SD323073

Tips, Tricks, and the Future for Forge Model Derivative Services

Kevin Vandecar – Developer Advocate
Autodesk

Denis Grigor – Developer Advocate
Autodesk

Learning Objectives

- Discover the basics and use of Model Derivative services
- Discuss tips for successful translations
- Learn about tricks to be aware of for certain file formats
- Learn about the future plans of the Forge Model Derivative services

Description

In this presentation, we'll cover some of the tips and tricks for successful model derivative service translations. We'll cover how to handle assemblies and advanced translation submissions, as well as tricks for certain file formats (for example, with the 3ds Max format, did you know you can use the archive format to include textures?). We'll end the presentation with a discussion about the future of model derivative services and how it fits into the future of Forge.

Speaker(s)

Kevin Vandecar is a Forge developer advocate and also the manager for the Media & Entertainment and Manufacturing Autodesk Developer Network Workgroups. His specialty is 3ds Max software customization and programming areas, including the new Forge Design Automation for 3ds Max service.

Denis Grigor: Within Autodesk, he 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.

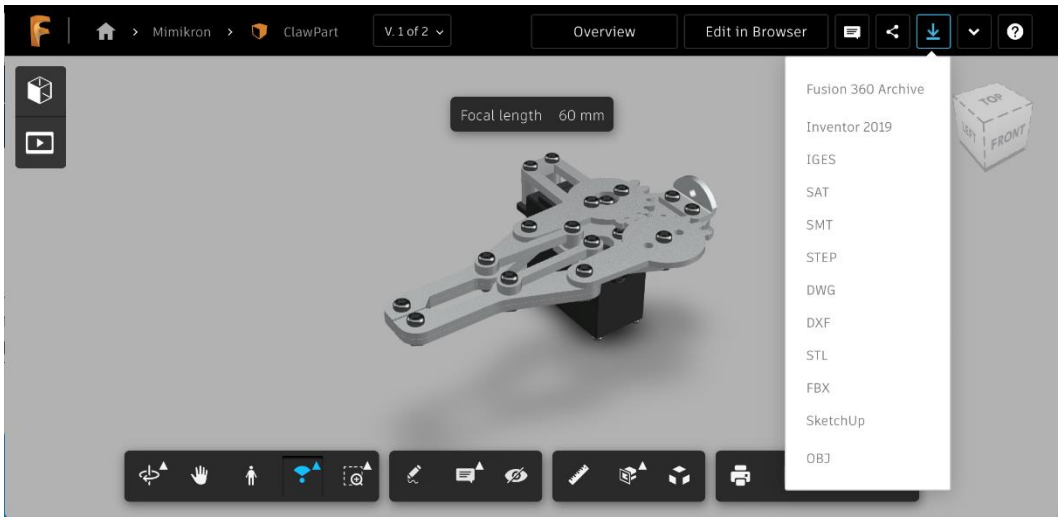
Introduction to Model Derivative Service

Autodesk is very well known for its desktop applications. But for many years there have been connections to the internet and cloud. These cloud powered features are becoming normal and blurring the line between desktop and cloud applications.

These cloud features are based on technologies that previously were available only for Autodesk internal use. The main idea behind the Forge platform *(https://forge.autodesk.com/)* is to make this common functionality available outside to developers, partners and customers to build new tools and customize existing tools.

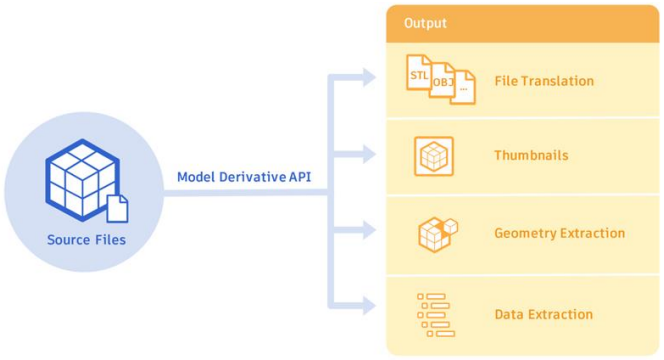
In this class we will concentrate our attention on Forge Model Derivative service. Originally known as part for View and Data API, it is very closely linked to the Forge Viewer as it is the step needed to convert the model for the viewing.

Since then it has become mature enough to be a separate service. It can be found directly or indirectly in many Autodesk products and services and it is more than just a pre-step for visualizing a model in the Viewer:



Basics and use of Model Derivative services

As mentioned above, the Model Derivative service provides more than just the traditional "Translate designs into SVF format for rendering in the Forge Viewer". In fact, "this API got its name because derived means to receive or obtain from a source or origin and in the case, metadata can be derived from original source data or a different file format can be derived from data in its original format":



Traditionally it is seen as a translator to SVF, but translation into other formats is also very useful for some workflows. For example:

Source Format	Derivative Format							
RVT	DWG	IFC					SVF	thumbnail
IAM			IGES	OBJ	STEP	STL	SVF	thumbnail
F2D	DWG							
F3D	DWG	FBX	IGES	OBJ	STEP	STL	SVF	thumbnail
FBX			IGES	OBJ	STEP	STL	SVF	thumbnail

Some translations could be used to strip/filter the data you need. In case of Revit, usually when you want to export DWG from a Revit file, you would need to do it explicitly through Revit. But, as we see in the above table, the RVT file can be translated into DWG. It will automatically extract 2D views of the model and return them as DWG.

Tip: Did you know there is a Model Derivative API to request the currently supported formats? See here: https://forge.autodesk.com/en/docs/model-derivative/v2/reference/http/formats-GET/ *This API returns the formats that are supported, with a subsection of the formats that can be used to generate those formats. For example, in the 'dwg' section, you can see f2d, f3d, and rvt can be used to extract DWG files:*

```
"dwg": [
            "f2d",
            "f3d",
            "rvt"
```

Why would you want to know at runtime? Maybe you support DWG only in your workflow, so you could know which formats are supporting it at any given time. Or to allow you to determine those formats that are currently supported for viewing if you provide a generic viewing solution.

Model Derivative service – Not just a step for the Forge Viewer

Although translation is an important part of Model Derivative service, it is often underestimated and includes features for Data Extraction and Geometry Extraction.

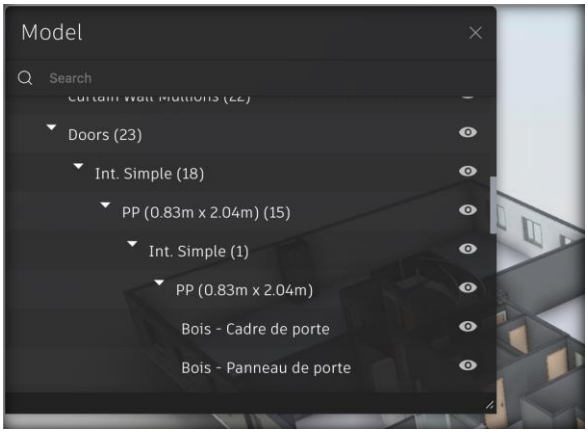
Data Extraction refers to getting the metadata associated to each component, without loading the model *(into Viewer or somewhere else).* You can consider the metadata as a database for the information coming from the translated models, that you can query anytime and as many times as you want, and at no cost *(once the model is translated)*. Note that depending on the source format and associated translator, the metadata returned will vary. For example, in a data rich example like Revit (that has led to the whole BIM movement), the source model and translator supports a very rich set of data extraction. But for a format like 3ds Max (*.max) format, where the objects are often generic geometry and meshes, there is not much metadata to be provided.

Tip: It is always good to experiment with the formats you are interested in, and test for the metadata and geometry extraction. If one translation does not provide everything, you can consider trying another. For example, an FBX file from Maya will yield different results of geometry make-up than OBJ format. Pay attention to the exporters and save options as well.

From the API, you can access the model structure and get the same components hierarchy as the Viewer displays. In fact, the viewer is using this API as well:


```
{
  "objectid": 134,
  "name": "Doors",
  "objects": [
    {
      "objectid": 135,
      "name": "Int. Simple",
      "objects": [
        {
          "objectid": 136,
          "name": "PP (0.83m x 2.04m)",
          "objects": [
            {
              "objectid": 137,
              "name": "Int. Simple",
              "objects": [
                {
                  "objectid": 138,
                  "name": "PP (0.83m x 2.04m)",
                  "objects": [
                    {
                      "objectid": 139,
                      "name": "Bois - Cadre de porte"
                    },
                    {
                      "objectid": 140,
                      "name": "Bois - Panneau de porte"
                    },
                    {
                      "objectid": 141,
                      "name": "Finition Peinture - Blanc satin\u00e9"
```


The data returned from the API

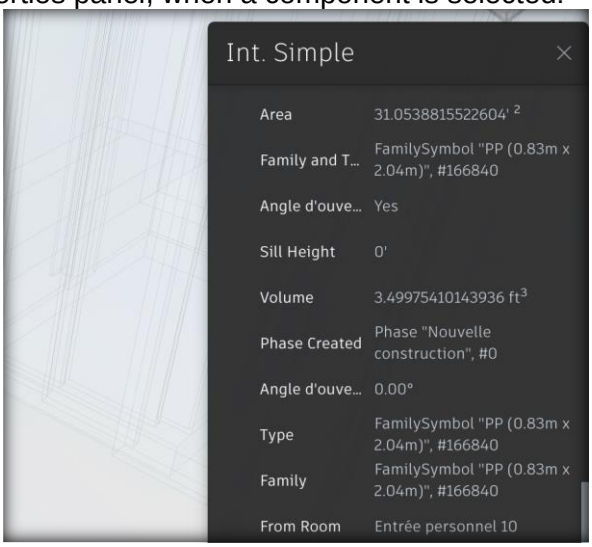


The data as shown in the Forge Viewer

In addition to the model structure, you can get the properties for any component of the model and get the same data we usually see in the properties panel, when a component is selected:


```
{
  "objectid": 137,
  "name": "Int. Simple",
  "externalId": "1/3/0/0/0",
  "properties": {
    "Element": {
      "Angle d'ouverture 2D fixe": "Yes",
      "Angle d'ouverture 3D": "0.00\u00b0",
      "Area": "31.054 ft^2",
      "Category": "Doors",
      "Family": [
        "Int. Simple",
        "FamilySymbol \"PP (0.83m x 2.04m)\", #166840"
      ],
      "Family and Type": "FamilySymbol \"PP (0.83m x 2.04m)\", #166840",
      "From Room": "Entr\u00e9e personnel 10",
      "Head Height": "6.693 ft",
      "Host Id": "Wall \"Voile BA 18\", #248972",
      "Id": "252589",
      "Level": "Level \"Niveau 0\", #608",
      "Mark": "1",
      "Name": "PP (0.83m x 2.04m)",
      "Phase Created": "Phase \"Nouvelle construction\", #0",
      "Sill Height": "0.000 ft",
      "Type": [
        "PP (0.83m x 2.04m)",
        "FamilySymbol \"PP (0.83m x 2.04m)\", #166840"
      ],
      "Type Id": "FamilySymbol \"PP (0.83m x 2.04m)\", #166840",
      "Visible dans les nomenclatures": "No",
      "Volume": "3.500 ft^3",
```


The data returned from the API



The data as shown in the Forge Viewer

For example, in use cases of cost analysis or data aggregation *(i.e. get number and total weight of components made out of Aluminum 5052-H38 in the model)*, that is often seen implemented as a Forge Viewer extension, could be made in the backend *(integrated in a pipeline* to get the necessary data (without viewing at all).

*Tip: When query the data we use the **:urn/metadata/:guid/properties** endpoint to get a JSON with metadata for all components of the model. If the JSON is bigger than 2097152 bytes compressed, you will get an error:*

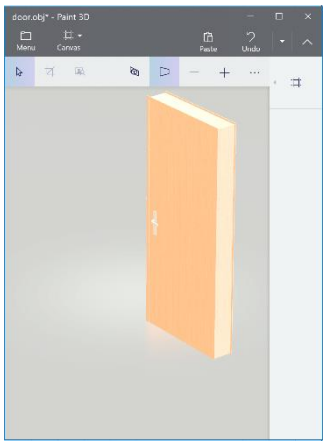
```
413 Request Entity Too Large
{Diagnostic": "Please use the 'forceget' parameter to force querying the data."}
```

*and the suggestion on how to fix it – just add '**?forceget=true**' to the end of your GET request and things will work as expected.*

*Tip: If you are not interested in getting the entire JSON with metadata for all components of the model, but instead want to get the metadata of a certain component – just add '**?objectid=XXX**' to the end of your GET request and get a small JSON with metadata for only the component with the specified 'objectid'.*

Geometry Extraction refers to getting the shape of the needed component or a group of components in form of OBJ along with materials and textures, without loading the model *(into Viewer or somewhere else).*

To achieve this, we need to know the "objectid" of the components/objects for which you want to extract the geometry and this info can be extracted from the model structure data:



Having this hierarchy, we can either extract a needed component or a group of components.

*Tip: Even if in the list of supported translations (https://forge.autodesk.com/en/docs/model-derivative/v2/developers_guide/supported-translations/), for many formats the OBJ is not mentioned as a destination format (i.e. Navisworks format can only be translated to SVF). But, using the Geometry Extraction feature makes it possible to get OBJ from any SVF. Just specify the "**objectid**" of the root, which is usually objectid = 1, and you'll get an OBJ of the entire model.*

NEU\.\d+$	OBJ	SVF	thumbnail
NWC		SVF	thumbnail
NWD		SVF	thumbnail

The rule is simple: if the model supports as SVF translation, AND the model or parts of it has a volume, then it is likely that this approach can be applied to extract the volume aspects of the model into OBJ.

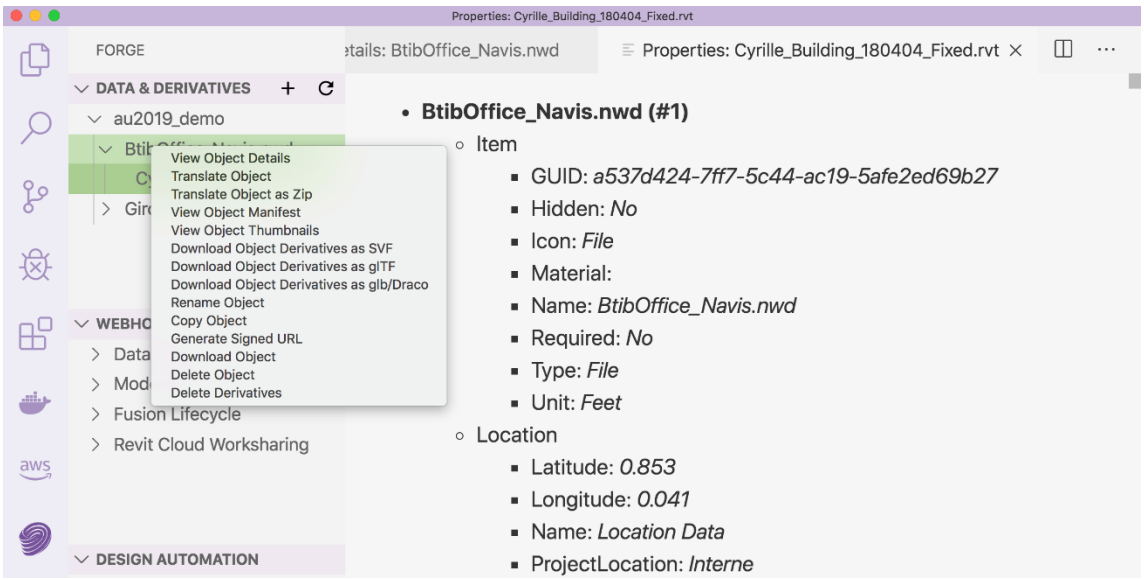
Tools to help master Model Derivative

The Model Derivative Service, as well as all other Forge APIs are easy to master and can be integrated either into your pipeline or use it to extend and enhance existing application. As you have probably noticed with the Forge Developer portal documentation, they are typically using CURL. Although CURL simplifies and clearly demonstrates the use of the APIs, it is a very manual process and there are other tools that can help you to learn and experiment faster.

For example, one of our engineers integrated Forge APIs into Visual Studio Code that help to:

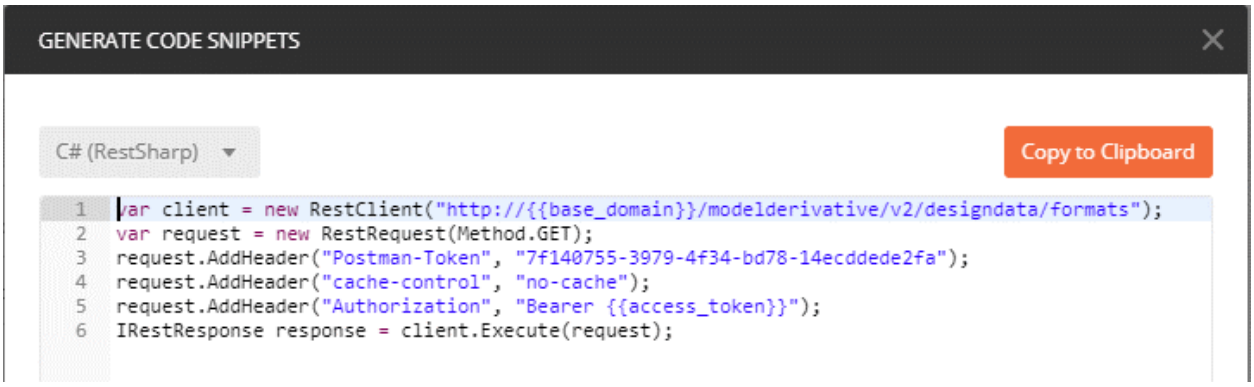
- creation of buckets,
- uploading of the objects into those buckets,
- trigger translation
- inspect the object tree and the properties

  Some of things we discussed above can be achieved using this tool in a more convenient way, but the main purpose of this tool here is to illustrate how Forge APIs can be easily integrated into any app (at least those supporting plugins/extensions).



More information on how to get and setup this Visual Studio Extension can be found at https://forge.autodesk.com/blog/forge-visual-studio-code

Postman can also accomplish this in a similar way. Postman also has the ability to generate API calls in the language and syntax of your choice. For example, here is the GET formats API call in C# (using the restsharp library):



There are a few Postman collections out there already that you can import and customize as you want. See these links for two different collections:

https://forge.autodesk.com/blog/my-postman-collection
https://github.com/yiskang/forge.api.postman

Special Cases

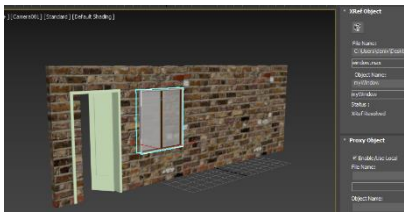
These tools will help you for generic cases, but for particular situations, you'll have to go back to API and consider the issues. Some issues that may come up:

1. **the file is too large**:

   When uploading our model to a bucket for further translation, we usually use ***buckets/:bucketKey/objects/:objectName*** endpoint. However, if the file is larger than 100Mb it is recommended to use ***buckets/:bucketKey/objects/:objectName/resumable*** endpoint.
   Even if it is not directly related to Model Derivative Service (MD), in case of a pipeline, this scenario should also be investigate as reason of MD failure.

2. **the file has XRefs**:

   When the source file is just a self-contained file, the workflow is simple – just send it to be translated. On the other hand, if the source file is a composite one, be it an Inventor assembly file, or a 3ds Max file with XRefs and textures – then the approach is a bit different:



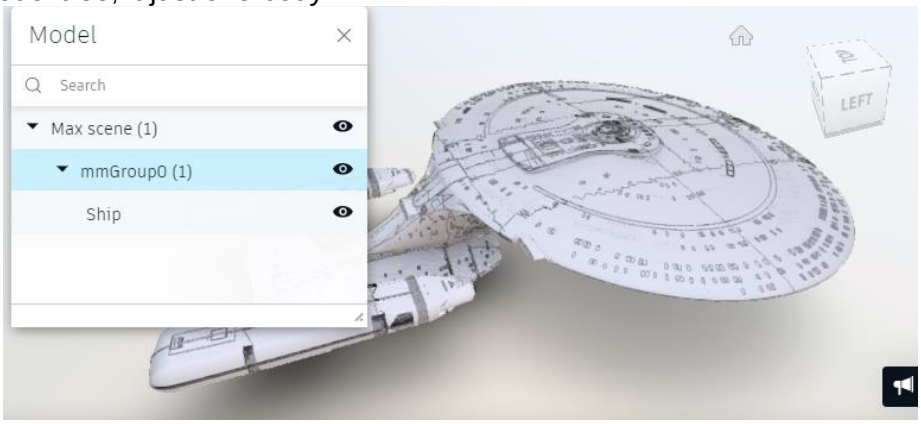
```
job_endpoint = '/modelderivative/v2/designdata/job'
header = {
    "Authorization": "Bearer " + token,
    "Content-Type": "application/json",
}

payload = json.dumps({
    "input" : {
        "urn": '"' + encoded_urn + '"',
        "rootFilename": "wall.max",
         "compressedUrn": True
    },
    "output": {
        "formats": [
            {
                "type": "svf",
                "views": [
                    "2d",
                    "3d"
                ]
            }
```

The solution is simple and it requires to pack all the dependencies along with the main file in one archive and then submit the job along with 2 additional mentions: ***rootFilename*** - the name of the main file and ***compressedUrn*** – set it as ***true*** to indicate that the source file is an archive.

Tip: Obviously, in case of 3ds Max and a few other formats, it might make more sense to handle differently. Again, in the 3ds Max context, you can use the “Archive” format that is simply a Zip file that contains ALL the externally references items (including textures). Another option for 3ds Max and other formats can be to export the scene into FBX using the “media embedded” option from the export plugin. But also remember that changing to another format, might strip the metadata you need.

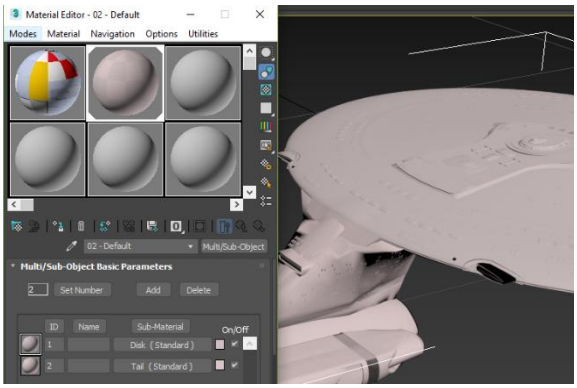
Some source file formats also have behaviors in translation that can be used for many useful things.

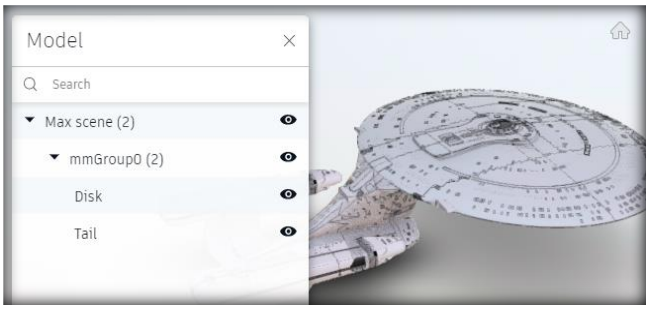
For example, if you have a model in STL format that is just a single body, where it should be multiple components that can be selected independently, let’s look at a trick to help with this using 3ds Max. As a simple illustration, let’s say this is an NCC Enterprise Ship and as we can see in the model tree, it just one body:



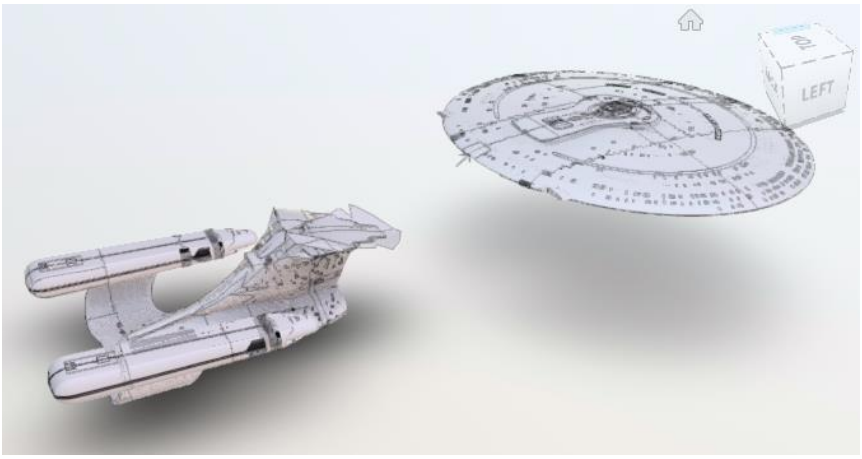
Anyone who watches Star Trek knows that ships of this class, are in fact 2 parts acting as a single body and in time of peril, they can disconnect and act independently. To separate the Disk from the Tail we could manually split the body using a 3D editor into in 2 separate meshes. But this could take some time and tool knowledge, and we actually may not want 2 meshes in the source model. Using 3ds Max where there is a more flexible way. If you assign different material id’s to polygons of the mesh and then assign a Multi-Material to the mesh, then upon translation all polygons with the same material id will be considered as a component with the name of the Sub-Material you assigned to them.

In this case, the Disk and the Tail was assigned different material ids with the Sub-Material names.





Having 2 components, we can also show off by using the “explode tool”, to illustrate how these 2 parts separate from each other:



This might look like a silly example, but it is just an illustration how this technique could be applied to transform mono-part models into multi-part ones, with further customizations.

In another example, by default the translator for Revit models does not support the idea of “rooms” in a model. But Navisworks, does. A simple solution here is to bring the Revit model into Navisworks and then translate as a Navisworks file.

Tip: How can you easily test the variety of formats and different application experiences in your workflow? If you are a developer and need to work with Autodesk desktop products, consider joining the ADN program. This program is meant to help developers and customers working with desktop applications to be able to customize and develop good workflows and solutions in conjunction with Autodesk products. Included with most membership types is access to nearly all the desktop products for development purposes. If you are working with a variety of formats, this might be a cost-effective option. Additionally, if you plan to use Forge Design Automation for Revit, 3ds Max, Inventor, and/or AutoCAD it is another positive reason. Although ADN membership is in no way required for Forge use, it can enhance your ability to work with these tightly integrated Autodesk products.

The future plans of the Forge Model Derivative services

The Model Derivative Service continuously evolves through optimizations, bug fixes and new features which are heavily tested internally and only after that released to public. From what to be expected, we can mention the following:

Efficient Derivative Access:

The current workflow to translate a file is the following:

- having an object in a bucket will trigger the translation;
- check if the model is translated by getting the manifest;
- get the derivative.

This workflow will be changed in the following way:

- no need to get the manifest, switching to Resource-based Request API:
  - GET /derivativeservice/v2/viewable/3d/<seed urn>
  - GET /derivativeservice/v2/convertible/dwg/<seed urn>
- The response and the status code will indicate the state:
  - If derivative exists, returns 200 OK, with the binary stream or signed URL
  - If derivative does not exist, but is under construction, returns 202 ACCEPTED, and the progress
  - If derivative does not exist, and there is no running generation, returns 404 NOT FOUND

Resources and credits:

- https://forge.autodesk.com/blog/forge-visual-studio-code
- https://forge.autodesk.com/blog/faster-get-hierarchy-api-and-how-solve-error-413
- https://forge.autodesk.com/blog/theres-table-derivable-file-formats-autodesk-forge-model-derivative-api
- https://forge.autodesk.com/blog/translate-composite-models-files-references
- https://forge.autodesk.com/blog/export-dwg-rvt-forge-translation-service
- https://notebooks.azure.com/denis-grigor/projects/md-notebook