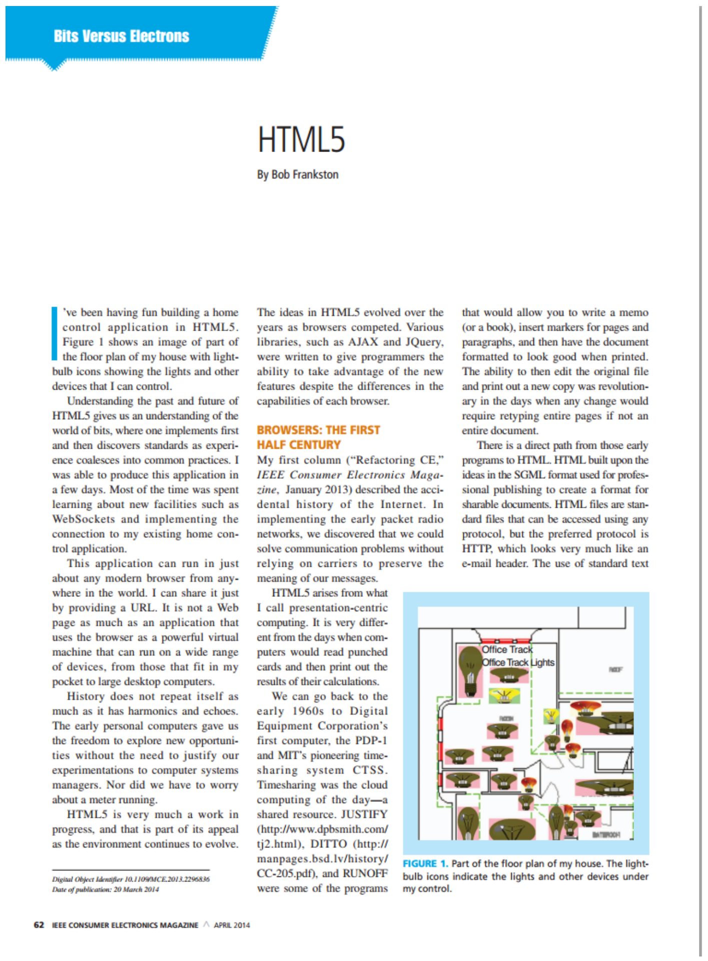
HTML5



Bits Versus Electrons

HTML5

By Bob Frankston

I've been having fun building a home control application in HTML5. Figure 1 shows an image of part of the floor plan of my house with lightbulb icons showing the lights and other devices that I can control.

Understanding the past and future of HTML5 gives us an understanding of the world of bits, where one implements first and then discovers standards as experience coalesces into common practices. I was able to produce this application in a few days. Most of the time was spent learning about new facilities such as WebSockets and implementing the connection to my existing home control application.

This application can run in just about any modern browser from anywhere in the world. I can share it just by providing a URL. It is not a Web page as much as an application that uses the browser as a powerful virtual machine that can run on a wide range of devices, from those that fit in my pocket to large desktop computers.

History does not repeat itself as much as it has harmonics and echoes. The early personal computers gave us the freedom to explore new opportunities without the need to justify our experimentations to computer systems managers. Nor did we have to worry about a meter running.

HTML5 is very much a work in progress, and that is part of its appeal as the environment continues to evolve.

Digital Object Identifier 10.1109/MCE.2013.2296836
Date of publication: 20 March 2014

The ideas in HTML5 evolved over the years as browsers competed. Various libraries, such as AJAX and JQuery, were written to give programmers the ability to take advantage of the new features despite the differences in the capabilities of each browser.

BROWSERS: THE FIRST HALF CENTURY

My first column ("Refactoring CE," *IEEE Consumer Electronics Magazine*, January 2013) described the accidental history of the Internet. In implementing the early packet radio networks, we discovered that we could solve communication problems without relying on carriers to preserve the meaning of our messages.

HTML5 arises from what I call presentation-centric computing. It is very different from the days when computers would read punched cards and then print out the results of their calculations.

We can go back to the early 1960s to Digital Equipment Corporation's first computer, the PDP-1 and MIT's pioneering time-sharing system CTSS. Timesharing was the cloud computing of the day—a shared resource. JUSTIFY (http://www.dpbsmith.com/tj2.html), DITTO (http://manpages.bsd.lv/history/CC-205.pdf), and RUNOFF were some of the programs that would allow you to write a memo (or a book), insert markers for pages and paragraphs, and then have the document formatted to look good when printed. The ability to then edit the original file and print out a new copy was revolutionary in the days when any change would require retyping entire pages if not an entire document.

There is a direct path from those early programs to HTML. HTML built upon the ideas in the SGML format used for professional publishing to create a format for sharable documents. HTML files are standard files that can be accessed using any protocol, but the preferred protocol is HTTP, which looks very much like an e-mail header. The use of standard text

FIGURE 1. Part of the floor plan of my house. The lightbulb icons indicate the lights and other devices under my control.

62 IEEE CONSUMER ELECTRONICS MAGAZINE ^ APRIL 2014

If you can you should read the IEEE Version of this article where it is better formatted and also provides the IEEE with feedback about readership.

Note that my previous article is "*Life (yet to be) Scripted*".

I've been having fun building a home control application in HTML5. You can see a part of the image of a floor in my house with light bulb icons showing the lights and other devices that I can control.

Understanding the past and future of HTML5 gives us an understanding of the world of bits where one implements first and then discovers standards as experience coalesces into common practices. I was able to produce this application in a few days – most of the time was spent learning about new facilities such as WebSockets and implementing connection to my existing home control application.

This is an application which can run in just about any modern browser from anywhere in the world. I can share it just by providing a URL. It's not a web page as much as an application that uses the browser as a powerful virtual machine that can run on a wide range of devices from those that fit in my pocket to large desktop computers.

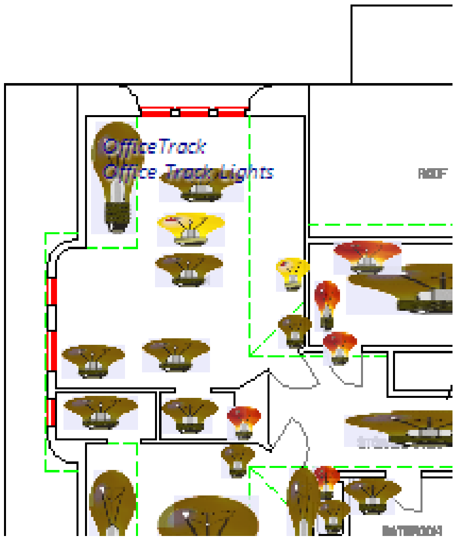
History doesn't repeat itself as much as it has harmonics and echoes. The early personal computers gave us the freedom to explore new opportunities without the need to justify our experimentations to computer systems managers. Nor did we have to worry about a meter running.

HTML5 is very much a work-in-progress and that's part of its appeal as the environment continues to evolve. The ideas in HTML5 evolved over the years as browsers competing. Various libraries such as AJAX and JQuery were written to give programmers the ability to take advantage of the new features despite the differences in the capabilities of each browser.

Browsers: The First Half Century

My first column (Refactoring CE, *IEEE Consumer Electronics Magazine*, January 2013) described the accidental history of the Internet. In implementing the early packet radio networks we discovered that we could solve communications problems without relying on carriers to preserve the meaning of our messages.

HTML5 arises from what I call presentation-centric computing. It's very different from the days when computers would read punched cards and then print out the results of their calculations.



We can go back to the early 1960's to DEC's first computer, the PDP-1 and MIT's pioneering timesharing system CTSS. Timesharing was the cloud computing of the day – a shared resource. JUSTIFY[i], DITTO[ii] and RUNOFF were some of the programs that would allow you to write a memo (or a book) and insert markers for pages and paragraphs and then have the document formatted to look good when printed. The ability to then edit the original file and print out a new copy was revolutionary in the days when any change would require retyping entire pages if not an entire document!

There is a direct path from those early programs to HTML. HTML built upon the ideas in the SGML format used for professional publishing to create a format for sharable documents. HTML files were standard files that could be accessed using any protocol but the preferred protocol is HTTP which looks very much like an email header. The use of standard text files allowed people to implement and experiment without having to build special tools. Anyone could read an HTML file or an HTTP message and that readability engendered a degree of trust.

This open format along with browsers that ignored unrecognized tags allow for experimentation with new tags such as <table>. The addition of JavaScript and the ability to embed new objects enabled further evolution. The initial JavaScript took about 10 days and has been evolving ever since.

The asynchronous loading of elements created an opportunity for the browser to expand its role from being a program that merely presented remote data to one that could have local smarts and fetch remote data as needed. I remember experimenting with those new capabilities in the 1990's.

Steve Ward took this effort a step further with his CURL language in 1998. But the market has taken a slower route to local programming in HTML5.

The constraints of HTML and the browsers contributed to success of the web. First is the concern about making it safe to use browser-based applications. This means the environment isn't suitable for all applications but limiting oneself to working within the browser removes a major barrier to the adoption of the application.

The asynchronous programming environment encourages a very responsive programming style while avoiding many of the complexities of using threads in native applications. Here too there is a benefit in choosing to limit oneself to the browser environment.

Programs can take advantage of the rendering heritage by manipulating objects and classes rather than having to do their own drawing. For example one makes an object (or a whole class of objects) appear or disappear by just setting the "visibility" property with a single line of code. The browser quickly updates presentation to reflect that change.

The loose programming environment encourages experimentation and users can tolerate most failures. Of course a heavy duty commercial site has high standards but, for the most part, users can handle issues, especially if they are fixed by something as simple as a redisplay.

The JavaScript language is also continuing to evolve. TypeScript (which I am using) and CoffeeScript represent the future direction of JavaScript and features from these preprocessors will become part of the standards over time.

Of course an interpretative environment like JavaScript has to be slow. Or does it? It's amazing what one could accomplish when there is the incentive to improve. JavaScript performance has improved rapidly using such techniques as recognizing patterns at runtime!

HTML5 Today

The process of implement first and standardize later has worked very well. Today's HTML5 is a plateau in a continuing evolution. Features that were formerly available in libraries are now natively available in current browsers. The development environments have also improved both within the browsers and externally.

Today's browser is a rich programming environment with strong text, video and audio presentation capabilities as well as vector and pixel-based graphics. The application can also use local storage on the platforms so it can function as a standalone application. WebSockets offers powerful communications capabilities. WebRTC brings voice communications capabilities to HTML5 -- basically making a telephone just another software component.

PhoneGap is one of the wrappers which can be used to make an HTML5 application run as a native application across a number of platforms. Adobe has a service that will take your application and adapt it to each platform for free (at least for modest use)!

RESTful servers complement the browser by building on the HTTP stateless relationships. This makes it easy for the browser application to access both local and remote services. You can think of a RESTful interface as a web form used by a program.

I can't begin to explain the full richness of today's HTML5 environment – all I can do is whet your appetite and encourage experimentation.

The Future?

It's not easy to separate the future from the present as much of the future is here now in the form of ongoing experiments and capabilities.

In my home control example I had to create icons to represent the devices in my house and do the programming necessary so that clicking the icon would control the actual device and, conversely, reflect any changes in the state of the actual device in the icon.

This capability is already available in the form of widgets that can be embedded on web pages but the standards are not quite there. In fact the idea of micro-formats in the form of XML objects is an old one but such standards have had difficulty gaining traction on their own. But as standards and common framework develops they will come into their own.

As a strong reminder that this isn't just about the Internet as such, we can transfer this information via NFC or QR Codes or any other means. A URL is simply one way to provide a reference. It uses the DNS as its dictionary.

Just as personal computers pulled back control from the centralized computing cloud of the 1970's, the local computing capabilities of the HTML5 environment offers the possibility of increased local control of computing. The so-called cloud or shared capabilities will remain important but the relationship has changed. Rather than just browsing the cloud, a browser is now a peer that uses the cloud as a resource. Don't let the legacy use of "browser" confuse you.

In thinking about the future it's useful to look at the accidental path that took us to HTML5. I'm surprised at how complex the HTML platform is. I had advocated far simpler platforms. Perhaps it's similar to the tension between simpler processor architectures (RISC) and more complex ones (CISC). It's very expensive to create complex systems but when investment is concentrated on a common goal we can build very rich platforms. HTML5 has lots of special case quirks because it's more an agglomeration of experiments than a single engineered whole. But as a by-product we also have a very powerful general purpose platform that can be used for any purpose.

The reason I use the term "refactoring" is that complexity is not intrinsic. We can ignore that complexity of implementing the platform itself and instead take advantage of the (relative) simplicity of using the platform to create applications which take advantage of the new opportunities.

What happens to the telecommunications industry when WebRTC makes telephony just another application? What happens to the television business when a "smart TV" is just another generic device with a screen and an HTML5 engine? Stay tuned to find out …

[i] http://www.dpbsmith.com/tj2.html
[ii] http://manpages.bsd.lv/history/CC-205.pdf