



# ChemDoodle Web Components

By ICLKevin

Created Sep 16 2009 - 2:08pm

## Protein Data Bank Demo and Canvas Technology Analysis

In this article, I provide some background and analysis of the javascript/*Canvas* technology along with future prospects followed by a step-by-step walkthrough on how to quickly put together a rich web application with the [ChemDoodle Web Components](#) [1]. In the walkthrough, I produce a small application that queries the [Protein Data Bank](#) [2] (PDB), displays 3D animations of the results, and saves a viewing history. The example is of moderate difficulty, and the reader should have prior knowledge of HTML. Javascript is also important, but those experienced in other programming languages should be able to follow the code with ease.

### About the Author

Kevin Theisen is the President of [iChemLabs](#) [3], which funds, develops and hosts the open source ChemDoodle Web Components.

### Introduction and Background

ChemDoodle Web Components are pure javascript objects derived from [ChemDoodle](#) [4]™ to solve common chemistry related tasks on the web. These components are powerful, fully customizable, easy to implement, and are free under the open source GPL license (this does not mean that your website needs to be GPL). They leverage the technology of the [HTML 5](#) [5] specification's [Canvas element](#) [6]. The *Canvas* element allows web developers to dynamically draw 2D vector graphics in a web page without the use of 3rd party plugins, such as [Flash](#) [7] or [Silverlight](#) [8]. The HTML 5 specification has not been completely adopted by all browsers yet, but it should be fully implemented on most browsers soon. On Mac OSX, [Apple Safari 4+](#) [9], [Mozilla Firefox 3.5+](#) [10] and [Google Chromium](#) [11] all support *Canvas* beautifully. By utilizing this powerful element, we were able to develop a software development kit, complete with graphical user interface components, completely in javascript. By using javascript to create scientific applications, implementation is simplified, extensibility is improved, security issues are handled by the browser and the most widespread programming language in the world is now more accessible to scientists.

The ChemDoodle Web Components currently provide 6 extendable graphical components as well as a growing cheminformatics library:

1. **Viewer**- view a static styled drawing
2. **Rotator**- provide a rotating animation of structures
3. **Transformer**- rotate, scale and translate structures
4. **Doodler**- draw structures with an interface that mimics ChemDoodle
5. **MolGrabber**- provide access to databases such as [PubChem](#) [12] right on your website
6. **File Loader**- allow users to load their own molecule files straight to your web based algorithms

### 3D Graphics and OpenGL ES

The *Canvas* element, as currently implemented, allows for only 2D graphics. That is all about to change. Only a couple days ago, [WebGL](#) [13] (complete with a very impressive video!) was quietly introduced into the nightly build of [WebKit](#) [14]. Some readers may already be very familiar with WebKit, as it is the open source browser engine that runs Safari. WebGL opens new possibilities

for the *Canvas* element, and I quote the best explanation I've seen so far, "The goal of WebGL is to expose the low-level OpenGL ES 2.0 APIs through JavaScript so that they can be used to draw hardware-accelerated 3D graphics in the HTML Canvas element" ([source](#) <sup>[15]</sup>). We will soon be able to develop fully native web applications with 2D and 3D graphical user interfaces completely in javascript for web browsers. Imagine advanced quantum computations with beautiful output, wide-spread open source simulation packages, and functional molecular modelers all optimized for the web and easily integrated into Web 3.0 sites! We will be developing 3D ChemDoodle Web Components as soon as WebGL is adopted by the browsers, and we aim to provide the funding, development and support for this open source package that the community requires.

While [OpenGL ES](#) <sup>[16]</sup> (OpenGL for **E**mbedded **S**ystems) is only a subset of OpenGL, it has proven to be quite effective in producing great 3D molecular graphics on non-desktop platforms. Brad Larson at [Sunset Lake Software](#) <sup>[17]</sup> mastered the technology to create the impressive [Molecules](#) <sup>[18]</sup> app on the iPhone. Expect this technology to allow for quick and engrossing graphics without the need for fully installed applications on the desktop.

## Future Expectations

Rich Apodaca continues to be at the forefront of emerging web technologies and provides insight on their impact to the sciences, including [javascript/Canvas](#) <sup>[19]</sup>, on his blog [Depth-First](#) <sup>[20]</sup>. Over the past year, they have seen the *Canvas* technology develop and have investigated many of its shortcomings. The community continues to push for more robust *Canvas* features and there has been significant progress and the technology is entirely capable of supporting 2D graphical scientific applications.

One issue is with competing technologies, such as applets, other multimedia plugins and those technologies not yet developed. While it is true that the internet is a continuously morphing environment, this author is confident that *Canvas* technology will be running the websites of the future. The reasoning is that it is a native browser technology, and users will be likely to choose it over cumbersome 3rd party plugins. Take CSS for instance, if website publishers are given a choice between using CSS or some other 3rd party plugin to organize their HTML pages, the resounding answer would be CSS. Additionally, web site visitors don't want to wait for applets to load and developers don't want the hassles associated with them. In order to compete in the future, developers will have to create engrossing Web 3.0 sites, and *Canvas* is the answer.

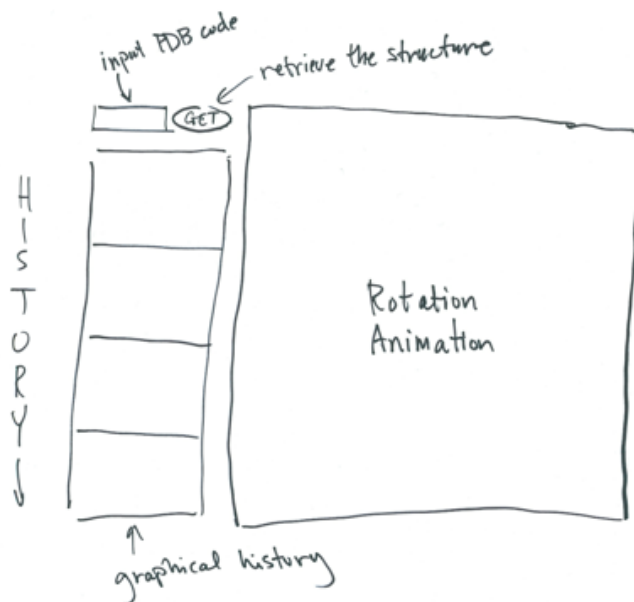
In another interesting twist a few months ago, [Google Chrome OS](#) <sup>[21]</sup> was announced. It is an interesting proposal, assuming users that spend most of their time on the web really only need a web browser on their computer. The conclusion is that a huge part of the market will benefit from a browser based OS that has minimal loading times and takes users straight to the web. For scientists, this seems a little far-fetched, as the very complex applications we have become reliant on are desktop applications. Porting them to javascript or even beginning to solve such a transition seems very unlikely. This is compounded by the difficulty inherent with creating javascript applications: debugging is incredibly tedious and there isn't a good IDE in sight. However, javascript is the most distributed runtime on the planet and we can be sure there will be several situations where scientists will require just web applications that are continuously updated and easily distributed. Certainly, if Google Chrome OS takes over, we will have a basis to continue to create great software.

## Walkthrough for a Demo PDB Web Application

Now that I have provided some background on the technology, I will be demonstrating how to write a web application using the ChemDoodle Web Components. The application has a simple goal: visitors search the PDB using structure codes and then view an animation of the result, while the most recently viewed structures are saved in a graphical history. The [finished product can be seen here](#) <sup>[22]</sup>. Use the browser's *View Source* function to see the source in its entirety. Note that previously *Canvas* referred to the HTML 5 element. *Canvas* from now on will refer to the base ChemDoodle Web Component class of the same name.

To begin, I always sketch a layout of how the application appears in my mind. I pictured a simple

layout, with the history to the left and search form at the top-left corner.



Conveniently, the use of the ChemDoodle Web Components allows me to produce any layout with any functionality I wish. Relying on applets may confine us to using a predefined interface. Since I want the left and right sections to align horizontally, I'll use a HTML *table* for simplicity. The table will have 1 row and 3 columns, 1 for an image used to provide explanation for the empty history components, 1 for the history and search form, and 1 for the animation.

```

1. <table>
2.     <tr>
3.         <td>
4.             <!--image to provide some explanation of the history components-->
5.             
6.         </td>
7.         <td>
8.             <!--the search form on top, with the history components below-->
9.         </td>
10.        <td>
11.            <!--the large rotation animation-->
12.        </td>
13.    </tr>
14. </table>

```

The first cell contains a single image. The second cell is the most complex, so I discuss it last. In the third cell I place a ChemDoodle Web Component `RotatorCanvas` that will display an animation of the main structure rotating about the y-axis. The `file2js` PHP script is provided with the ChemDoodle Web Components, and is an example of how to pass molecular data to the javascript components.

```

1.     <script>
2.         // construct the rotator component with 600x600px dimensions;
3.         // this variable is actually defined before the search form to ensure its existence
4.         // during page compilation
5.         rotate3D = new RotatorCanvas('rotate3D', 600, 600, true);
6.
7.         // call the yet to be written setupVisuals() function to standardize the style
8.         // for all components
9.         setupVisuals(rotate3D);
10.
11.        // rotate around the y-axis
12.        rotate3D.yIncrement = -Math [23].PI / 360;
13.        rotate3D.xIncrement = 0;
14.        rotate3D.zIncrement = 0;
15.    </script>

```

```

16.         // read in the PDB file and turn off ring perception (to improve efficiency, rings
17.         // aren't necessary for this app)
18.         var mol = readPDB(<?php file2js('./molecules/1BNA.pdb'); ?>);
19.         mol.findRings = false;
20.
21.         // load the molecule into the component
22.         rotate3D.loadMolecule(mol);
23.
24.         // begin rotation
25.         rotate3D.startRotation();
26.     </script>

```

The second cell has two types of components, the HTML form for PDB code input, and the user's most recent viewing history. The HTML form is a little complex as a trick is used to pass the PDB file to the javascript functions. Due to security issues, javascript cannot access local or server files, so a server side script accesses the data instead, as defined in the sample PHP file CDWProteinDataBank.php. This is an asynchronous call, and when finished, the PHP script loads the file into the hidden iframe below, which *onload*, calls the javascript function GetPdbFromFrame(). The CDWProteinDataBank.php file is provided with the ChemDoodle Web Components and should be used as an example only. You should write your own server side scripts to appropriately suit your own website's security and logistic concerns.

```

1.     <center>
2.         <!--generate the HTML [24] form that calls the server side CDWProteinDataBank script to
3.             retrieve the input structure. This is an asynchronous call, and when finished,
4.             the PHP script loads the file into the hidden iframe below, which onload, calls
5.             the javascript function GetPdbFromFrame(), which has not yet been defined. The
6.             ValidateMolecule() function ensures that a valid string was input into the text field-->
7.         <form name="PDBForm" method="POST" action="/CDWProteinDataBank.php" target="HiddenPDBFrame"
8.             onSubmit="ValidateMolecule(PDBForm); return false;">
9.             <!--text field-->
10.            <input type="text" name="q" value="" size="4" maxlength="4" />
11.            <!--submit button that displays 'Retrieve'-->
12.            <input type="submit" name="submitbutton" value="Retrieve" />
13.        </form>
14.
15.        <!--this hidden iframe holds the PDB file to be loaded by the javascript component, as
16.            javascript cannot directly access files for security reasons. Look at
17.            CDWProteinDataBank.php and the GetPdbFromFrame() function in this demo to see
18.            exactly how this works-->
19.        <iframe id="HMGF-rotate3D" name="HiddenPDBFrame" height="0" width="0"
20.            style="display:none;" onLoad="GetPdbFromFrame('HMGF-rotate3D', rotate3D)">
21.        </iframe>
22.    </center>

```

The user's viewing history will be placed below the search form. I would like this to be a graphical history, so I want the images of previous structures to be shown here. In addition, clicking the structure should transfer it to the main rotator and hovering should highlight it for visual feedback. The possibilities are endless, and I will extend the ChemDoodle Web Components' Canvas base class to provide this functionality after this section. I call the class HistoryCanvas, and very simply, I create four of them with a dimension that allows for horizontal alignment with the rotator canvas, and stack them on top of each other. I have now laid out all the components on the HTML page.

```

1.     <script>
2.         // initialize a history component to 140x140px, which fits well;
3.         // the HistoryCanvas class is one we will extend from the ChemDoodle Web Components
4.         // to match the intended functionality
5.         var mem1 = new HistoryCanvas('mem1', 140, 140, true);
6.
7.         // call the same setupVisuals() function as the rotator to standardize the graphics
8.         setupVisuals(mem1);
9.
10.        // load a placeholder molecule into the first history component to draw attention
11.        var mol = readPDB( <?php file2js('./molecules/1TRZ.pdb'); ?>);
12.        mol.findRings = false;
13.        mem1.loadMolecule(mol);
14.    </script>

```

```

15.     <br>
16.     <script>
17.         // repeat for 3 more history components
18.         var mem2 = new HistoryCanvas('mem2', 140, 140, true);
19.         setupVisuals(mem2);
20.         mem2.repaint();
21.     </script>
22.     <br>
23.     <script>
24.         var mem3 = new HistoryCanvas('mem3', 140, 140, true);
25.         setupVisuals(mem3);
26.         mem3.repaint();
27.     </script>
28.     <br>
29.     <script>
30.         var mem4 = new HistoryCanvas('mem4', 140, 140, true);
31.         setupVisuals(mem4);
32.         mem4.repaint();
33.     </script>

```

After the components have been defined, the rest of the functions need to be written. I'll begin with the most important function, which is actually a class. In javascript, all objects are functions and functions are objects. So we create the class by creating a function and declaring member methods and parameters. The history components to be added should display the structure, and should also be clickable. Once clicked, the saved molecule loads into the main rotator. Also, hovering over a history component should highlight it, like a hyperlink. The class is aptly named HistoryCanvas. In javascript, to create a subclass, I define the *prototype* of the child class. The *prototype* of the HistoryCanvas class is set to the ChemDoodle Web Components' Canvas class to absorb its graphical functionality.

```

1.     <script>
2.         // the HistoryCanvas class, takes the HTML id, width and height dimensions as parameters
3.         function HistoryCanvas(id, width, height) {
4.             // the static all Array contains all the components to receive
5.             // mouse and keyboard events from the Canvas user input manager
6.             all[all.length] = this;
7.
8.             // call the super's create() method to create the component and lay it out on
9.             // the HTML page
10.            this.create(id, width, height);
11.
12.            // save the last mouse event point
13.            this.p = null;
14.
15.            // override the Canvas abstract drawChildExtras() function, it is an optional
16.            // override, and does nothing if not overrode;
17.            // this function will draw extra graphics after the molecule has been drawn, a
18.            // green highlight will be drawn around the border of the component if the last
19.            // mouse event point is in the component
20.            this.drawChildExtras = function(ctx) {
21.                if (this.p != null) {
22.                    ctx.strokeStyle = '#ADFF75';
23.                    ctx.lineWidth = '4';
24.                    ctx.strokeRect(0, 0, this.width, this.height);
25.                }
26.            }
27.
28.            // another optional override, this receives the mouseup event from the Canvas
29.            // input manager;
30.            // when clicked, the component will place a copy of its current contents into the
31.            // main rotator after the molecule from the rotator has been pushed into the user's
32.            // viewing history
33.            this.mouseup = function(p) {
34.                if (p.x > 0 && p.y > 0 && p.x < this.width && p.y < this.height) {
35.                    var save = copy(this.molecule);
36.                    save.findRings = false;
37.                    cycleMolecules();
38.                    rotate3D.loadMolecule(save);
39.                }
40.            }

```

```

41.
42.         // another optional override, this receives the mouseexit event from the Canvas
43.         // input manager;
44.         // clears the highlight
45.         this.mouseexit = function(p) {
46.             if (this.p != null) {
47.                 this.p = null;
48.                 this.repaint();
49.             }
50.         }
51.
52.         // another optional override, this receives the mouse move event from the Canvas
53.         // input manager
54.         // shows the highlight if necessary
55.         this.move = function(p) {
56.             if (this.p == null && p != null) {
57.                 this.p = p;
58.                 this.repaint();
59.             }
60.         }
61.         return true;
62.     }
63.     // the HistoryCanvas class extends the ChemDoodle Web Components' base Canvas class
64.     HistoryCanvas.prototype = new Canvas [25]();
65. </script>

```

Now that the HistoryCanvas class has been defined, there are three functions left to be completed: setupVisuals(), GetPdbFromFrame(), and cycleMolecules(). The first, setupVisuals(), just sets the visual specifications of an input component so that all created components can be standardized by one function.

```

1.     <script>
2.         // this function sets up the visual specifications for the components, a single method
3.         // is used to standardize the graphics
4.         function setupVisuals(canvas) {
5.             // use Jmol colors for bonds, this will gradient between the two constituent atoms
6.             canvas.specs.bonds_useJmolColors = true;
7.
8.             // thicken the bonds
9.             canvas.specs.bonds_width = 3;
10.
11.            // turn on this setting for better contrast between overlapping bonds
12.            canvas.specs.bonds_clearOverlaps = true;
13.
14.            // turn off atoms, so we get a wireframe like model
15.            canvas.specs.atoms_display = false;
16.
17.            // set the background to black for a classic modeling look
18.            canvas.specs.backgroundColor = 'black';
19.        }
20.    </script>

```

The second method, GetPdbFromFrame(), is the function to be called after the PDB file has been retrieved and inserted into the hidden frame. The function retrieves the file, parses it, pushes the currently rotating structure into the user's viewing history and loads the new structure.

```

1.     <script>
2.         // this function is called when the hidden iframe is filled with a PDB file;
3.         // this function will read the PDB file from the hidden iframe, parse it, and load
4.         // it into the rotator
5.         function GetPdbFromFrame(frameId, canvas) {
6.             // if canvas (input component) is null, return, this should never be the case
7.             if (!canvas) {
8.                 return;
9.             }
10.
11.            // get the PDB file from the hidden frame
12.            var mol = document.getElementById(frameId).contentDocument.body.innerHTML;
13.

```

```

14.         // if the server encountered an error, this will be output, can be changed in
15.         // CDWProteinDataBank.php
16.         if (mol.match('^ChemDoodle Web Components Query Error.')) {
17.             alert(mol);
18.         }
19.         else {
20.             // stop the main rotation during the load
21.             canvas.stopRotation();
22.
23.             // push the current structure in the rotator down into the user's viewing
24.             // history
25.             cycleMolecules();
26.
27.             // parse and create the molecule
28.             var m = readPDB(mol);
29.             m.findRings = false;
30.
31.             // load the molecule into the rotator
32.             canvas.loadMolecule(m);
33.
34.             // reanimate the component
35.             canvas.startRotation();
36.         }
37.     }
38. </script>

```

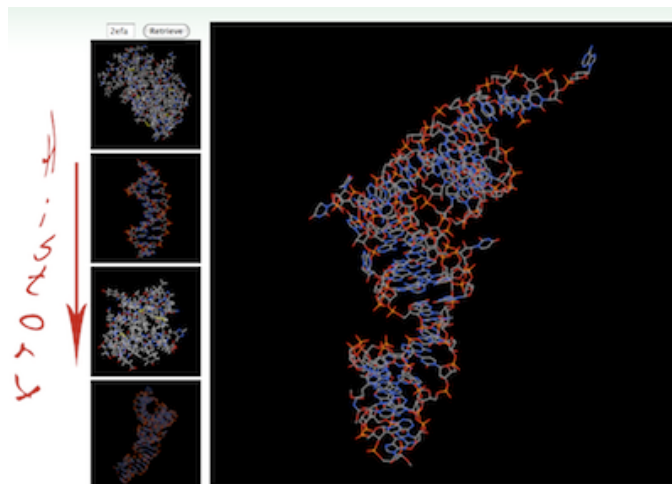
The last function, cycleMolecules(), pushes the currently animating structure into the user's viewing history. The scale attribute is transferred as well as the structure so it doesn't need to be recalculated. The top history component must load the molecule from the rotate3D object to make sure to fit it properly, since the component sizes are different.

```

1.     <script>
2.         // this simple function just pushes the molecules down through history
3.         // note that the molecule, as well as the scale is passed, so no recalculations
4.         // are required mem1 must load the molecule from the rotate3D object to make sure
5.         // to fit it properly
6.         function cycleMolecules() {
7.             if (mem3.molecule != null) {
8.                 mem4.molecule = mem3.molecule;
9.                 mem4.specs.scale = mem3.specs.scale;
10.                mem4.repaint();
11.            }
12.            if (mem2.molecule != null) {
13.                mem3.molecule = mem2.molecule;
14.                mem3.specs.scale = mem2.specs.scale;
15.                mem3.repaint();
16.            }
17.            if (mem1.molecule != null) {
18.                mem2.molecule = mem1.molecule;
19.                mem2.specs.scale = mem1.specs.scale;
20.                mem2.repaint();
21.            }
22.            if (rotate3D.molecule != null) {
23.                mem1.loadMolecule(rotate3D.molecule);
24.            }
25.        }
26.     </script>

```

And that finishes the project. Load it up to see the web app in action!



## Closing Statement

The ChemDoodle Web Components provide scientists with a javascript cheminformatics library and GUI toolkit for quickly producing web applications using the HTML 5 specification's *Canvas* element. With only a few functions, an entire application for querying the PDB and displaying nice animations was built. The future of these technologies is very promising and iChemLabs aims to support this open source project to significantly benefit the scientific community. For any questions, comments or requests, please email the author at [kevin@ichemlabs.com](mailto:kevin@ichemlabs.com) [26].

© 2006 iChemLabs, LLC. All Rights Reserved.

**Source URL (retrieved on Sep 16 2009 - 9:12pm):** <http://www.ichemlabs.com/content/chemdoodle-web-components-0>

### Links:

- [1] <http://web.chemdoodle.com>
- [2] <http://www.rcsb.org/pdb/home/home.do>
- [3] <http://www.ichemlabs.com>
- [4] <http://www.chemdoodle.com>
- [5] <http://www.w3.org/TR/html5/>
- [6] [https://developer.mozilla.org/en/Canvas\\_tutorial](https://developer.mozilla.org/en/Canvas_tutorial)
- [7] <http://www.adobe.com/products/flashplayer/>
- [8] <http://silverlight.net/>
- [9] <http://www.apple.com/safari/>
- [10] <http://www.mozilla.com/en-US/firefox/firefox.html>
- [11] <http://blog.chromium.org/>
- [12] <http://pubchem.ncbi.nlm.nih.gov/>
- [13] <http://blog.wolfire.com/2009/09/preview-of-webkits-webgl-canvas3d/>
- [14] <http://webkit.org/>
- [15] <http://arstechnica.com/open-source/news/2009/09/webkit-adoption-shows-strong-momentum-for-webgl-3d-graphics.ars>
- [16] <http://www.khronos.org/opengles/>
- [17] <http://www.sunsetlakesoftware.com/>
- [18] <http://www.sunsetlakesoftware.com/molecules>
- [19] <http://depth-first.com/articles/2008/07/15/javascript-for-cheminformatics>
- [20] <http://depth-first.com/>
- [21] <http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html>
- [22] <http://web.chemdoodle.com/pdb.php>
- [23] <http://www.google.com/search?hl=en&q=allinurl:Math+java.sun.com&btnI=I'm+Feeling+Lucky>
- [24] <http://www.google.com/search?hl=en&q=allinurl:HTML+java.sun.com&btnI=I'm+Feeling+Lucky>
- [25] <http://www.google.com/search?hl=en&q=allinurl:Canvas+java.sun.com&btnI=I'm+Feeling+Lucky>
- [26] <mailto:kevin@ichemlabs.com>