FDC322554

Extend the Viewer

Petr Broz
Autodesk Inc.

Learning Objectives

- Discover the basics of Forge;
- Learn how to interact with the Viewer;
- Learn how to manipulate elements;
- Learn tips and tricks to extend the Viewer.

Description

Endless possibilities, that's what we can do with the Forge Viewer. Want to learn the basics? This class will explore how to create custom extensions to interact with the Viewer. To attend this class, make sure to have a running Forge application with the Viewer, or you should have attended either "Upload and View Your Models with Forge" or "Connect to Fusion Team and BIM 360 Data." Bring your own laptop or use the lab equipment. This class requires basic knowledge of JavaScript.

Your Forge DevCon Experts

Petr Broz
Petr is a senior developer advocate at Autodesk. He has been developing some of the company's web applications and services since 2011, and now he aims to help others build amazing, creative solutions using Autodesk Forge APIs. Petr's technical focus areas include everything around web and cloud, 3D graphics, and all kinds of "non-real reality".

Denis Grigor
Within Autodesk, Denis Grigor is providing programming support to 3ds Max and Forge external developers. This mix of desktop and cloud exposure, helps him in his endeavor to safely erase the borders between them.

Forge Viewer

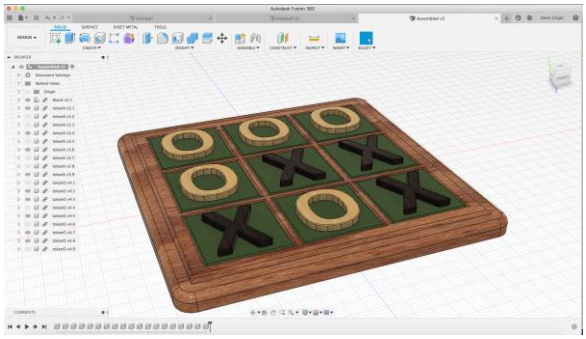
Traditionally Autodesk is known for desktop type applications, yet along the way, more and more cloud powered features are integrated into desktop applications, thus somehow watering the line between desktop and cloud application.

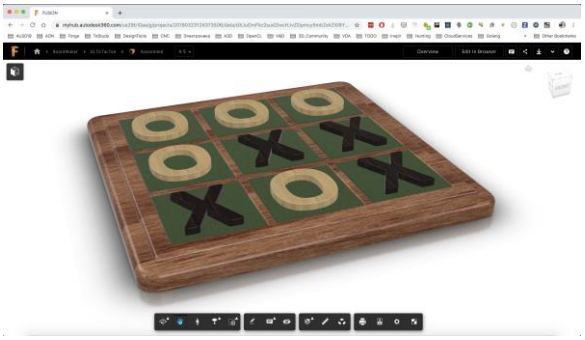
These cloud powered features are based on technologies that previously were available only for internal use, while the main idea behind Forge platform (https://forge.autodesk.com/) was to make it available outside, to third party developers.

In this class we will concentrate our attention on Forge Viewer, which for several years was available in A360, Fusion360 and lately in Revit, 3ds Max etc. as Shared Views.

The main idea of the Forge Viewer is the visualization of models in a web browser, while keeping all the information related to the model (like component hierarchy, materials, properties and other metadata). Previously, when you wanted to share your model with somebody, you had to expect that the counterpart can open your model, which usually meant that it needed an installed software capable opening the certain format.

Through use of the Forge Viewer, this is not needed anymore, as the model is visualized in a web browser and a link can be shared and made available to anybody who has a browser (literally meaning to anybody).





In Autodesk products, the Forge Viewer is tailored for above mentioned scenario, while in hands of a developer, since it is just a JavaScript library built on top of three.js, the imagination is the only limit and in what follows we will illustrate how easy the Forge Viewer can be customized through use of extensions.

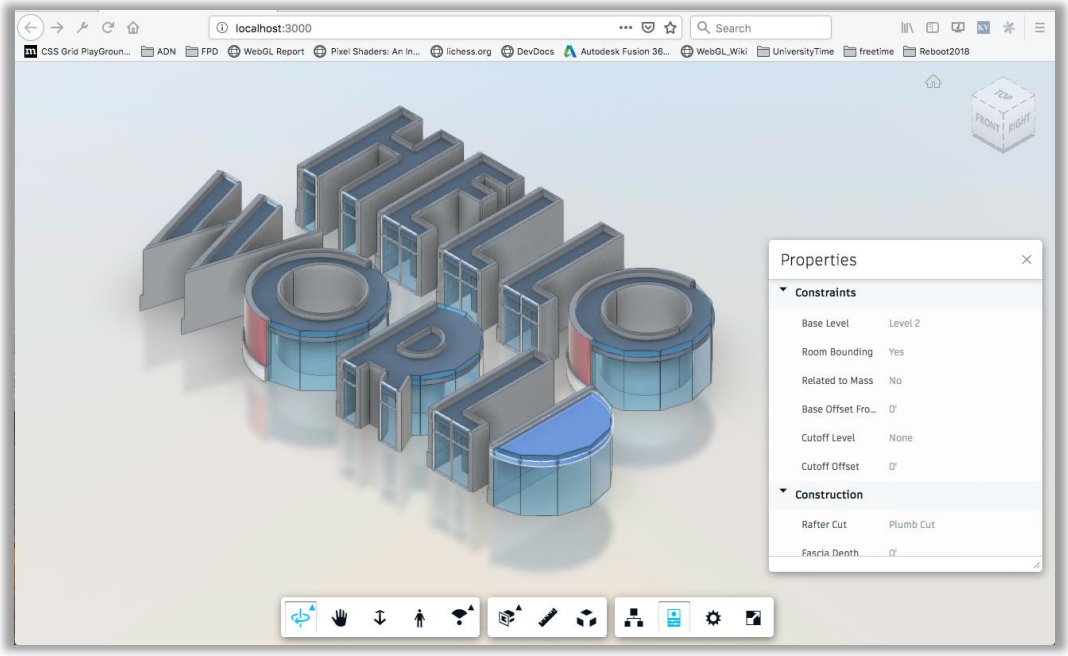
Basic Extension

The first step is of course bringing the model into browser using Forge Viewer.

As already mentioned, Forge Viewer is a simple JavaScript library, thus it can be easily embedded into an HTML page using the following minimal code (providing the needed URN and an access token):

```
<!-- Autodesk Forge Viewer files -->
<link rel="stylesheet"
      href="https://developer.api.autodesk.com/modelderivative/v2/viewers/7.*/style.min.css"
      type="text/css">
<script src="https://developer.api.autodesk.com/modelderivative/v2/viewers/7.*/viewer3D.min.js"></script>


<!-- Developer JS -->
<script>
    let viewer;
    const options = {
        env: 'AutodeskProduction',
        getAccessToken: "<YOUR_APPLICATION_TOKEN>",
    };
    Autodesk.Viewing.Initializer(options, () => {
        viewer = new Autodesk.Viewing.GuiViewer3D(document.getElementById('forgeViewer'));
        viewer.start();
        const documentId = 'urn:<YOUR_URN_ID>';
        Autodesk.Viewing.Document.load(documentId, onDocumentLoadSuccess);
    });

    function onDocumentLoadSuccess(doc) {
        const viewables = doc.getRoot().getDefaultGeometry();
        viewer.loadDocumentNode(doc, viewables).then(i => {
        });
    }
```

This code will "automagically" present your model in a way similar to the following screenshot:



As you can see, the original hierarchical structure was preserved, and the metadata associated with each component was preserved as well. The main benefit compared with other in-browser viewing technologies is this exact preservation of useful data. In case of a house model, when you click on a window, you are not presented with information about vertices, triangles etc., but information like height, width, type of materials and many other relevant properties. This is exactly why we call it “an industrial grade viewer”.

At this step, we have our model visualized in a web browser and, as you can see, it comes with already integrated navigation controls, camera control (perspective/orthographic), component tree, measuring tools, property tools and many other goodies by default.

Forge Viewer uses the concept of “extensions”, providing a mechanism to write custom code that interacts with the viewer, and the official documentation even provides a step-by-step tutorial on writing such an extension. In fact, all tools you see on the bottom toolbar are extensions that can be easily used as reference implementations for your own tools.

Writing an extension is not a rocket science, the minimum amount of code being:

```
class TemplateExtension extends Autodesk.Viewing.Extension {
    constructor(viewer, options) {
        super(viewer, options);
        this.viewer = viewer;
    }

    load() {
        console.log('TemplateExtension is loaded!');
        return true;
    }
    unload() {
        console.log('TemplateExtension is now unloaded!');
        return true;
    }
}

Autodesk.Viewing.theExtensionManager.registerExtension('TemplateExtension',
    TemplateExtension);
```

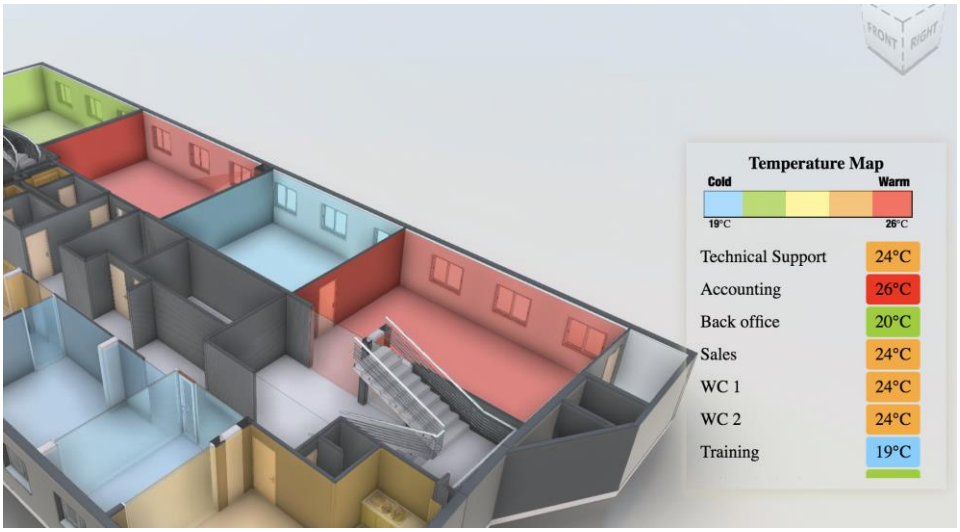
This gives you the simplest extension, that can be loaded as any other extension through call to:

```
<script src="./templateExt.js"></script>

...

<script>

    // ...

    viewer.loadExtension('TemplateExtension');


    // ...
</script>
```

Check https://learnforge.autodesk.io/#/viewer/extensions/skeleton for a more information on the Extension Skeleton.

Extension UI

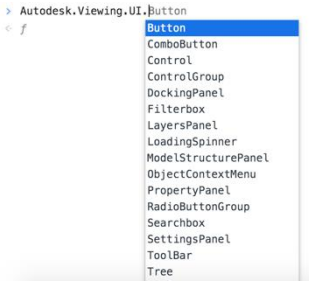
It is not mandatory for a Viewer Extension to have a UI, or you can rely on your own UI:



But sooner or later you will encounter problems with:

- ***Positioning***: you have to take care of dragging and component auto layout;
- ***Styles***: you will have to take care of styling so that UI looks seamless;
- ***Intergration***: you will have to make sure that your custom UI components are not overlapping and well interact/snap with Viewer UI components.

From this perspective, for building UI for a Viewer Extension, it is advised to rely on Viewer API which provides a lot of components which play nicely with each other:



For more information on adding UI to extension, check
https://learnforge.autodesk.io/#/viewer/extensions/panel

Accessing Scene and Metadata

Once everything is setup, there are a couple of objects that you should be aware of if you want to retrieve some data or do some Viewer manipulation/customization:

1. ***instanceTree*** – imagine it as the guy who knows all about the nodes in your scene. You have an ***id*** of a ***node*** and want to get info about it, like *name*, *children*, *fragments* – this is your guy and you can get it in your extension by having:

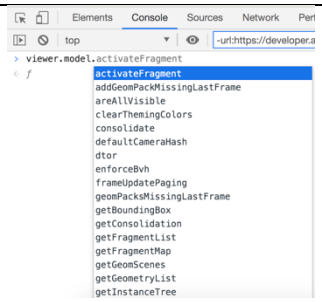
```
let tree = viewer.model.getInstanceTree();
```

One thing to keep in mind is that your extension might be loaded faster that the model is loaded (and the tree created), thus you'll get a weird error about a missing function and so on.
This is why, using events, in this case Autodesk.Viewing.OBJECT_TREE_CREATED_EVENT is a must for this very case and all you need is:

2. ***model*** – this is obviously an object which knows everything about your model.
   Want to get the properties of a component or material it is using? He can help with that.
   All you need, is call him by having:

```
let model = viewer.model;
```

Tip: To learn faster some viewer components, you can use the Console from Developer Tools, usually available in any browser, to investigate the needed objects.
For example, what methods are available for the model:




```
load() {
    console.log('MyExtension is loaded!');
    this.viewer.addEventListener(Autodesk.Viewing.OBJECT_TREE_CREATED_EVENT,
        this.customize);

    return true;
}

customize() {
    this.viewer.removeEventListener(Autodesk.Viewing.OBJECT_TREE_CREATED_EVENT,
        this.customize);

    //Start coding here ...
}
```