

SD323111

3ds Max Design Automation—Locally and in the Cloud

Kevin Vandecar – Developer Advocate
Autodesk

Lanh Hong – Developer Advocate
Autodesk

Learning Objectives

- Learn how to build automation routines for 3ds Max
- Learn how to use 3dsmaxbatch tool to automate tasks
- Learn how to expose functionality from C++, .NET, Python, and MAXScript to enable configurable automation
- Learn how to use Forge Design Automation for 3ds Max to automate your tasks in the cloud

Description

This class will discuss the various techniques for automating 3ds Max software. We'll first cover techniques to build automated routines to drive complex tasks within 3ds Max. 3ds Max includes a local tool called 3dsmaxbatch that enables local automation of complex tasks. This functionality is also available in the cloud under the Forge Design Automation API. We'll discuss MAXScript, Python, C++, and .NET approaches to automation, and how to connect your routines to the web to drive configurable automation.

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.

Lanh Hong joined Autodesk as an intern in the summer of 2017 and knew she wanted to come back after graduation. Upon finishing her bachelor's in Computer Science at the University of California, Davis in 2018, she was given the opportunity to join Autodesk as a Developer Advocate working closely with the Autodesk Maya API.

Build automation routines for 3ds Max

3ds Max has always been a powerhouse of customization, so if you are currently using 3ds Max, it's likely you have used at least MAXScript to drive some custom behavior. There are so many resources out there for MAXScript and plugins, even if you are not a coder, it's easy to find custom routines.

What is Automation?

Any routine that can run unattended to do batch processing. Examples include:

Batch process files sets

- Export MAX scenes to other formats. For example to the FBX format
- Import other formats into 3ds Max scene format. For example, import FBX.
- Modify MAX scenes requiring common automated changes

Tools to build, modify, or analyze content

- Configurator for content
- for example: based on user input, build a specialized asset

Modify content

- for example: LOD or Mesh manipulation

Analysis tool

- for example: given model is analyzed for defect or problem criteria

Use 3dsmaxbatch tool to automate tasks

The 3dsmaxbatch.exe tool has been provided with 3ds Max since 2018 release. Note that there were a few adjustments after 2018 updates 2 and 3 , but from there on it has been consistent and ready for use locally. This is also the default Forge Design Automation engine for 3ds Max.

The complete and latest docs are provided here:

<http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=GUID-48A78515-C24B-4E46-AC5F-884FBCF40D59>

Remember that Design Automation for 3ds Max will support back to the 2018 version, and is using the latest Update, so you should not have to worry about the “legacy” behavior.

The best approach for automation is to test and work through all your kinks locally in a known environment. Once you are confident in the automation, you can deploy to a local pipeline, or then extend it to Forge Design Automation.

Because 3dsmaxbatch is accepting scripts (MAXScript or Python) to initiate the automation, you will need at least some scripting code to get things started. There are essentially two ways to handle this:

1. Build a script that executes functionality directly. When passed into the 3dsmaxbatch, it will be executed immediately and launch any automation in that job execution.
2. Using the Application Plug-in Package system. This was originally created to handle app store plugins, but it is also a very powerful loader that allows multiple versions to be supported in the same bundle, plus other configuration options. It also resides outside of the 3ds Max installation folder structure, so provides a nice way to stay independent of the 3ds Max installation. The specification file “PackageContents.xml” has a tag called “post-start-up scripts parts” that will load and evaluate a script after 3ds Max is fully running. This is another way to directly kick-off automation. See here for complete documentation:

http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=developer_writing_plug_ins_packaging_plugins_packagexml_format_html

Expose functionality from C++, .NET, Python, and MAXScript to enable configurable automation

3ds Max has a very rich customization system and provides several environments for programming behavior. This includes:

- MAXScript – a powerful and mature scripting system specific to 3ds Max. The syntax and API are unique to 3ds Max,
- Python – The python environment is basically a wrapper around the MAXScript APIs. You can use Python syntax to call the MAXScript APIs directly.
- C++ SDK – the 3ds Max SDK is primarily C++ based. In order to use the SDK, you will need to install it separately. After 3ds Max itself is installed, you can restart the installer and select tools->SDK. Or even easier is to use the MSI file directly from the distribution file set. Typically, this is located here: <distribution folder>\x64\Tools\MAXSDK. Although there are ways you can code “start-up” execution from a C++ plugin, it is better to expose the functionality to MAXScript so it can be called directly there. You can use Parameter Blocks or Function Publishing for this:
http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=developer_3ds_max_sdk_features_function_publishing_html
- .NET API – The .NET API is a wrapper for the C++ SDK. Even though it is not required to install the SDK to use .NET 3ds Max API assemblies, you should consider installing it. Because the .NET API is a wrapper, the samples and other support from the SDK can be very helpful when working in .NET API. The easiest way to expose functionality here is to use a static class and static methods. There are helper MAXScript functions that will allow you to access the class and methods, that you can then call directly. The only drawback here is that you can only support standard data types and it is not easy to handle 3ds Max types.

Tips:

When developing, keep logic separate from UI

- Allows you to call the logic without needing UI interaction
- Define clean and clear parameters for maintainability

Plugins using Parameter Blocks

- Get/Set for parameters
- Even if Auto-UI, you can still set the parameters from code

Be aware that most C++ plugins are using parameter blocks. Even if there is no “exposed” API, you can usually configure a plugin's data by get/set on the parameters stored in the parameter block. There are two versions of the Parameter block system:

1. ParamBlock1 is the older system. Most plugins within 3ds Max and the SDK samples have been moved to ParamBlock2. But be aware that there is a chance you can still run across an older parameter block. If you have plugins using the older system, there is a porting guide here:
http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=developer_3ds_max_sdk_features_working_with_plugin_parameter_blocks_pb1_to_pb2_conversion_html
2. ParamBlock2 is the current system. See here for complete details:
http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=developer_3ds_max_sdk_features_working_with_plugin_parameter_blocks_html

If your plugin is simple and is driven mainly by parameter blocks, you really do not need to use function publishing unless you want to provide a more robust way to drive the plugin.

Using the parameter block from the ProOptimizer plugin is how the .NET plugin for the 3ds Max Design Automation sample is working. The automation plugin is located in the design automation repo. See here: <https://github.com/kevinvandecar/design.automation.3dsmax-csharp-meshoptimizer/>

Using MAXScript (or Python) is certainly an easy way to build up routines. These can still be installed as an Application Plug-in Package bundle. Again this allows you to keep your routines separate from the actual 3ds Max installation folder structure. Plus you can also support multiple versions and differences easily in the same bundle.

Let's take a look at how a C++ Plugin can be automated. Again, the direct and easiest ways is to use the parameter blocks. In this example, we look at the C++ source code from the bend modifier:

```
bend_angle, _T("BendAngle"),
    TYPE_FLOAT, P_RESET_DEFAULT|P_ANIMATABLE,
IDS_ANGLE,
    p_default, 0.0f,
    p_range, -BIGFLOAT, BIGFLOAT,
    p_ui, TYPE_SPINNER,
    EDITTYPE_FLOAT, IDC_ANGLE, IDC_ANGLESPINNER,
0.5f,
```

```
p_end,
```

Then from MAXScript you can add the modifier to every object in a scene and set the values like this:

```
for obj in Geometry do (
    bendModADN = Bend_OSM_ADN()
    addModifier obj bendModADN
    bendModADN.BendAngle = 45
    bendModADN.BendDir = 70
)
```

Now let's take a look at a more complex example using the .NET API:

First, using .NET API we are configuring the C++ plugin that comes with 3ds Max called ProOptimizer. This plugin is a modifier that does mesh reduction by setting a percentage value.

Add the modifier:

```
IObject obj = node.ObjectRef;
IIDerivedObject dobj = global.CreateDerivedObject(obj);
object objMod = ip.CreateInstance(SClass_ID.Osm, cid as IClass_ID);
IModifier mod = (IModifier)objMod;

dobj.AddModifier(mod, null, 0); // top of stack
node.ObjectRef = dobj;
```

Insure the ProOptimizer modifier exists on an object, and then configure it.

```
IModifier mod = GetModifier(node, cidOsmProoptimizer);
if (mod != null)
{
    // In order to get the "Calculate" parameter to trigger
    // the modifier to execute, we have to enable some UI elements.
    ip.CmdPanelOpen = true; // ensures the command panel in general is open
    ip.SelectNode(node, true); // Select the node to make it active
    ip.CommandPanelTaskMode = 2; //TASK_MODE_MODIFY. This makes the modifier panel active
    // Now we can set the parameters on the modifier, and at end "calculate" the results.
    IIParamBlock2 pb = mod.GetParamBlock(0);
    pb.SetValue((int)ProOptimizerPBValues.optimizer_main_ratio, t, VertexPercent, 0);
    pb.SetValue((int)ProOptimizerPBValues.optimizer_options_keep_uv, t, 1, 0);
    pb.SetValue((int)ProOptimizerPBValues.optimizer_options_keep_normals, t, 0, 0);
    // There is no true way to know if this was valid/invalid for the mesh,
    // so we check the outer level routine on triobject for changes. **
    pb.SetValue((int)ProOptimizerPBValues.optimizer_main_calculate, t, 1, 0);
    ip.ClearNodeSelection(false);
}
}
```

These routines are wrapped inside a static class and methods. For example:

```
namespace Autodesk.Forge.Sample.DesignAutomation.Max
{
```

```
static public class RuntimeExecute
{
    static public int ProOptimizeMesh()
    {
        ...
    }
}
```

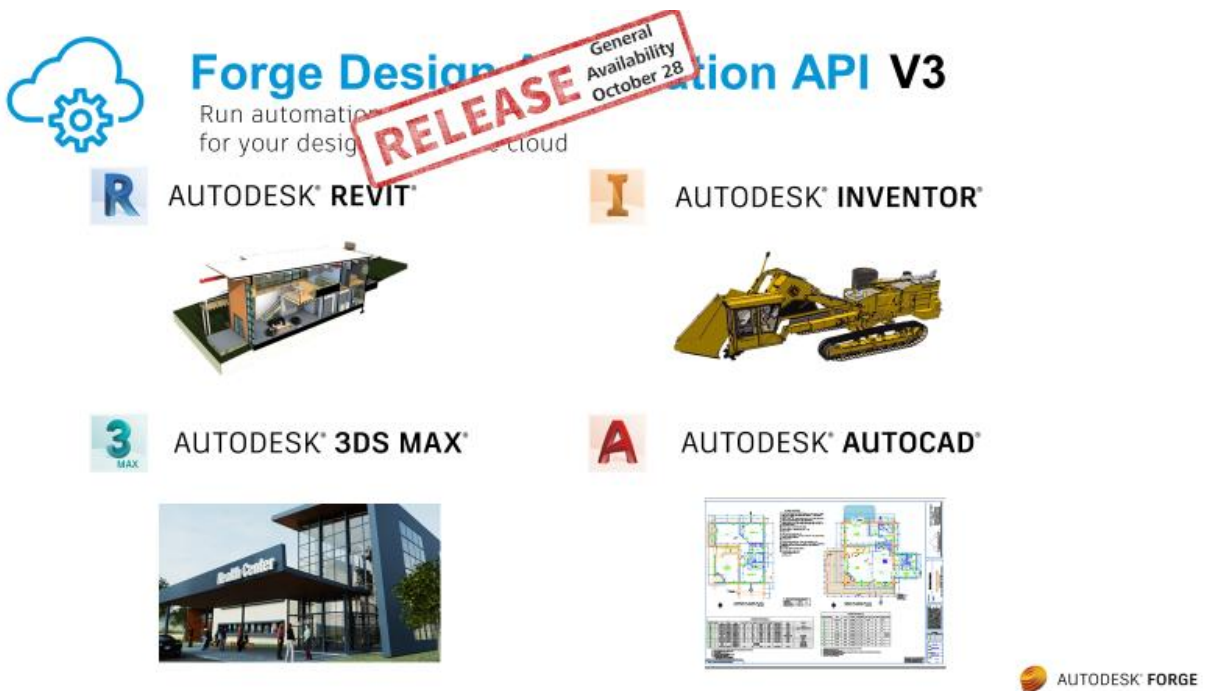
From MAXScript we can call it like this...

```
da = dotNetClass("Autodesk.Forge.Sample.DesignAutomation.Max.RuntimeExecute")
da.ProOptimizeMesh()
```

The MAXscript function “dotNetClass” gains access to the .NET class and methods exposed there. See here for details: <http://help.autodesk.com/view/3DSMAX/2020/ENU/?guid=GUID-779FD7AC-953D-4567-B2A8-60B1D8695B95>

Use Forge Design Automation for 3ds Max to automate your tasks in the cloud

Forge Design Automation V3 has been recently released after being in beta for nearly a year. During this time Autodesk collected much feedback and have now delivered a stable set of traditional desktop engines in the Forge cloud. The Design Automation APIs support 3ds Max, Revit, Inventor, and AutoCAD.

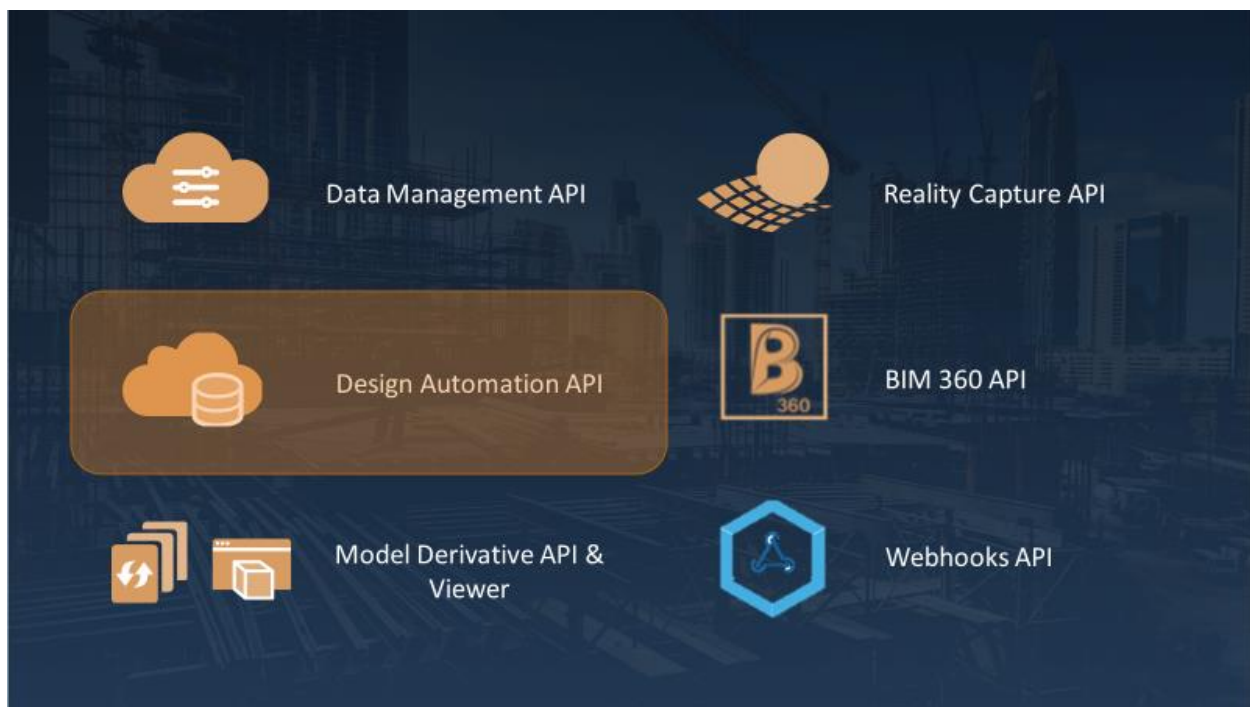


The graphic features a central blue cloud icon with a gear inside. To its right, the text reads "Forge Design Automation API V3" in large blue letters, with "Run automation for your design in the cloud" in smaller text below. A prominent red stamp with white text says "RELEASE" and "General Availability October 28". Below this, four software logos are arranged in a 2x2 grid: Autodesk Revit (blue 'R'), Autodesk Inventor (orange 'I'), Autodesk 3ds Max (green '3'), and Autodesk AutoCAD (red 'A'). Each logo is accompanied by a small image representing the software: a building cutaway for Revit, a yellow excavator for Inventor, a modern building for 3ds Max, and a CAD drawing for AutoCAD. The Autodesk Forge logo is in the bottom right corner.

The Design Automation API itself is identical for all engines. This allows you to build reusable frameworks and tools that can automate any of the four engines with simple code changes. It also allows you to easily work with more than one engine at the same time.

Let's start with a look at how the Forge Design Automation API fits into the overall Forge technology stack. Currently the APIs include:

- [Data Management](#)
- [Reality Capture](#)
- [BIM 360](#)
- [Webhooks API](#)
- [Model Derivative API](#) and [Viewer JavaScript library](#)
- [Design Automation API](#)



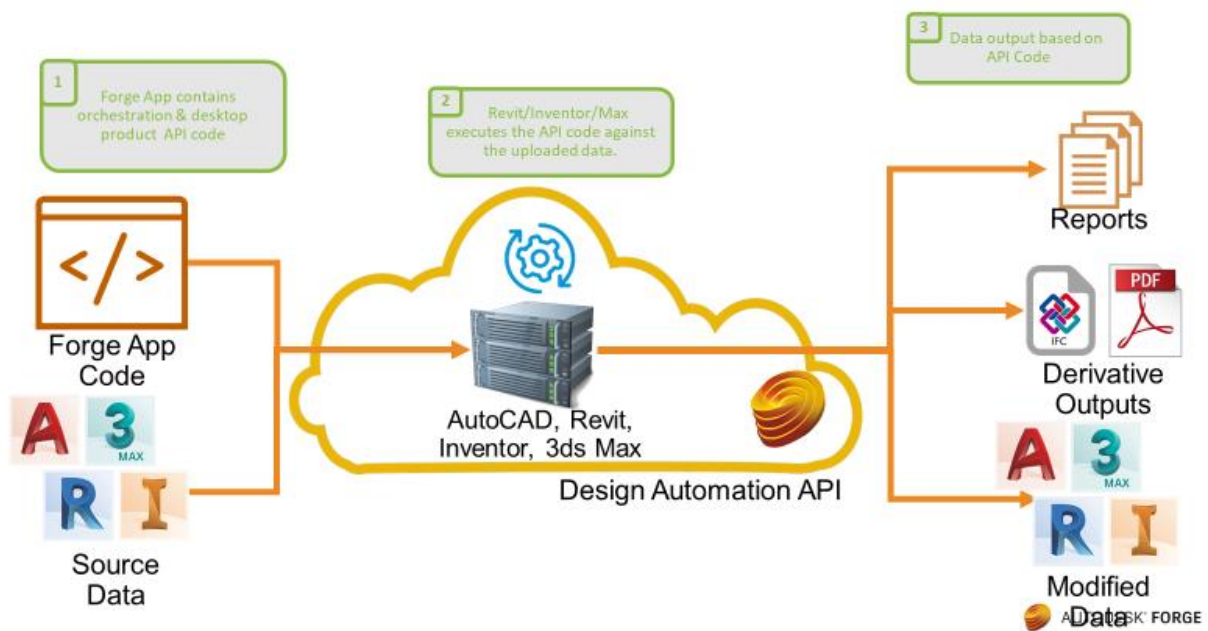
Because Design Automation is just another set of end points in the Forge technology stack, this allows you to easily connect multiple aspects of your automation and workflow together in a single web app. For example, after processing something in 3ds Max Design automation, maybe you would like to preview the geometry. Here you can combine Design Automation for 3ds Max with the Viewer and Model Derivative functionality. Or perhaps you want to use reality Capture to generate a 3d Asset, and then further process it within 3ds Max automation.

The benefits to using Forge Design Automation, instead of local automation include:

- No local resources needed
- No licenses consumed
- No 3ds Max “compatible” hardware
- No branding requirements
- Build custom experiences

For example, with an inhouse automation, you might use Forge Design Automation to run regular jobs at any time, and free up a 3ds Max license for creative work. Or you might build a commercial online tool that provides assets to your customer in 3ds Max format, or any of the other formats that 3ds Max can export.

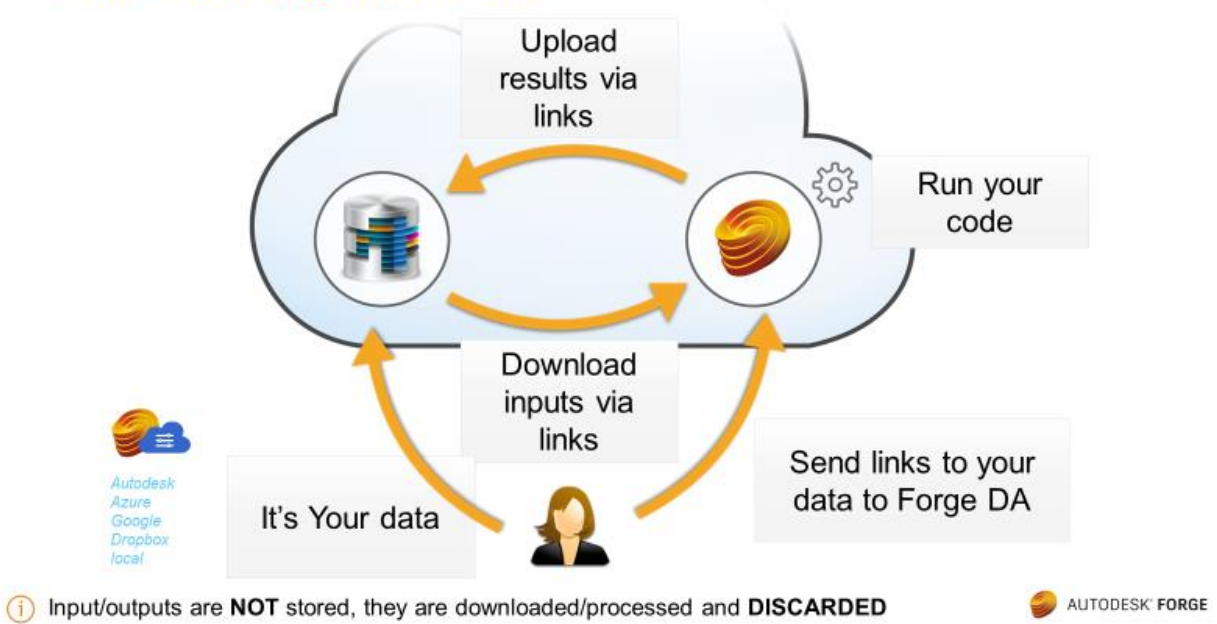
How does it work? Basically, it is very similar to running locally, except you will need to use the Forge Design Automation APIs to manage the jobs. The fundamental idea is like this:



Here you can see there is input data, processing, and output/modified data.

The key idea is to be sure you understand that the data is yours and can be stored anywhere. For example, another cloud storage service (ie. Amazon, Dropbox) or Autodesk hub storage (a360 or BIM 360), or can even be locally uploaded. In that case even in-house centralized network storage can be used. Once the data is downloaded from your storage to the Design Automation instance, it is then processed as you request. Once the job is finished, the output is then uploaded back to your desired location. After the Design Automation job is finished, the instance is shutdown and no data is maintained.

Data Processing on Demand



The Design Automation API is built out as industry standard REST API endpoints. The key concepts are listed in the chart below:

The Design Automation API

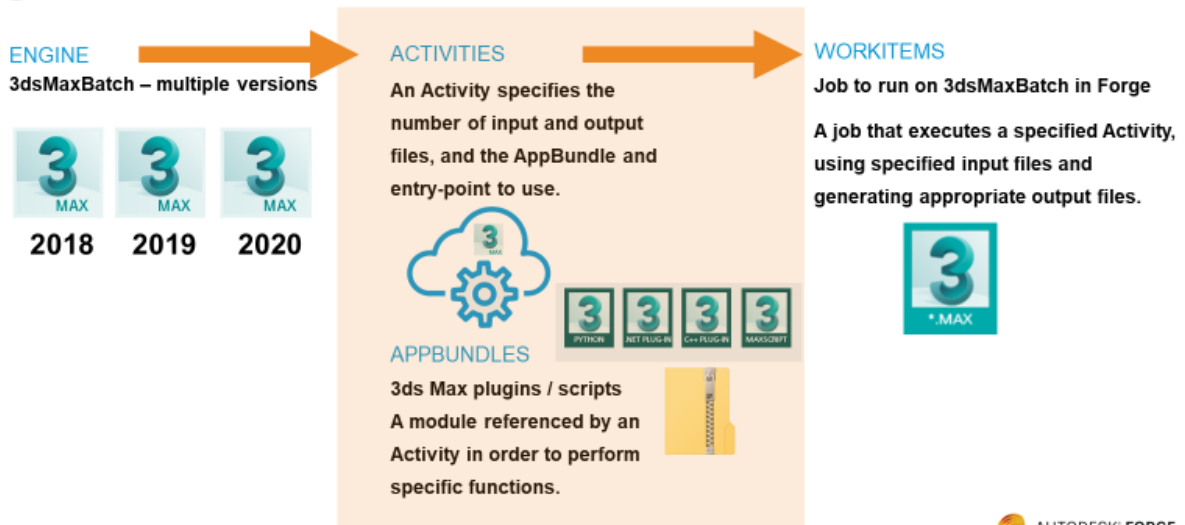
V3 HTTP Endpoint	Programming concept	Product concept	V2 HTTP Endpoint
/workitems	Function call	Product execution, job	/WorkItems
/activities	Function definition	Input/Output types, parameters	/Activities
/appbundles	Shared library	scripts / plugins	/AppPackages
/engines	Instruction set	Product (3dsmax) to use	/Engines

For details of each of these concepts, please refer to the online documentation here: https://forge.autodesk.com/en/docs/design-automation/v3/developers_guide/basics/ and https://forge.autodesk.com/en/docs/design-automation/v3/developers_guide/field-guide/

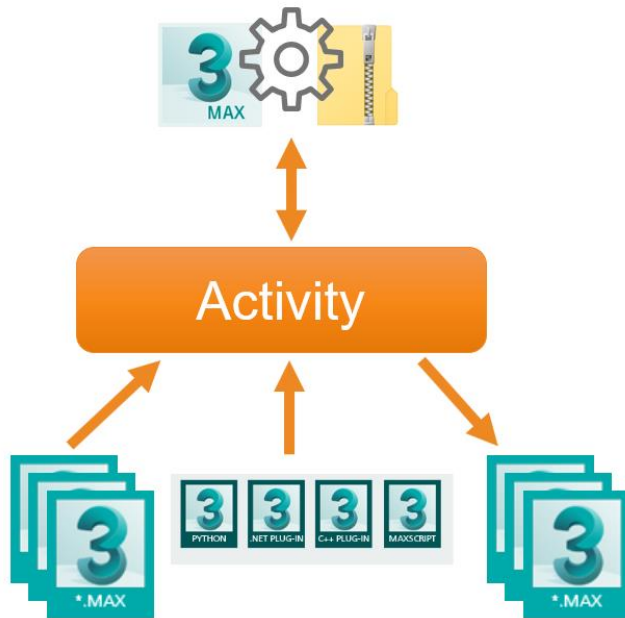
In the 3ds Max context, see below for how each concept relates to the others and are used in conjunction to setup an automation job.



3ds Max Design Automation Terminology



The Activity is the major aspect that defines the automation itself. It describes the AppBundle to use, how many and what are the input files, optionally can use zip files (data sets), types of parameter, and how many and what are the outputs. You can also indicate whether they are shared or not, and by Forge ID or with everyone. You can specify an alias to help you identify each activity and its state. For example, production, beta or alpha. And finally, you can also versions the activities.



Here is an example JSON data structure for a 3ds Max activity that exports the input scene to FBX and provides it back as the output. There is no bundle in this example, and the execution script is provided directly here in the activity:

```
{
  "id": "ExportToFBX",
  "commandLine": "$ (engine.path)/3dsmaxbatch.exe -sceneFile
\\$(args[InputFile].path)\" \"$(settings[script].path)\",
  "description": "Export a single max file to FBX",
  "appbundles": [
  ],
  "engine": "Autodesk.3dsMax+2020",
  "parameters": {
    "InputFile": {
      "zip": false,
      "description": "Input 3ds Max file",
      "ondemand": false,
      "required": true,
      "verb": "get",
      "localName": "input.max"
    },
    "OutputFile": {
      "zip": false,
      "ondemand": false,
      "verb": "put",
      "description": "Output FBX file",
      "required": true,
      "localName": "output.fbx"
    }
  }
},
```

```

"settings": {
  "script": "exportFile (sysInfo.currentdir + \"/output.fbx\") #noPrompt
using:FBXEXP"
}
}

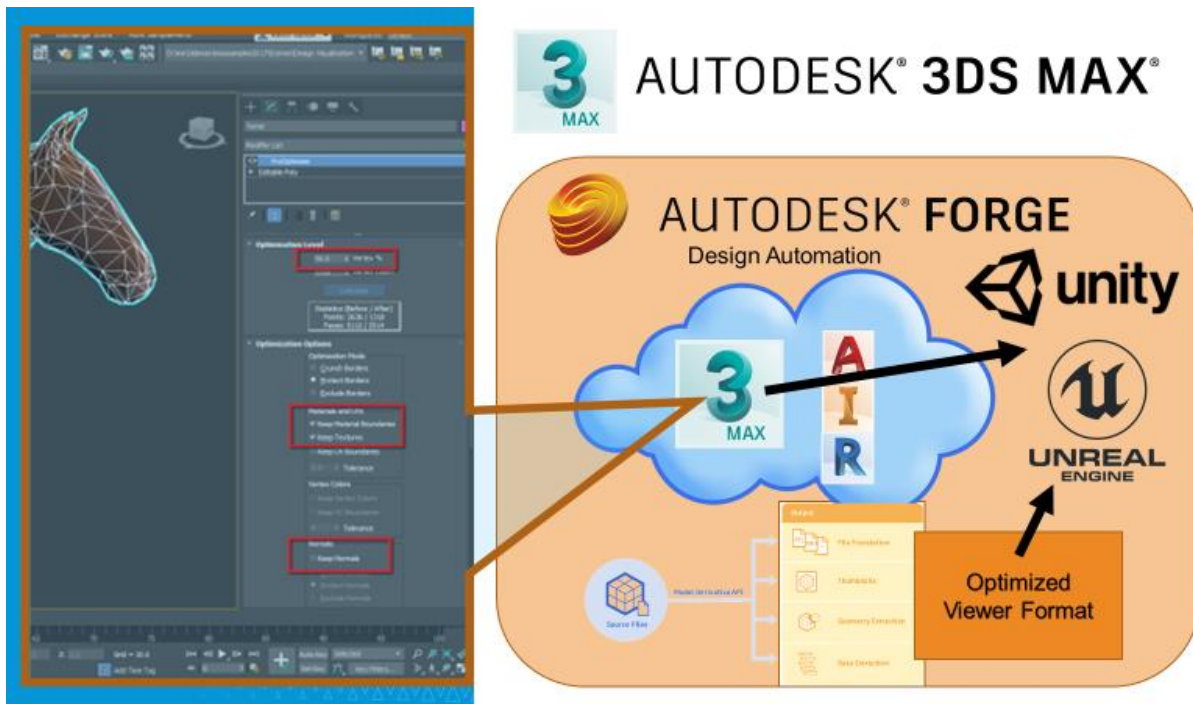
```

There are also callbacks that are an important aspect to Design Automation interaction.

- OnDemand callback
 - Allows you to provide additional input data to the WorkItem during execution, on as needed basis
- OnProgress callback
 - Allows you to check for the status of WorkItem execution on a periodic basis
- OnComplete callback
 - Lets you know when the WorkItem execution has ended without the need to actively check the status

The OnComplete callback is one that you would likely always use.

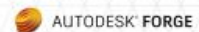
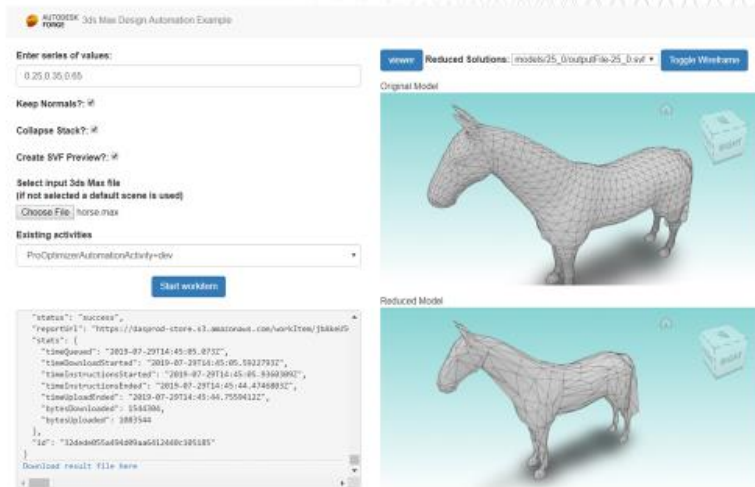
Forge Design Automation for 3ds Max Example



This example is taking an input 3ds Max scene and reducing the mesh level of detail using the ProOptimizer feature. Once the optimization is finished, then the plug-in saves a new 3ds Max scene file, exports an FBX file, and also directly produces a SVF export. SVF is the format that the Forge Viewer is using, and so this allows us to also preview the design.

Example: Modify input Scene file

- Automate ProOptimizer 3ds Max feature using .NET and node.js
- Simple MAXScript to initiate
- JSON data from website input to configure the automation task
- Generate multiple outputs as MAX / FBX
- Generate Forge Viewer format for visual review / comparison



A few final things to consider...

- You have access to multiple versions of 3ds Max (currently 2018, 2019, 2020)
- The jobs are run in parallel, so you can break things down as you want to optimize performance and speed.
- Full MAXScript support (as automation). MAXScript/Python can drive functionality in C++ and .NET
- Custom plug-ins (C++ and .NET)
- No license needed, just pay for use
- Rendering with Arnold for now will have a watermark
- The job runs in a “sandbox” that is protected. This means:
 - One job per worker, nothing shared with anyone else
 - Protects your code, protects the Autodesk infrastructure
 - No access outside of working directory
 - No Internet access
 - What the user is executing should be able to run in low integrity
 - The execution should be self-contained to the low integrity folders
- Be aware of the quotas and limits. Especially:
 - Input and Output file size
 - Job duration

The currently defined quotas for all Design Automation engines are listed here:

https://forge.autodesk.com/en/docs/design-automation/v3/developers_guide/quotas/

Support

Forge Portal - Design Automation API v3 Documentation

https://forge.autodesk.com/en/docs/design-automation/v3/developers_guide/overview/
<https://forge.autodesk.com/en/docs/design-automation/v3/tutorials/3dsmax>

Ask your questions on StackOverflow and use the tag `autodesk-designautomation`. This helps us to find your questions and ensure it is answered. We feel the community is already becoming quite mature and so there are great experiences that everyone is willing to share there.

<https://stackoverflow.com/questions/tagged/autodesk-designautomation>

You can also ask directly if desired. See complete details here:

<https://forge.autodesk.com/en/support/get-help>

Samples are provided via github. The main organization is: Autodesk-Forge and you will find many repos there with code samples in a variety of languages and options.

Tutorials

<https://forge.autodesk.com/developer/getting-started>

<https://forge.autodesk.com/en/docs/design-automation/v3/tutorials/3dsmax/>

<https://learnforge.autodesk.io/#/tutorials/modifymodels> (design automation)

Samples

<https://forge.autodesk.com/code-samples>

<https://github.com/kevinvandecar/design.automation.3dsmax-csharp-meshoptimizer>

<https://github.com/lanhhong/design.automation.3dsmax-nodejs-meshoptimizer>