AUTODESK UNIVERSITY

Getting Started with the Fusion 360 API

Patrick Rainsberry

Senior Product Manager, Fusion 360 | @prrainsberry

About the speaker

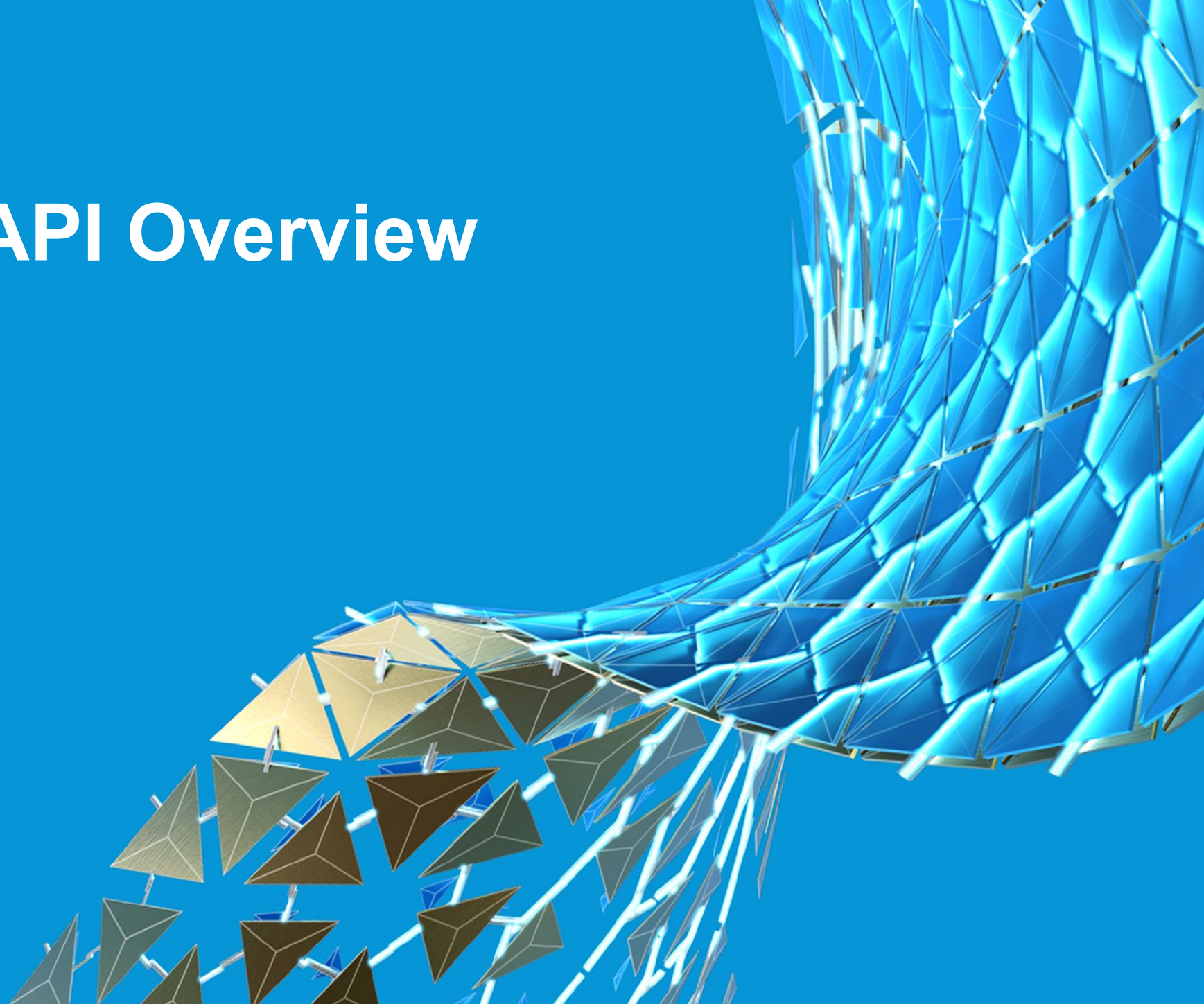
Patrick Rainsberry

I am a mechanical engineer. I have an undergrad degree from UC Berkeley and a Masters from UCLA. I also have an MBA from the University of La Verne. I have been working in the CAD industry for over 15 years as well as some time spent as a design engineer. Currently I am a Product Manager for Fusion 360, working on various projects related to desktop and cloud API's and various other projects related to the data management experience in Fusion 360.

Outline

- API Overview
- Key concepts of the API
- Building an Add-In
- Resources

Fusion 360 API Overview

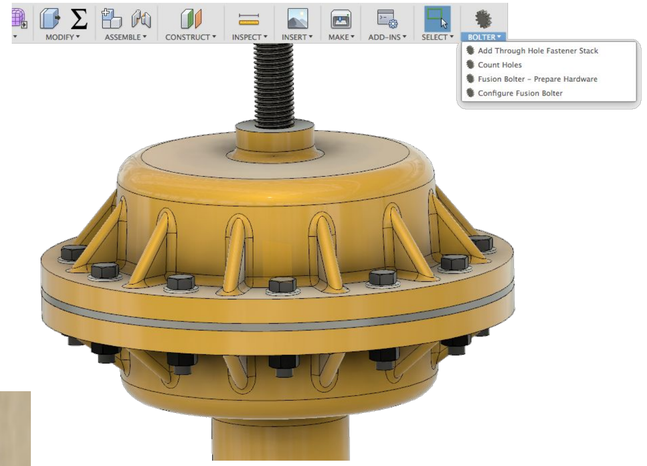


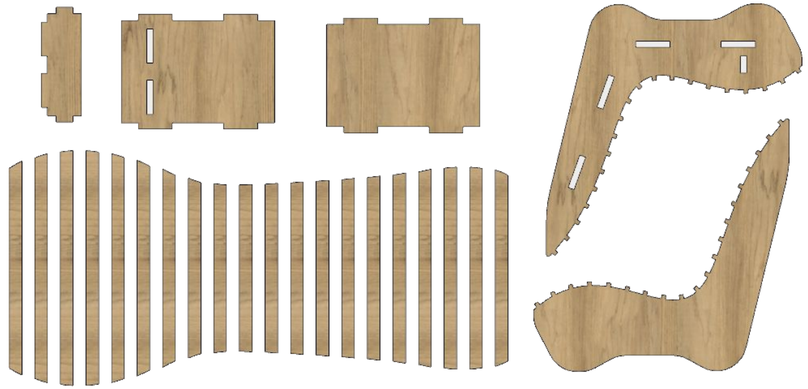
Things to Automate

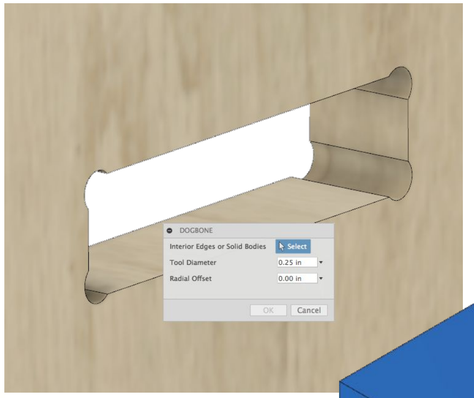
Repetitive tasks

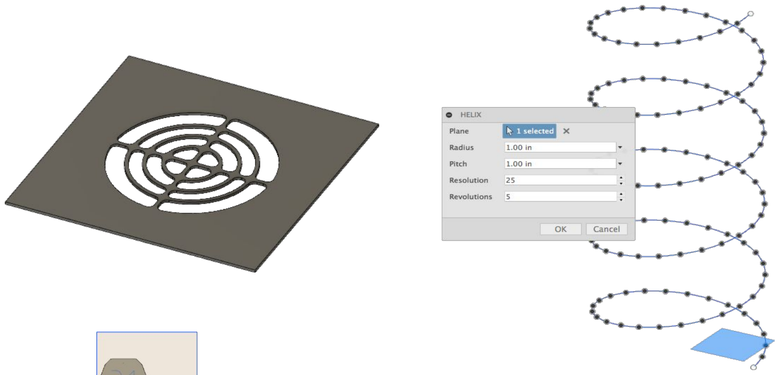
Data import / export

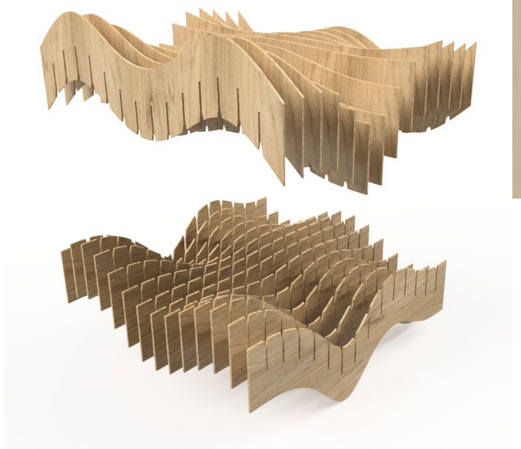
Complex operations

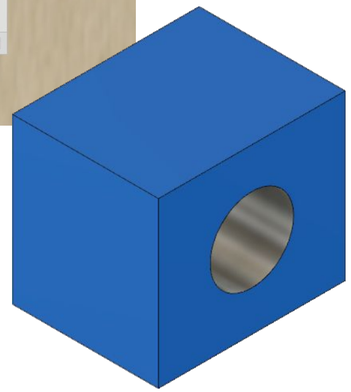


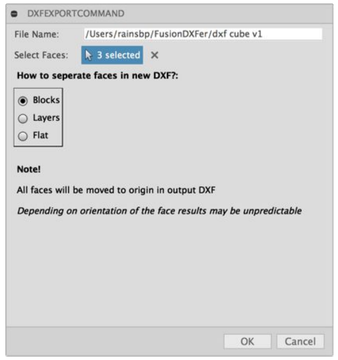


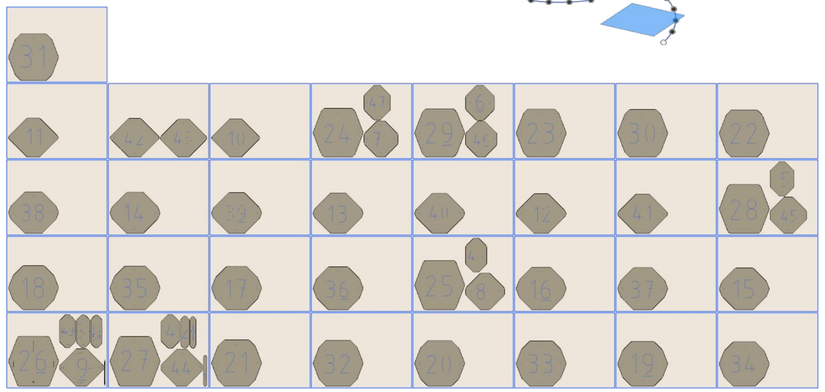












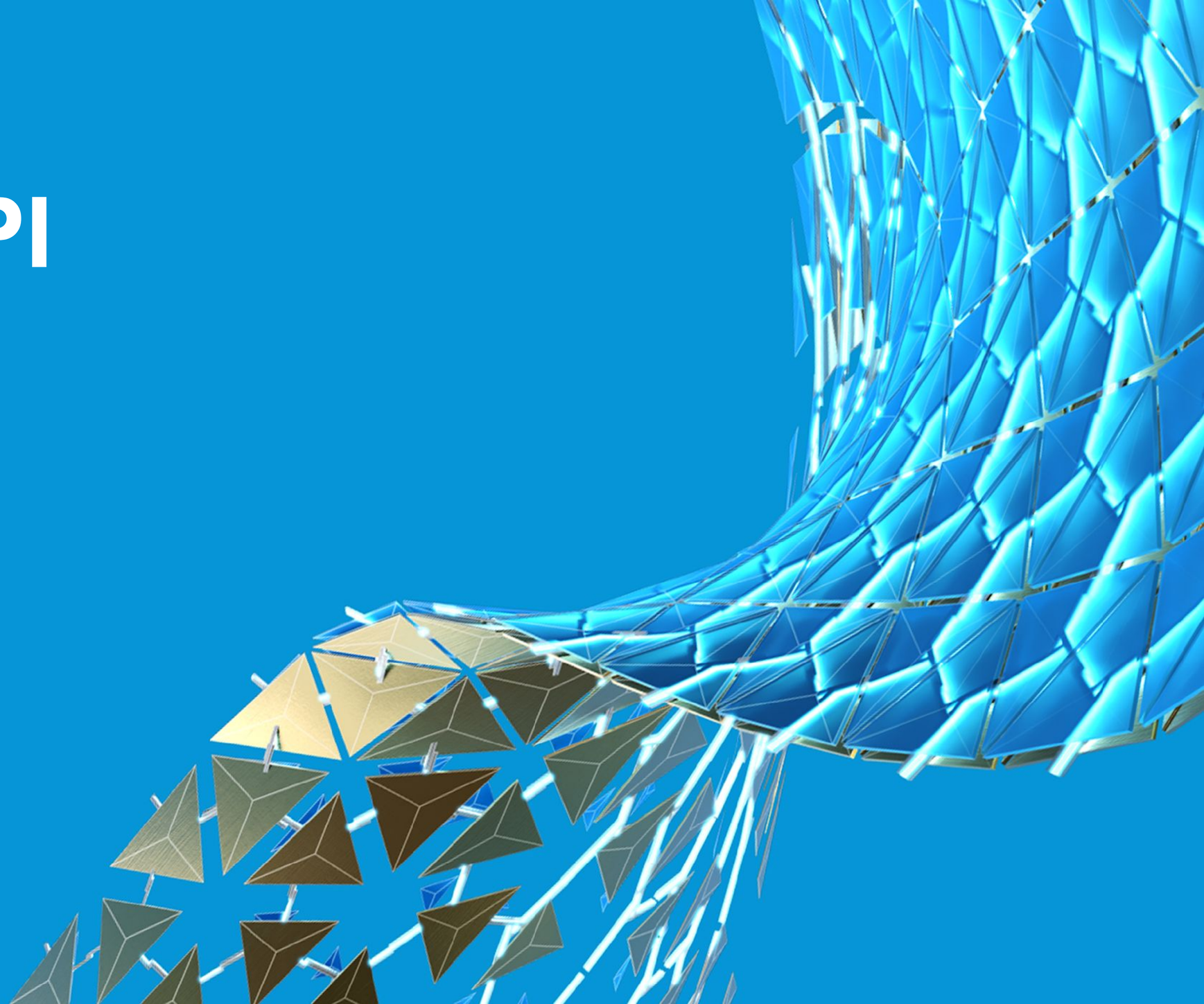
Fusion 360 API

- **Platform independent API supports OSX and Windows**
- **Designed to be program language independent, currently supports:**
  - Python
  - C++
- **Python is a widely used general-purpose, high-level programming language that is designed to be concise and human readable.**

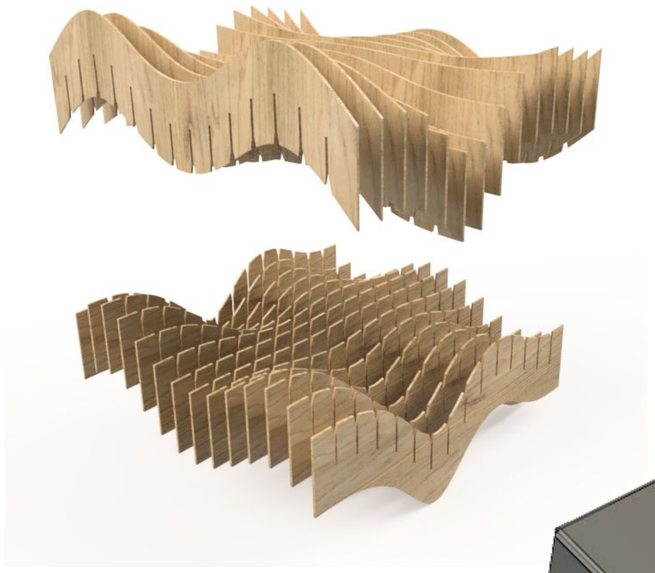
Primary Areas of the API

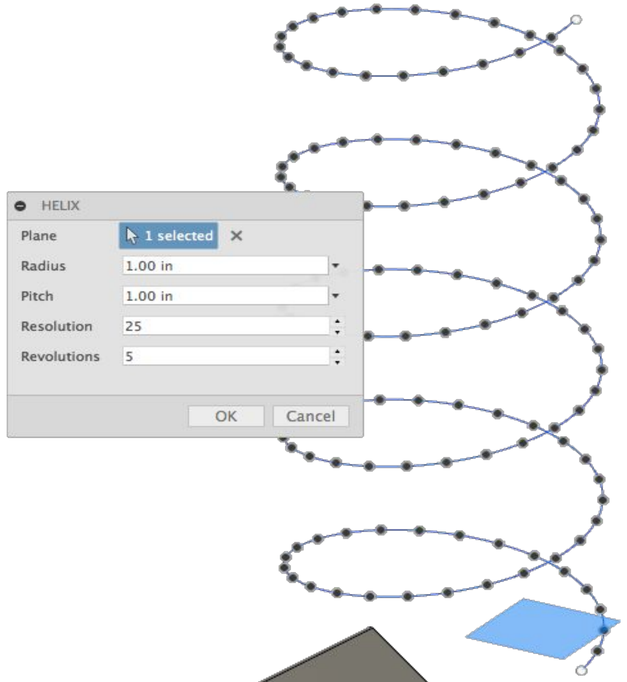
- **Design**
  - Automate creation and editing of solid and surface geometry
  - Interrogate and analyze geometry
- **CAM**
  - Interrogate basic CAM information
  - Automate post processing
- **Data**
  - Import/Export Data
  - Interrogate and manipulate Fusion 360 Data
- **Other Useful Concepts**
  - Custom Graphics
  - Palettes
  - Application Events
  - Attributes
  - Temporary BREP Manager

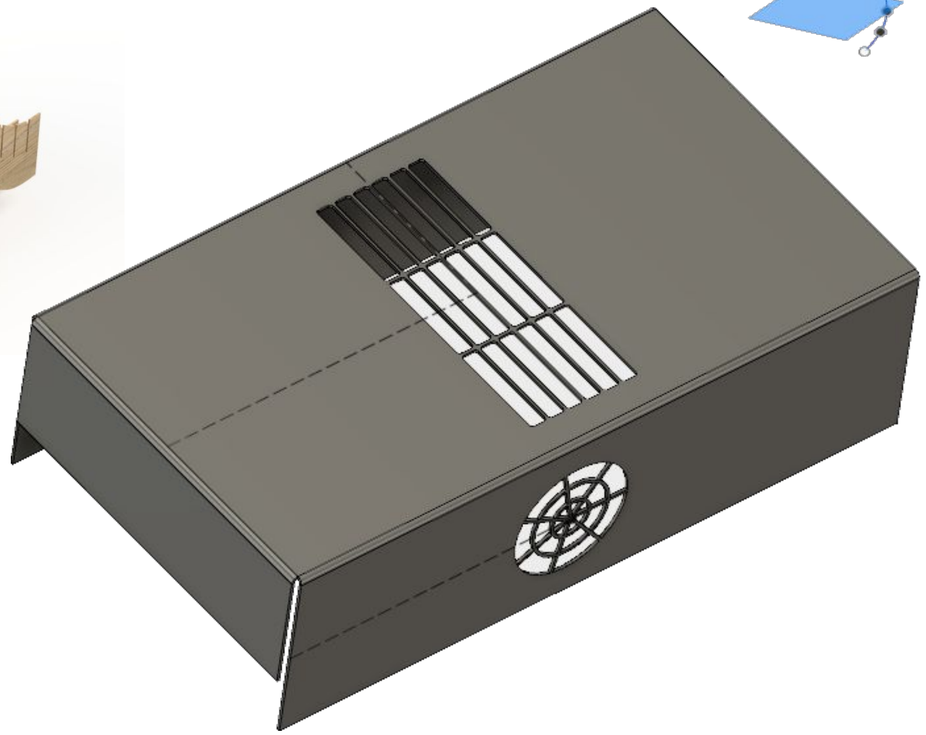
Design API



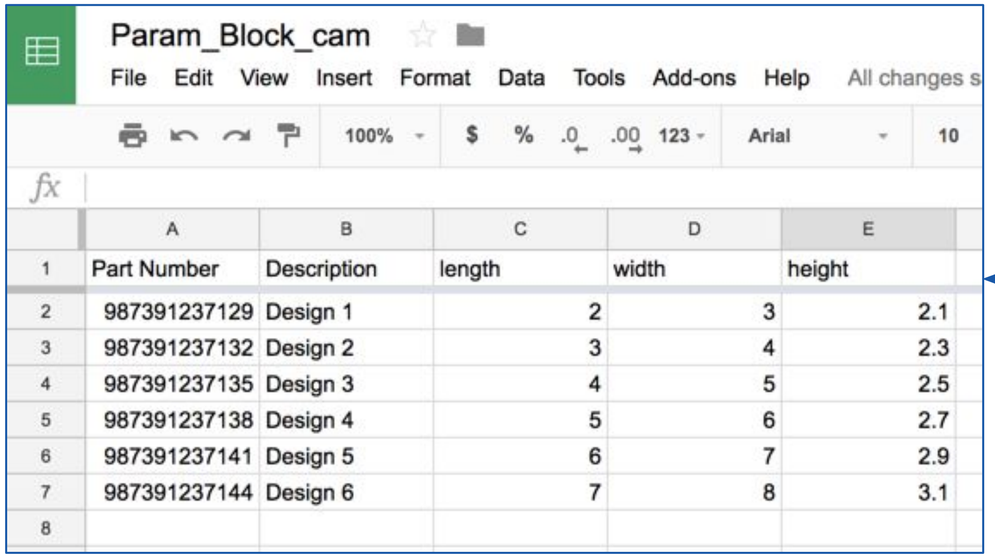
Automate Geometry Creation



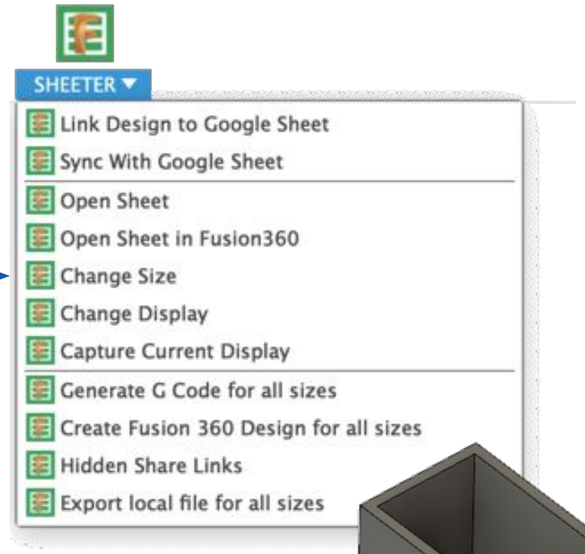


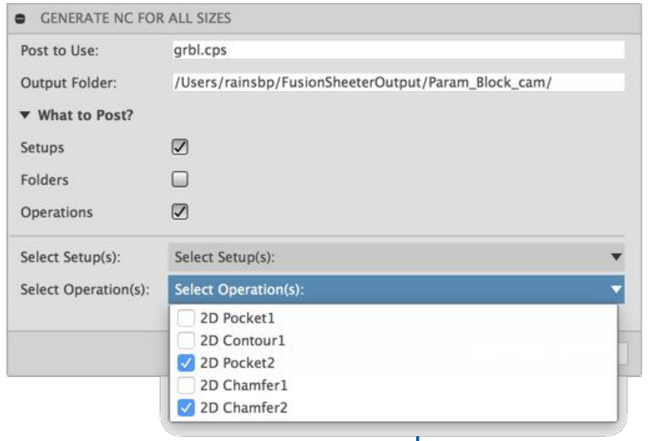


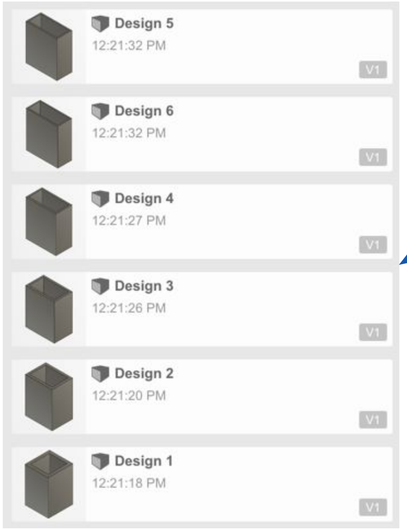
Automate Geometry Modification

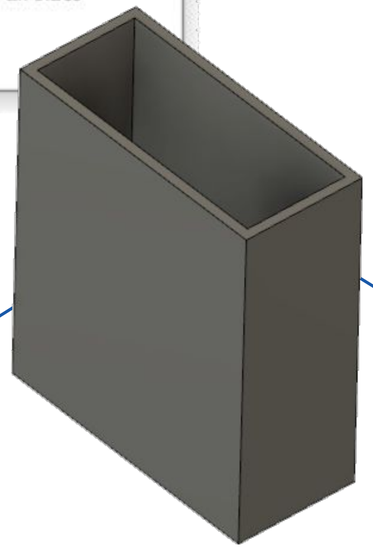


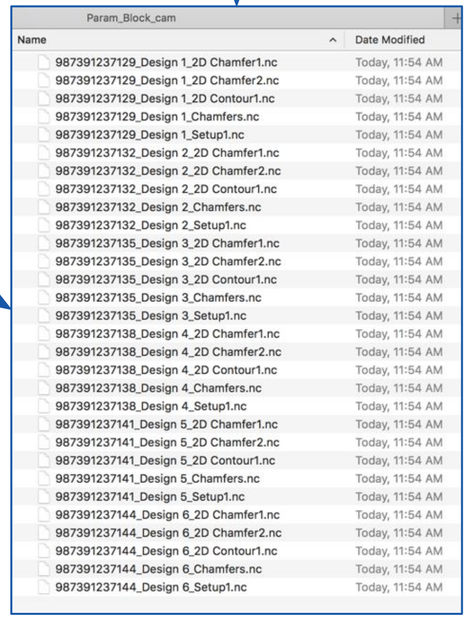
	A	B	C	D	E
1	Part Number	Description	length	width	height
2	987391237129	Design 1	2	3	2.1
3	987391237132	Design 2	3	4	2.3
4	987391237135	Design 3	4	5	2.5
5	987391237138	Design 4	5	6	2.7
6	987391237141	Design 5	6	7	2.9
7	987391237144	Design 6	7	8	3.1
8					







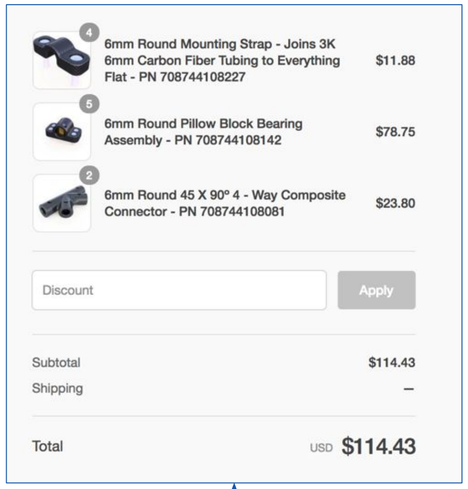


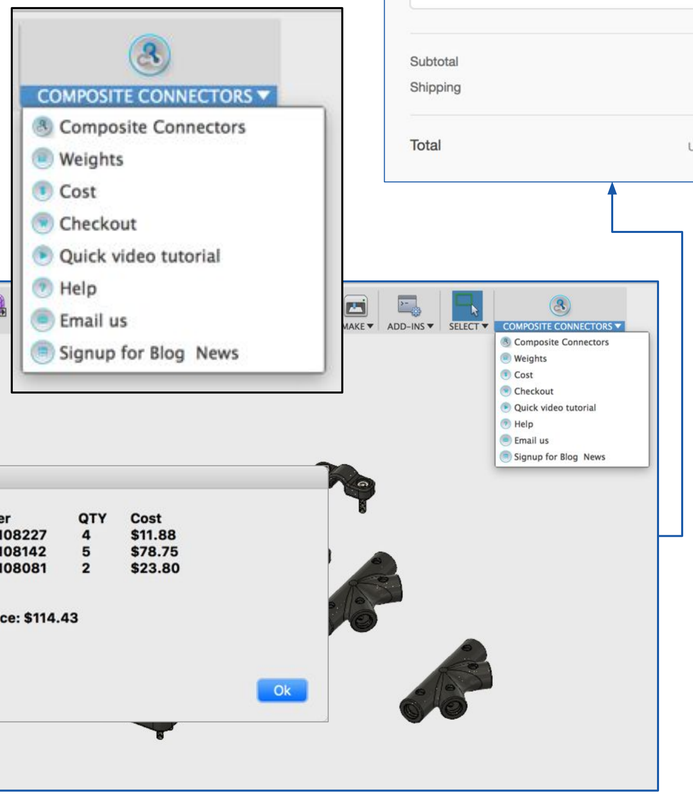


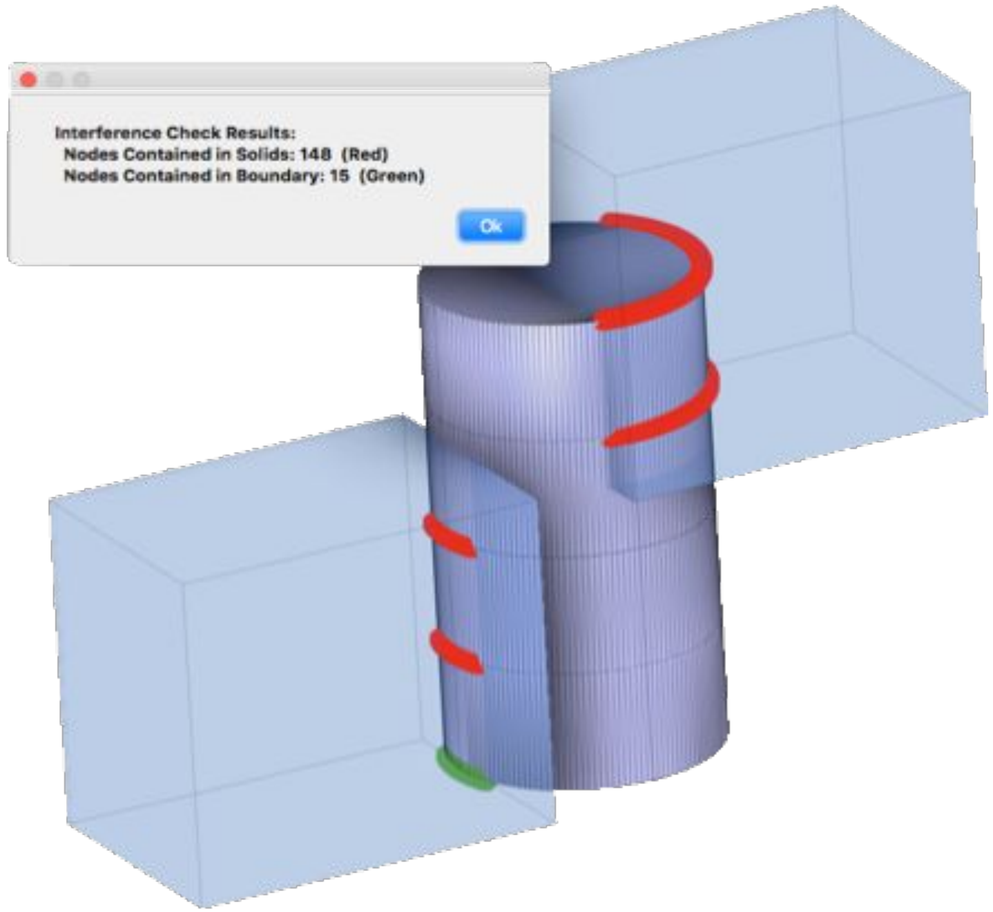
Google Sheets Integration

- Synchronize Parameters
- Export multiple sizes
- Post process multiple sizes
- Save display states

Interrogate and Analyze Geometry

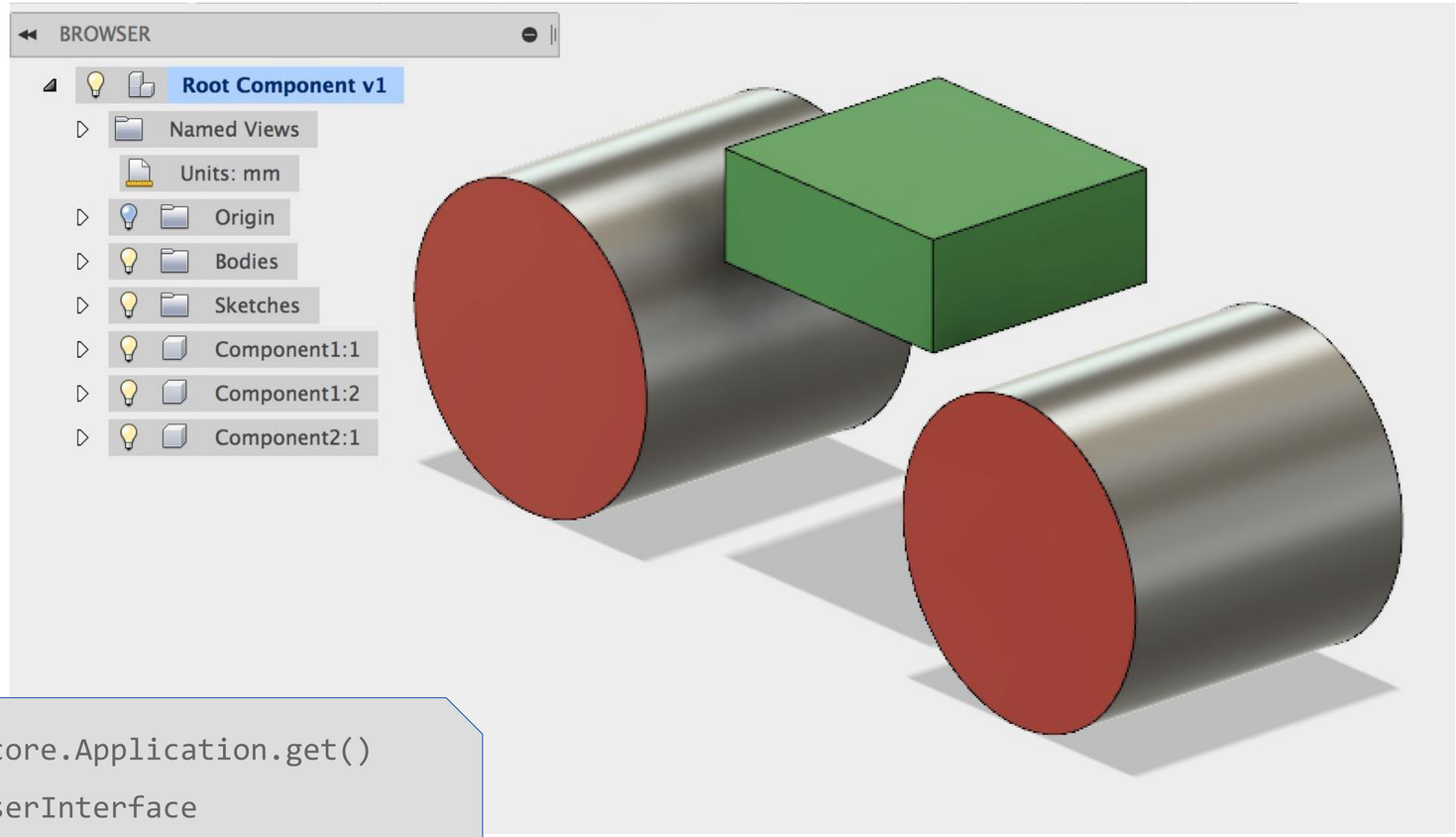






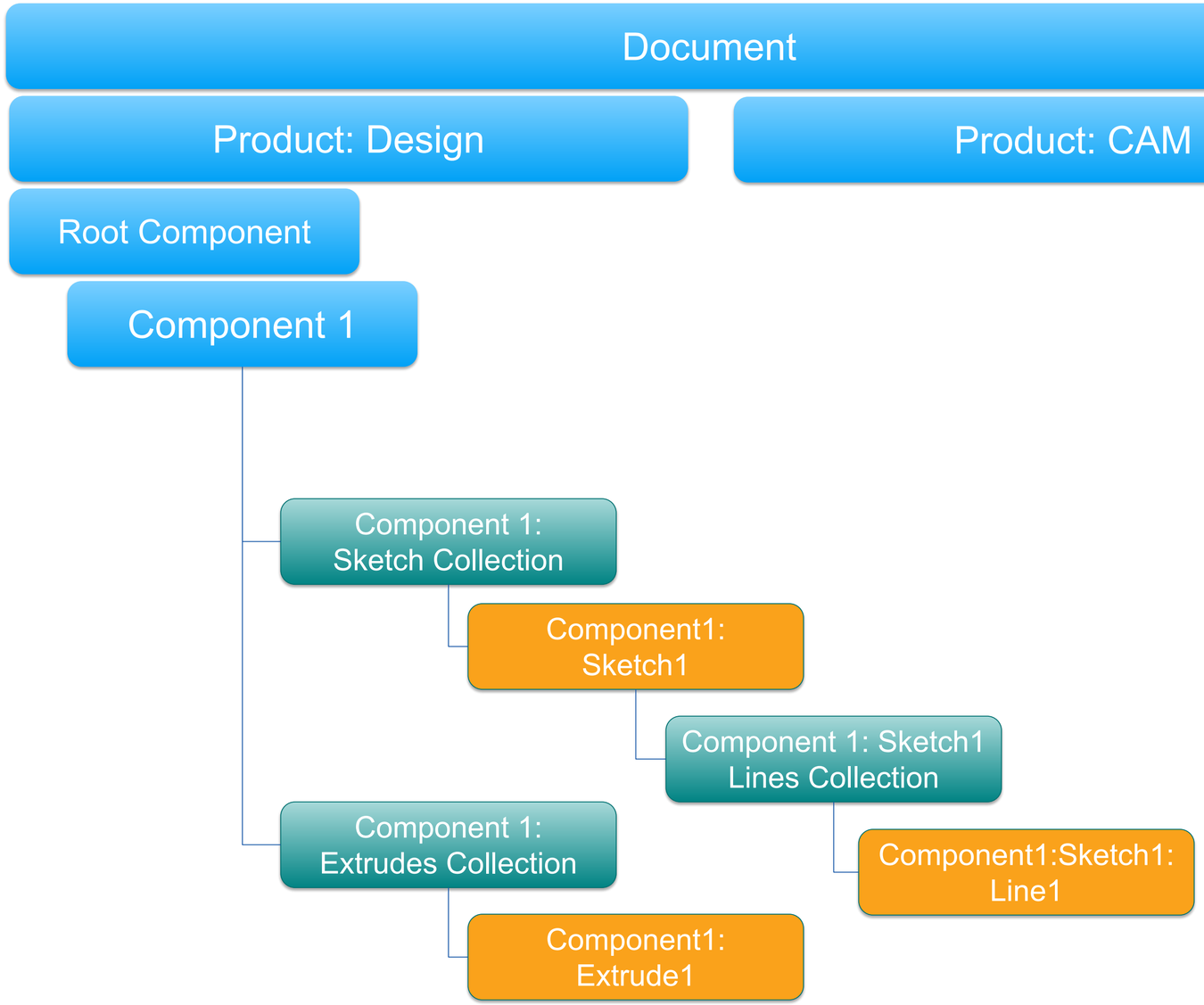
Fusion 360 Document Structure

- Document
  - Product: Design
    - Root Component
      - Component 1
        - Component 1: Occurrence 1
          - Component1:1 /RedFace (proxy)
        - Component 1: Occurrence 2
          - Component1:2 /RedFace (proxy)
      - Component2
        - Component 2: Occurrence 1
  - Product: CAM



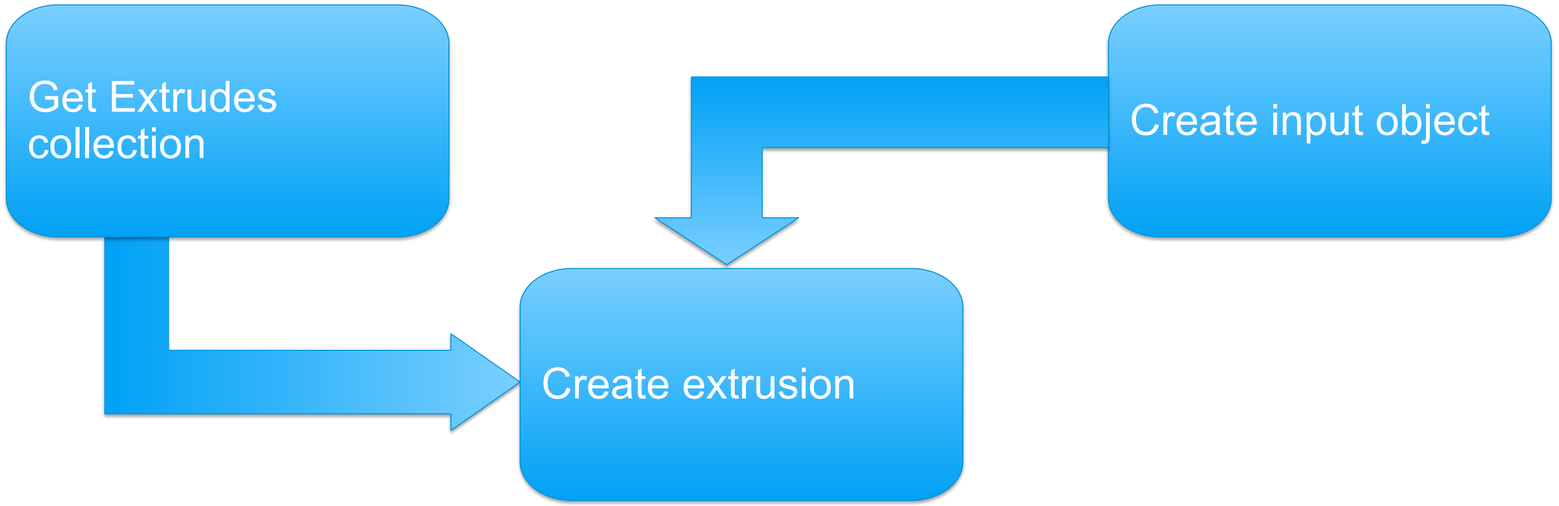

```
app = adsk.core.Application.get()
ui  = app.userInterface
design = app.activeProduct
```

Features and Collections

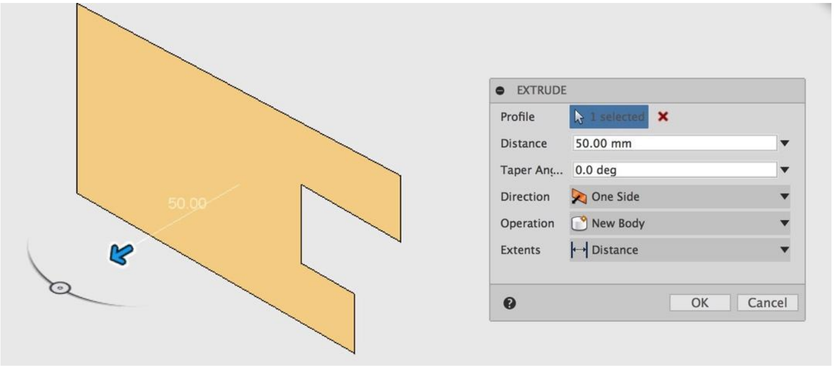



```
# Get reference to the root component
rootComp = design.rootComponent

#Get reference to the sketches and plane
sketches = rootComp.sketches

#Create a new sketch and get lines reference
sketch = sketches.add(rootComp.xYConstructionPlane)
lines = sketch.sketchCurves.sketchLines
lines.addByTwoPoints(point0, point1)
```

Creating Features (Extrude)



- Define parameters for feature
- Define profile(s) for feature
- Define distance




```
# Get the profile defined by the sketch
profile = sketch.profiles.item(0)

# Create an extrusion input
extrudes = rootComp.features.extrudeFeatures
operation_type = adsk.fusion.FeatureOperations.NewBodyFeatureOperation
ext_input = extrudes.createInput(profile, operation_type)
```

Units in Fusion 360

Fusion Default Model Units

cm (*areas and volumes are cm^2 and cm^3*)

radians

kg

```
# Define that the extent is a distance extent of 1 cm
distance = adsk.core.ValueInput.createByReal(1)

# Set the distance extent to be single direction
ext_input.setDistanceExtent(False, distance)

# Set the extrude to be a solid one
ext_input.isSolid = True

# Create the extrusion
extrudes.add(ext_input)
```

Active units and feature definitions

Scripts must adapt to user changing units

Most features look for “Value Inputs” not raw values

var x = adsk.core.ValueInput.createByReal(23)

var x = adsk.core.ValueInput.createByString(“23 in”);

UnitsManager is a utility for values and units.

convert(1.5, "in", "ft") -> 0.125

evaluateExpression(“3 in * 5 in”, “in”) -> 38.1

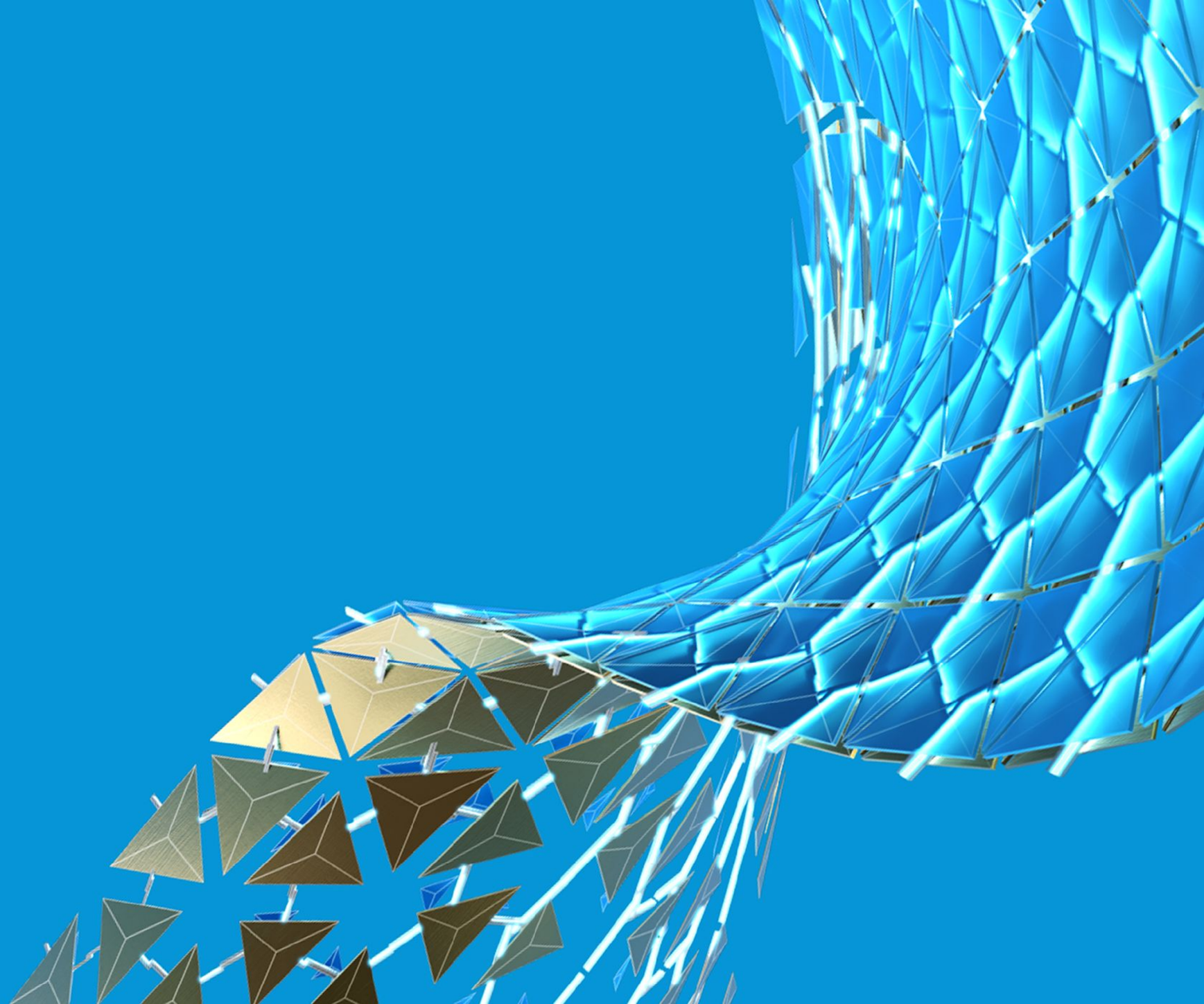
formatInternalValue(2000, “ft*ft*ft“, true) -> "0.070629 ft^3"

standardizeExpression("1.5", "in") -> "1.5 in"

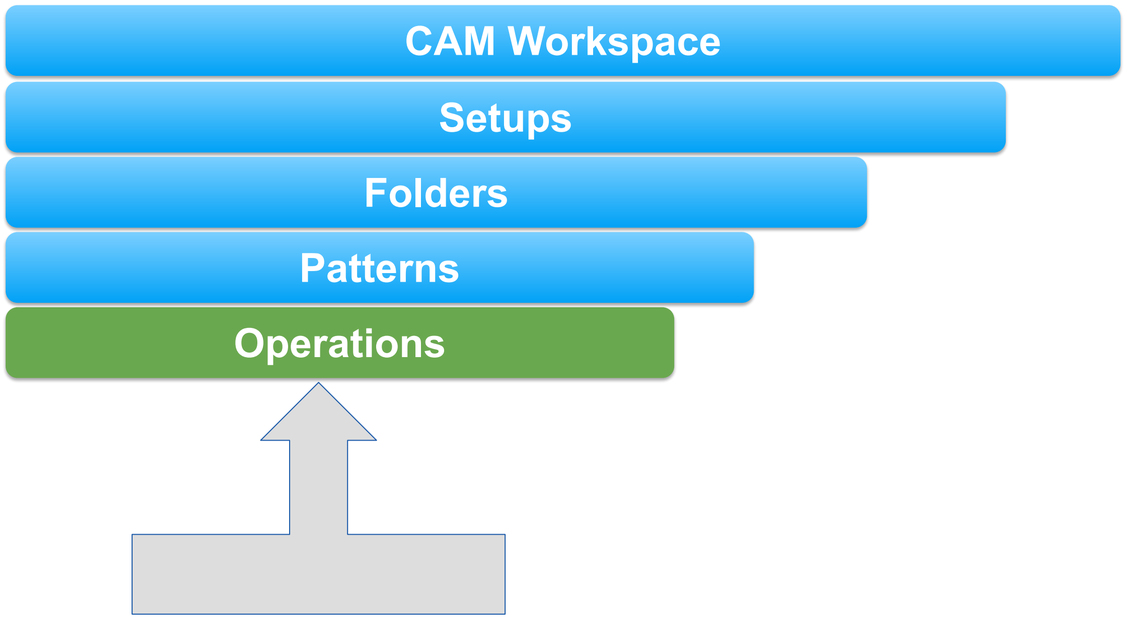
Full Script

```python
# Author-Patrick Rainsberry
# Description-Basic Script to create a block

import adsk.core, adsk.fusion, adsk.cam, traceback

def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
        design = app.activeProduct

        # Get reference to the root component
        rootComp = design.rootComponent

        #Get reference to the sketchs and plane
        sketches = rootComp.sketches
        xyPlane = rootComp.xYConstructionPlane

        #Create a new sketch and get lines reference
        sketch = sketches.add(xyPlane)
        lines = sketch.sketchCurves.sketchLines

        # Use autodesk methods to create input geometry
        point0 = adsk.core.Point3D.create(0, 0, 0)
        point1 = adsk.core.Point3D.create(0, 1, 0)
        point2 = adsk.core.Point3D.create(1, 1, 0)
        point3 = adsk.core.Point3D.create(1, 0, 0)

        # Create lines
        lines.addByTwoPoints(point0, point1)
        lines.addByTwoPoints(point1, point2)
        lines.addByTwoPoints(point2, point3)
        lines.addByTwoPoints(point3, point0)

        # Get the profile defined by the square
        profile = sketch.profiles.item(0)

        # Create an extrusion input
        extrudes = rootComp.features.extrudeFeatures
        operation_type = adsk.fusion.FeatureOperations.NewBodyFeatureOperation
        ext_input = extrudes.createInput(profile, operation_type)

        # Define that the extent is a distance extent of 1 cm
        distance = adsk.core.ValueInput.createByReal(1)

        # Set the distance extent to be single direction
        ext_input.setDistanceExtent(False, distance)

        # Set the extrude to be a solid one
        ext_input.isSolid = True

        # Create the extrusion
        extrudes.add(ext_input)

    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

CAM API



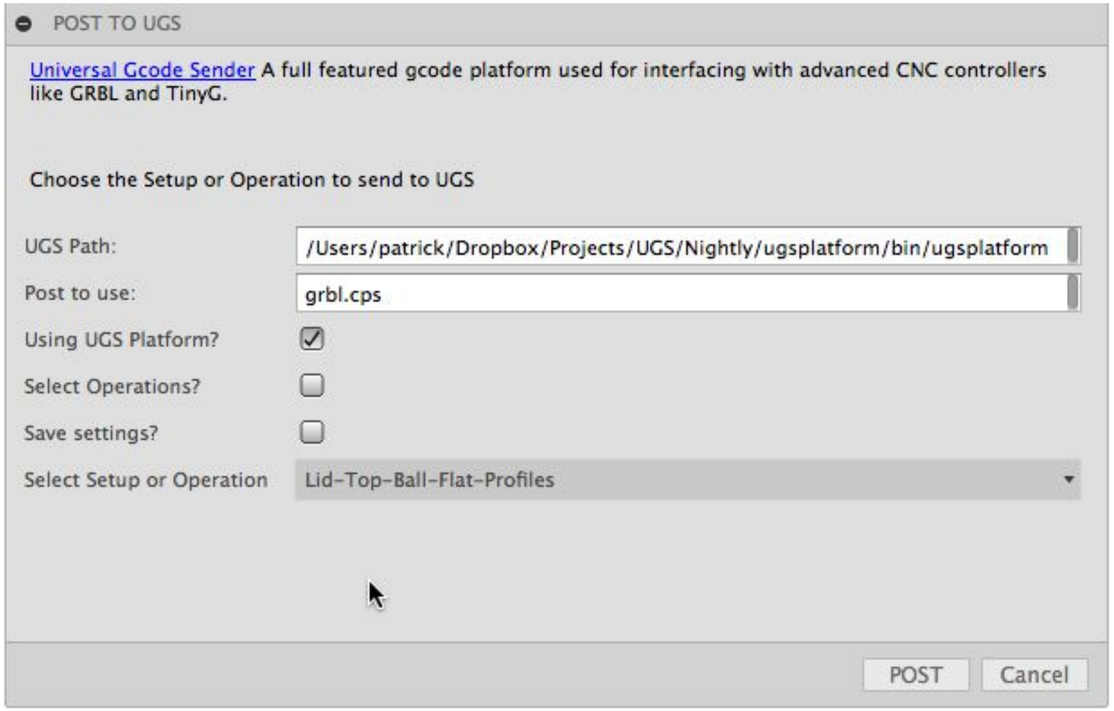
Interrogate Basic CAM Information



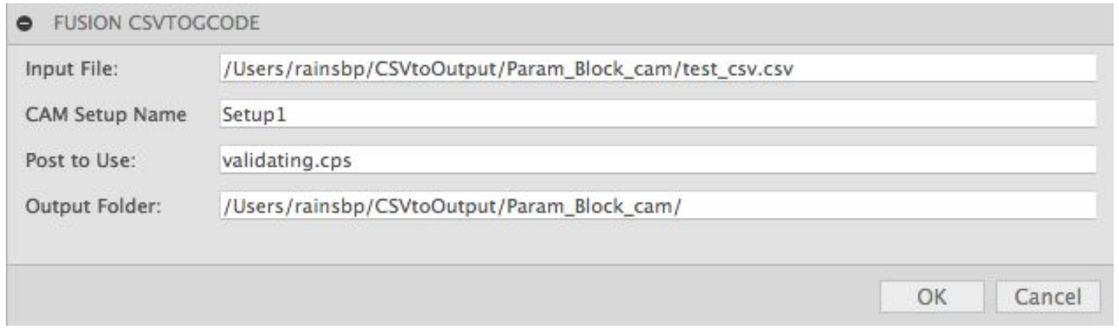
Create from Template
View Properties
Edit Properties

Regenerate Toolpaths
Create Setup Sheets
Post Process one or more items
Get machining time for 1 or more items

Automate Post Processing



Post process and automatically send to controller software

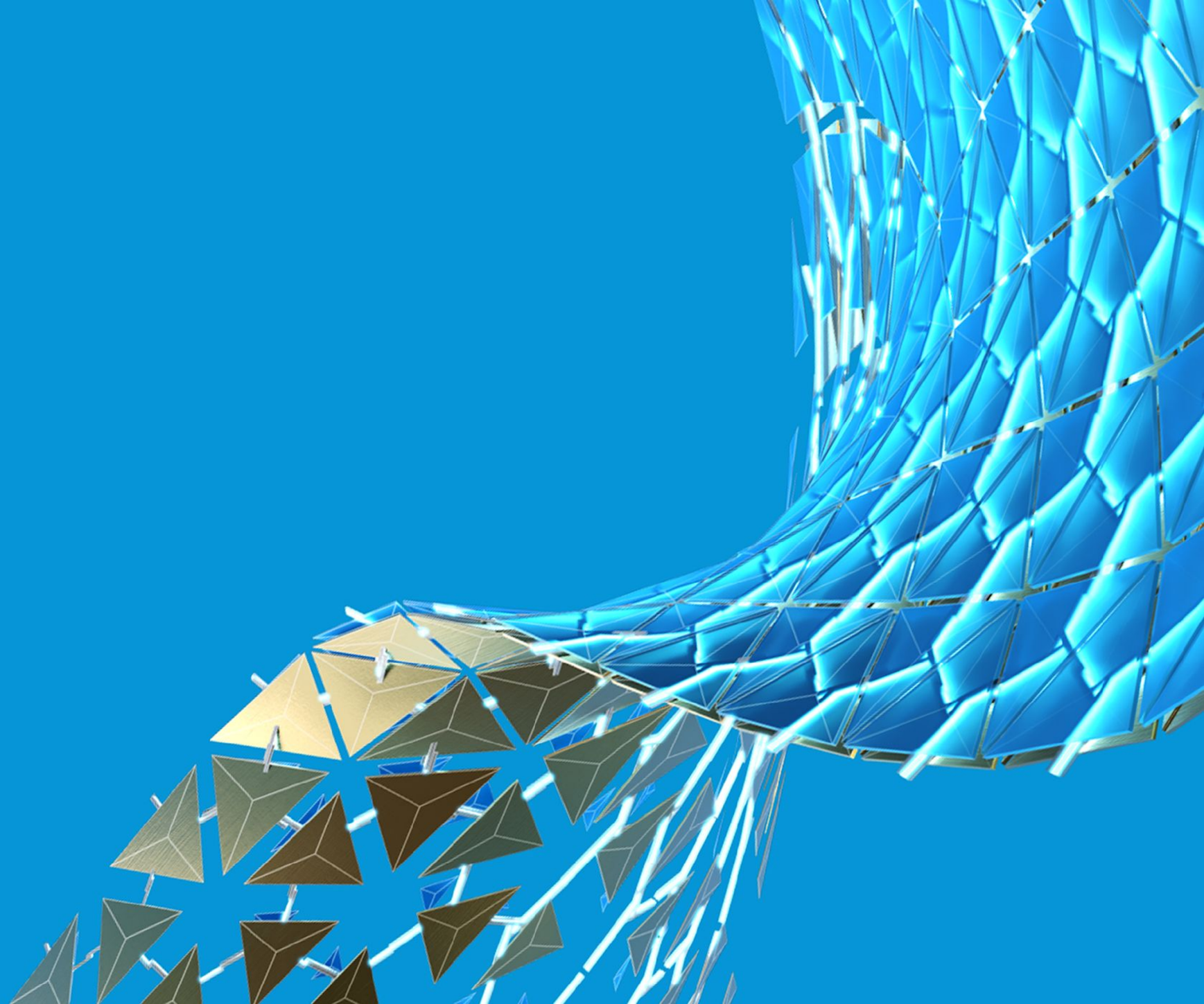


Read parameters, post process, for every row in a csv file

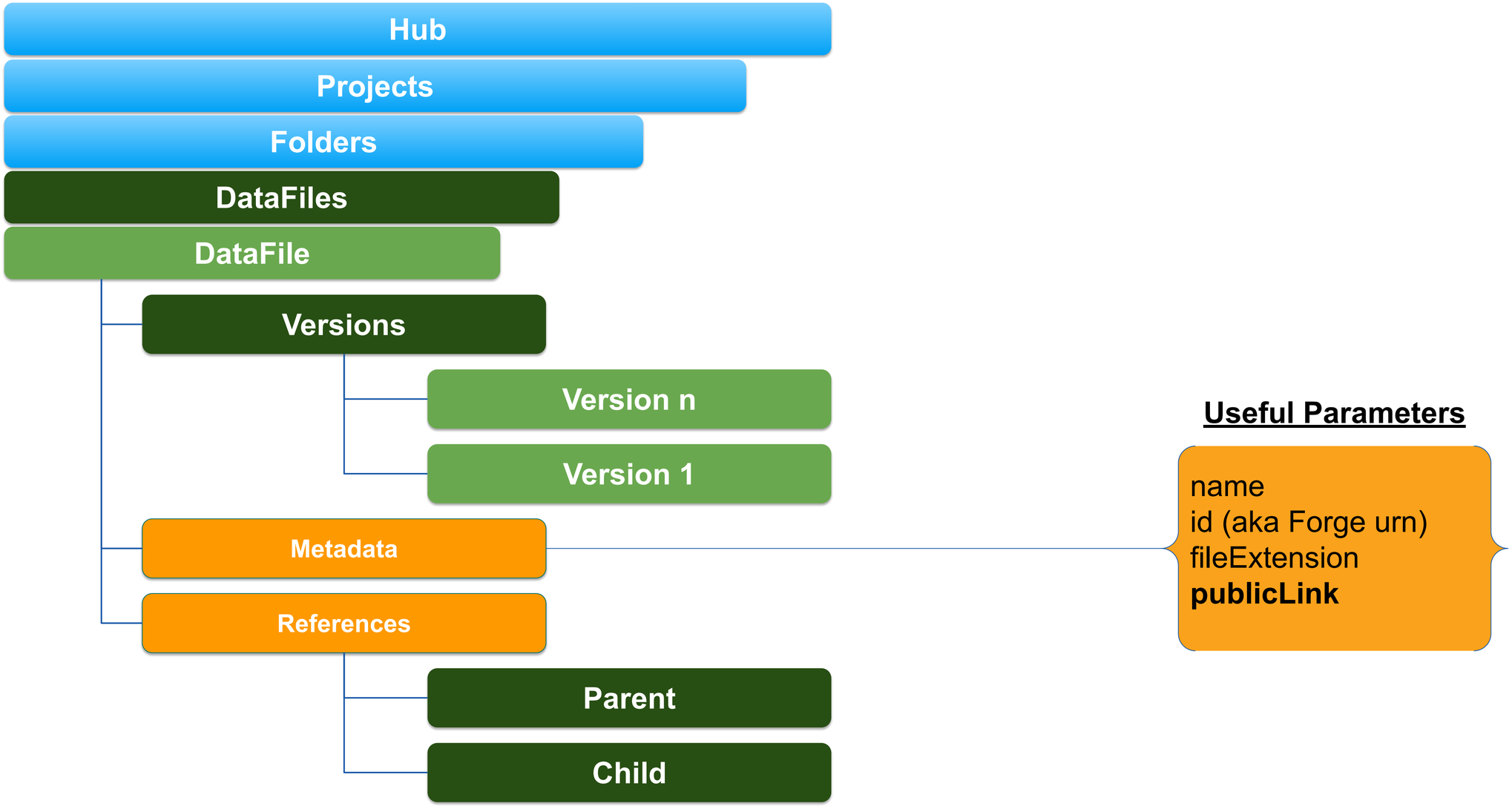
CAM API Sample

```
import time
TIMEOUT = 10

# Find setup
for setup in cam.setups:
  if setup.name == setup_name:
      to_post = setup

      # Update tool path
      future = cam.generateToolpath(to_post)

      check = 0
      while not future.isGenerationCompleted:
          adsk.doEvents()
          time.sleep(1)
          check += 1
          if check > TIMEOUT:
              ao['ui'].messageBox('Timeout')
              break

      # Set the post options
      post_config = os.path.join(cam.genericPostFolder, post_name)
      units = adsk.cam.PostOutputUnitOptions.DocumentUnitsOutput

      # create the postInput object
      post_input = adsk.cam.PostProcessInput.create(setup_name, post_config, output_folder, units)
      post_input.isOpenInEditor = False

      cam.postProcess(to_post, post_input)
```

Data API

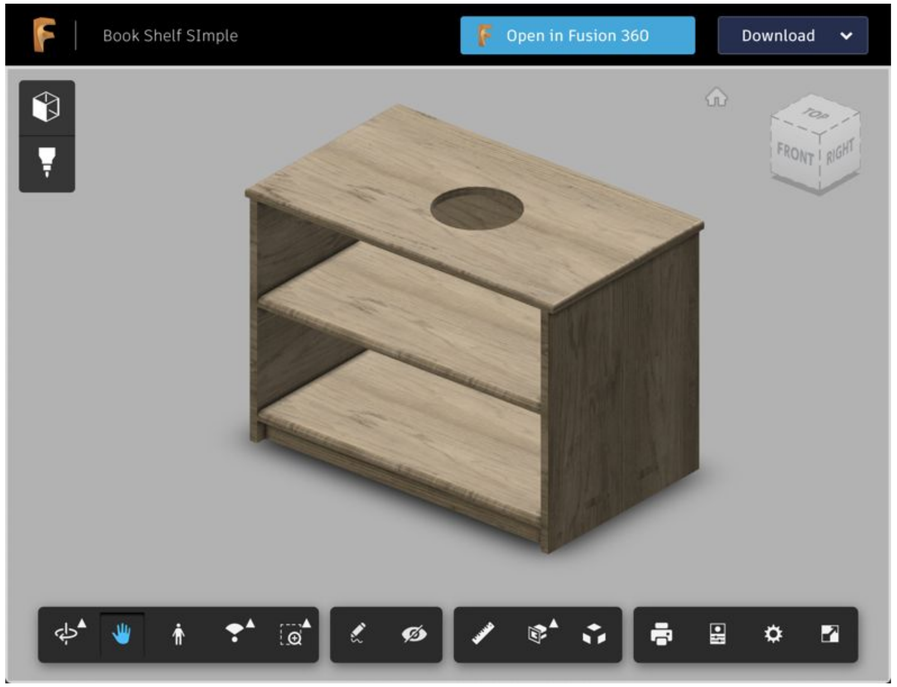


Interrogate and Manipulate Fusion 360 Data



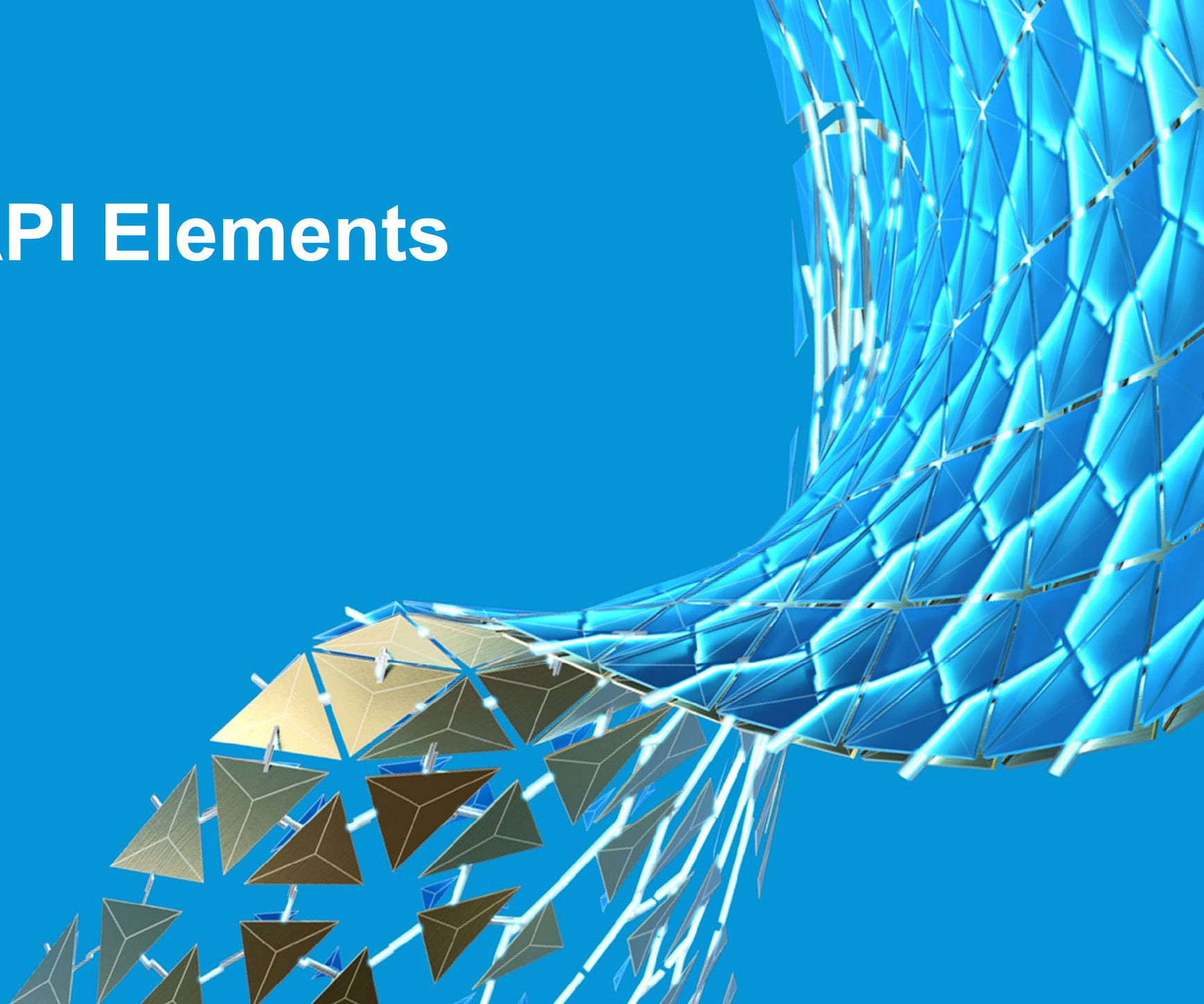
Data API Example

```
for data_file in app.activeDocument.dataFile.parentFolder.dataFiles:
    if data_file.fileExtension == "f3d":
        if test_name == data_file.name:
            short_public_link = data_file.publicLink
            public_link = un_shorten_url(short_public_link)
            custom_properties = {
                "short_public_link": short_public_link,
                "public_link": public_link,
                "public_link_id": public_link.split("/")[-1],
                "forge_urn": data_file.id,
                "forge_id": data_file.id.split(":")[-1]
            }
```



https://a360.co/2I3ANla
https://autodesk3008.autodesk360.com/g/shares/SH56a43QTfd62c1cd9683a5210ff3637dbfa
SH56a43QTfd62c1cd9683a5210ff3637dbfa
urn:adsk.wipprod:dm.lineage:L-WnvzX-QiindaIZkCIelss
L-WnvzX-QiindaIZkCIelss

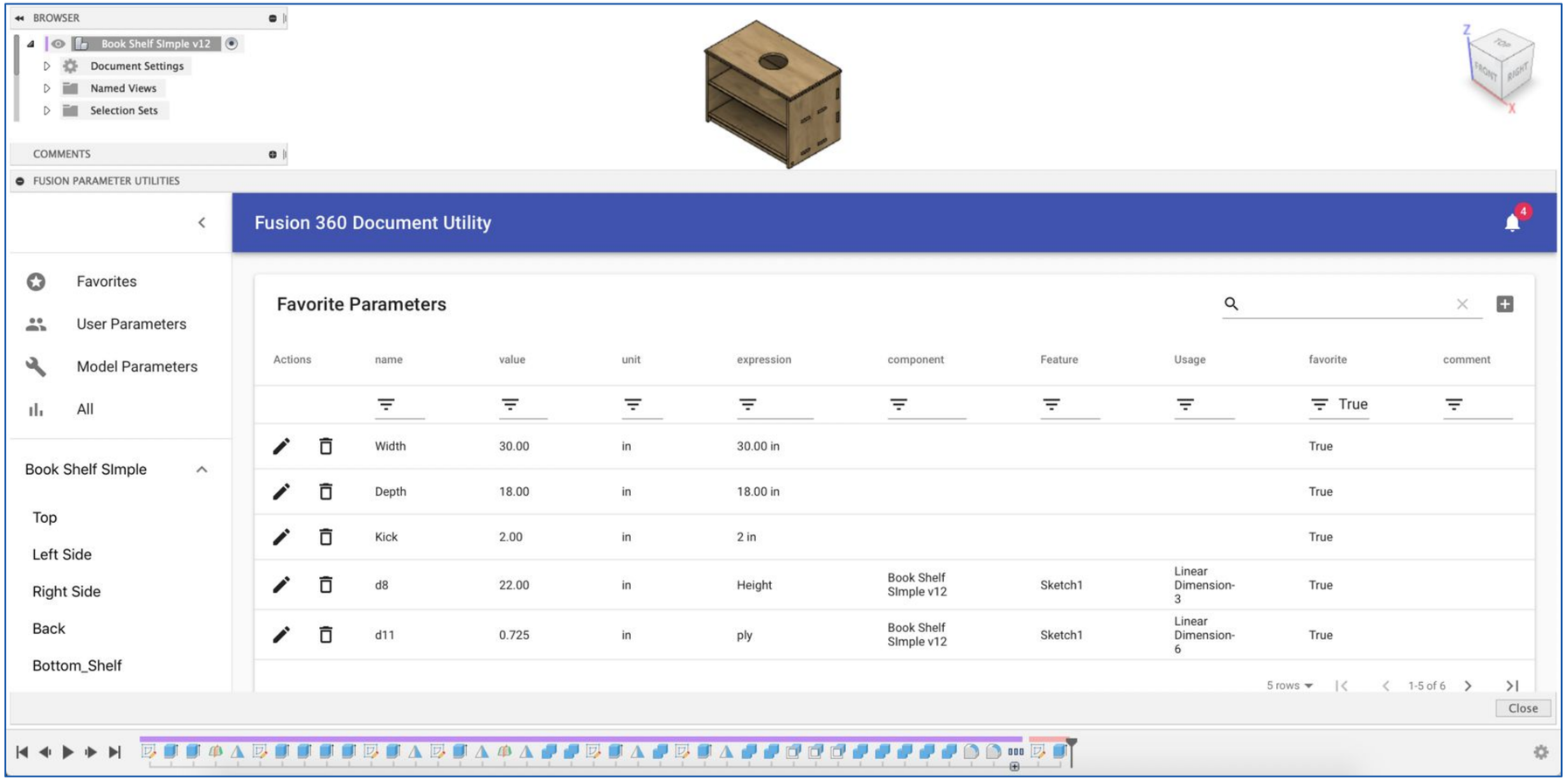
Additional API Elements

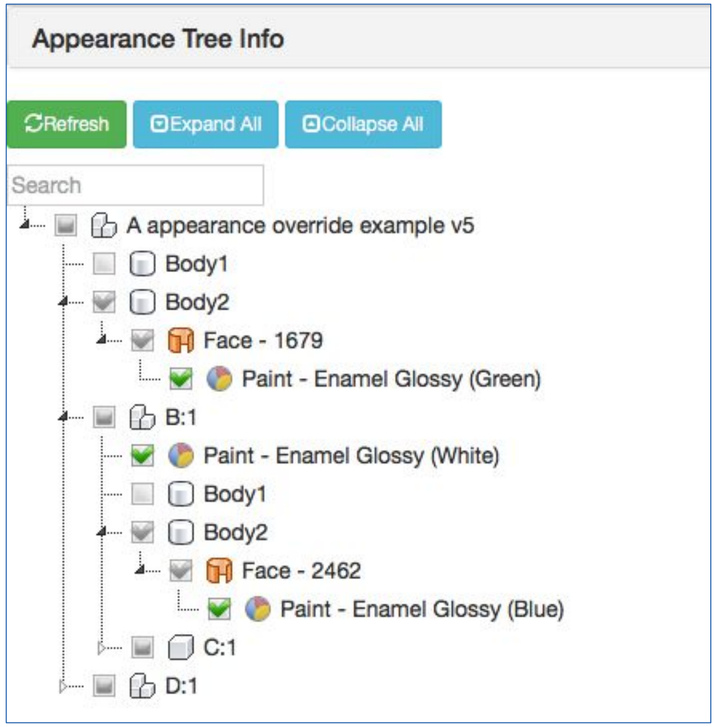


Palettes

Loads an html page in a frame in Fusion 360

Can send and receive information from the page.





<u>Leverage client side libraries:</u>

- jquery + jstree (above)
- react + material-ui + material-table (left)

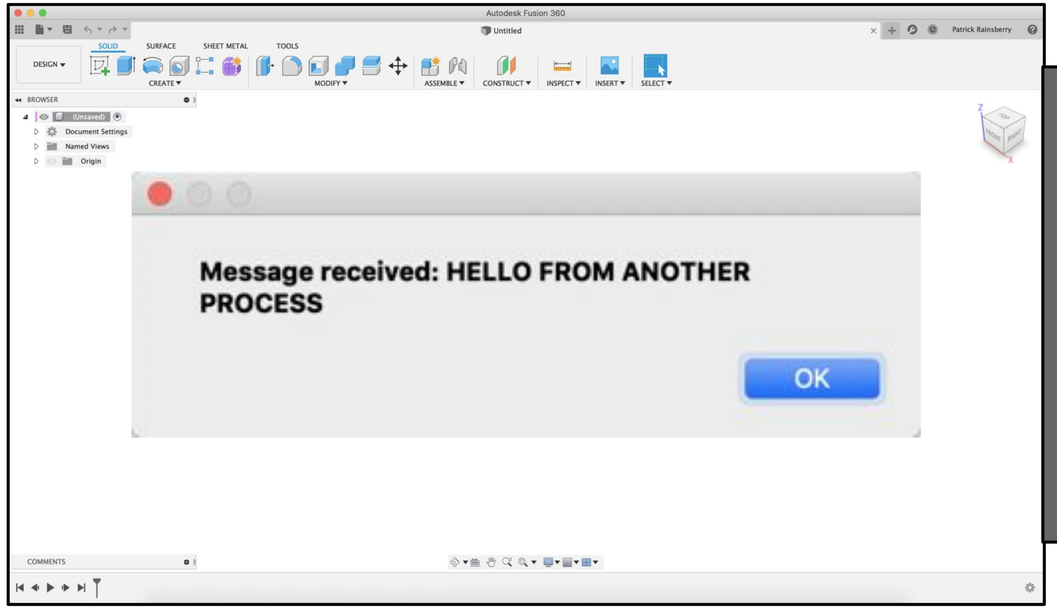
<u>Connect Directly to a web server:</u>

- Insert components from a catalog
- Synchronize data

Custom Events

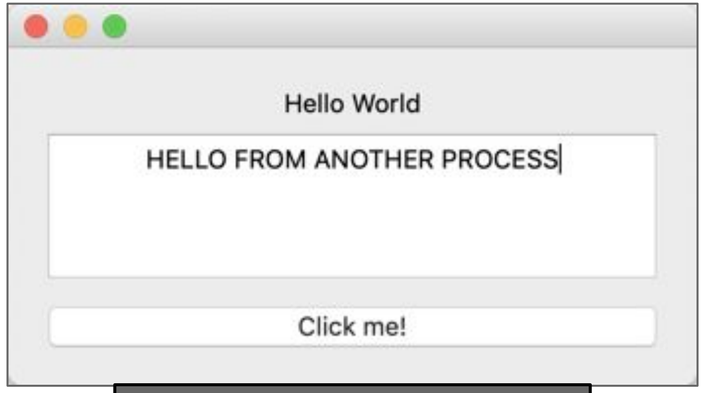
Custom Events:

- **Register a custom event and then**
- **Execute some function whenever it is called**
- **Particularly useful for running a separate thread**
- **Use to communicate with other apps, or handle data queues**



Custom Event Listener

Some other process



Connection Listener

Connection Client

<u>Thread spawned from Add-in</u>

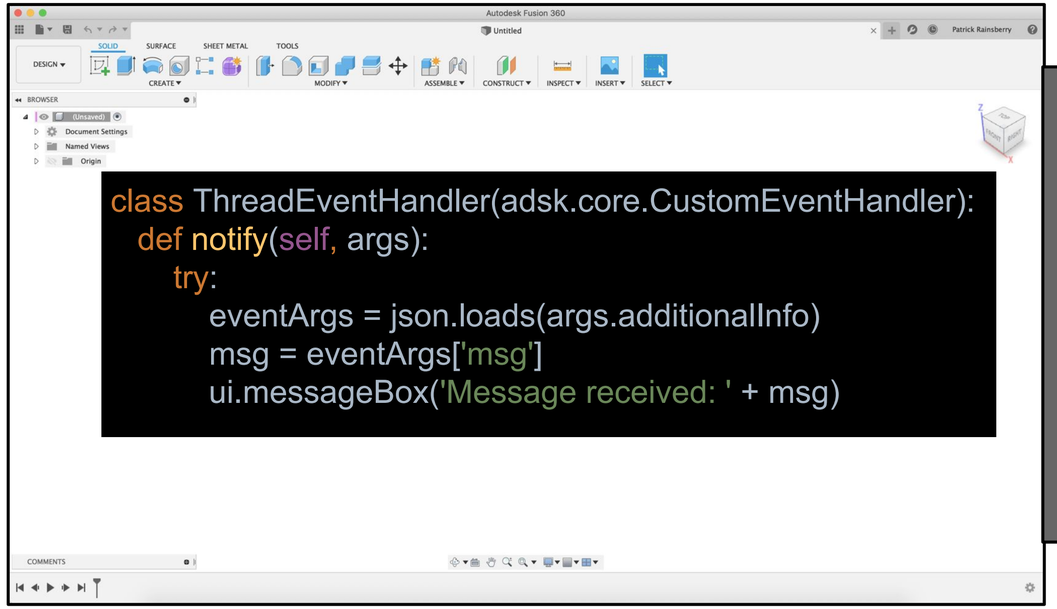
- Opens connection to other process
- Receives message from process
- Fires “Custom Event”

```
app.fireCustomEvent(myCustomEvent, json.dumps(args))
```

Custom Events

Custom Events:

- **Register a custom event and then**
- **Execute some function whenever it is called**
- **Particularly useful for running a separate thread**
- **Use to communicate with other apps, or handle data queues**




```
class ThreadEventHandler(adsk.core.CustomEventHandler):
    def notify(self, args):
        try:
            eventArgs = json.loads(args.additionalInfo)
            msg = eventArgs['msg']
            ui.messageBox('Message received: ' + msg)
```

Custom Event Listener

Some other process

```
from multiprocessing.connection import Listener
address = ('localhost', 6000)
with Listener(address, authkey=b'secret password') as listener:
    with listener.accept() as conn:
        widget.conn = conn


# Elsewhere in application:
self.button.clicked.connect(conn.send([test, False]))
```

Connection Listener

Connection Client

```
from multiprocessing.connection import Client
address = ('localhost', 6000)
with Client(address, authkey=b'secret password') as conn:
    while not self.stopped.wait(5):
        msg = conn.recv()[0]
        args = {'msg': msg}
        app.fireCustomEvent(myCustomEvent, json.dumps(args))
```

Attributes

- **Attributes can be assigned to nearly any object in a Fusion 360 document (including the document itself)**
- **Attributes are a string value, but a VERY useful practice is to store a json string as the attribute:**
  - *json.dumps(some_python_dictionary)*
- **Objects can retrieved by searching for a particular attribute value or group in a document**
- **Another VERY useful technique is to assign a unique id to any object (geometry, etc.) that you want to maintain a reference to.**

`Fusion360Utilities.item_id`*(item, group_name)*

```
items_to_remember = []
for item in object_collection_of_interesting_fusion_objects:
  items_to_remember.append(item_id(item, "MyAppName"))
document_settings = {"items_to_remember": items_to_remember}
document.attributes.add("MyAppName", "settings", json.dumps(document_settings))
```

`Fusion360Utilities.get_item_by_id`*(this_item_id, app_name)*

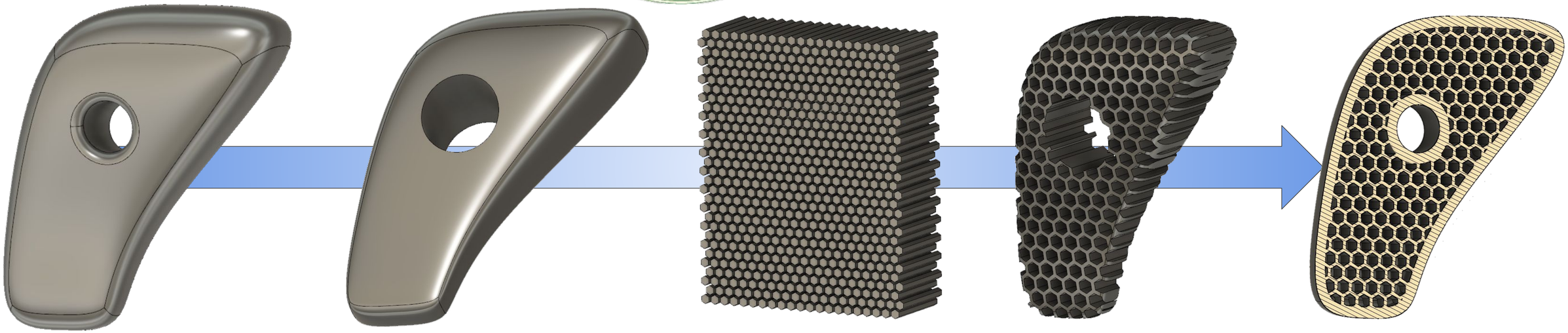
```
settings_attribute = document.attributes.itemByName("MyAppName", "settings")
settings = json.loads(settings_attribute.value)
remembered_items = []
for object_id in settings["items_to_remember"]:
  remembered_items.append(get_item_by_id(object_id, "MyAppName"))
```

Temporary BREP Manager

adsk.fusion.TemporaryBRepManager.get()

- Call functions directly into asm
- Geometry is “transient”
- Finally add geometry to a base feature
- EXTREMELY FAST

3,500 sketch curves ⇒
reduced to ~1,500 edges

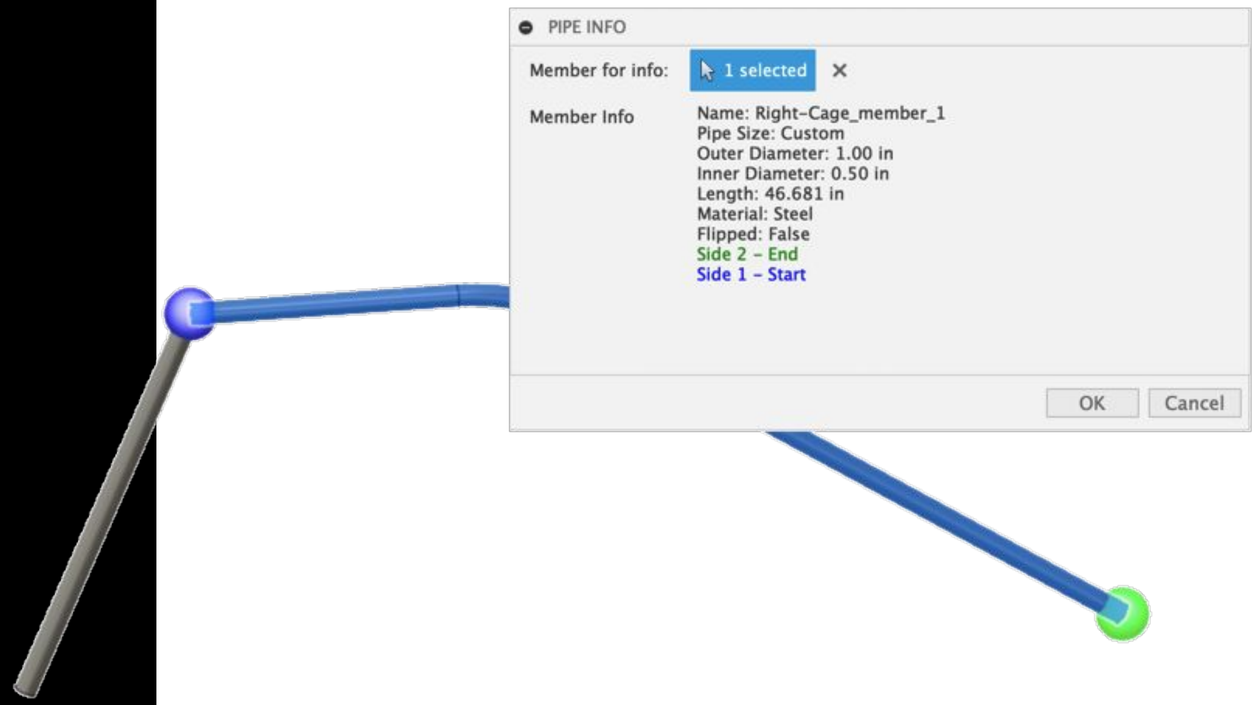


Custom Graphics

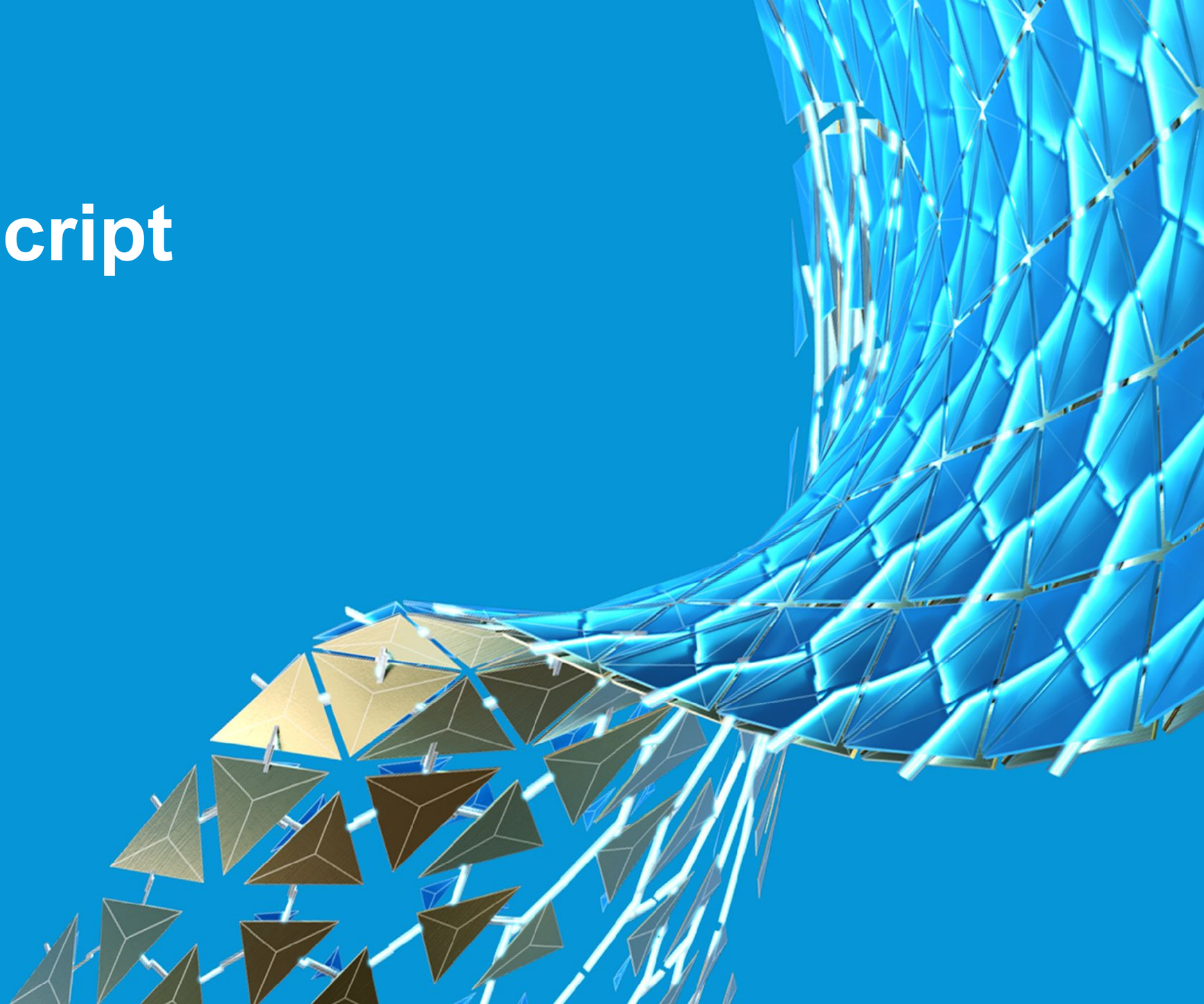
Works very well in conjunction with

temporaryBrepManager

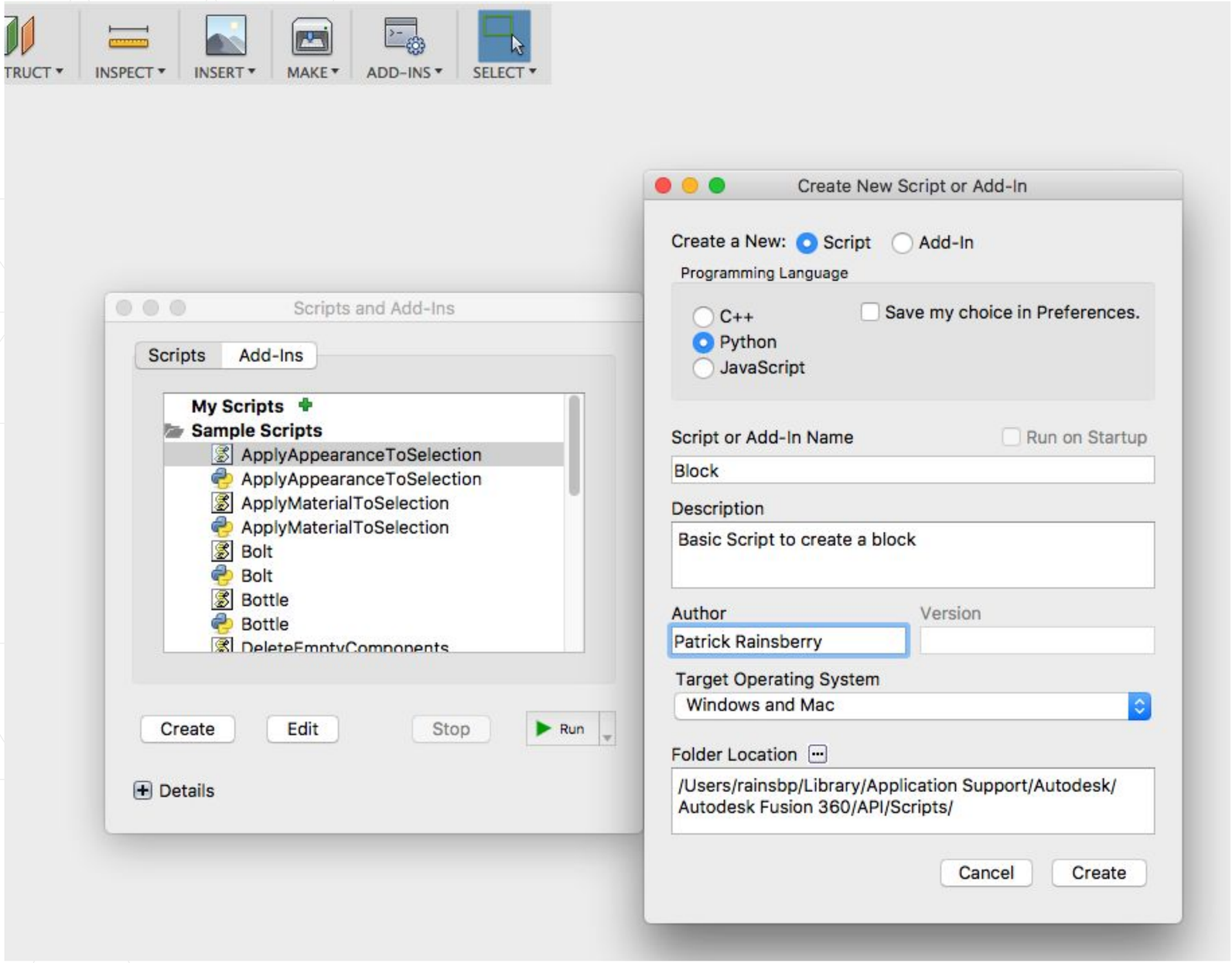
```
COLORS = {
  "blue": adsk.fusion.CustomGraphicsBasicMaterialColorEffect.create(
      adsk.core.Color.create(10, 10, 245, 255)
  ),
  "green": adsk.fusion.CustomGraphicsBasicMaterialColorEffect.create(
      adsk.core.Color.create(10, 245, 10, 255)
  )}

def create_graphics(center_point, color):
  ao = AppObjects()
  graphics_group = ao.root_comp.customGraphicsGroups.add()

  temp_brep_mgr = adsk.fusion.TemporaryBRepManager.get()
  sphere = temp_brep_mgr.createSphere(center_point, 3.0)
  sphere_graphic = graphics_group.addBRepBody(sphere)

  transform = adsk.core.Matrix3D.create()
  transform.translation = center_point.asVector()

  sphere_graphic.transform = transform
  sphere_graphic.color = COLORS[color]
```



Creating a Script



Create a New Script



Initial Script

```
#Author-Patrick Rainsberry
#Description-Basic Script to create a block

import adsk.core, adsk.fusion, adsk.cam, traceback

def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
        ui.messageBox('Hello script')

    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Our Script - Reference design object

```
#Author-Patrick Rainsberry
#Description-Basic Script to create a block

import adsk.core, adsk.fusion, adsk.cam, traceback
def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
```

All our code goes here

```
    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Our Script - Reference design object

```
#Author-Patrick Rainsberry
#Description-Basic Script to create a block

import adsk.core, adsk.fusion, adsk.cam, traceback
def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
        design = app.activeProduct



    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Get document components and create a sketch

```
# Get reference to the root component
rootComp = design.rootComponent

#Get reference to the sketches and plane
sketches = rootComp.sketches
xyPlane = rootComp.xYConstructionPlane

#Create a new sketch and get lines reference
sketch = sketches.add(xyPlane)
lines = sketch.sketchCurves.sketchLines
```

Create Points and Lines

```
# Use autodesk methods to create input geometry
point0 = adsk.core.Point3D.create(0, 0, 0)
point1 = adsk.core.Point3D.create(0, 1, 0)
point2 = adsk.core.Point3D.create(1, 1, 0)
point3 = adsk.core.Point3D.create(1, 0, 0)
```

Create Points and Lines

```
# Use autodesk methods to create input geometry
point0 = adsk.core.Point3D.create(0, 0, 0)
point1 = adsk.core.Point3D.create(0, 1, 0)
point2 = adsk.core.Point3D.create(1, 1, 0)
point3 = adsk.core.Point3D.create(1, 0, 0)

# Create lines
lines.addByTwoPoints(point0, point1)
lines.addByTwoPoints(point1, point2)
lines.addByTwoPoints(point2, point3)
lines.addByTwoPoints(point3, point0)
```

Create Extrusion Input

```
# Get the profile defined by the circle
profile = sketch.profiles.item(0)

# Create an extrusion input
extrudes = rootComp.features.extrudeFeatures
ext_input = extrudes.createInput(profile, adsk.fusion.FeatureOperations.NewBodyFeatureOperation)
```

Set Options for Extrude

```
# Define that the extent is a distance extent of 1 cm
distance = adsk.core.ValueInput.createByReal(1)

# Set the distance extent to be single direction
ext_input.setDistanceExtent(False, distance)

# Set the extrude to be a solid one
ext_input.isSolid = True

# Create the extrusion
extrudes.add(ext_input)
```

Full Script

```python
#Author-Patrick Rainsberry
#Description-Basic Script to create a block

import adsk.core, adsk.fusion, adsk.cam, traceback

def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
        design = app.activeProduct

        # Get reference to the root component
        rootComp = design.rootComponent

        #Get reference to the sketchs and plane
        sketches = rootComp.sketches
        xyPlane = rootComp.xYConstructionPlane

        #Create a new sketch and get lines reference
        sketch = sketches.add(xyPlane)
        lines = sketch.sketchCurves.sketchLines

        # Use autodesk methods to create input geometry
        point0 = adsk.core.Point3D.create(0, 0, 0)
        point1 = adsk.core.Point3D.create(0, 1, 0)
        point2 = adsk.core.Point3D.create(1, 1, 0)
        point3 = adsk.core.Point3D.create(1, 0, 0)

        # Create lines
        lines.addByTwoPoints(point0, point1)
        lines.addByTwoPoints(point1, point2)
        lines.addByTwoPoints(point2, point3)
        lines.addByTwoPoints(point3, point0)

        # Get the profile defined by the square
        profile = sketch.profiles.item(0)

        # Create an extrusion input
        extrudes = rootComp.features.extrudeFeatures
        ext_input = extrudes.createInput(profile, adsk.fusion.FeatureOperations.NewBodyFeatureOperation)

        # Define that the extent is a distance extent of 1 cm
        distance = adsk.core.ValueInput.createByReal(1)

        # Set the distance extent to be single direction
        ext_input.setDistanceExtent(False, distance)

        # Set the extrude to be a solid one
        ext_input.isSolid = True

        # Create the extrusion
        extrudes.add(ext_input)

    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Full Script

```
import adsk.core, adsk.fusion, adsk.cam, traceback
def run(context):
    ui = None
    try:
        app = adsk.core.Application.get()
        ui  = app.userInterface
        design = app.activeProduct
        rootComp = design.rootComponent
        sketches = rootComp.sketches
        xyPlane = rootComp.xYConstructionPlane
        sketch = sketches.add(xyPlane)
        lines = sketch.sketchCurves.sketchLines
        point0 = adsk.core.Point3D.create(0, 0, 0)
        point1 = adsk.core.Point3D.create(0, 1, 0)
        point2 = adsk.core.Point3D.create(1, 1, 0)
        point3 = adsk.core.Point3D.create(1, 0, 0)
        lines.addByTwoPoints(point0, point1)
        lines.addByTwoPoints(point1, point2)
        lines.addByTwoPoints(point2, point3)
        lines.addByTwoPoints(point3, point0)
        profile = sketch.profiles.item(0)
        extrudes = rootComp.features.extrudeFeatures
        ext_input = extrudes.createInput(profile, adsk.fusion.FeatureOperations.NewBodyFeatureOperation)
        distance = adsk.core.ValueInput.createByReal(1)
        ext_input.setDistanceExtent(False, distance)
        ext_input.isSolid = True
        extrudes.add(ext_input)
    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Control the block

Create Points

```
# Use autodesk methods to create input geometry
point0 = adsk.core.Point3D.create(0, 0, 0)
point1 = adsk.core.Point3D.create(0, 1, 0)
point2 = adsk.core.Point3D.create(1, 1, 0)
point3 = adsk.core.Point3D.create(1, 0, 0)
```

Create Points with Variables

```
length = 4
width = 2
height = 3

# Use autodesk methods to create input geometry
point0 = adsk.core.Point3D.create(0, 0, 0)
point1 = adsk.core.Point3D.create(length, 0, 0)
point2 = adsk.core.Point3D.create(length, width, 0)
point3 = adsk.core.Point3D.create(0, width, 0)
```

Set Options for Extrude with Variable

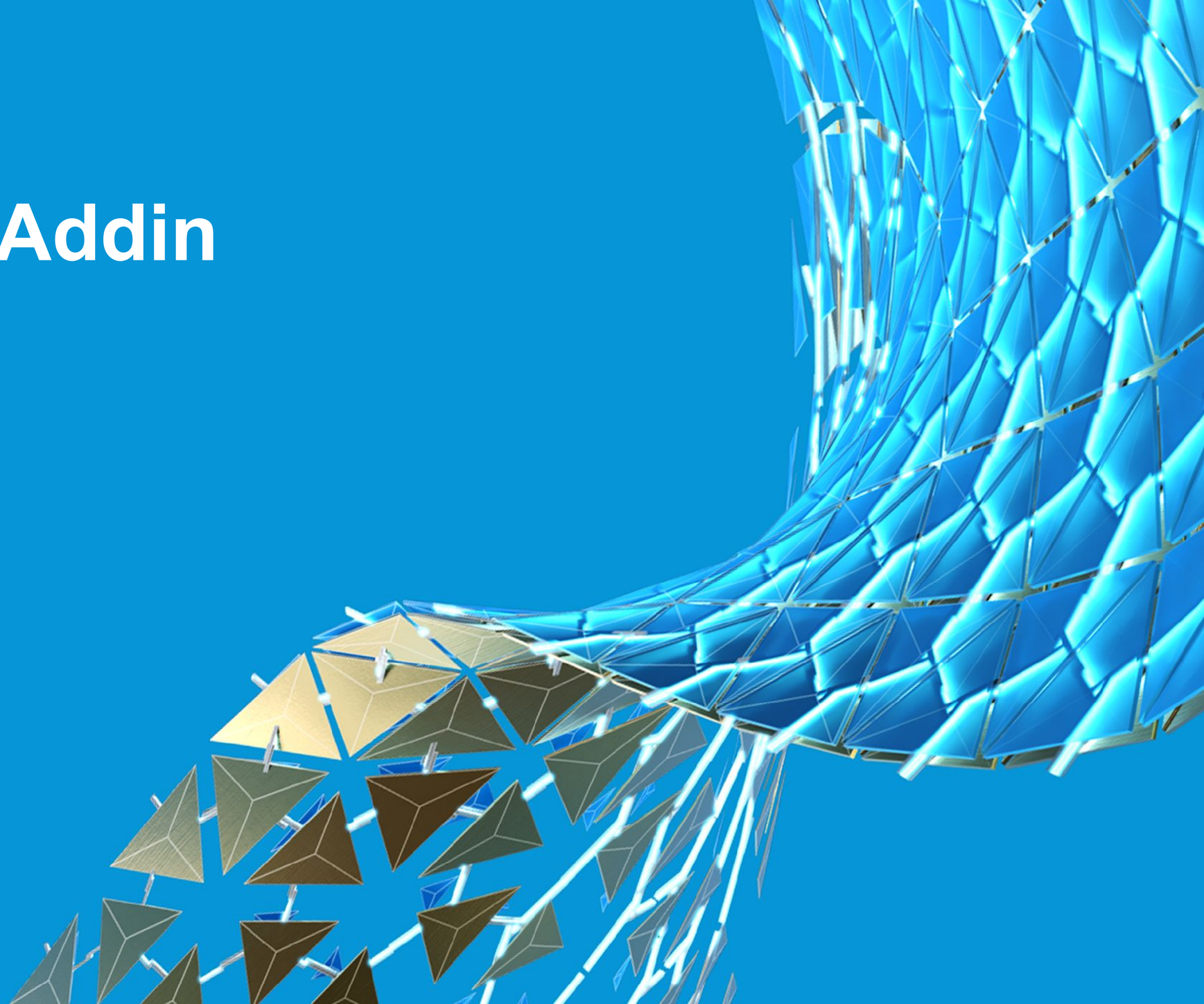
```
# Define that the extent is a distance extent of 1 cm height parameter
distance = adsk.core.ValueInput.createByReal(1)
distance = adsk.core.ValueInput.createByReal(height)

# Set the distance extent to be single direction
ext_input.setDistanceExtent(False, distance)

# Set the extrude to be a solid one
ext_input.isSolid = True
```

User Input

Basic User Input

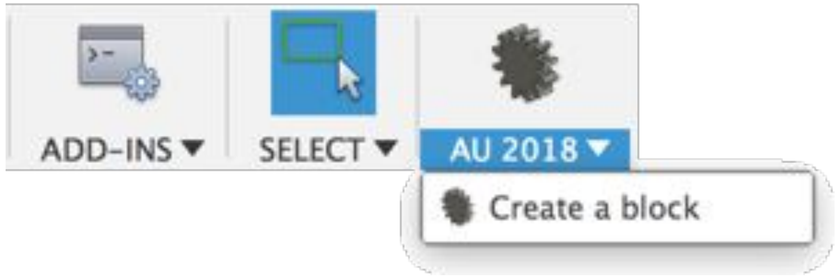
```
length = 4
depth = 2
height = 3

# Prompt user for values (Note: zero error checking)
length_input = ui.inputBox('Enter a length', 'Length', '3')
depth_input = ui.inputBox('Enter a depth', 'Depth', '1')
height_input = ui.inputBox('Enter a distance', 'Height', '2')

# Convert string to number from returned value
length = float(length_input[0])
depth = float(depth_input[0])
height = float(height_input[0])
```

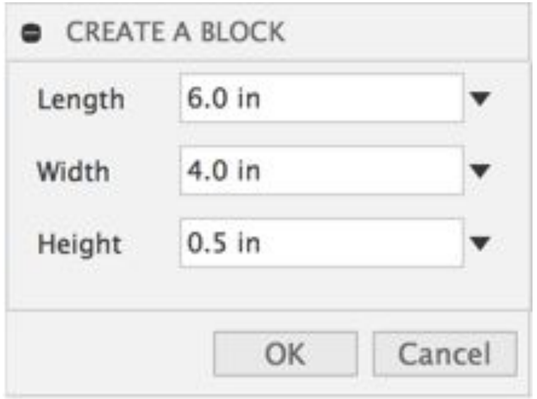
Creating an Addin



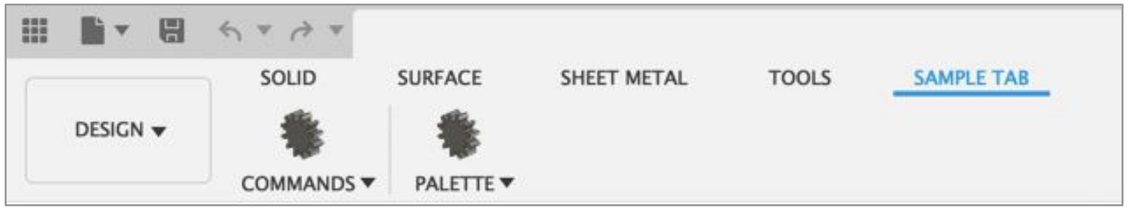
Add-ins vs. Scripts

- **Add-ins are always running (once started)**
- **They create a command in the UI (typically)**
- **When a user clicks the command it reacts:**
  - Typically would show a dialog box
  - User inputs values / makes selections
  - Add-in processes values and creates result
- **All actions of command result in single “undo” step**
  - *They may create many features in the timeline*





Commands



Workspace

TabToolBar

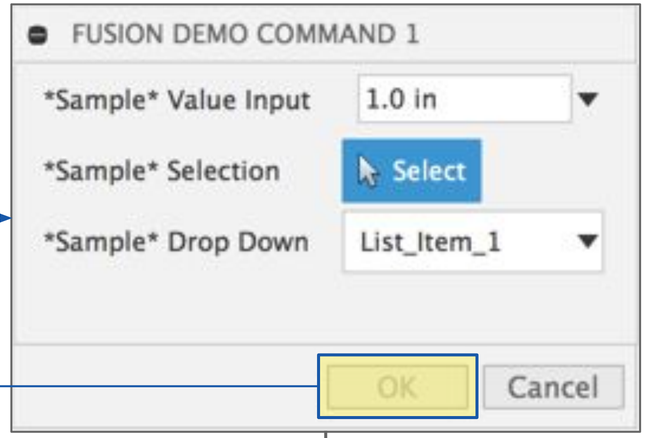
ToolBarPanel

CommandControl

CommandDefinition

CommandEvents

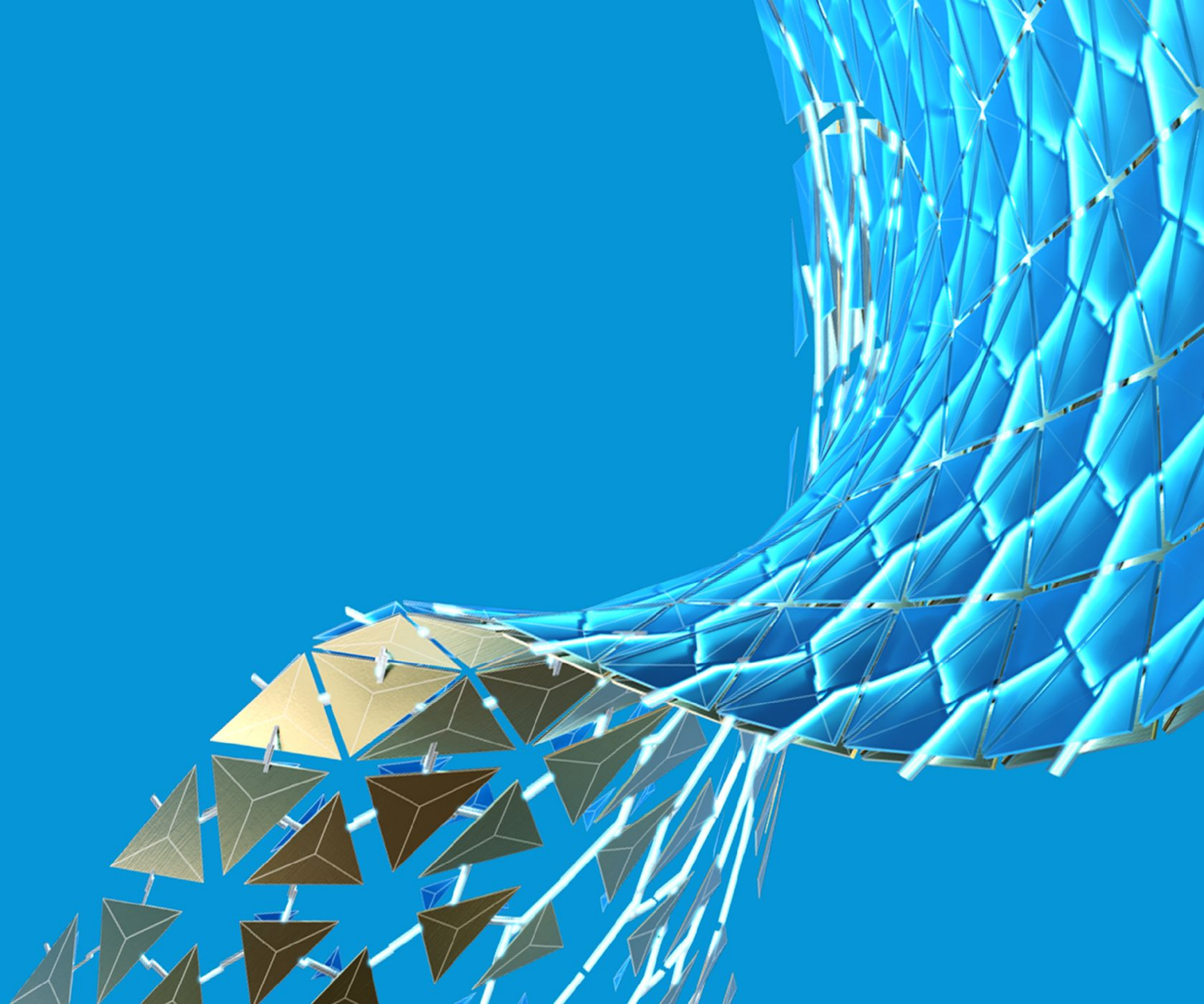
- CommandCreatedEvent
- CommandExecutedEvent
- CommandPreviewEvent
- CommandInputsChangedEvent
- CommandIDestroyedEvent



CommandInputs

- DropDownInput
  - ListItems
    - item
    - item
- SelectionInput
  - selection(0)
  - selection(1)
- ValueInput
  - Value

Apper



Using Apper

- **Apper is a wrapper to create Fusion 360 Add-ins**
- **The idea is to simplify the creation of add-ins for users**
- **Note this is a bit of a “pet project” and not really endorsed or maintained by anybody that actually knows what they are doing…**

- **Two main elements:**
  - Add-In Definition: **Fusion360App.py**
  - Commands: **Fusion360CommandBase.py**

https://apper.readthedocs.io/

Using Apper

- **Apper is a wrapper to create Fusion 360 Add-ins**
- **The idea is to simplify the creation of add-ins for users**
- **Note this is a bit of a “pet project” and not really endorsed or maintained by anybody that actually knows what they are doing…**

<u>Use at your own risk</u>

This sample is provided "As-is" with no guarantee of performance, reliability or warranty.

- **Two main elements:**
  - Add-In Definition: **Fusion360App.py**
  - Commands: **Fusion360CommandBase.py**

https://apper.readthedocs.io/

Cookiecutter

The easiest way to get started with apper is to start from a template project.

cookiecutter creates projects from project templates and is an amazing resource

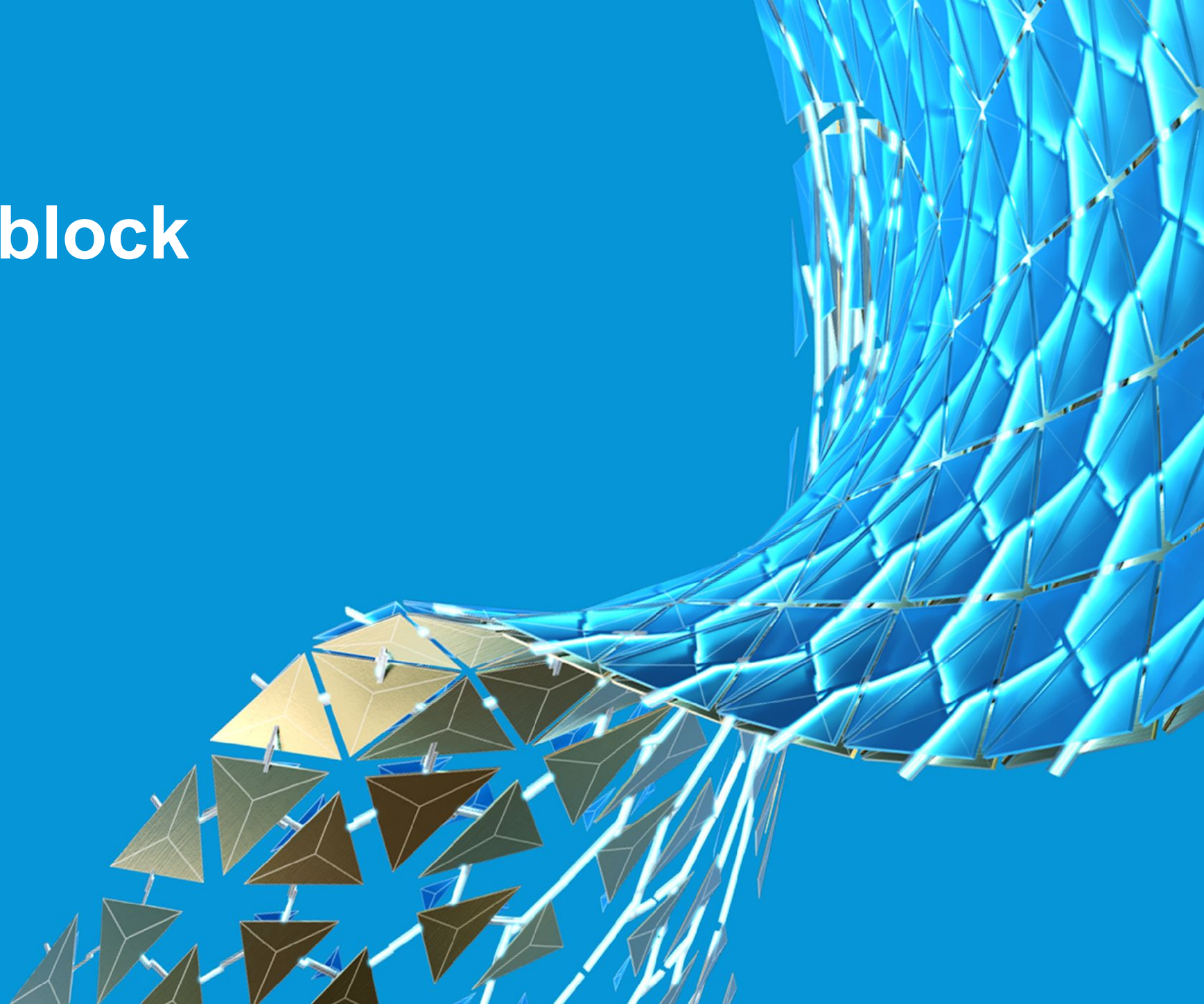
For more detailed installation instructions see their [documentation]

Learn more here:

https://apper.readthedocs.io/en/latest/usage/setup.html

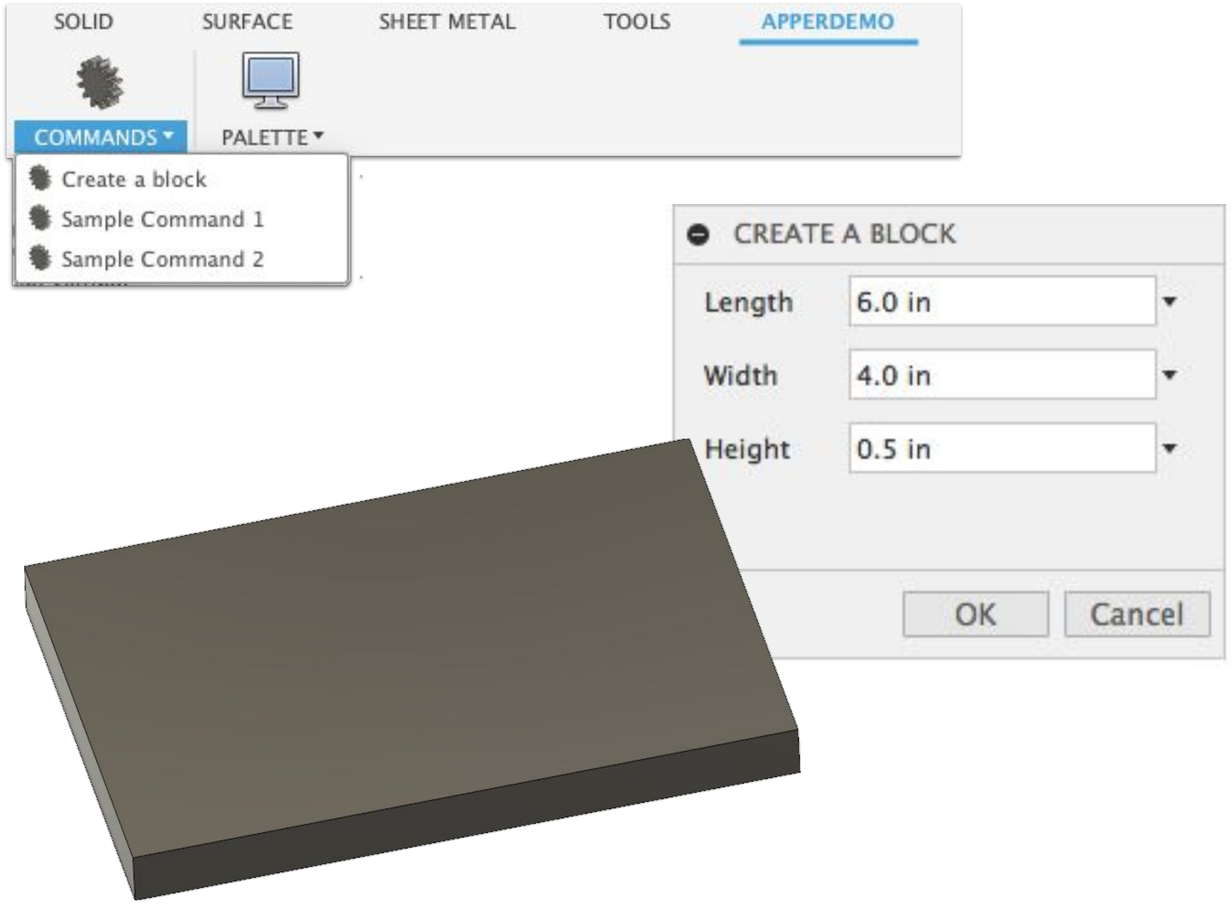
```
>>> pip3 install cookiecutter
>>> cd ~
>>> cd /Library/Application Support/Autodesk/Autodesk Fusion 360/API/AddIns/
>>> cookiecutter https://github.com/tapnair/cookiecutter-fusion360-addin.git
```

Refactor the block



Refactoring the Block

- Follow previous steps to create a new add-in
- Create a new python file in commands called: **BlockCommand.py**
- Take block code and move into a new function in **BlockCommand.py**
- Create UI elements to capture user input



BlockCommand.py

```python
import adsk.core
import adsk.fusion

from ..apper import apper

# Create Block based on user input
def make_block(length, width, height):

    ao = apper.AppObjects()

    # Get reference to the sketches and plane
    sketches = ao.root_comp.sketches
    xy_plane = ao.root_comp.xYConstructionPlane

    # Create a new sketch and get lines reference
    sketch = sketches.add(xy_plane)
    lines = sketch.sketchCurves.sketchLines

    # Use Autodesk methods to create input geometry
    point0 = adsk.core.Point3D.create(0, 0, 0)
    point1 = adsk.core.Point3D.create(length, 0, 0)
    point2 = adsk.core.Point3D.create(length, width, 0)
    point3 = adsk.core.Point3D.create(0, width, 0)

    # Create lines
    lines.addByTwoPoints(point0, point1)
    lines.addByTwoPoints(point1, point2)
    lines.addByTwoPoints(point2, point3)
    lines.addByTwoPoints(point3, point0)

    # Get the profile defined by the circle
    profile = sketch.profiles.item(0)

    # Create an extrusion input
    extrudes = ao.root_comp.features.extrudeFeatures
    ext_input = extrudes.createInput(profile, adsk.fusion.FeatureOperations.NewBodyFeatureOperation)

    # Define that the extent is a distance extent of height
    distance = adsk.core.ValueInput.createByReal(height)

    # Set the distance extent to be single direction
    ext_input.setDistanceExtent(False, distance)

    # Set the extrude to be a solid one
    ext_input.isSolid = True

    # Create the extrusion
    extrudes.add(ext_input)
```

Refactoring the Block

ApperDemo.py

```python
import adsk.core
import traceback


try:
    from . import config
    from .apper import apper

    from .commands.BlockCommand import BlockCommand

    # Create our addin definition object
    my_addin = apper.FusionApp(config.app_name, config.company_name, False)
    my_addin.root_path = config.app_path

    # General command showing inputs and user interaction
    my_addin.add_command(
        'Create a block',
        BlockCommand,
        {
            'cmd_description': 'Create a block from user input sizes',
            'cmd_id': 'BlockCommand',
            'workspace': 'FusionSolidEnvironment',
            'toolbar_panel_id': 'Commands',
            'cmd_resources': 'command_icons',
            'command_visible': True,
            'command_promoted': True,
        }
    )
```

BlockCommand.py

```python
# Class for Fusion 360 Block Command
class BlockCommand(apper.Fusion360CommandBase):

    # Run when the user presses OK
    def on_execute(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs, args, input_values):

        # Get the values from the user input
        length = input_values['length_input']
        width = input_values['width_input']
        height = input_values['height_input']

        # Run the block function
        make_block(length, width, height)

    # Run when the user selects your command icon from the Fusion 360 UI
    def on_create(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs):

        # Create a default value using a string
        default_length = adsk.core.ValueInput.createByString('6.0 in')
        default_width = adsk.core.ValueInput.createByString('4.0 in')
        default_height = adsk.core.ValueInput.createByString('.5 in')

        ao = apper.AppObjects()

        inputs.addValueInput('length_input', 'Length', ao.units_manager.defaultLengthUnits, default_length)
        inputs.addValueInput('width_input', 'Width', ao.units_manager.defaultLengthUnits, default_width)
        inputs.addValueInput('height_input', 'Height', ao.units_manager.defaultLengthUnits, default_height)
```

Function containing previous block creation code

https://apper.readthedocs.io/

Developer Resources

Useful Information and troubleshooting an add-in

The best place to get help is the Fusion 360 forum. Otherwise I find an infinite resource in places like stack exchange. Most of the challenges I come across are really python questions more than anything.

Useful Links:

Forum to ask questions:
https://forums.autodesk.com/t5/api-and-scripts/bd-p/22

Offline API DOcumentation (chm):
https://help.autodesk.com/cloudhelp/ENU/Fusion-360-API/SupportFiles/FusionAPI.chm

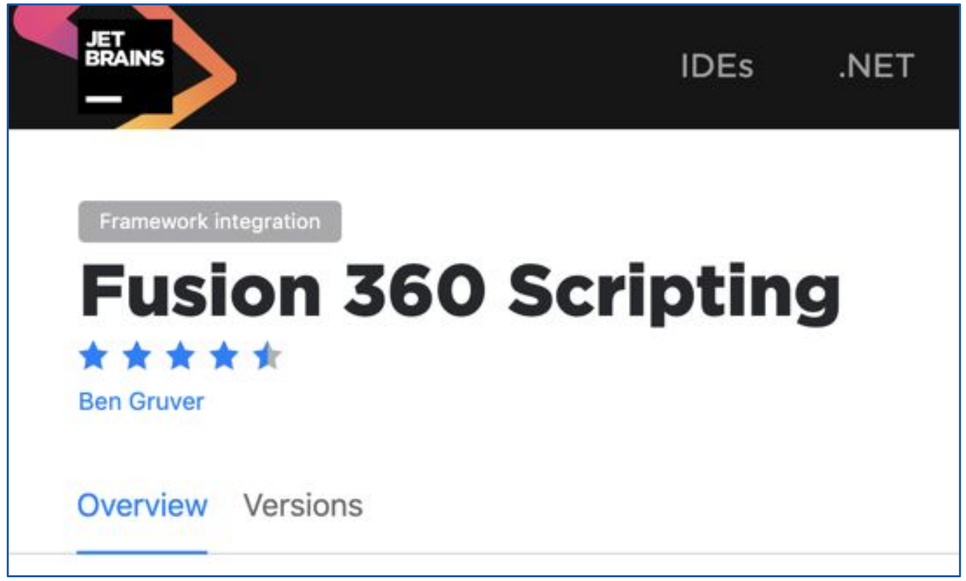
For more detailed information about editing and debugging your scripts and add-ins see the language specific topics (Python or C++) because the process is different depending on which programming language you're using:
Python Specific Issues
C++ Specific Issues

Samples:

My main page for these projects: https://tapnair.github.io/index.html

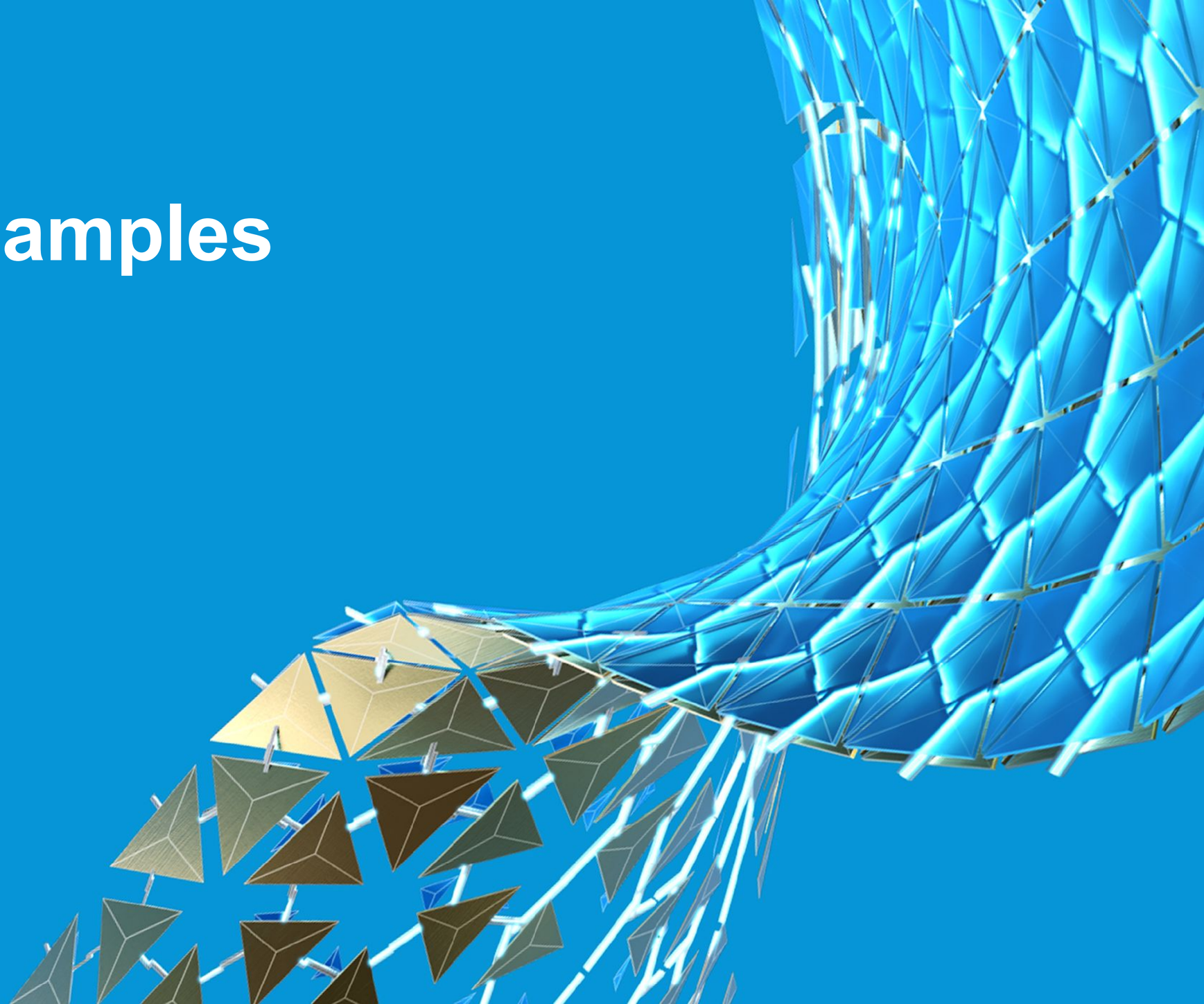
PyCharm Plugin

- **Run script in Fusion 360**
  - Launches a script in Fusion 360
  - As if you had run it from the AddIn window
- **Debug script in Fusion 360**
  - Launches a script in fusion 360 and attached a debugger
  - Stop at breakpoints
  - All the usual debuggery goodness.
  - Redirects stdout and stderr to the debugging console
- **Attach to Process**
  - Attaches to a Fusion 360 process without running a script.
  - Any breakpoints will be hit
    - *Assuming Fusion happens to run the break pointed code.*
    - *e.g. if you start the script manually in Fusion 360 itself.*
- **Automatically adds a dependency for the Fusion Python APIs**
  - Used for autocomplete, contextual docs, etc.



https://plugins.jetbrains.com/plugin/11343-fusion-360-scripting

Additional Samples



Samples

My main page for these projects: https://tapnair.github.io/index.html

ventMaker - Create custom vent features in Fusion 360. Circular, Slot and rectangular vents.

HelixGenerator - Generate Helical Curves in Fusion 360

ParamEdit - Quick editor to make changes to user parameters with real time update.

stateSaver - Save the current state of: hide/show, suppress/unsuppress, and user parameter values.

ShowHidden - Display utilities for Fusion 360. Show hidden or all: bodies, components and planes.

Project-Archiver - Automate the export of all designs in a project to a local archive directory.

copyPaste - Copy and paste bodies between documents in Fusion 360, explicitly breaking references

NESTER - Semi automated nesting of sheet/flat parts in Fusion 360.

OctoFusion - Automate the process of exporting a file and sending it to Octoprint.

UGS_Fusion - Automate the process of posting a file and opening it in Universal G-code Sender