

Autodesk View and Data API Webcast

Notes from the Autodesk View and Data API Webcast by Cyrille Fauvel and Stephen Preston, Thursday September 18, 2014.

Cool technology, based on WebGL.

I need to assemble components -- puzzle image.

How do you see the world? Is it in 2D or 3D? It does not matter; our viewer can handle both.

How it is created is irrelevant, any editor is fine, AutoCAD, Inventor, Revit and non-Autodesk.

Let's look at some examples:

Kean Walmsley made this Morgan three-wheeler, being shown at the Geneva car expo.

We were given the full model. It looks like an image, but it is indeed a full 3D model.

None of the UI components are shown, yet we can switch camera views as we wish.

We can also switch to the interior.

We have control through the viewer itself, and also directly through JavaScript.

Anything we show here today can be automated via JavaScript.

Here is another model from a customer, a big Inventor one from a customer.

The viewer has a simple explode functionality built in.

You can control the explode factor and still rotate in real time.

Another thing you can do:

I'll switch back to the home screen and switch on the UI components.

They show the Inventor model structure.

I know that this model contains lights, so I can ask for this subassembly, for anything containing 'light'.

You either search like that or simply click an element to isolate it.

You can easily see its properties, either using the mouse or the JavaScript API.

Another demonstration shows the Waltham office building model.

It is modelled with a lot of detail, right down to furniture level.

You can still rotate very quickly.

This is a Revit model, so the structure is different from the Inventor model.

We can look at a specific level and focus in on a specific chair and see the Revit family information for that instance.

If I am interested in the HVAC installation, I can select that category to isolate it.

Elements within the model have a unique id within the source model, and the viewer model tracks that identify and includes all of the model element properties.

You could use this for facility management or security; send someone out with this instead of a map.

Another example showing a simple API usage example is by Philippe, showing a simple view.

Go through the ADN gallery, load one, rotate around, look at structure and properties, everything imported by the translator into the model.

You drive the viewer programmatically to create an animation, including explosion and rotation.

It works in any WebGL enabled browser.

This entire gallery was created through the public API.

The thumbnails are also provided by the system.

Now you have a feel for what this is about, let us dive into a more detailed description.

Creating WebGL content can be a lot of work. We fully automate this process to make it a no-brainer to bring any model into the viewer and browser.

The API consists of two main components: the server and the client.

The former handles the translation process that creates information stored on the server and streamed to the WebGL component.

There is absolutely no install on your computer or device.

The model is streamed to your browser and you can immediately start viewing and interacting while the full definition and textures comes by and by.

The server API is a REST API using OAuth 2 authorisation.

The web client API is based on three.js, and you can use three.js functionality in addition to and in parallel with the viewer API.

Here is a list of the file formats supported.

In general, we can say that we really want to support every single industry standard format.

You only have to translate the model once.

The translation is read-only, so the resulting models cannot be modified.

To get started, go to the main page and register.
That equips you with an OAuth key pair to sign the API requests to the server.
The rest of the process is a simple five-step process.
Let's look at the OAuth process, which starts by requesting an access token.
It is valid for 15 minutes, which is enough for the following steps.
Next, create a bucket; think of it as a small database, a container for your models.
Organise your models in any way you want using these.
Next, upload your model. If it consists of several files, upload them all into the same bucket.
Request translation to convert your heavyweight CAD model into the lightweight stream.
We have three types of buckets, depending on how long you want to retain the result.
One important detail is the resulting model id, which is human readable.
The following API calls make use of the model URN, which is the base 64 encoded id.
<<<<Update file format list.>>>>
<<<<2D support is not as far along as 3D yet. Currently, we support 2D DWF.>>>>
<<<<Offline viewing? Ultimately, that is on the plan, important for mobile devices, not here yet, currently connected browser is required. Once the model is in the browser, you can disconnect and walk around, of course. We stream the information to the viewer, nothing is stored on the hard drive, everything is JSON. In theory, you can retrieve these JSON objects and store them to your hard drive.>>>>
<<<<The secret source of this service is handling all the translation and including the rich model data, which was previously inaccessible to the general public. Yes, models must be uploaded to the server. They are not stored there, only the resulting models with the embedded metadata. We will not hand out the translators to third party and cannot be shipped to third parties. If you cannot upload your models to our servers you cannot use this service.>>>>
The translation process can be stepped through conveniently using the curl script sample.
Step through and look at the human readable id.
Next is 'register', to request the translation.
Instead, switch bucket and request local html file for the example and post it to my local web server and browser: voila.
That wraps up the server REST API.
Now, let's talk about the client API.

We need a WebGL enabled browser, e.g. Chrome, Opera, Firefox, Safari, IE 11.
We can embed the viewer into an HTML page in an embed tag, like in Kean's blog.
Within the Typepad blog post, I can use the full interactive viewer 3D controls.
Another method is to create your own HTML 5 page and reference the viewer API JavaScript and CSS style sheet.
Next, create a div. The viewer uses that to create a canvas element, so don't try to do that yourself.
Initialise the viewer with your model URN and the current OAuth access token.
You can extend the basic viewer functionality using your own JavaScript code to access the model hierarchy, metadata, properties, handle events, camera, zoom, navigation, geometry, textures etc.
You read more about the Autodesk JavaScript API in the documentation plus all the additional three.js functionality.
Look at the resources for getting started, numerous public samples published on GitHub.
You need to substitute your own key, access token and URN.
You can test the API online on the console without coding at all, which may be cumbersome and error prone, though.
We are working on a very easy debug sample to simplify this analysis without coding.
We have a cloud and mobile DevBlog with on-going discussions on this, and lots of more demos:
Timeliner: a simple use case showing a Revit model and connecting it with a schedule, press play, and see the different components appearing while the timeline scrolls by, using the API to simply hide and display components.
SAP integration: we have this chair, move, explode, and interesting: click on a component, showing the information from the design file. It also has an SAP product id, and the rest of the information is retrieved real-time through a live link to an SAP database. Change the SAP data in real time and see the price update immediate. The unique id enables linking the model elements to any other external data and update that with no further translation required.
More information on WebGL:
Khronos group at www.khronos.com.
Three.js:
Fine control the viewer and add your own geometry on top of the model.
Tag geometry, add commands, and use the three.js library.
It significantly simplifies the WebGL code.

Let's wrap up with a JavaScript code example extracted source code from the ADN gallery.

This is the source code extracted by Chrome from the live web page.

App.js is based on Angular.js.

These two components are loaded.

The two URLs... the Autodesk server and the personal server.

The latter is invoked from the getToken implementation.

When we request the token, we ask our own web site, not the Autodesk developer one.

Do not put your key pair into your JavaScript, or they will be visible to others.

For security reasons, the server does not accept a request from JavaScript to create a token.

You have to do it from your server.

Q&A: please type questions.

[Q] Redlining?

[A] It is on the development schedule, not there yet, though. The development schedule is rash; updates come daily or weekly rather than monthly or annual.

[Q] Current status?

[A] Right now this is a beta. Play with it, ask for what you need. We can respond quickly to your requests.

[Q] Pricing?

[A] Currently zero. Eventually, it is not yet decided. We do not want to make a mistake or introduce any undesirable pricing. An option might be to charge per GB of translation and provide free viewing, with a cut-off level below which everything is completely free.

[Q] Languages?

[A] We currently show Python, PHP, ASP, ... anything you need can be provided.

[Q] Recording?

[A] A link will be forwarded to all participants.

[Q] Modify texture?

[A] Yes, absolutely, using the three.js library, probably.

[Q] Access underlying three.js data?

[A] Yes, totally, including addition of data and geometry, query underlying three.js parameters and viewer information.

[Q] Add and remove model elements?

[A] You can delete geometry at runtime. Changes will not be saved to the model. You would have to keep track of the changes on your own if you want to modify the original model, e.g. delete a window in Revit. Using the element GUID makes this easy.

[Q] Combine two models in the same view?

[A] You can do that yourself now; we will probably be supporting this more in future. Development is fast and our list is long and exciting.

[Q] 2D support?

[A] Currently, the 3D support is further along. Enhancing 2D is high on the list of priorities.

[Q] Add custom parameters?

[A] You would not add things to the model; you would link to an additional parallel database, like in the SAP sample.

[Q] Privacy?

[A] OAuth makes this secure and nobody can access your models.

[Q] Does it have to be the Autodesk server?

[A] Right now, it has to be the Autodesk server. In theory, other servers could be added. Explain your need to us, please. We do not store the models after translation. If you want to store the resulting stream, just intercept the JSON and do so.

[Q] Permanence?

[A] You can request a permanent bucket, and then delete it after the time span required.

Call to action:

Try it out!

Apply for a key, read the documentation, everything is public, samples on GitHub, download everything f interest, get it running try it out, live samples galore exist.