FDC126529

Deep Dive with the Model Derivative API

Sharmila Phadnis
Autodesk

Description

Model Derivative API is not just meant for translating files to a viewing file format (the 'SVF' file format) and loading the model for viewing. In this class, we will provide you with deeper understanding of the Model Derivative API, it's feature set and capabilities. We will cover the range of file formats that are available for translation; how to extract metadata & geometry from a model. This will be followed by demonstrating a use case developed by one of our Forge customers. The class will show C# and Node.js code samples.

Your Forge DevCon Expert(s)

Sharmila Phadnis has been working in Autodesk for the last 4 years. She started as a Software Engineer in the Office of the CTO working on research projects such as Dreamcatcher. She led the launch of Model Derivative API. She is currently responsible for managing the data pipeline, leading the efforts to build the data analytics tool and leveraging data to understand the usage patterns of the cloud platform services while taking value added actions. She also interfaces with AEC product line as a business partner stakeholder. Her areas of interest include API Design, Data and Analytics, Cloud Computing, User Research. Prior to joining Autodesk, Sharmila was working as a Software developer on SaaS products.

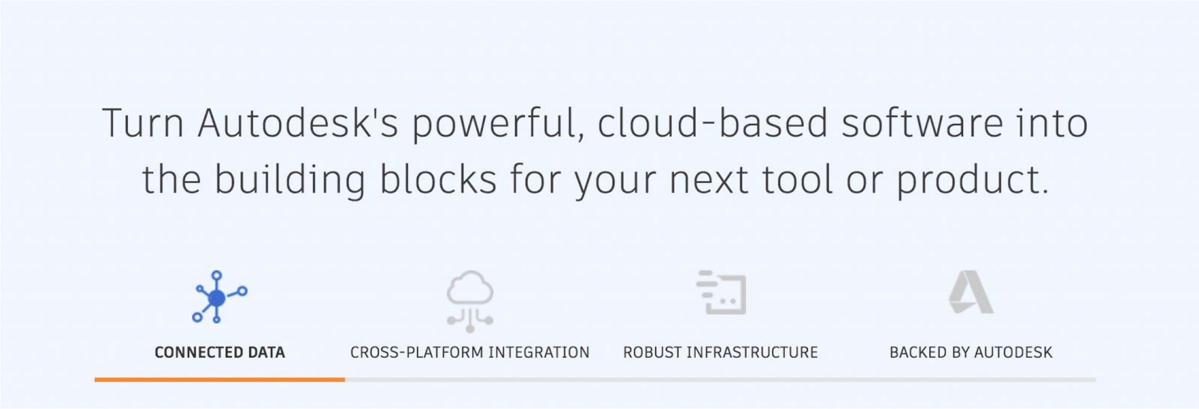
Email: sharmila.phadnis@autodesk.com
Twitter: @sharmilaphadnis

Introduction

This class will start by introducing what is Forge and why do we need it. We will skim through different Forge APIs such as Data Management API, Model Derivative API, BIM360 API. We will deep dive into Model Derivative API to understand the supported file formats and the metadata and data extraction capabilities. Followed by customer application, we will take a look at code samples that use the C# , Node.js Forge SDK to demonstrate the feature set of Model Derivative API by creating a Forge application.

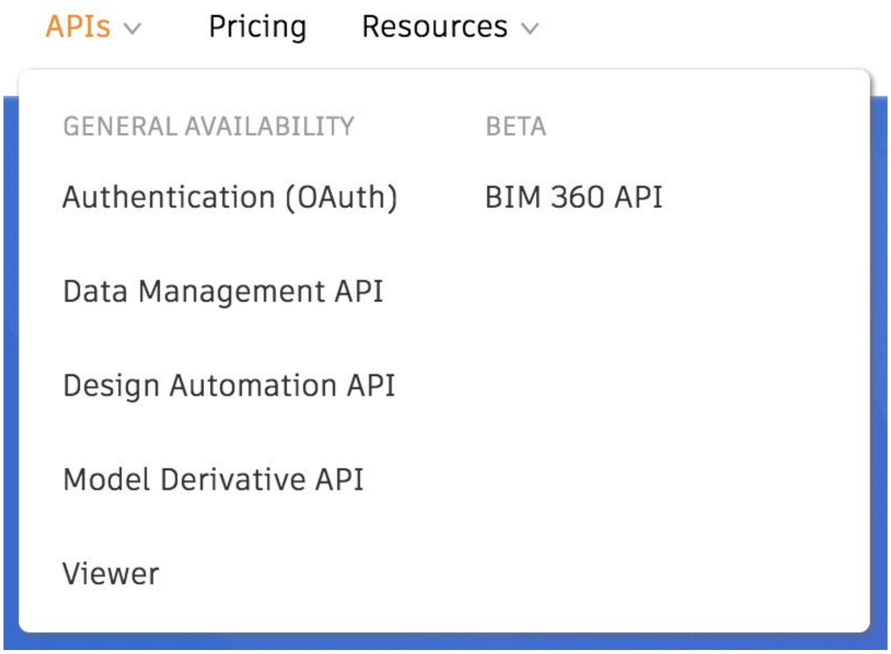
What is Forge AUTODESK FORGE and Why do we care

In the era of desktop, Autodesk IP was locked in it's software components and libraries. With the advent of cloud, the IP has been decoupled into a set of APIs that can be used to build powerful custom solutions outside the box.



Turn Autodesk's powerful, cloud-based software into the building blocks for your next tool or product.

Available Forge APIs



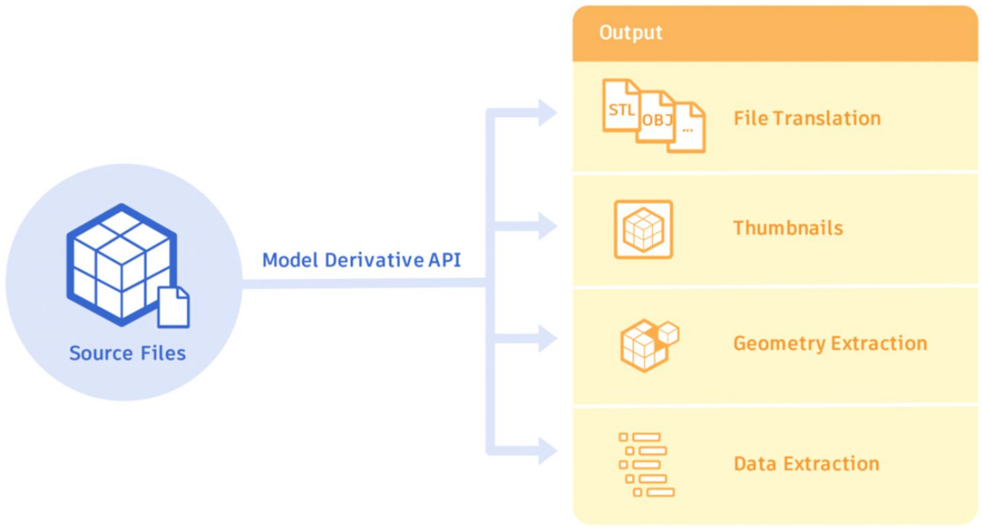
API Details

- **Viewer:** Viewer is used to display 2D and 3D design files from over 60 file formats for Web and mobile presentation. The Viewer is a customization WebGL based tool for loading models in the browser with the ability to comment, mark up and measure within the viewer.
- **Design Automation API:** With this tool, AutoCAD scripts can be executed in the cloud for workflows such as batch conversion of thousands of DWG files to PDF, can be run in the cloud to free up space on one's computer.
- **Model Derivative API:** Used for translating design files into different file formats, for extracting design data for use in other apps and retrieving isolated geometry.
- **Data Management API:** Used for accessing and managing data from multiple products such as A360, Fusion 360, BIM 360 Docs, BIM 360 Team and Forge native storage service with a single API call.
- **Authentication API :** Used to generate oauth tokens for authenticating and authorizing users to access and work with apps and interact with APIs

Model Derivative API

When a design file is viewed in A360 viewer, under the hood uploaded CAD file is first translated into a viewing format (aka ‘svf’ file format) through the Model Derivative API. Once the file is translated, corresponding derivative is viewed in the context of the viewer.

Model Derivative API is used for translating files but,it is also used to generate thumbnails from the model, extract all or specific parts of geometry and extract data from the model.



Supported Translations:

Model Derivative API supports translation of more than 60 file formats and includes most of the industry standard file formats from media & entertainment, architecture and engineering, manufacturing. A complete view of all supported translation is noted below. The json reponse list the supported output file format followed by a list of input file format available for translation. For example, given a dwg file format, Model Derivative API supports translation from f2d,f3d,rvt formats.

{"formats":{
"dwg":
["f2d","f3d","rvt"],
"fbx":["f3d"],"ifc":["rvt"],
"iges":

["f3d","fbx","iam","ipt","wire"],

"obj":["asm","f3d","fbx","iam","ipt","neu","prt","sldasm","sldprt","step","stp","stpz","wire","x_b","x_t","asm\\.\\d+$","neu\\.\\d+$","prt\\.\\d+$"],

"step":

["f3d","fbx","iam","ipt","wire"]

,"stl":

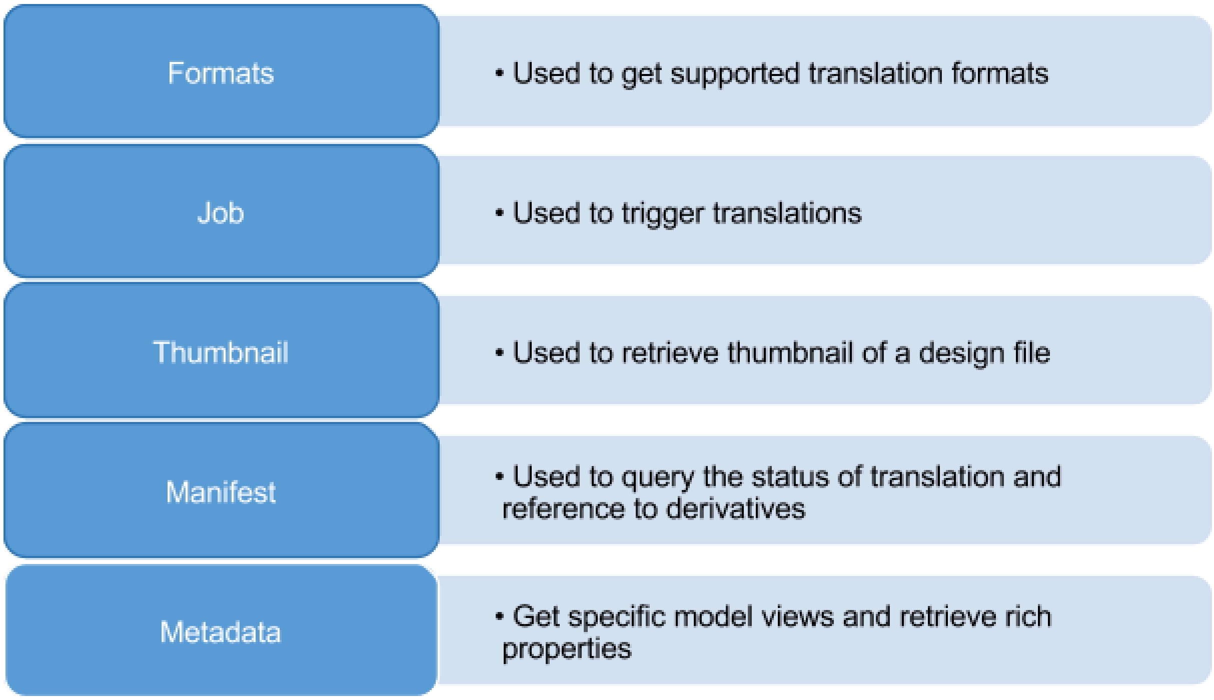
["f3d","fbx","iam","ipt","wire"],

"svf":["3dm","3ds","asm","catpart","catproduct","cgr","collaboration","dae","dgn","dlv3","dwf","dwfx","dwg","dwt","dxf","emodel","exp","f3d","fbx","g","gbxml","glb","gltf","iam","idw","ifc","ige","iges","igs","ipt","iwm","jt","max","model","neu","nwc","nwd","obj","pdf","prt","psmodel","rcp","rvt","sab","sat","session","skp","sldasm","sldprt","smb","smt","ste","step","stl","stla","stlb","stp","stpz","wire","x_b","x_t","xas","xpr","zip","asm\\.\\d+$","neu\\.\\d+$","prt\\.\\d+$"],

"thumbnail":["3dm","3ds","asm","catpart","catproduct","cgr","collaboration","dae","dgn","dlv3","dwf","dwfx","dwg","dwgx","dwt","dxf","emodel","exp","f3d","fbx","g","gbxml","glb","gltf","iam","idw","ifc","ige","iges","igs","ipt","iwm","jt","max","model","neu","nwc","nwd","obj","pdf","prt","psmodel","rcp","rva","rvt","sab","sat","session","skp","sldasm","sldprt","smb","smt","ste","step","stl","stla","stlb","stp","stpz","wire","x_b","x_t","xas","xpr","zip","asm\\.\\d+$","neu\\.\\d+$","prt\\.\\d+$"]}}

Features of Model Derivative API

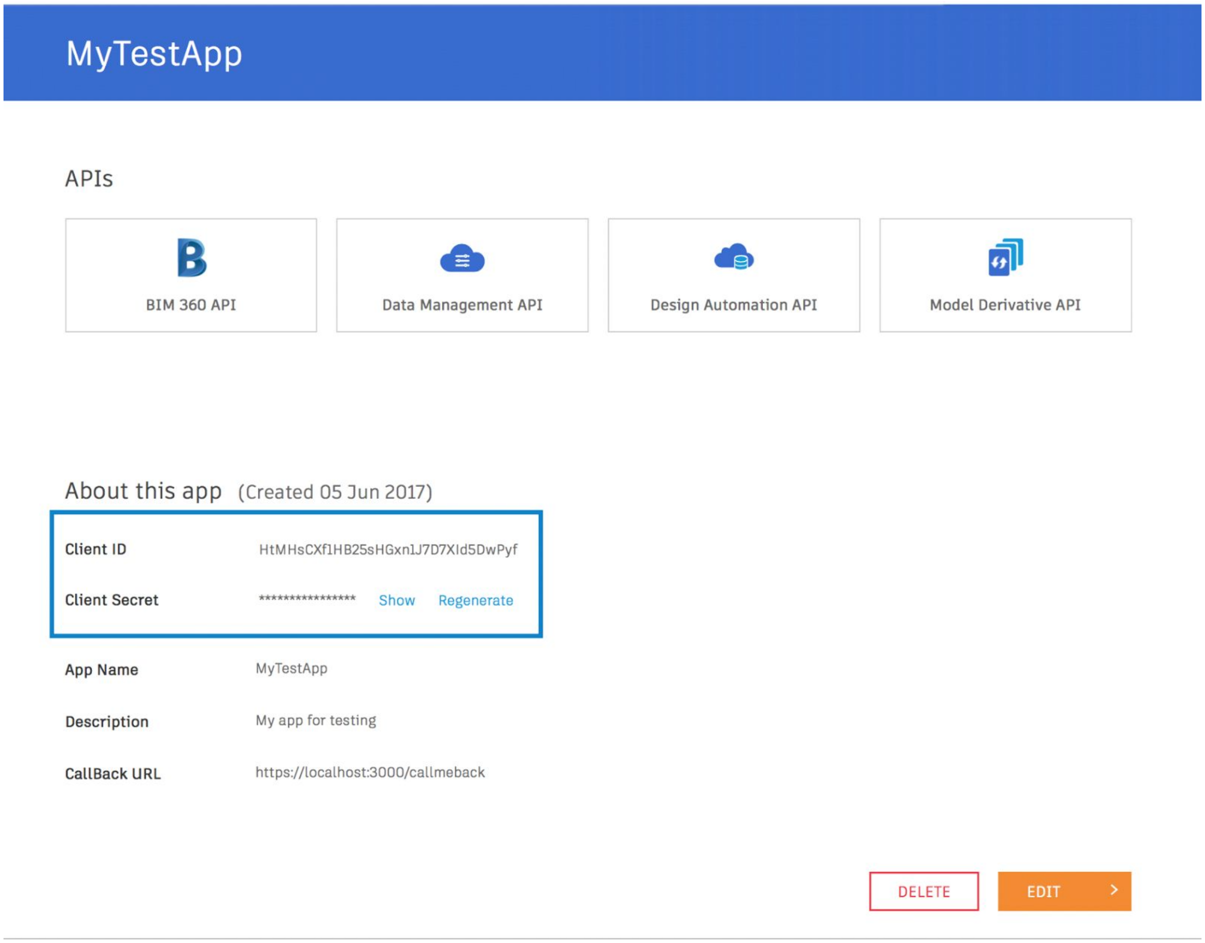
Features of Model Derivative API are listed below, exposed via endpoints for consumption through REST or by leveraging the SDKs.



Formats	• Used to get supported translation formats
Job	• Used to trigger translations
Thumbnail	• Used to retrieve thumbnail of a design file
Manifest	• Used to query the status of translation and reference to derivatives
Metadata	• Get specific model views and retrieve rich properties

Getting Started

Create an App on developer.autodesk.com



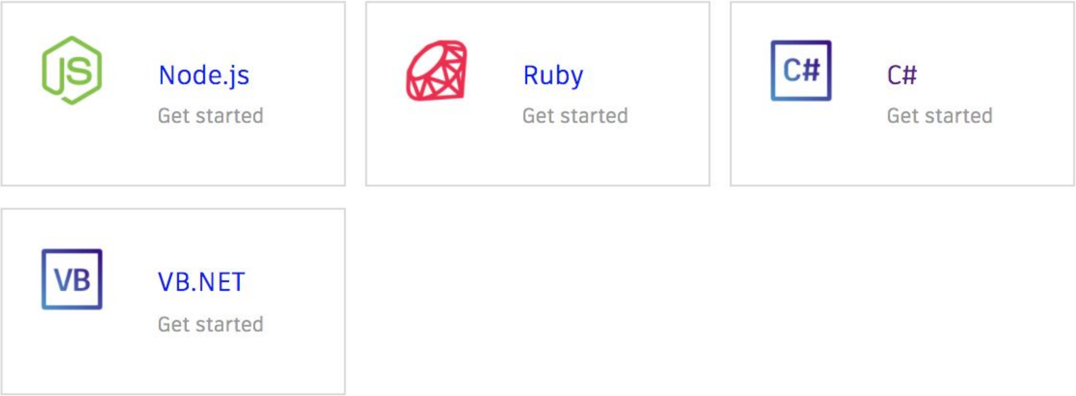
MyTestApp

APIs

About this app (Created 05 Jun 2017)

Client ID	HtMHsCXf1HB25sHGxn1J7D7XId5DwPyf
Client Secret	****************
App Name	MyTestApp
Description	My app for testing
CallBack URL	https://localhost:3000/callmeback

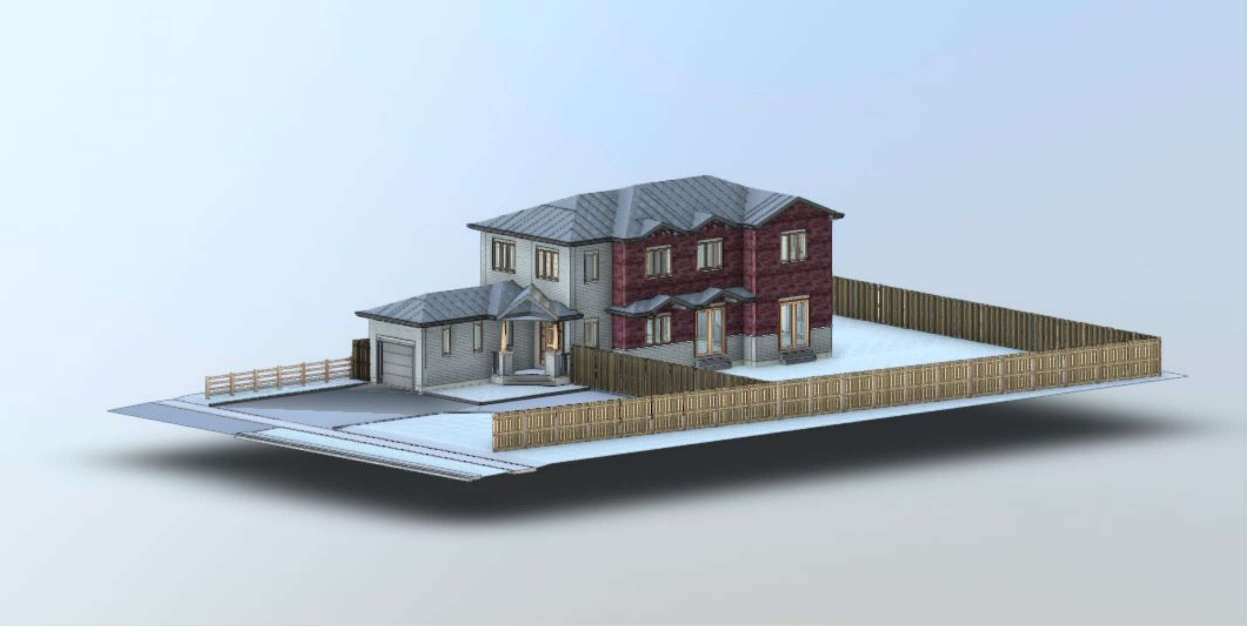
Supported SDK



Walkthrough with Code Sample

This section will walkthrough a code sample in visual studio IDE that covers each step in the process including uploading a model and using Model Derivative API features.

Step 0 - Select a Model



Step 1 – Generate a token

```
Scope[] scopes = new Scope[] { Scope.DataRead, Scope.DataWrite, Scope.BucketCreate,
Scope.BucketRead };
            oauth2TwoLegged = new TwoLeggedApi();
            twoLeggedCredentials = oauth2TwoLegged.Authenticate(FORGE_CLIENT_ID /*Replace
with your Client ID*/, FORGE_CLIENT_SECRET/*Replace with your Client Secret*/,
oAuthConstants.CLIENT_CREDENTIALS, scopes);
            bucketsApi.Configuration.AccessToken = twoLeggedCredentials.access_token;
            objectsApi.Configuration.AccessToken = twoLeggedCredentials.access_token;

            derivativesApi.Configuration.AccessToken = twoLeggedCredentials.access_token;
```

Step 2 – Create a Bucket

```
Console.WriteLine("***** Sending createBucket request");
            PostBucketsPayload payload = new PostBucketsPayload(BUCKET_KEY, null,
PostBucketsPayload.PolicyKeyEnum.Persistent);
            dynamic response = bucketsApi.CreateBucket(payload, "US");

            Console.WriteLine("***** Response for createBucket: " + response.ToString());
```

Step 3 – Upload a Model

```
Console.WriteLine("***** Sending uploadFile request");
            string path = FILE_PATH;
            if (!File.Exists(path))
                path = @"..\..\" + FILE_PATH;
            using (StreamReader streamReader = new StreamReader(path))
            {
                dynamic response = objectsApi.UploadObject(BUCKET_KEY,
                    FILE_NAME, (int)streamReader.BaseStream.Length,
streamReader.BaseStream,
                    "application/octet-stream");
                Console.WriteLine("***** Response for uploadFile: ");
                Console.WriteLine("Uploaded object Details - Location: " +
response.location
                    + ", Size: " + response.size);
                return (response);

            }
```

Step 4 – Translate the file to SVF

```
       try {
            Console.WriteLine("***** Sending Derivative API translate request");
            JobPayloadInput jobInput = new JobPayloadInput(
                System.Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(urn))
            );
             List<JobPayloadItem> outputs = new List<JobPayloadItem>()
                {
                 new JobPayloadItem(
                   JobPayloadItem.TypeEnum.Step,
                   new List<JobPayloadItem.ViewsEnum>()
                   {
                     JobPayloadItem.ViewsEnum._2d,
                     JobPayloadItem.ViewsEnum._3d
                   })
                };
                JobPayload job = new JobPayload(jobInput, new JobPayloadOutput(outputs));
            dynamic response = derivativesApi.Translate(job, true);
            Console.WriteLine("***** Response for Translating File to SVF: " +
response.ToString());

            if (response.result == "success" || response.result == "created")
            {
                base64Urn = response.urn;
                dynamic manifest = verifyJobComplete(base64Urn);
                if (manifest.status == "success")
                {
                    return true;
                }
            }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error translating file : " + ex.Message);
            }
            return false;
```

Step 5 – Translate the file to STEP

```
       try
            {
                Console.WriteLine("***** Sending Derivative API translate request");
                JobPayloadInput jobInput = new JobPayloadInput(urn);
                List<JobPayloadItem> outputs = new List<JobPayloadItem>()
                {
                 new JobPayloadItem(
                   JobPayloadItem.TypeEnum.Step,
                   new List<JobPayloadItem.ViewsEnum>()
                   {
                     JobPayloadItem.ViewsEnum._2d,
                     JobPayloadItem.ViewsEnum._3d
                   })
                };
                JobPayload job = new JobPayload(jobInput, new JobPayloadOutput(outputs));
                dynamic response = derivativesApi.Translate(job, true);
                Console.WriteLine("***** Response for Translating File to STEP: " +
response.ToString());

                if (response.result == "success" || response.result == "created")
                {
                    base64Urn = response.urn;
                    dynamic manifest = verifyJobComplete(base64Urn);
                    if (manifest.status == "success")
                    {
                        return true;
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error translating file : " + ex.Message);
            }
            return false;
```

Step 6 – Check if the file translation is complete

```
            while (true)
            {
                dynamic response = derivativesApi.GetManifest(base64Urn);
                if (hasOwnProperty(response, "progress") && response.progress ==
"complete")
                {
                    Console.WriteLine("***** Finished translating your file to SVF -
status: " + response.status
                        + ", progress: " + response.progress);
                    return (response);
                }
                else
                {
                    Console.WriteLine("***** Haven't finished translating your file to
SVF - status: " + response.status
                        + ", progress: " + response.progress);
                    Thread.Sleep(1000);
                }

            }
```

Step 7 – Open the viewer

```
Console.WriteLine("***** Opening SVF file in viewer with urn:" + base64Urn);
            string st = _html.Replace("__URN__", base64Urn).Replace("__ACCESS_TOKEN__",
twoLeggedCredentials.access_token);
            System.IO.File.WriteAllText("viewer.html", st);

            System.Diagnostics.Process.Start(new
System.Diagnostics.ProcessStartInfo("viewer.html"));
```

HTML Code to load the viwer.

```
<html>
<head>
       <meta charset=""UTF-8"">
       <script
src=""https://developer.api.autodesk.com/viewingservice/v1/viewers/three.min.css""></scri
pt>
       <link rel=""stylesheet""
href=""https://developer.api.autodesk.com/viewingservice/v1/viewers/style.min.css"" />
       <script
src=""https://developer.api.autodesk.com/viewingservice/v1/viewers/viewer3D.min.js""></sc
ript>
</head>
<body onload=""initialize()"">
<div id=""viewer"" style=""position:absolute; width:90%; height:90%;""></div>
<script>
       function authMe () { return ('__ACCESS_TOKEN__') ; }

       function initialize () {
              var options ={
                     'document' : ""urn:__URN__"",
                     'env': 'AutodeskProduction',
                     'getAccessToken': authMe
              } ;
              var viewerElement =document.getElementById ('viewer') ;
              //var viewer =new Autodesk.Viewing.Viewer3D (viewerElement, {}) ; / No
toolbar
              var viewer =new Autodesk.Viewing.Private.GuiViewer3D (viewerElement, {}) ;
// With toolbar
              Autodesk.Viewing.Initializer (options, function () {
                     viewer.initialize () ;
                     loadDocument (viewer, options.document) ;
              }) ;
       }
       function loadDocument (viewer, documentId) {
              // Find the first 3d geometry and load that.
              Autodesk.Viewing.Document.load (
                     documentId,
                     function (doc) { // onLoadCallback
                            var geometryItems =[] ;
                            geometryItems
=Autodesk.Viewing.Document.getSubItemsWithProperties (
                                   doc.getRootItem (),
                                   { 'type' : 'geometry', 'role' : '3d' },
                                   true
                            ) ;
                            if ( geometryItems.length <= 0 ) {
                                   geometryItems
=Autodesk.Viewing.Document.getSubItemsWithProperties (
                                          doc.getRootItem (),
                                          { 'type': 'geometry', 'role': '2d' },
                                          true
                                   ) ;
                            }
                            if ( geometryItems.length > 0 )
                                   viewer.load (
                                          doc.getViewablePath (geometryItems [0])//,
                                          //null, null, null,
                                          //doc.acmSessionId /*session for DM*/
                                   ) ;
                     },
                     function (errorMsg) { // onErrorCallback
                            alert(""Load Error: "" + errorMsg) ;
                     }
              ) ;
       }
</script>
</body>
</html>";

        #endregion

    }

}
```

Step 8 : Extract Metadata and properties from the Model (not in the context of the viewer)

```
dynamic metadata =  derivativesApi.GetMetadata(urn);
            foreach (KeyValuePair<string, dynamic> metadataItem in new
DynamicDictionaryItems(metadata.data.metadata))
            {
                dynamic hierarchy =  derivativesApi.GetModelviewMetadata(urn,
metadataItem.Value.guid);

                dynamic properties =  derivativesApi.GetModelviewProperties(urn,
metadataItem.Value.guid);

                Console.WriteLine(hierarchy);
                Console.WriteLine(properties);


            }
```

Step 9 – Extract Geometry

```
            {
                JobPayloadInput jobInput = new JobPayloadInput(urn);
                List<JobPayloadItem> outputs = new List<JobPayloadItem>()
                {
                 new JobPayloadItem(
                   JobPayloadItem.TypeEnum.Obj,
                   new List<JobPayloadItem.ViewsEnum>()
                   {
                     JobPayloadItem.ViewsEnum._2d,
                     JobPayloadItem.ViewsEnum._3d
                   },
                   new
JobObjOutputPayloadAdvanced(){ModelGuid="4f981e94-8241-4eaf-b08b-cd337c6b8b1f",
ObjectIds=selectedObjectIds,ExportFileStructure=JobObjOutputPayloadAdvanced.ExportFileStr
uctureEnum.Single })
                   };

                JobPayload job = new JobPayload(jobInput, new JobPayloadOutput(outputs));
                dynamic response = derivativesApi.Translate(job, true);
                Console.WriteLine("***** Response for Extracting File to OBJ: " +
response.ToString());

                if (response.result == "success" || response.result == "created")
                {
                    base64Urn = response.urn;
                    dynamic manifest = verifyJobComplete(base64Urn);
                    if (manifest.status == "success")
                    {
                        return true;
                    }
                }
            return false;
```

Walkthrough of Customer Sample

This section will cover one or more customer samples in the class.

Addtional samples

a) Introduction to Model Derivative API : This sample will connect to your Autodesk A360 account, displays the selected model in the viewer , shows the properties of the selected model view and provides an ability to download the translated version of the selected part(s).

https://github.com/Autodesk-Forge/model.derivative-nodejs-sample

b) Export Revit Data : The app lets you export Revit data, hosted on BIM360, into an Excel file

https://github.com/Autodesk-Forge/bim360appstore-model.derivative-nodejs-xls.exporter

Upcoming APIs

Design Automation API for Revit IO, Inventor IO: Used for batch processing of Revit and Inventor data from Models.

Reality Capture API: Used to create 3D Mesh from overlapping photos. Eg- 3DR is using Autodesk's ReCap API to capture data using drones, process the photos in the cloud using Recap API.

Web hooks: Used for creating custom callback on existing events of the API that require polling such as checking for job completion status in Model Derivative API.

Additional Resources

Developer Portal: developer.autodesk.com
Forge website: forge.autodesk.com
Code Samples: https://github.com/Autodesk-Forge

Twitter: @autodeskforge

Interesting Forge samples:

http://labs.blogs.com/its_alive_in_the_lab/2017/09/pierre-masson-sample-autodesk-forge-viewer-application-with-source-code.html

https://derivatives.autodesk.io/