AUTODESK UNIVERSITY 2014

Connect AutoCAD to the Web with HTML5 and JavaScript

Kean Walmsley – Autodesk

SD5009 AutoCAD 2015 software's JavaScript API (application programming interface) enables a whole new generation of web-enabled applications. This session will focus on how you can use this API to connect web-based applications with AutoCAD software. We'll look at how you can use HTML5 to implement user interface components inside AutoCAD software (modal and modeless dialogs, palettes, and new document windows), including calling into core AutoCAD software capabilities and custom Microsoft .NET/C++ code via the JavaScript API. To illustrate how to address various workflows, we'll take a look at a few sample integrations, including integrating Paper.js to preview 2D geometry; integrating Three.js to preview 3D geometry; integrating Isomer for isometric previews of 3D geometry; and integrating Shapeshifter to preview and import highly complex 3D geometry.

Learning Objectives

At the end of this class, you will be able to:

- Understand why and how AutoCAD software now has a JavaScript API
- Discover the various user interface capabilities that are now HTML5 enabled
- Learn how to communicate from an HTML5 app with AutoCAD software and .NET code
- Learn how to integrate a variety of third-party JavaScript libraries to solve various problems

About the Speaker

Kean Walmsley has been with Autodesk, Inc., since 1995, working for most of that time for Autodesk Developer Network in a variety of roles and in a number of different countries. Kean's current role is software architect for the AutoCAD software family of products, and he continues to write regular posts for his popular development-oriented blog, Through the Interface. Kean currently lives and works in Switzerland.

kean.walmsley@autodesk.com

Introduction to AutoCAD's JavaScript API

[Source: http://through-the-interface.typepad.com/through_the_interface/2013/03/autocad-2014-for-developers.html]

With it being the "lingua franca" of the web (including the mobile web and increasingly the world of cross-platform development), JavaScript is the latest programming language to be enabled to work with AutoCAD. I, for one, welcome our new web-centric overlords!

Just as with the introduction of AutoCAD's .NET API, back in AutoCAD 2005, we weren't "done" in just one release. The first release of the JavaScript API has been implemented with very focused goals, to allow you to:

- Drive "in canvas" user interface activities, such as querying for user input and displaying transient graphics
- Manipulate the current view: zoom, pan, switch Visual Style, etc.
- Implement "out of canvas" user interface dialogs via HTML5 that connect back to AutoCAD via JavaScript

The Design Feed feature inside AutoCAD makes use of this new capability, so we're "drinking our own champagne" (apparently another – more polite, vegetarian compatible? – way of saying "eating your own dog food") with this API from day one.

So how do these goals translate to exposure of individual parts of AutoCAD's core object model? Here are the pieces targeted in the V1 release:

- Prompts
- Transient graphics and input
- Dragger (e.g. Jigs)
- View manipulation
- Commands
- Windows
- Drawing object operations and events
- Bindable Object Layer (BOL)
- Application object (miscellaneous properties)

You'll be able to use these capabilities inside a JavaScript file or HTML page and load it inside AutoCAD using the new WEBLOAD command, Application.LoadJSScript() from .NET or acjsLoadJSScript() from ObjectARX.

That said, the way you're most likely to implement JavaScript functionality in this release of AutoCAD is to implement an HTML5 palette that can be loaded into AutoCAD via a new overload of the PaletteSet.Add() method (one which now takes a URI parameter). You'll need a local module (.NET or ObjectARX) to load the palette by calling this API method, but the UI itself can be hosted elsewhere and therefore be updated more easily.

If dealing with standard modal/modeless windows, you'll use a different protocol:

- In .NET: new overloads for the Application class methods, ShowModalWindow() and ShowModelessWindow()
- In ObjectARX: acedShowHTMLModalWindow(), acedShowHTMLModelessWindow() and acedAddHTMLPalette()
- In LISP: (showHTMLModalWindow) [as it only deals with modal UI]

Now as a V1 implementation, there are certainly holes in the API coverage. If you want to fill these, as a stop-gap, you can implement additional capabilities in .NET (using the JavaScriptCallback attribute to mark a method accordingly) or ObjectARX (using acjsDefun()) and then call them from JavaScript. The functions get called with a JSON fragment containing any arguments being passed. We'll look at this mechanism in more detail in a future post.

Because it's now possible to host such program content on the web – and not in a local folder – we've added the TRUSTEDDOMAINS sysvar (which is analogous to TRUSTEDPATHS for the local system) to enable specific internet domains to be marked as trusted. This URL list is not accessible via the OPTIONS dialog, but you can append to it easily via LISP [e.g. (setvar "TRUSTEDDOMAINS" (strcat (getvar "TRUSTEDDOMAINS") ";through-the-interface.typepad.com"))] or via the Registry.

While the JavaScript API allows us to provide a more convenient protocol for certain operations – such as Acad.Editor.CurrentViewport.zoomWindow(pt1, pt2, true) to perform a Zoom Window with an animated view transition – even in addition to the ability to host content centrally it's not enough to justify the development and introduction of a new API. The plan is to take this API technology beyond the desktop, at some point in the future. Exciting times!

[Source: http://through-the-interface.typepad.com/through_the_interface/2013/04/under-the-hood-autocads-javascript-api.html]

To get JavaScript working inside AutoCAD, we've integrated the open source Chromium component into the product. This gives us the all-important V8 environment for executing JavaScript as well as a WebKit-based browser for rendering HTML content. You may have heard that Google is forking WebKit to continue development on their Blink rendering engine. This shouldn't change anything for us, moving forwards: we may well choose to – at some point – move to a newer version of Chromium, and it'll happen to be based on Blink.

One of the main reasons for going with Chromium rather than WebKit is that its JavaScript engine – V8 – is extensible. This allows us to add our AutoCAD-specific JavaScript APIs into the engine, making them available to code being executed by it.

Chromium is hosted in a separate process to AutoCAD: AcWebBrowser.exe. This process communicates back to AutoCAD via IPC – which should be just fine even for interactive processes such as jigs (where we have to pass a point from AutoCAD to our JavaScript code for each mouse movement). That's not to say there isn't some marshalling overhead associated with this communication – there is – but the benefits of having a separate process executing the code – particular around fault isolation – led to us making this trade-off.

The browser component loads what's known as the JavaScript Shaping Layer when it starts. This is a generic API implementation layer that communicates with the host environment – in our case AutoCAD – via a small number of entry/exit points: exec(), execAsync() and registerCallback(). Any arguments that are needed for these functions – such as the options provided to the getInteger() method – will be encoded in JSON to be communicated via the IPC mechanism.

The beauty of limiting the implementation to make use of these few functions is that it becomes much more easy to translate across to other platforms – the layer itself is portable and having fewer connection points will reduce the amount of native development needed on each platform. At least that's our expectation. That doesn't mean, of course, that no work is needed to actually fill out the implementation behind these entry/exit points – that is work that clearly needs to happen – but we're keeping the number of core connection points down to a small number and designing them to be as streamlined as possible, which should also help minimise the impact felt by the IPC marshalling overhead.

This marshalling happens between a native proxy inside AcWebBrowser.exe process and a native stub inside AutoCAD. With – as mentioned earlier – the marshalling of data happening via JSON.

Something that's interesting for developers – and I know Philippe Leefsma has been looking into this and will no doubt post something soon on the AutoCAD DevBlog – is the ability to use

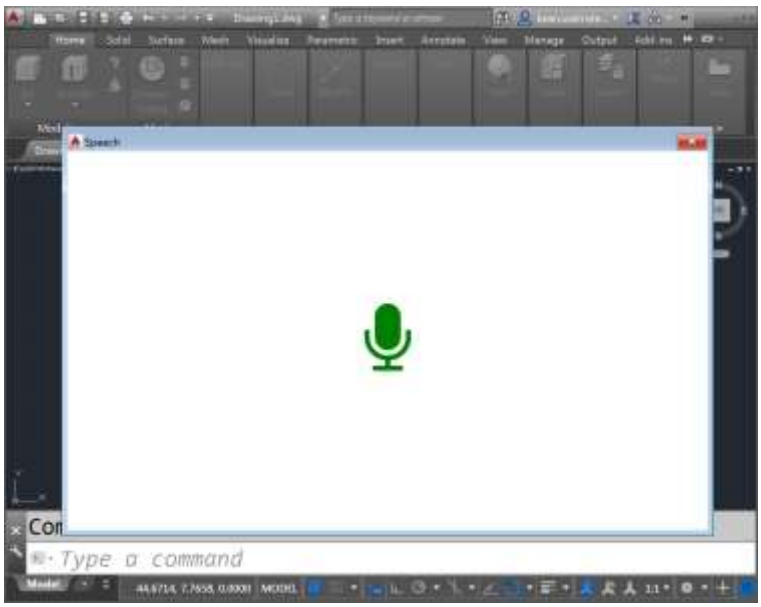
acjsDefun() to register your own stub functions that can be recipients of the calls coming in via IPC. You'll presumably need to essentially extend the Shaping Layer with your complementary JavaScript code that makes use of exec(), execAsync() and/or registerCallback() calls, but you won't need to implement native code to extend the proxy implementation inside the browser process.

One thing that's worth mentioning about the JavaScript Shaping Layer we've created: it's being written in TypeScript, which gives us the ability to work in a more object-oriented fashion than JavaScript inherently provides, before compiling it to JavaScript and publishing the output. This is certainly something that people using the JavaScript API will want to look at doing themselves, depending on the complexity of the application they're working on.

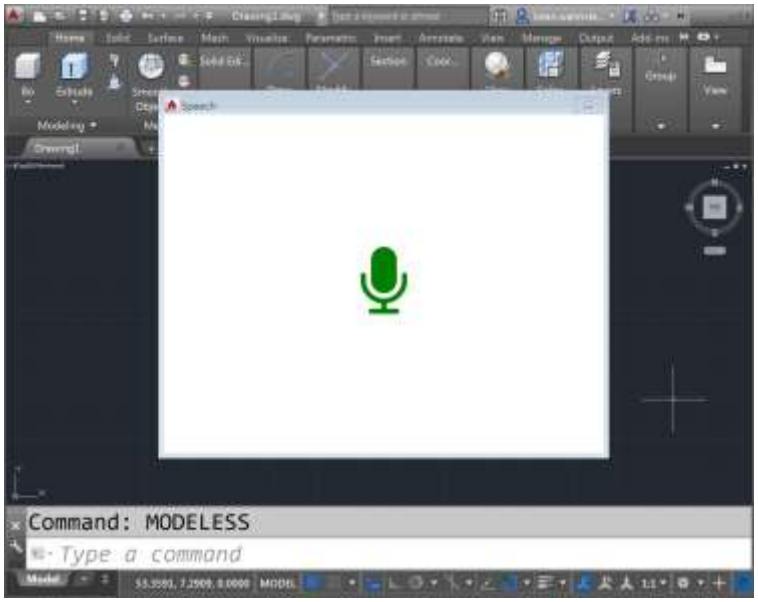
HTML5 User Interfaces in AutoCAD

For this section we're going to focus on using .NET to load a HTML5 page into four different kinds of dialog. The HTML is a simple (although unfortunately currently non-functional) page to toggle some kind of speech recognition in AutoCAD. There's a picture of a microphone centered on the page that toggles between red and green when you click on it.

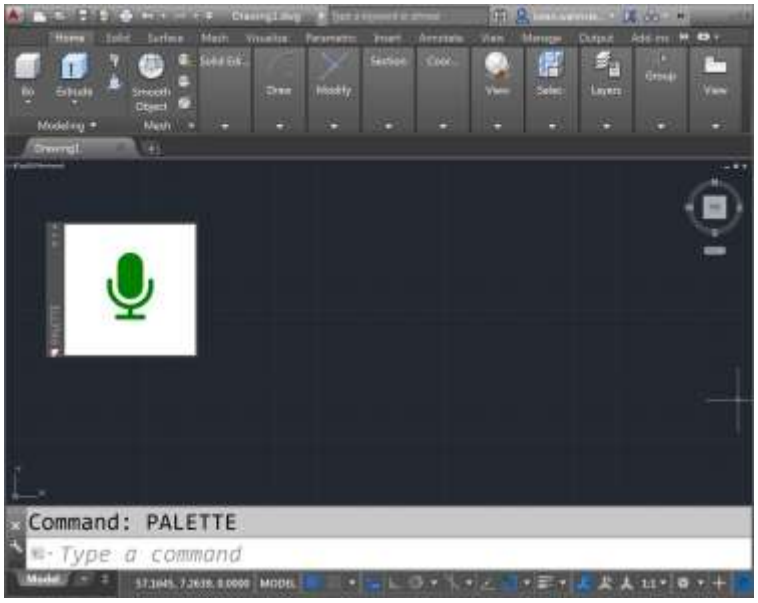
Here's a modal dialog loaded using `Application.ShowModalWindow(uri);`



Here it is loaded using `Application.ShowModelessWindow(uri);`

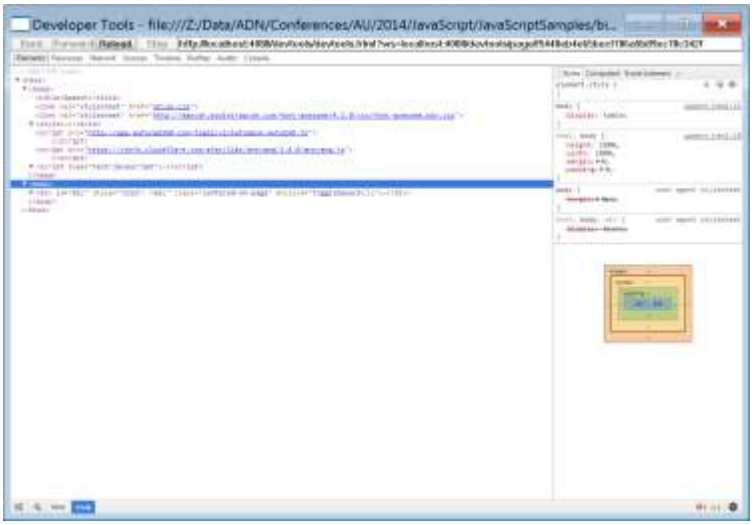


And again as a palette created using `var ps = new PaletteSet(…); ps.Add("…", uri);`



And finally as a document window, added using
`Application.DocumentWindowCollection.AddDocumentWindow("…", uri);`

From each of these dialogs it's possible to launch the Chromium debugging tools by hitting F12. With a little practice, this dialog can be used to diagnose all kinds of tricky issues.



Interfacing with .NET from HTML & JavaScript

The Shaping Layer has clear design goals, and it's natural that some of your needs aren't met by its current state. The good news is that it's straightforward to extend: you can easily add custom .NET code that gets called from JavaScript for hardcore, in-process processing (for instance).

Here's what you need to do from a JavaScript file – this mirrors a lot of the code in AutoCAD.js:

```
function doSomethingInAutoCAD(jsonArgs) {
  var jsonResponse =
    exec(
      JSON.stringify({
        functionName: 'MyFunction',
        invokeAsCommand: false,
        functionParams: jsonArgs
      })
    );
  var jsonObj = JSON.parse(jsonResponse);
  if (jsonObj.retCode !== Acad.ErrorStatus.eJsOk) {
    throw Error(jsonObj.retErrorString);
  }
  return jsonObj.result;
}
```

You can see that it defines a function that takes some arguments – formatted as a single JSON string – which then simply calls exec() (one of the custom extensions we've made to JavaScript when hosted in V8, if you remember) to call our .NET function inside AutoCAD:

```
[JavaScriptCallback("MyFunction")]
public string MyFunction(string jsonArgs)
{
  // Do something with the jsonArgs provided
  // ...

  // Set the results as a JSON string in jsonRes
  // (be sure to set 'retCode' and 'result')
  // ...

  return jsonRes;
}
```

Tagging the function with the JavaScriptCallback attribute – and providing a matching name – allows exec() to find and execute the function. Any data returned as JSON will then be available inside JavaScript, of course.

Interaction scenarios

There are two typical interaction scenarios for HTML/JavaScript UIs inside AutoCAD that might drive communication between JavaScript and .NET.

The first is where some kind of user event – such as clicking on a button in the HTML page – should cause custom .NET code to be executed inside AutoCAD and the results returned to JavaScript. The JavaScript associated to the UI element's onclick() event should call through AutoCAD's Shaping Layer (extended with some custom JS code) requesting for a .NET function to be executed. This will then execute, returning the results as JSON for the calling JavaScript code to parse and then use the results as it sees fit (perhaps to update the HTML UI).

The second is a bit more rare, but still very relevant: this is where some event inside AutoCAD – perhaps the creation of new geometry – needs to drive an update to the HTML UI. In this scenario the JavaScript code needs to register a callback using registerCallback(). At this point the callback becomes available for .NET or ObjectARX code to execute using acjsInvokeAsync(). This could be called from a commandEnded event handler, of instance. The marshaled JSON data can then be parsed in JavaScript and used to update the HTML UI.

Taking it to the next level with JavaScript

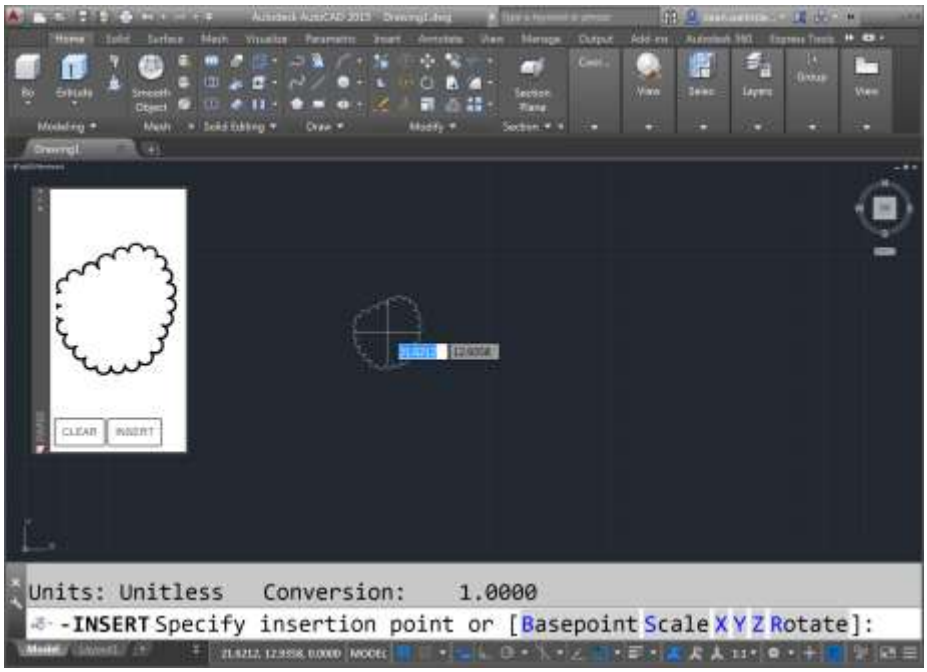
In many ways JavaScript's key strength is around library availability. There's a plethora of open source libraries on GitHub that do amazing things with JavaScript. Some are application frameworks that allow you to easily create professional UIs, others are focused on simplifying the use of WebGL or the HTML5 canvas for display of graphics.

It's this latter category that we're going to focus on. Here are the libraries we'll demonstrate integrated into AutoCAD:

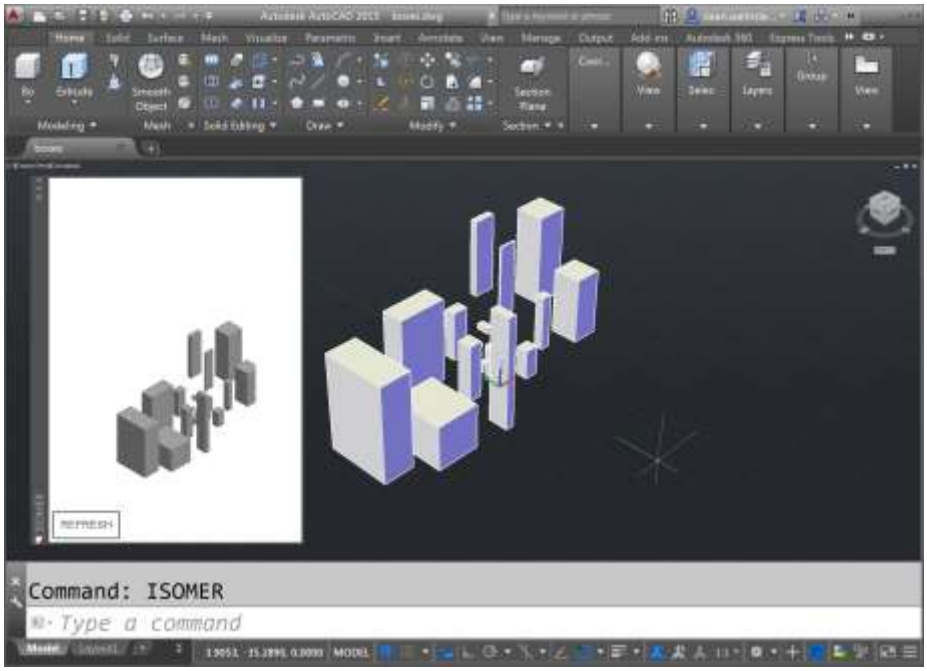
- Paper.js
  - A capable 2D graphics library for HTML5 canvas, based on Scriptographer, as scripting environment for Adobe Illustrator
- Isomer
  - Another HTML5 canvas toolkit for creating isometric geometry
    - You create 3D primitives, they get displayed isometrically in 2D
- D3.js & d3pie.js
  - Data-Driven Documents – simplifies creation of dynamic charts
    - d3pie adds specific support for pie charts
- Three.js
  - Object-based 3D graphics toolkit based on WebGL
    - The basis for Autodesk's A360 viewer and web-service
    - WebGL is finally seeing widespread browser adoption

Here are a few snapshots of the various demos.

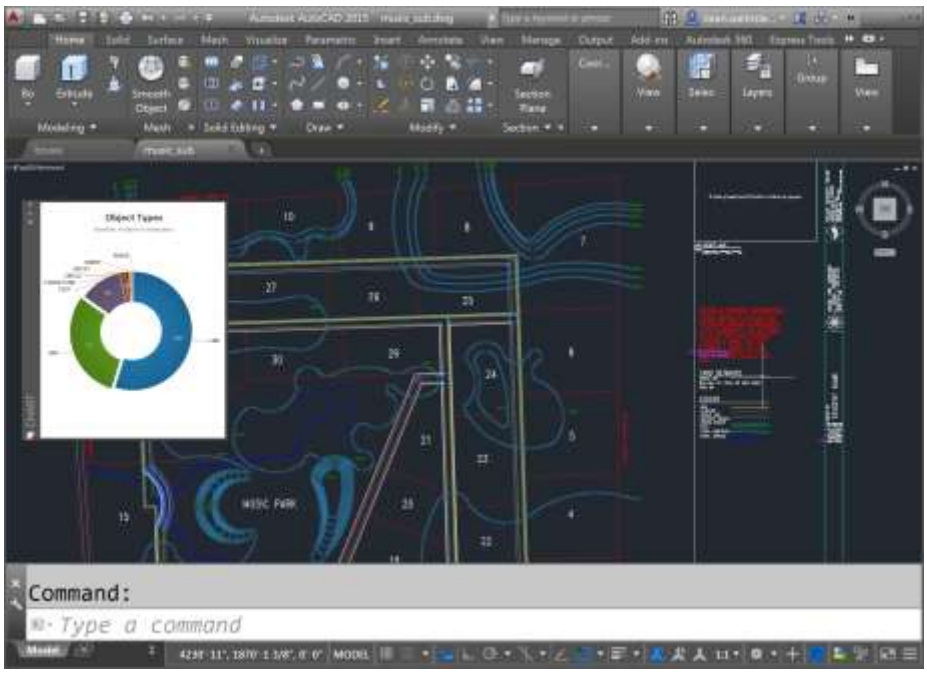
Paper.js revision cloud:



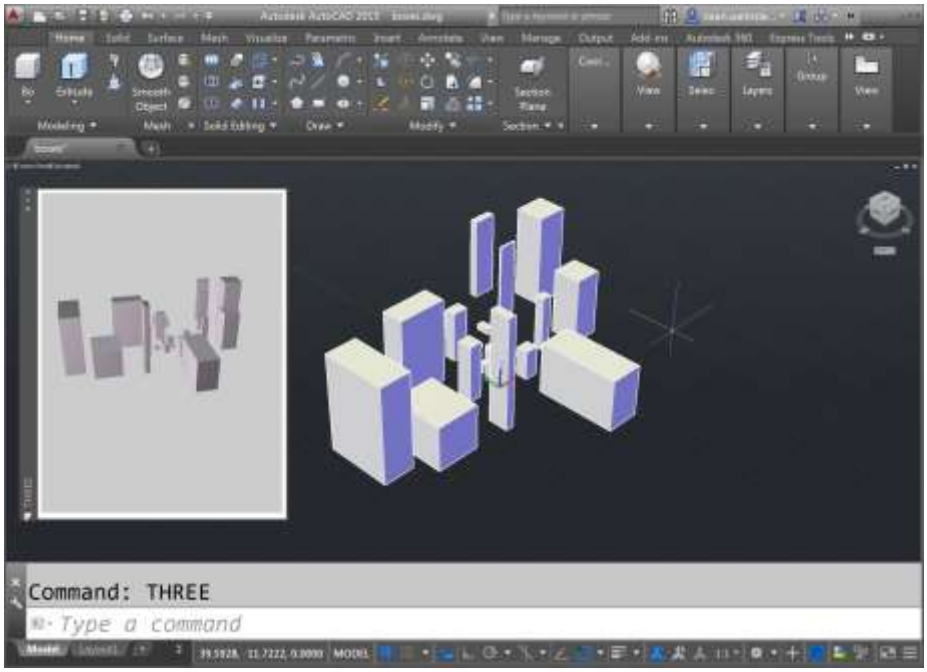
Isomer:



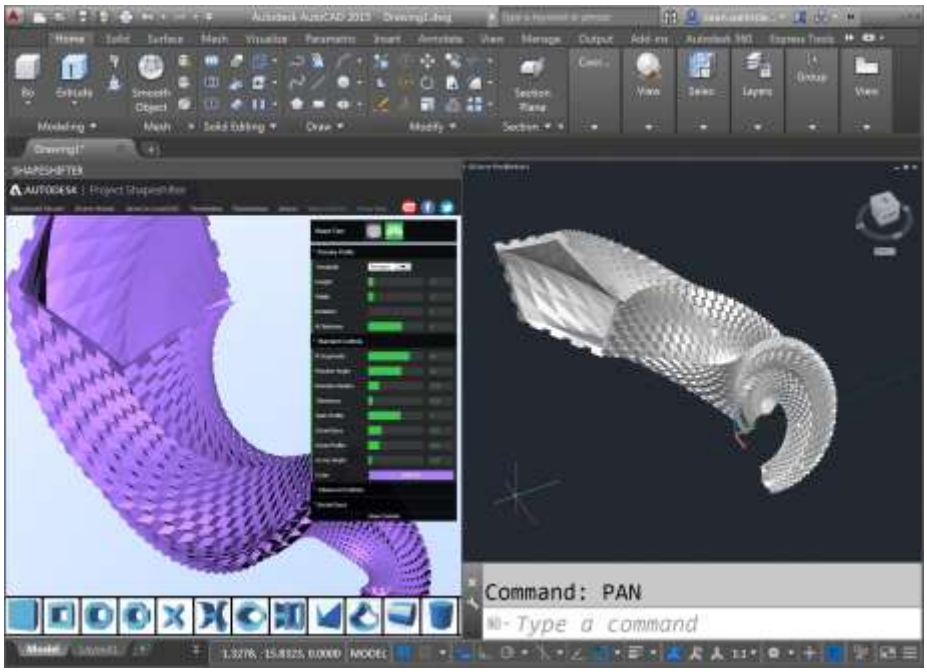
D3.js and d3pie.js:



Three.js:



Finally, here's a prototype showing a custom version of ShapeShifter – which uses AngularJS for UI – which sends data to AutoCAD to create a SubDMesh. This is the only sample that doesn't have the HTML content shared (the client code is still included in the sample project for reference, however).



This is an exciting time to be working with JavaScript and other web-centric technology. AutoCAD's new JavaScript API opens up the product to this world and creates some really interesting opportunities both for technology integration and the creation of (eventually) cross-platform UI assets.

Appendix: AutoCAD & JavaScript-related blog posts on Through the Interface.

AutoCAD 2014 for developers

Zooming to an AutoCAD entity using JavaScript

Implementing an AutoCAD palette using HTML5 and JavaScript

Under the hood: AutoCAD's JavaScript API

Jigging an AutoCAD circle using JavaScript

Jigging an AutoCAD circle from a JavaScript-defined command

Complementing AutoCAD's JavaScript API using .NET

Returning data from .NET to JavaScript inside AutoCAD

JavaScript in AutoCAD: Introducing the series

JavaScript in AutoCAD: Revision clouds using Paper.js

JavaScript in AutoCAD: Isometric views of 3D solids

JavaScript in AutoCAD: Viewing 3D solids using Three.js

Displaying the area of the last AutoCAD entity in an HTML palette using JavaScript and .NET

Connecting Three.js to an AutoCAD model – Part 1

Connecting Three.js to an AutoCAD model – Part 2

Displaying a graph of AutoCAD drawing objects using JavaScript and .NET