I encourage you to use the sites I've mentioned here, and to consider contributing updates to them so you can help ensure that quality, up-to-date information about the Web platform is available to us all.

— Mike Smith works on W3C core specifications for browser technologies. He previously worked for Opera Software.

Comment

From the Vault

Getting Started with The Cloud: The Ecosystem

This first article from a popular series on Dr.Dobbs.com discusses general cloud-related issues and looks specifically at both the Amazon and Google cloud architectures.

— *DDJ*

By Allen Holub

A friend recently reported a conversation he had with one of those wide-eyed, gee-golly developers who's half Techie and half Moonie. When asked what he was working on, the speaker came back with, "cloud cloud cloud cloud cloud cloud," and my friend said, "but, what if ...," to which the speaker replied, "cloud cloud cloud cloud cloud," to which my friend said, "but that won't work because...," to which the developer responded, "cloud cloud cloud cloud cloud" — and so it went. Many use "cloud" as a synonym for "good." Cloud architectures indeed have a lot going for them, but they're not a panacea, and you need to know what you're doing to jump to the cloud successfully.

This article both discusses general cloud-related issues and looks specifically at the Amazon and Google cloud architectures. Subsequent articles will be more practical, delving deeply into code that comprises cloud-based applications, but let's start with some background.

What Is The Cloud?

First, what exactly does "cloud computing" mean? The term "cloud" dates to the early days of the Internet; back before domains existed (yes, there was such a time). An email address was essentially a route specified in what was called "bang notation." To send an email, you needed to know the name of every machine between you and the recipient. Here's a particularly nasty example that I pulled out of an old newsgroup post:

```
dog.ee.lbl.gov!ucbvax!cis.ohio-state.edu!zaphod.mps.ohio-
state.edu!qt.cs.utexas.edu!cs.utexas.edu!utgpu!utzoo!sq!msb
```

The "dog" and "msb" are the sending and receiving machines. The rest is the route from one machine to the other. The routes didn't have to be fully specified — most email systems knew about a handful of major hubs, so a minimum-length address just specified a route from that hub to you — but there was zero flexibility.

Things changed with the introduction of domains. Instead of an explicit route, you sent an email to a gateway machine, and the email's recipient got the mail from a different gateway machine. The servers through which the mail passed on the way from one gateway to the other were anonymous, and the network topology was unknown. The word "cloud" was coined to describe that amorphous network. You didn't know what went on inside the cloud (and believe me, you didn't want to). As long as the mail ended up at the right place, everything was copacetic.

So, here's my rather strict definition of "cloud": A network of computers in an unknown topology, arranged in such a way that you don't need to know anything about this network except how to talk to a machine at the edge.

A "cloud application" is then an application deployed to the cloud itself, not to a specific machine. The application could be running on one or more machines that may or may not be physically collocated. Its data store could also be distributed, and may not be on the same machine as the application.

In my own mind, I see a difference between cloud applications and Web 2.0 or AJAX applications, which we used to call "thin client" applications. Here, the UI is a standalone program (typically written in JavaScript and running in the browser), and it talks to a server (which is primarily a data repository) using HTTP.

Web 2.0 applications are typically implemented using a traditional client-server architecture (one server hosted on an ISP talking to multiple browser-based clients). However, there's no reason why you can't have Web 2.0 cloud applications. In fact, that's most likely the way that all applications will work five years from now. It's useful, however, to separate the concepts in your head. Most current Web 2.0 applications are not cloud based.

What Difference Does It Make?

So, why would you want a cloud application instead of a simple client/server arrangement? Consider the following ping times:

```
> ping www.google.cn
PING www.google.cn (203.208.37.99): 56 data bytes
64 bytes from 203.208.37.99: icmp_seq=0 ttl=239 time=273.340 ms
64 bytes from 203.208.37.99: icmp_seq=1 ttl=239 time=478.394 ms
64 bytes from 203.208.37.99: icmp_seq=2 ttl=239 time=421.920 ms
64 bytes from 203.208.37.99: icmp_seq=3 ttl=239 time=343.003 ms
64 bytes from 203.208.37.99: icmp_seq=4 ttl=239 time=263.843 ms
64 bytes from 203.208.37.99: icmp_seq=5 ttl=239 time=482.231 ms
...
```

The round-trip time between my desk in Berkeley, California, and one of Google's servers in Hong Kong ranges from a bit over a quarter to almost half a second, and we're sending only 56 bytes. There's essentially no server overhead, but we're hostage to both distance and the speed of the routers through which the data is passing. (A tracert reports only 17 hops, so the latency is probably all distance.) The picture is different when the server is close by. Here are the results from Berkeley to San Jose (12 hops):

```
> ping google.com
PING google.com (74.125.224.52): 56 data bytes
64 bytes from 74.125.224.52: icmp_seq=0 ttl=54 time=19.815 ms
64 bytes from 74.125.224.52: icmp_seq=1 ttl=54 time=20.466 ms
64 bytes from 74.125.224.52: icmp_seq=2 ttl=54 time=35.547 ms
...
```

A cloud application (or at least the instance of the application that we're talking to), would ideally be running on the machine with the best access time. That's the main advantage — the cloud can effectively reconfigure itself to take care of pesky details like network latency.

However, there's actually no way to guarantee that this reconfiguration will actually happen, which brings us to the dark underbelly of a cloud app: We need to program for the worst case.

Imagine a cloud app that's doing some kind of word completion. Every time you type a character, it's sent off to a server, which finds words prefixed with whatever you've typed. The server sends back a list of possible matches, and your program displays these. Most of the cloud books, in fact, demonstrate this sort of thing in exactly that way — the local application talks to the server with literally every keystroke. Given the look-up times, etc., your user isn't going to be particularly happy with your worst-case response time. You can, however, rethink your strategy. When the first few characters are typed, the server could send you a large, perhaps exhaustive, list of every word that could possibly start with those characters. Thereafter, the application can use that list to update its display rather than going back to the server with every key press. By eliminating the redundant network queries, we make the application much more responsive.

On the plus side, the cloud is amorphous and can indeed reconfigure itself based on observed load. If Google notices that there is a lot of traffic between Berkeley and Hong Kong, it may well replicate the Hong Kong server somewhere in California, and the latency would suddenly improve. The same applies to your cloud application: It will, ideally, be running on several geographically distributed servers, with the topology scaling to accommodate actual requests. In other words, the size of the network (and your cloud-services provider) matters. For cloud services to be effective, the provider has to be large. If you deploy to the Google or Amazon infrastructure, you're effectively leveraging the flexibility inherent in a very large network. By my rather strict definition, an application running on a single server, whether it's an ISP or so-called cloud host, isn't a cloud application at all because it looses the scalability and flexible topology of a true cloud infrastructure.

Scalability and Cloud-Service Architecture

A significant advantage to a cloud infrastructure is automatic scalability, but here's one place where the basic architecture matters. Amazon's Elastic Compute Cloud (EC2), like most cloud providers, rents you a "virtual machine" to host your application. Your VM may or may not share a physical machine with other apps, and it has an unknown number of physical processors attached to it. At its heart, though, your EC2 VM is just a Linux (or Windows) box, and you can configure it however you want. You typically pay only for the time that the VM is actually busy doing something, which is great for a software startup that's effectively getting rack space for free. As the load increases, so do your expenses (but hopefully, so does your revenue). You use your VM pretty much the same way you'd use a shell login to a shared server at an ISP, deploying with FTP, etc.

The inherent flexibility of a hosted-VM approach is particularly important the day that your application gets reviewed in *The New York Times*, and suddenly you have 1000 hits/second. Your ISP-hosted shared server would just crash at this point. An EC2 VM will scale, however, running on a dedicated machine if necessary, with cores added as necessary. Amazon will automatically increase the "umph" of your VM — giving you more machine cycles on the physical machine, for example, or assigning more cores to your application. Of course, you'll pay for this extra umph.

The main downside of this approach is that there is an effective upper limit on the scalability. Adding cores can get you only so far, and there's a diminishing return on the number of cores. Eventually, you're

using everything that the machine can give you. What if that's still not enough to handle the volume? In theory, your app can be placed on several machines at this juncture, with Amazon handling the load balancing (you can run several EC2 VMs in separate physical locations that you specify), but that scaling doesn't happen automatically, and the app has to be written with scaling in mind.

That is, if you're really planning on scaling, you have to do exactly the same amount of programming work that you'd do if you were running the application on multiple machines in your own data center. This is a nontrivial amount of work. So, Amazon and its brethren give you a lot of flexibility in configuration. You can put anything you want on your virtual Linux box, write your app in any language, augment it with custom processes; go crazy! The downside is that you have to worry about administration and scaling, and that can add to the complexity (and cost) of the application very quickly.

Fortunately, there is another approach — the one used by the Google "App Engine." Google doesn't rent you a VM at all — you have no control of the operating system and can't install arbitrary applications on "your" machine. Instead, you rent time on a virtual application server (think Tomcat). You write your application in an approved language (Python or Java) and you deploy your application directly to Google's app server, not to the operating system. For example, if you're using Java, your application is a standard Java "web app" packaged into a WAR file and deployed to Google exactly the same way that you'd deploy to a Tomcat instance, by uploading the WAR. Google handles the Tomcat part. (It's not actually using Tomcat, but I usually test locally using Tomcat and haven't found any problems. Google's own development tools use Jetty.) Part of deploying the app is telling Google what URL to use to access it. You can use both a Google-provided URL (*something*.appspot.com), or a subdomain of your own domain.

I personally prefer the Google approach for several reasons. First, I hate doing system-administration work. Since Amazon just gives me a box with an OS on it, I'm forced into that role if I use EC2. Google, on the other hand, does all the SA work for me. All I need to worry about is my application. Second, Google's virtual application server can, itself, scale and take my application along with it. For example, the app server could, at least in theory, run on multiple machines simultaneously in the same way that you can cluster Tomcat instances. Scaling, then, is in no way limited by the number of cores or speed of a single machine. As a consequence, my application can be vastly simpler, since I don't have to deal with the scalability issues in the source code. Finally, Google provides a rich set of development tools (mostly Eclipse extensions) that ease development considerably, though many of those tools will work with EC2 as well. For example, the Google Web Toolkit (GWT) provides you with a way to build browser-agnostic AJAX front end in Java. GWT includes a Java-to-JavaScript compiler that translates your code into platform-independent JavaScript when it's time to deploy — but when you're developing, it's all Java. That means that you can use the Eclipse debugger on both the client and server side, trace execution from client to server, etc., all within a single development environment. GWT applications will even run fine under Tomcat on an EC2 instance. I can develop much faster with GWT than I ever could when I was writing JavaScript by hand.

On the downside (to paraphrase Henry Ford): Your app can come in any color, provided that it's black. Your choice of implementation language is Java (my own predilections preclude writing an enterprise application in Python). You have to structure your application as a Java

web application, built around servlets, and you have to access your data using JDO or JPA (there's no JDBC support).

Google's working on adding SQL (The service is currently in limited preview, see https://developers.google.com/cloud-sql/), but it's not there yet, and is available only to "App Engine For Business" customers. Unfortunately, Google's pricing model for the "For Business" customers effectively makes SQL inaccessible to a standard web application meant as a public Software-as-a-Service (SaaS) app. Google charges $8 per year per user for a "For Business" application, which makes sense if you're implementing your HR application on Google instead of running it in your own data center. But a per-user fee is nonsensical if you're writing a SaaS app to expose to the entire Internet. The standard (not "For Business) App Engine charges are based on CPU and data usage, not the per-user model. Google has made similarly stupid (a technical term we analysts use) choices on other fronts as well. For example, a standard App Engine application can use SSL only if you deploy the page to a Google URL (MyDomain.appspot.com), which could be disconcerting to one of your users if they look at the address bar. Similarly, though you can host subdomains on the Google App Engine, you cannot host your main domain on Google. You have to get an account with a standard ISP, and then redirect access to a Google-hosted subdomain. (For what it's worth, *www.foo.com* is a subdomain, so it can be hosted on Google. It's the *foo.com*, without the *www*, that's the problem.)

Amazon EC2, on the other hand, gives you several database choices: You can run an RDMS on your VM, you can use Amazon's Relational Database Service (RDS), or you can use Amazon's SimpleDB service if you're doing something very simple. You can easily host your domain on an EC2 instance, and you can easily access that domain using SSL (because you're just accessing your own instance of Apache, running on the VM).

So, to sum up the differences before moving on to other issues: Google provides a better programming environment, with easy deployment, and very good scalability; but, Google's services are marred by an inability to easily host your top-level domain, inability to use SSL with your domain's URL, and lack of SQL support. The last two can be resolved if you're an "App Engine For Business" user, but the pricing model for that service effectively makes it useful only for large companies who want to move in-house applications from their own data centers to Google, something that I have a hard time believing will happen.) Amazon has none of those particular problems, but system administration is difficult with EC2, and scalability is not fully automated. It's the scalability issue that's the show stopper for me, so I'm using the Google App Engine in spite of its limitations.

Storage

Before finishing up, there are a few other issues that you need to consider that are just part and parcel of cloud applications, regardless of the provider.

First of all, you need to rethink your relationship with your data. Data may or may not be replicated by the service, and it may or may not be stored on the same physical machine (or on the machine that's running the service that accesses the data). Securing the data can be a significant problem. You can encrypt sensitive information like credit-card numbers, but encrypting everything is not a practical solution because you can't issue queries on encrypted data. More to the point, if you're doing the encryption, then there's always a point at which the

data is unencrypted and your encryption key is in plain sight. If your application is running on someone else's server, you're potentially vulnerable. Of course, a dedicated VM is less vulnerable than a shared server at an ISP that allows shell access, but you'll never be as secure as you would be in your own data center.

Also, bear in mind that your data is certainly replicated on many servers. This organization can be an important performance enhancer. Consider a video-streaming application that stores its data in the cloud. The actual application may be in only one place, but once we start streaming, we hope that we're connected to the version of the data with the fastest transmission time. However, more servers in more data centers means more vulnerability. And every cloud-service-provider employee who has administrative access to a physical cloud server also has access to your data, so your vulnerability is replicated along with the data.

There's also the issue of back up. A cloud provider may or may not actually back up your data. Google, for example, does tape back ups of gmail, but as far as I know, does not back up its general storage system. Feel free to correct me if I'm wrong, but practically speaking, cloud applications have to assume that there's no underlying backup mechanism. The data is replicated on many servers, and the servers themselves use RAID drives (if they have disks in them), so it would take a global catastrophe to loose your data altogether, but it's not possible to go back in time as you could do using a back up tape. You can write a web application that transfers data from Google to your own local repository, but that's actually a surprisingly difficult (and painfully slow) operation to perform using JDO/JPA, which is the only access method that Google provides.

Security

The biggest problem with cloud applications is actually application security. Security is, of course, a huge problem with most software. Programmers are simply unaware of how to write secure applications, and management is typically unwilling to spend the paltry sum required for the training that would eliminate 90% of the problem. It's symptomatic that most of the really big breaches I've read about in the past few years have been done using SQL Injection, which is not only a venerable, well-understood exploit, but is literally a trivial matter to defeat. This is a case where 10 minutes of training could eliminate millions of dollars of vulnerability, but the training still doesn't happen.

One of the problems is perception. Application security has nothing at all to do with things like firewalls and SSL. The IT department is simply not involved. Most exploits attack an application through a bug of some sort, usually a minor one. And most hackers get at that bug simply by using the program in predictable ways, just like all your other users do. There are no secret back doors, and the vulnerabilities are typically in plain sight. For example, the classic example of a SQL-injection exploit that you find in the books shows you how to get a dump of someone's entire database by exploiting a very-simple bug in a website's password-recovery page (access to which doesn't typically require a password, I would hope). It doesn't matter whether you access that page using HTTPS; if you can access the page at all, you can do the damage.

So, the biggest security problem with a cloud-based Web 2.0 application is the size of the *attack surface* — the number of places where a hacker can potentially use a bug to break into the system. Most Web 2.0 applications use remote procedure calls, or an equivalent mechanism like REST, heavily; and every one of those calls — in fact, every ar-

gument to every one of those calls — represents a potential vulnerability. It's easy to deal with these problems if you know about them (e.g., check that all arguments are valid and reasonable on both the client and server side; if you're using Google's Web Toolkit, you can literally use the same Java code on both sides to do the checking).

The real solution to this problem is simple: training.

The Entire Ecosystem

There are a few loose ends to cover. First, Google provides a reasonably rich set of support services for your web application. Get complete details at http://code.google.com/appengine/docs/java/apis.html, but here's a list of functionalities to be aware of. I'll demonstrate a few of these in future articles.

- Blobstore: Lets you store very large objects that can't be handled by the standard JDO mechanism, and serve them directly to your users if you like. It's handy for things like big images.
- Capabilities: A management API that lets you dynamically detect whether other Google services are operational. You can use it to disable features of your own app when a Google service on which it depends goes down for maintenance.
- Channel: A mechanism for pushing information down to a browser-based client, so that client can update itself without polling.
- Images: A library for doing simple image manipulation. Supports transformations like resizing, rotation, darkening and lightening the image, etc.
- Mail: Lets you both send and receive email from your application. You send using standard JavaMail APIs. You receive by writing a servlet that waits for email to arrive. (Google receives the email, then posts it to your servlet.) This facility is useful, but Google limits the number of emails that you can send in a day to 500, so you can't use this service for mailing lists or bulk mail.
- Memcache: A mechanism for caching chunks of data in "memory." This is a wrapper around Java's JCache APIs. Caching through this API is better than rolling your own cache because Memcache can scale properly if the application is running on multiple machines.
- Multitenency: Effectively adds namespaces to the storage system so that you can partition your data easily.
- OAuth: Provides a mechanism to grant third-party access to Google services. For example, a customer of yours could use this mechanism to allow your application to access his Google Docs files for persistent storage.
- Task Queues: Allows your application to execute background tasks that are not necessarily triggered by a user action.
- URL Fetch: A wrapper around java.net.URL and related classes that lets you access other web content using URLs. Handy for doing things like sending bulk email from a non-Google server.
- Users: Allows you to use Google's log-in mechanism for your application. That is, one of your users can log in to your application using Google's login page. I have mixed feelings about this service because it's one of the few services that doesn't just implement a standard Java library. If you use it as your sole login mechanism, you're effectively giving your user list to Google, and I'd rather know who my users are, thank you.
- XMPP: Support for XMPP-compatible IM services (like Google Talk).

APIs

The final thing to think about are the services that can coexist with your web application. Google, under the moniker "GData," provides API access to literally all of its web applications — from Calendar to YouTube, making it easy to do things like integrate a Google Docs page into your application or update an appointment on a Google Calendar.

The complete list of APIs is available for browsing at http://code.google.com/apis/gdata/docs/directory.html. Most of these are simple REST-based APIs. You typically encode a request in the URL and HTTP GET or POST, and receive a result in JSON. However, Google provides both Java and Python libraries that wrap the REST APIs, and it also provides an Eclipse plug-in that makes it easier to write to the APIs.

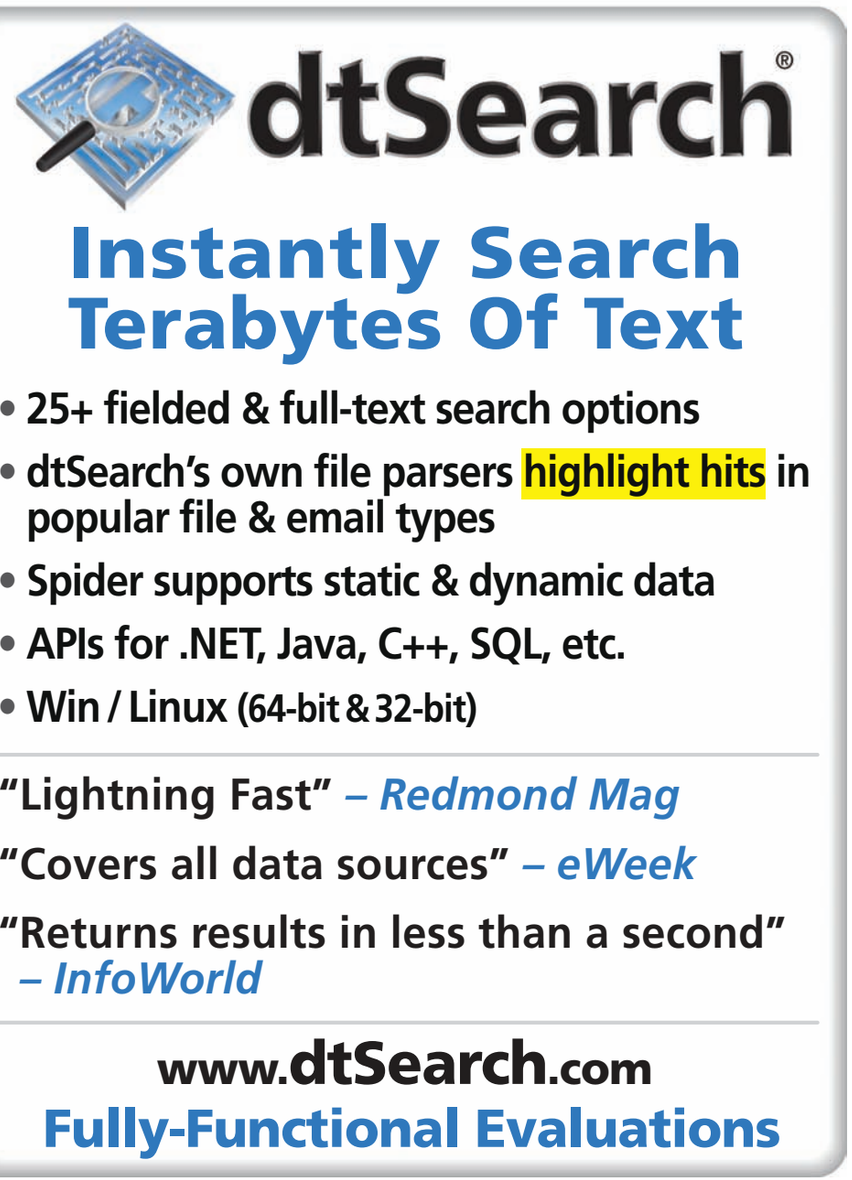
Related Articles

Getting Started With the Cloud: Logging On With Google OAuth (http://www.drdobbs.com/web-development/229625374).

Getting Started with Google Apps and OAuth (http://www.drdobbs.com/web-development/229401853).

— Allen Holub provides technical training, OO design and Agile-process consulting, and web-application/SaaS development services. He is the author of **Holub on Patterns: Learning Design Patterns by Looking at Code** *(http://is.gd/fn4lrA), and numerous articles for SD Times, JavaWorld, and IBM Developer Works. Contact him via http://www.holub.com/contact.*

Comment



This Month on DrDobbs.com

Items of special interest posted on www.drdobbs.com over the past month that you may have missed

ORACLE AND THE END OF PROGRAMMING AS WE KNOW IT

If Oracle prevails in its claim that APIs can be copyrighted, nearly every aspect of programming will be changed for the worse.

http://www.drdobbs.com/jvm/232901227

UNDERSTANDING CLIENT-SIDE STORAGE IN WEB APPS

Pamela Fox compares the four major APIs for browser-side storage and explains how to balance performance, data size, and browser support.

http://www.drdobbs.com/web-development/232900805

USING SQLITE ON ANDROID

With a little care, SQLite can be used as a data store or full database on Android devices.

http://www.drdobbs.com/database/232900584

STOCK IN TRADE

The classic trade in computer science is trading memory for execution speed.

http://www.drdobbs.com/blogs/embedded-systems/232901504

CALLING CONSTRUCTORS WITH PLACEMENT NEW

Trying to create an object at a specific memory address highlights the peculiarities of C++ constructors.

http://www.drdobbs.com/cpp/232901023

THE HUMBLE BOOLEAN DESERVES HELP

The Boolean data type rarely gets the attention it merits. As a result, it's stuck in a world of exactly two values, with little margin for safety.

http://www.drdobbs.com/architecture-and-design/232900836

VOLDEMORT TYPES IN D

Sometimes, the confluence of existing features can yield unexpected surprises.

http://www.drdobbs.com/blogs/cpp/232901591

EASY-TO-USE ARM SYSTEM ON A BOARD

The Raspberry Pi is ushering in a new era of powerful, stunningly low-cost PCs on a board not much larger than an Arduino Uno.

http://www.drdobbs.com/blogs/tools/232901444

IN THIS ISSUE

Dr. Dobb's

Andrew Binstock Editor in Chief, Dr. Dobb's
alb@drdobbs.com

Deirdre Blake Managing Editor, Dr. Dobb's
dblake@techweb.com

Amy Stephens Copyeditor, Dr. Dobb's
astephens@techweb.com

Sean Coady Webmaster, Dr. Dobb's
scoady@techweb.com

J. D. Hildebrand Editor at Large, Dr. Dobb's
jdhildebrand@drdobbs.com

Jon Erickson Editor in Chief Emeritus , Dr. Dobb's

CONTRIBUTING EDITORS
Scott Ambler
Mike Riley
Herb Sutter

DR DOBB'S
UBM TECHWEB
303 Second Street,
Suite 900, South Tower
San Francisco, CA 94107
1-415-947-6000

INFORMATIONWEEK

Rob Preston VP and Editor In Chief, InformationWeek
rpreston@techweb.com 516-562-5692

John Foley Editor, InformationWeek
jpfoley@techweb.com 516-562-7189

Chris Murphy Editor, InformationWeek
cjmurphy@techweb.com 414-906-5331

Art Wittmann VP and Director, Analytics, InformationWeek
awittmann@techweb.com 408-416-3227

Alexander Wolfe Editor In Chief, InformationWeek.com
awolfe@techweb.com 516-562-7821

Stacey Peterson Executive Editor, Quality, InformationWeek speterson@techweb.com 516-562-5933

Lorna Garey Executive Editor, Analytics, InformationWeek
lgarey@techweb.com 978-694-1681

Stephanie Stahl Executive Editor, InformationWeek
sstahl@techweb.com 703-266-6030

Fritz Nelson VP and Guest Editorial Director
fnelson@techweb.com 949-223-3608

David Berlind Chief Content Officer, TechWeb
dberlind@techweb.com 978-462-5315

REPORTERS

Charles Babcock Editor At Large
Open source, infrastructure, virtualization
cbabcock@techweb.com 415-947-6133

Thomas Claburn Editor At Large
Security, search, Web applications
tclaburn@techweb.com 415-947-6820

Paul McDougall Editor At Large
Software, IT services, outsourcing
pmcdougall@techweb.com

Marianne Kolbasuk McGee Senior Writer IT management and careers
mmcgee@techweb.com 508-697-0083

J. Nicholas Hoover Senior Editor
Desktop software, Enterprise 2.0, collaboration
nhoover@techweb.com 516-562-5032

Andrew Conry-Murray New Products and Business Editor
Information and content management
acmurray@techweb.com 724-266-1310

W. David Gardner News Writer
Networking, telecom
wdavidg@earthlink.net

Antone Gonsalves News Writer
Processors, PCs, servers
antoneg@pacbell.net

Eric Zeman
Mobile and Wireless
eric@zemanmedia.com

CONTRIBUTORS

Michael Biddick mbiddick@nwc.com
Michael A. Davis mdavis@nwc.com
Jonathan Feldman jfeldman@nwc.com
Randy George rgeorge@nwc.com
Michael Healey mhealey@nwc.com

EDITORS

Jim Donahue Chief Copy Editor
jdonahue@techweb.com

ART/DESIGN

Mary Ellen Forte Senior Art Director
mforte@techweb.com

Sek Leung Senior Designer
sleung@techweb.com

INFORMATIONWEEK ANALYTICS
analytics.informationweek.com

Art Wittmann VP and Director
awittmann@techweb.com 408-416-3227

Lorna Garey Executive Editor, Analytics
lgarey@techweb.com 978-694-1681

Heather Vallis Managing Editor, Research
hvallis@techweb.com 508-416-1101

INFORMATIONWEEK.COM

Benjamin Tomkins Managing Editor
btomkins@techweb.com 516-562-5336

Roma Nowak Senior Director, Online Operations and Production
rnowak@techweb.com 516-562-5274

Tom LaSusa Managing Editor, Newsletters
tlasusa@techweb.com

Jeanette Hafke Web Production Manager
jhafke@techweb.com

Joy Culbertson Web Producer
jculbertson@techweb.com

Nevin Berger Senior Director, User Experience
nberger@techweb.com

Steve Gilliard Senior Director, Web Development
sgilliard@techweb.com

Copyright 2012 United Business Media LLC. All rights reserved.

INFORMATIONWEEK ADVISORY BOARD

Dave Bent
Senior VP and CIO
United Stationers

Robert Carter
Executive VP and CIO
FedEx

Michael Cuddy
VP and CIO
Toromont Industries

Laurie Douglas
Senior CIO
Publix Super Markets

Dan Drawbaugh
CIO
University of Pittsburgh Medical Center

Jerry Johnson
CIO
Pacific Northwest National Laboratory

Kent Kushar
VP and CIO
E.&J. Gallo Winery

Carolyn LEclispe CDTon
Director, E-Services
California Office of the CIO

Jason Junenard
Managing Director
Wells Fargo Securities

Randall Mott
Sr. Executive VP and CIO
Hewlett-Packard

Denis O'Leary
Former Executive VP
Chase.com

Mykolas Rambus
CEO
Wealth-X

M.R. Rangaswami
Founder
Sand Hill Group

Manjit Singh
CIO
Las Vegas Sands

David Smoley
CIO
Flextronics

Ralph J. Szygenda
Former Group VP and CIO
General Motors

Peter Whatnell
CIO
Sunoco

UBM TECHWEB

Tony L. Uphoff CEO

John Dennehy CFO

David Michael CIO

Joseph Braue Sr. VP, Light Reading Communications Network

Scott Vaughan CMO

Ed Grossman Executive Vice President, InformationWeek Business Technology Network

John Ecke VP and Group Publisher, Financial Technology Network, InformationWeek Government, and InformationWeek Healthcare

Martha Schwartz EVP, Group Sales, InformationWeek Business Technology Network

Beth Rivera Senior VP, Human Resources

Eric Lundquist VP and Guest Editorial Analyst, InformationWeek Business Technology Network

David Berlind
Chief Content Officer, TechWeb, and Editor in Chief, TechWeb.com

Fritz Nelson VP and Guest Editorial Director, InformationWeek Business Technology Network, and Executive Producer, TechWeb TV

UNITED BUSINESS MEDIA LLC

Pat Nohilly Sr. VP, Strategic Development and Business Administration

Marie Myers Sr. VP, Manufacturing

INFORMATIONWEEK VIDEO

informationweek.com/tv

Fritz Nelson Executive Producer
fnelson@techweb.com

INFORMATIONWEEK BUSINESS TECHNOLOGY NETWORK

DarkReading.com
Security
Tim Wilson, Site Editor
wilson@darkreading.com

IntelligentEnterprise.com
App Architecture
Doug Henschen, Editor in Chief
dhenschen@techweb.com

NetworkComputing.com
Networking , Communications, and Storage
Mike Fratto, Site Editor
mfratto@techweb.com

PlugIntoTheCloud.com
Cloud Computing
John Foley, Site Editor
jpfoley@techweb.com

InformationWeek SMB
Technology for Small and Midsize Business
Benjamin Tomkins, Site Editor
btomkins@techweb.com

Dr. Dobb's
Good Stuff for Serious Developers
Andrew Binstock
Editor in Chief
alb@drdobbs.com

Dr.Dobb's Business Contacts

INFORMATIONWEEK BUSINESS TECHNOLOGY NETWORK

EVP of Group Sales, InformationWeek Business Technology Network, Martha Schwartz
(212) 600-3015, mschwartz@techweb.com

Sales Assistant, Salvatore Silletti
(212) 600-3327, ssilletti@techweb.com

SALES CONTACTS—WEST

Western U.S. (Pacific and Mountain states) and Western Canada (British Columbia, Alberta)

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

Strategic Accounts

Account Director, Sandra Kupiec
(415) 947-6922, skupiec@techweb.com

Account Manager, Vesna Beso
(415) 947-6104, vbeso@techweb.com

Account Executive, Matthew Cohen-Meyer
(415) 947-6214, mmeyer@techweb.com

MARKETING

VP, Marketing, Winnie Ng-Schuchman
(631) 406-6507, wng@techweb.com

Marketing Director, Angela Lee-Moll
(516) 562-5803, aleemoll@techweb.com

Marketing Manager, Monique Kakegawa
(949) 223-3609, mluttrell@techweb.com

Director, Client Marketing, Michelle Somers
(516) 562-7928, msomers@techweb.com

SALES CONTACTS—EAST

Midwest, South, Northeast U.S. and Eastern Canada (Saskatchewan, Ontario, Quebec, New Brunswick)

District Manager, Steven Sorhaindo
(212) 600-3092, ssorhaindo@techweb.com

Strategic Accounts

District Manager, Mary Hyland
(516) 562-5120, mhyland@techweb.com

Account Manager, Tara Bradeen
(212) 600-3387, tbradeen@techweb.com

Account Manager, Jennifer Gambino
(516) 562-5651, jgambino@techweb.com

Account Manager, Elyse Cowen
(212) 600-3051, ecowen@techweb.com

Sales Assistant, Kathleen Jurina
(212) 600-3170, kjurina@techweb.com

AUDIENCE DEVELOPMENT

Director, Karen McAleer
(516) 562-7833, kmcaleer@techweb.com

BUSINESS OFFICE

General Manager, Marian Dujmovits

United Business Media LLC
600 Community Drive
Manhasset, N.Y. 11030 (516) 562-5000
Copyright 2012. All rights reserved.

Entire contents Copyright © 2012, Techweb/United Business Media LLC, except where otherwise noted. No portion of this publication may be reproduced, stored, transmitted in any form, including computer retrieval, without written permission from the publisher. All Rights Reserved. Articles express the opinion of the author and are not necessarily the opinion of the publisher. Published by Techweb, United Business Media, 303 Second Street, Suite 900 South Tower, San Francisco, CA 94107 USA 415-947-6000.

UBM TECHWEB

Tony L. Uphoff CEO

John Dennehy CFO

David Michael CIO

Joseph Braue Sr. VP, Light Reading Communications Network

Scott Vaughan CMO

Ed Grossman Executive Vice President, InformationWeek Business Technology Network

John Ecke VP and Group Publisher, Financial Technology Network, InformationWeek Government, InformationWeek Healthcare

Martha Schwartz EVP, Group Sales, InformationWeek Business Technology Network

Beth Rivera Senior VP, Human Resources

David Berlind Chief Content Officer, TechWeb, and Editor in Chief, TechWeb.com

Fritz Nelson VP, Editorial Director, InformationWeek Business Technology Network, and Executive Producer, TechWeb TV

Eric Lundquist VP and Editorial Analyst, InformationWeek Business Technology Network

UNITED BUSINESS MEDIA LLC

Pat Nohilly Sr. VP, Strategic Development and Business Admin.

Marie Myers Sr. VP, Manufacturing