Fusion 360 API

Patrick Rainsberry, Autodesk

Business Strategy, Fusion 360

AUTODESK®
FUSION 360™
ACADEMY

https://github.com/tapnair/Fusion360APIClass

Download samples and references

Outline

- Get started with the Fusion 360 API
- Learn how to write a basic script
- Learn how to create a custom add-in or feature
  - Using Add-in Skeleton
  - Automating Tasks
    - Holes in a plate - CSV Import
    - Change something, export, g-code
    - Solid Infill - Analyze model to create new geometry
  - Other Examples
- Learn how to find necessary information and troubleshoot a Fusion 360 add-in

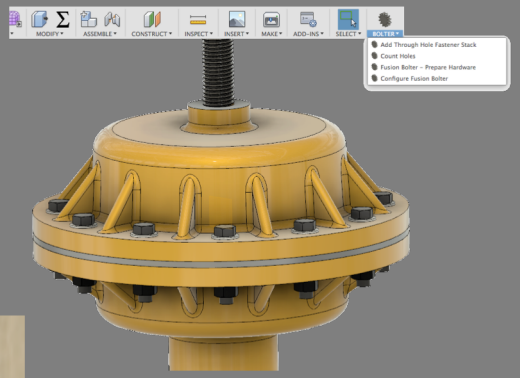
API Overview

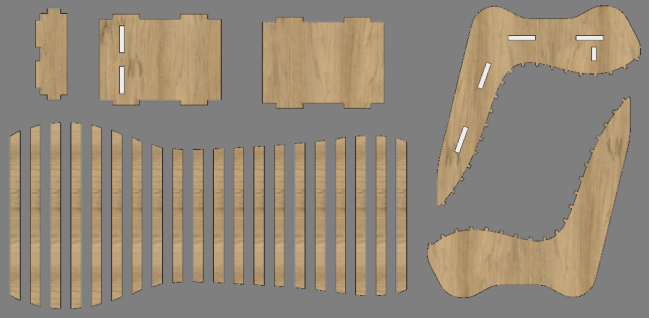
Things to Automate

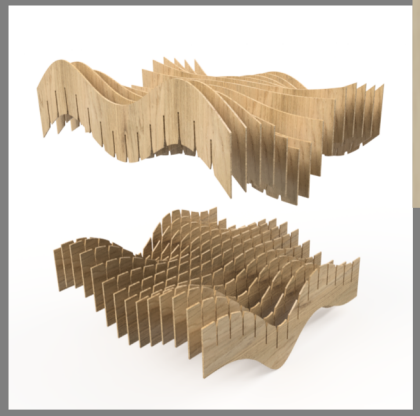
Repetitive tasks

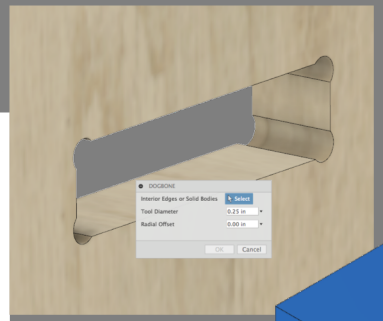
Data import / export

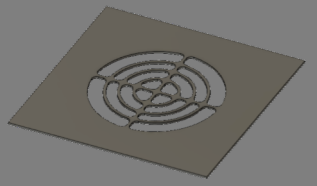
Complex operations

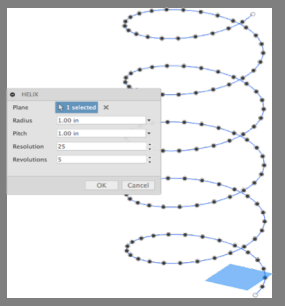


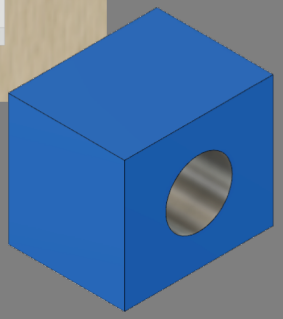


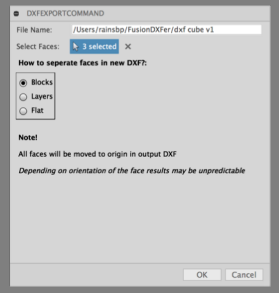


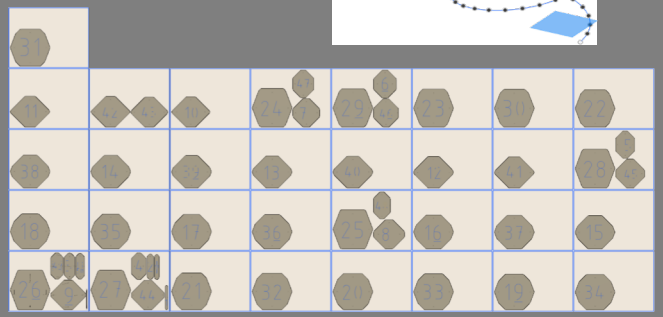










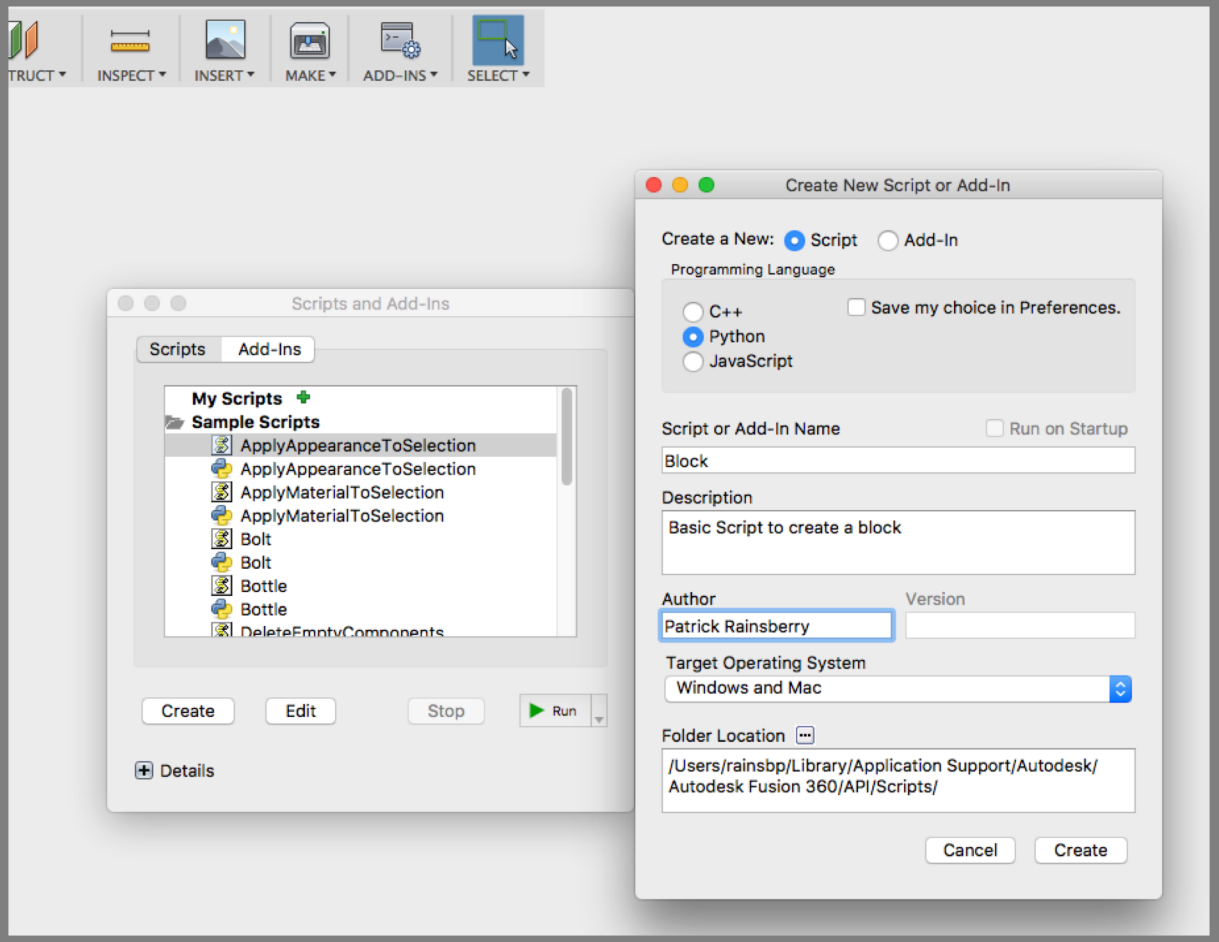


Fusion 360 API

- 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.
  - Supports object-oriented, imperative and functional programing
  - Dynamically typed (interpreted)
  - Automatic memory management
  - A large set of standard and third party libraries

Make a Block

Create a New Script



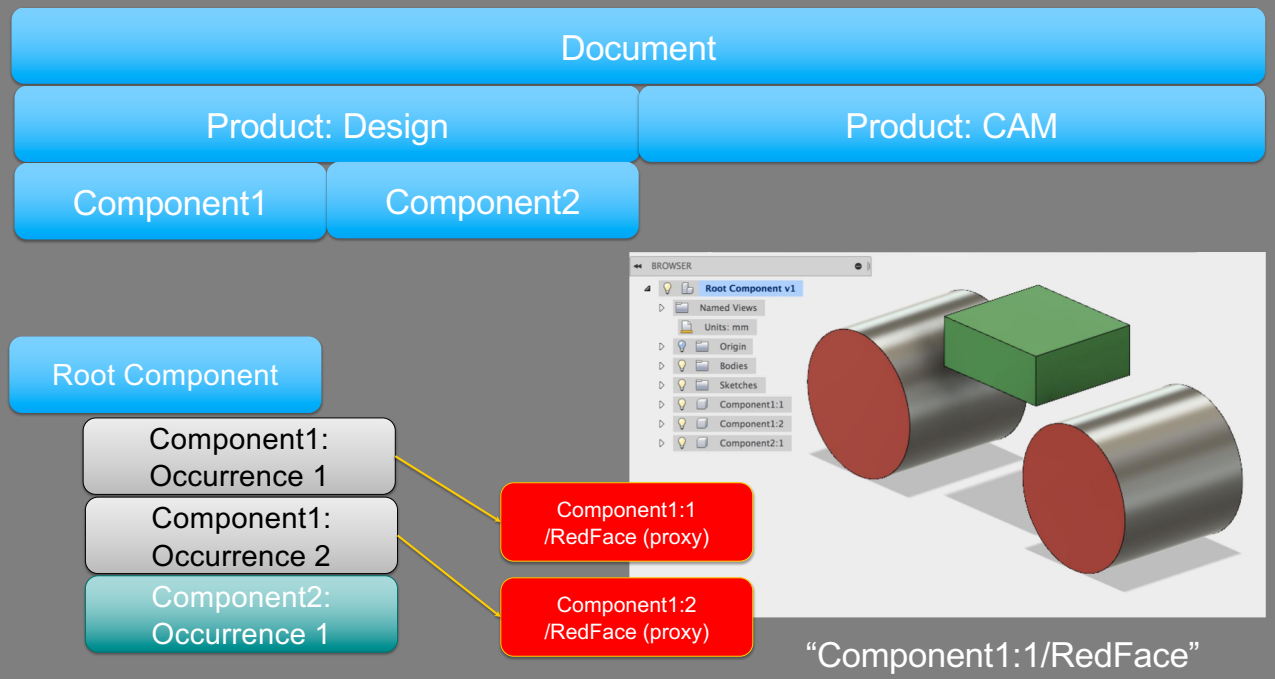
Initial Script

```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
        ui.messageBox('Hello script')

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

Our Script - Reference design object

```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

        All our code goes here

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

Fusion 360 Document Structure



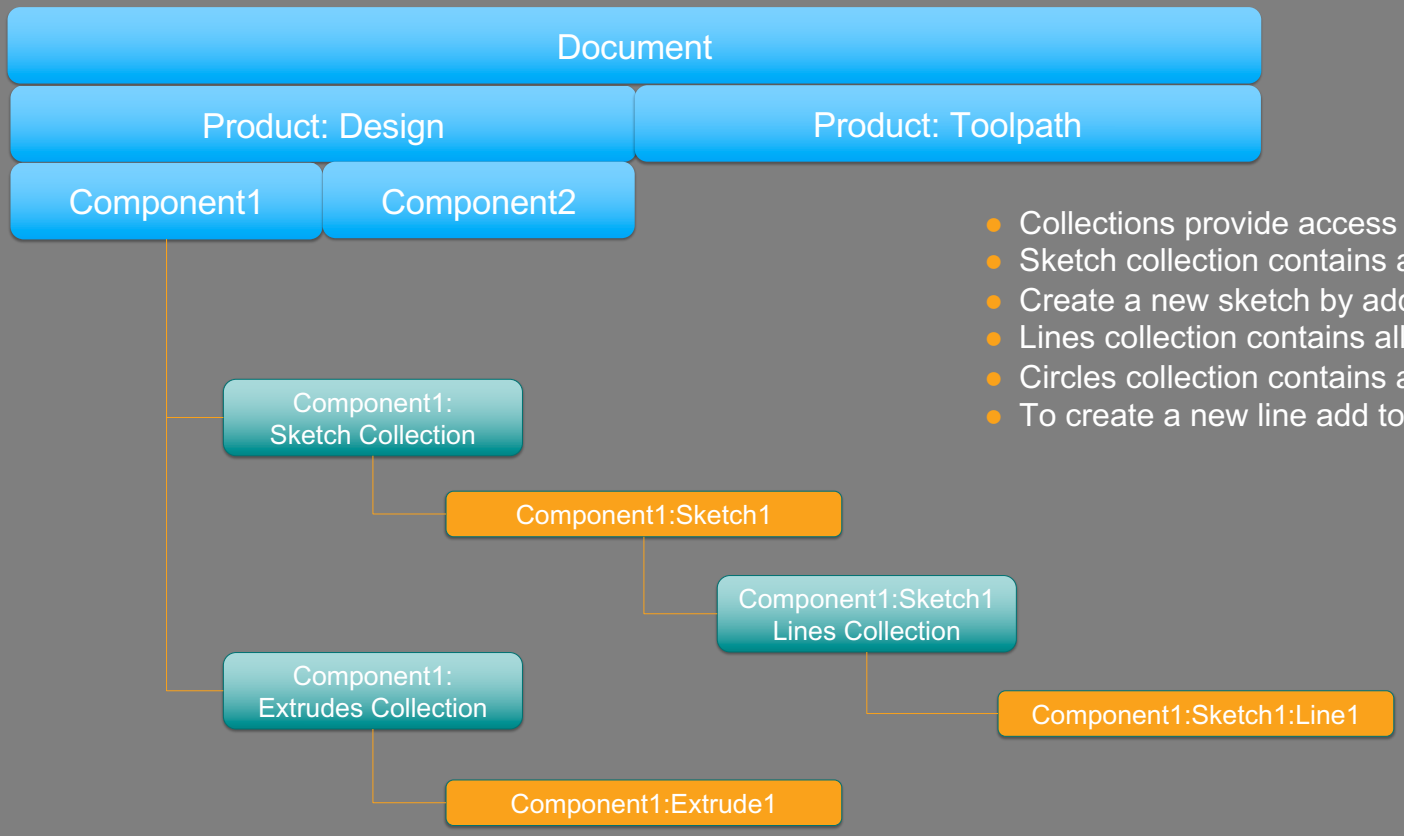
"Component1:1/RedFace"

- Document: Data panel item. ~Fusion File
- Product: Data type. Design, Tool-path, etc.
- Component: Unique part geometry
- Occurrence: Instance of a component
- Proxy: Reference to occurrence geometry

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()))
```

Features and Collections



- Collections provide access to a common set of objects.
- Sketch collection contains all sketches in a component
- Create a new sketch by adding an item to the collection
- Lines collection contains all lines within a sketch
- Circles collection contains all circles in the sketch, etc...
- To create a new line add to the Lines collection

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
```

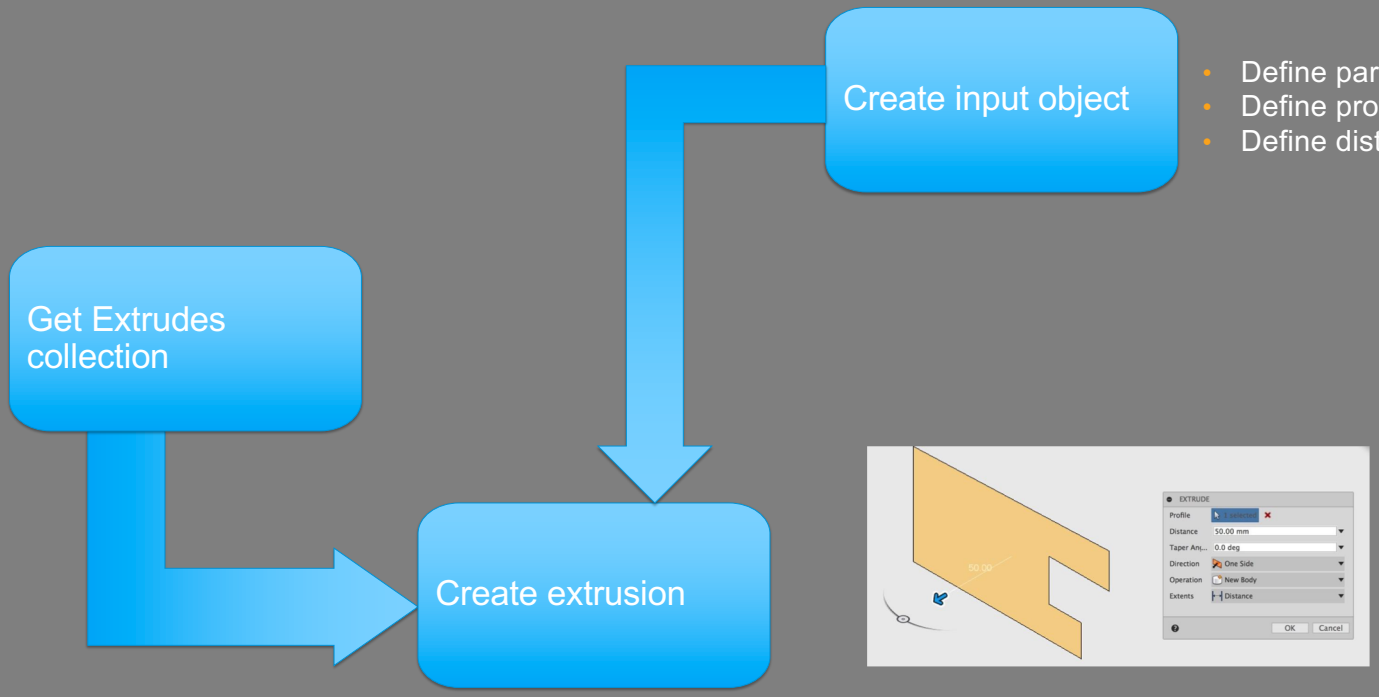
Create Points and Lines

```
# 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)
```

Creating Features (Extrude)



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

By adding a new extrude to the extrudes collection

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)
```

Units in Fusion 360

Fusion Default Model Units

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

radians

kg

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”);

Users can input any unit

3

3 in + 5 in

3 m ^ 2

3 in + 5 mm

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"

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

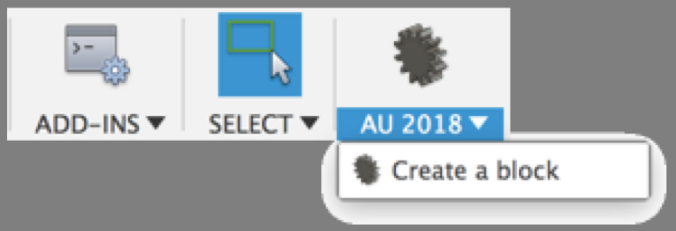
Basic User Input

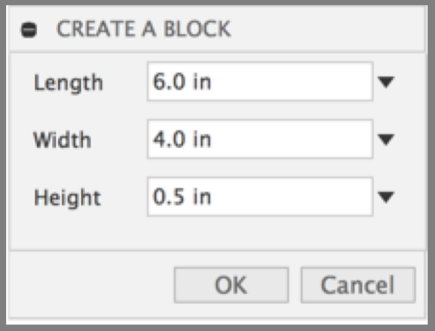
```
length = 4
depth = 2
height = 3

# 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

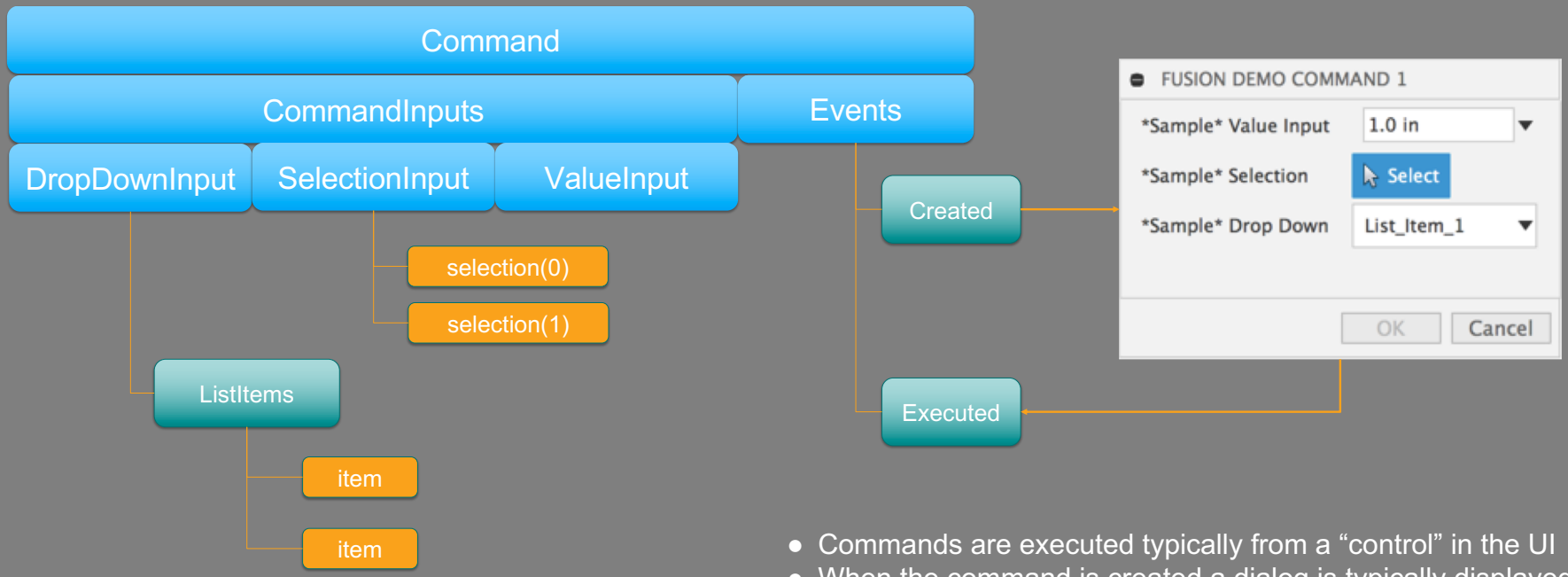
Add-ins vs. Scripts

- 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



- Commands are executed typically from a “control” in the UI
- When the command is created a dialog is typically displayed
- A user inputs values and/or selections
- Those inputs are used to create some actions in Fusion 360

Addin Skeleton

Using Add-in Skeleton

- Addin Skeleton is a wrapper to create basic Fusion 360 Addins
- 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:
  - Addin definition: **Fusion360AddinSkeleton.py**
  - Commands: **Demo1Command.py**

Using Add-in Skeleton

- Addin Skeleton is a wrapper to create basic Fusion 360 Addins
- 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…

Use at your own risk

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

- Two main elements:
  - Addin definition: **Fusion360AddinSkeleton.py**
  - Commands: **Demo1Command.py**

Create a new addin

Get the template and create your add-in directory

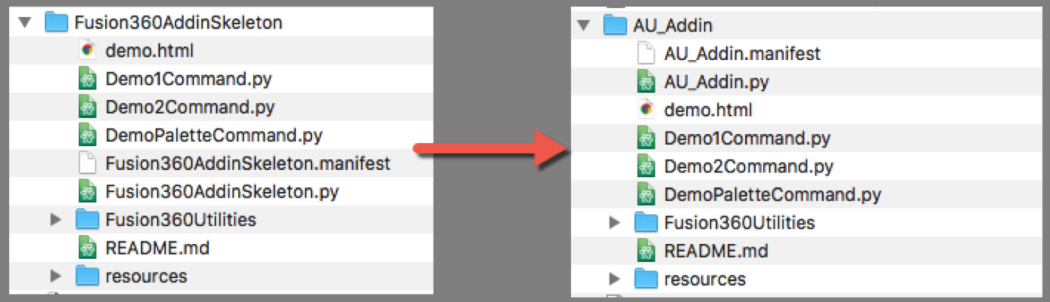
- Download or clone this repo: https://github.com/tapnair/Fusion360AddinSkeleton

Move the folder into your add-ins directory. Look here for more information:
https://tapnair.github.io/installation.html

Files in the Fusion360Utilities folder should not be modified.

Rename the following items to your desired addin name:

- The top level folder
- Fusion360AddinSkeleton.py
- Fusion360AddinSkeleton.manifest



Edit the manifest file and update the fields accordingly

Addin Definition

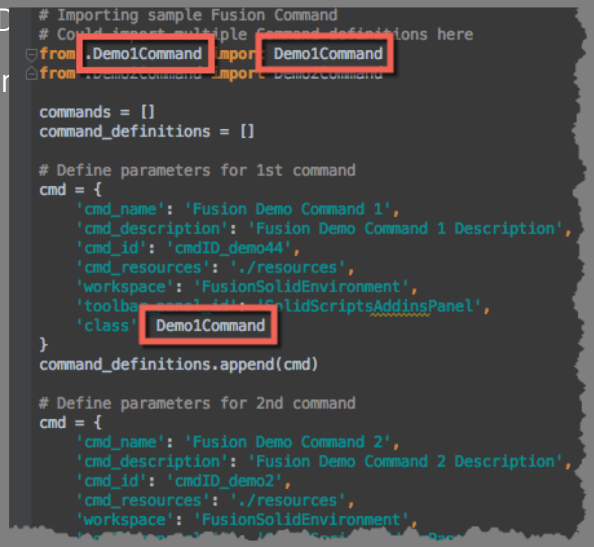
Open the newly renamed python file

The current file will create two commands in the Fusion 360 UI in the Addins D

Change the names and description strings here to your desired naming conver

You can define many commands here.

You can also pass other values as necessary into the commands.



Command Definitions

The Fusion360CommandBase class wraps the common tasks used when creating a Fusion 360 addin.

- Edit Demo1Command.py and add functionality to the desired methods.
- onCreate: Build your UI components here
- onExecute: Will be executed when user selects OK in command dialog.
- DemoCommand1 creates a very basic UI and then accesses the input parameters.

On Create

When the user clicks the command icon in the Fusion UI (The command control) this function will be executed

By referencing the *inputs* object you can easily add dialog box elements to your command

Sometimes you may want to read some data or analyze the model BEFORE creating the dialog box

```
# Run when the user selects your command icon from the Fusion 360 UI
# Typically used to create and display a command dialog box
# The following is a basic sample of a dialog UI
def on_create(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs):

    # Create a default value using a string
    ao = AppObjects()
    default_value = adsk.core.ValueInput.createByString('1.0 in')
    default_units = ao.units_manager.defaultLengthUnits
    inputs.addValueInput('value_input_id', '*Sample* Value Input', default_units, default_value)

    # Other Input types
    inputs.addBoolValueInput('bool_input_id', '*Sample* Check Box', True)
    inputs.addStringValueInput('string_input_id', '*Sample* String Value', 'Some Default Value')
    inputs.addSelectionInput('selection_input_id', '*Sample* Selection', 'Select Something')

    # Read Only Text Box
    inputs.addTextBoxCommandInput('text_box_input_id', 'Selection Type: ', 'Nothing Selected', 1, True)

    # Create a Drop Down
    drop_down_input = inputs.addDropDownCommandInput('drop_down_input_id', '*Sample* Drop Down',
                                                     adsk.core.DropDownStyles.TextListDropDownStyle)
    drop_down_items = drop_down_input.listItems
    drop_down_items.add('List_Item_1', True, '')
    drop_down_items.add('List_Item_2', False, '')
```

On Input Changed

When a user changes anything in the command dialog this method is executed.

Typically used for making changes to the command dialog itself.

For example if a user selects STL as an export type, you can then display an option to show a refinement option

```
# Run when any input is changed.
# Can be used to check a value and then update the add-in UI accordingly
def on_input_changed(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs, changed_input, input_values):

    # Selections are returned as a list so lets get the first one
    all_selections = input_values.get('selection_input_id', None)

    if all_selections is not None:
        the_first_selection = all_selections[0]

        # Update the text of the string value input to show the type of object selected
        text_box_input = inputs.itemById('text_box_input_id')
        text_box_input.text = the_first_selection.objectType
```

On Preview / On Destroy

On preview will also execute on any changes to the command inputs

- Code in this function will cause the graphics to refresh.
- Note if your addin is complex it may be useful to only preview a subset of the full operations

On Destroy executes after the command has run

- You can use this to do any clean up that may otherwise be difficult until after the command has completed
- Like firing a second command for example

```
# Run whenever a user makes any change to a value or selection in the addin UI
# Commands in here will be run through the Fusion processor and changes will be reflected in  Fusion graphics area
def on_preview(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs, args, input_values):
    pass

# Run after the command is finished.
# Can be used to launch another command automatically or do other clean up.
def on_destroy(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs, reason, input_values):
    pass
```

On Execute

```
# Run when the user presses OK
# This is typically where your main program logic would go
def on_execute(self, command: adsk.core.Command, inputs: adsk.core.CommandInputs, args, input_values):

    # Get the values from the user input
    the_value = input_values['value_input_id']
    the_boolean = input_values['bool_input_id']
    the_string = input_values['string_input_id']
    all_selections = input_values['selection_input_id']
    the_drop_down = input_values['drop_down_input_id']

    # Selections are returned as a list so lets get the first one and its name
    the_first_selection = all_selections[0]
    the_selection_type = the_first_selection.objectType

    # Get a reference to all relevant application objects in a dictionary
    ao = AppObjects()

    converted_value = ao.units_manager.formatInternalValue(the_value, 'in', True)

    ao.ui.messageBox('The value, in internal units, you entered was:  {} \n'.format(the_value) +
                     'The value, in inches, you entered was:  {} \n'.format(converted_value) +
                     'The boolean value checked was:  {} \n'.format(the_boolean) +
                     'The string you typed was:  {} \n'.format(the_string) +
                     'The type of the first object you selected is:  {} \n'.format(the_selection_type) +
                     'The drop down item you selected is:  {}'.format(the_drop_down)
                     )
```

Extra Capabilities: input_values

In the on_execute, on_preview, on_input_changed methods there is a parameter called "input_values"

This parameter is a dictionary containing the relevant values for all of the user inputs.

- The key is the name of the input.
- The value is dependant on the type input:
  - Value type inputs will have their actual value stored (string or number depending)
  - List type inputs (drop downs, etc) will have the name of the selected item as the value (string)
  - Selection inputs are returned as an array of the selected objects (even if 1 item is selected)

```
# Get the values from the user input
the_value = input_values['value_input_id']
the_boolean = input_values['bool_input_id']
the_string = input_values['string_input_id']
all_selections = input_values['selection_input_id']
the_drop_down = input_values['drop_down_input_id']

# Selections are returned as a list so lets get the first one and its name
the_first_selection = all_selections[0]
the_selection_type = the_first_selection.objectType
```

Note: you can still access the raw command inputs object with the "inputs" variable. This would behave similar to any of the examples in the API documentation. i.e. length = inputs.itemById('length').value

Extra Capabilities: AppObjects

This is a helper class that can be used to easily access of many useful fusion 360 objects.

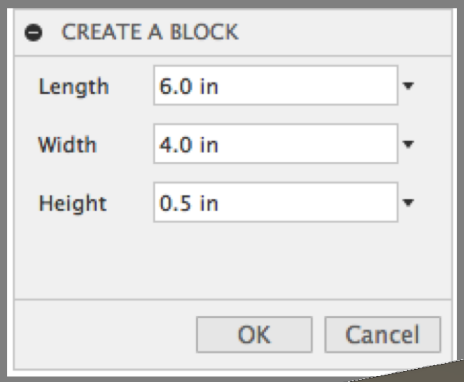
It contains many properties:

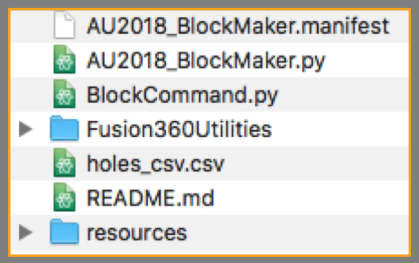
```
from .Fusion360Utilities.Fusion360Utilities import AppObjects
ao = AppObjects()
ao.ui.messageBox('Hello World!')
```

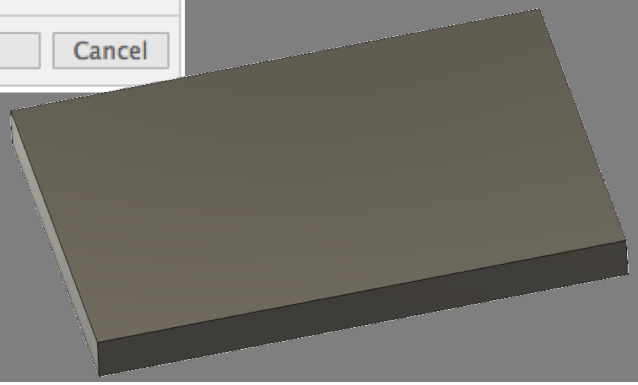
- app - Application Object
- document - Active Document
- product - Active Product
- design - Design Product (if it exists)
- cam - CAM Product (if it exists)
- ui - User Interface
- import_manager - Application Import Manager
- export_manager - Export Manager (if the active product is Design)
- units_manager - Fusion Units Manager (if the active product is design) or Units Manager
- root_comp - Root Component (if the active product is design)
- time_line - (if the active product is design and the type os Parametric Design Type)

Refactoring the Block

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







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

    ao = AppObjects()

    # Get reference to the sketchs 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

FA19_BlockMaker.py

```python
# Author-Patrick
# Description-Basic demo of creating a block
from .BlockCommand import BlockCommand

commands = []
command_definitions = []

# Define parameters for 1st command
cmd = {
    'cmd_name': 'Create a block',
    'cmd_description': 'Create a block',
    'cmd_id': 'cmdID_BlockCommand',
    'cmd_resources': './resources',
    'workspace': 'FusionSolidEnvironment',
    'toolbar_panel_id': 'AU 2018',
    'command_promoted': True,
    'class': BlockCommand
}
command_definitions.append(cmd)


# Set to True to display various useful messages when debugging your app
debug = False

# Don't change anything below here:
for cmd_def in command_definitions:
    command = cmd_def['class'](cmd_def, debug)
    commands.append(command)


def run(context):
    for run_command in commands:
        run_command.on_run()


def stop(context):
    for stop_command in commands:
        stop_command.on_stop()
```

BlockMakerCommand.py

```python
# Class for Fusion 360 Block Command
class BlockCommand(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 = 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

Connecting to External Data

Holes in a Plate - Reading a CSV file

User Input
- Plate Size (X, Y, Z)
- CSV File Name

Create Plate
- Create sketch
- Create solid extrude

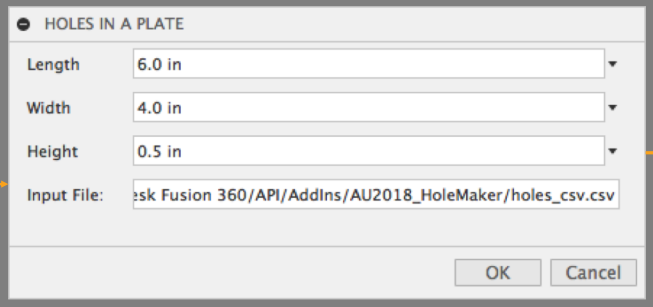
Read CSV File

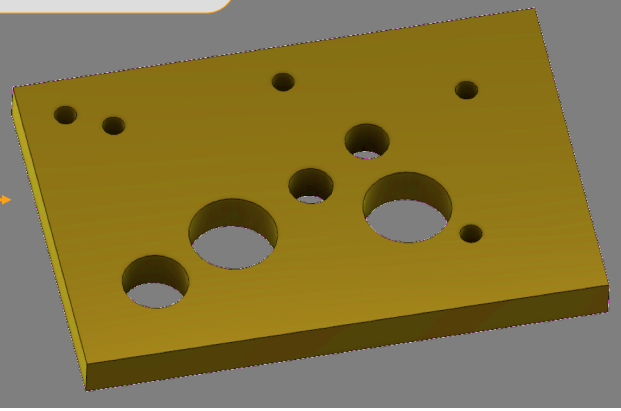
Make Holes
- Get all circle profiles
- Create cut Extrude

For each row in the csv file

Sketch Circle

	A	B	C
1	x	y	radius
2	1	1	0.125
3	3	2	0.25
4	2.25	1.5	0.25
5	5	3	0.375
6	4	2.5	0.5
7	1.375	3	0.125
8	3	0.5	0.125
9	2	2.5	0.5
10	5	0.75	0.125
11	5.5	0.5	0.125
12			





Holes in a Plate - Reading a CSV file

```python
def make_holes(hole_list):

    ao = AppObjects()

    profile_collection = adsk.core.ObjectCollection.create()

    # 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)
    circles = sketch.sketchCurves.sketchCircles

    input_units = 'in'

    for hole in hole_list:
        x = ao.units_manager.evaluateExpression(hole['x'], input_units)
        y = ao.units_manager.evaluateExpression(hole['y'], input_units)
        radius = ao.units_manager.evaluateExpression(hole['radius'], input_units)
        # x = float(hole['x'])
        # y = float(hole['y'])
        # radius = float(hole['radius'])

        center_point = adsk.core.Point3D.create(x, y, 0)
        circles.addByCenterRadius(center_point, radius)

    for profile in sketch.profiles:
        profile_collection.add(profile)

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

    # Set the distance extent to be single direction
    extent_all = adsk.fusion.ThroughAllExtentDefinition.create()
    ext_input.setOneSideExtent(extent_all, adsk.fusion.ExtentDirections.PositiveExtentDirection)

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

```python
# Function to convert a csv file to a list of dictionaries.
# Takes in one variable called "csv_file_name"
def csv_dict_list(csv_file_name):

    # Open variable-based csv,
    # Iterate over the rows and map values to a list of dictionaries
    reader = csv.DictReader(open(csv_file_name, 'r'))
    dict_list = []
    for line in reader:
        dict_list.append(line)
    return dict_list
```

```python
# Simple add-in to create a plate with holes
class HolesCommand(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']
        the_file_name = input_values['the_file_name']

        # Read in the csv hole and return a list of 1 dictionary per hole
        hole_list = csv_dict_list(the_file_name)

        # Create a block based on user input
        make_block(length, width, height)

        # Create holes based on the imported csv
        make_holes(hole_list)
```

Automating Geometry Changes and Outputs

User Input
- input file
- output location
- options

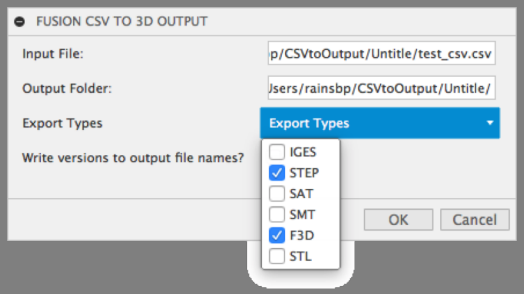
Read CSV File

For each row in the csv file

Update Model Parameters

Export gcode
- Switch to CAM
- Update toolpaths
- Post nc file

Export 3D Model
- export each file type



Add-in to create multiple variations of a design

- Export 3D files (Step, IGES, SAT, f3d)
- Output g-code for an existing setup or operation

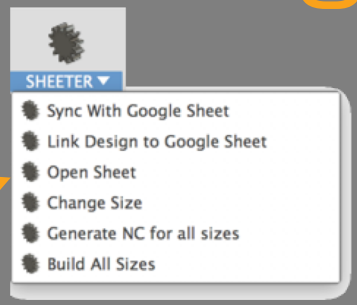
Google Sheets Integration

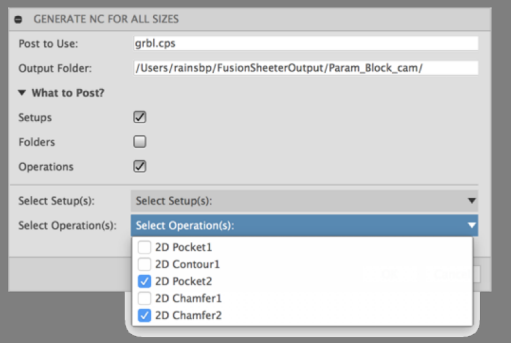
Param_Block_cam

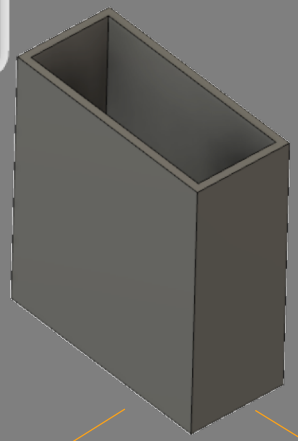
File Edit View Insert Format Data Tools Add-ons Help All changes s

100% $ % .0 .00 123 Arial 10

	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					



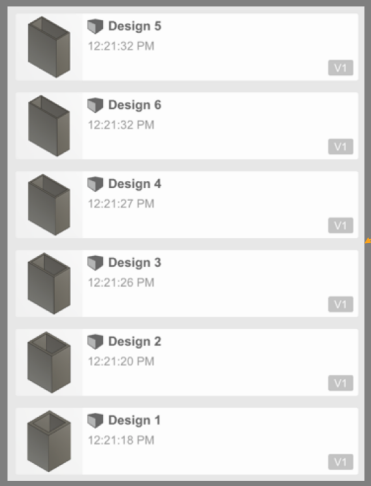


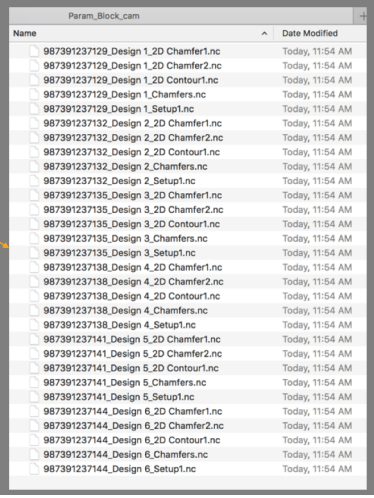


Use Google Sheets to drive model:

- Parameters
- Feature Suppression
- BOM Data

Change Sizes
Generate Fusion models for all sizes
Generate CAM (nc) for all sizes





Working with Geometry

Infill Example - Geometry Pattern

User Input
- Cell Size
- Shape Type

→

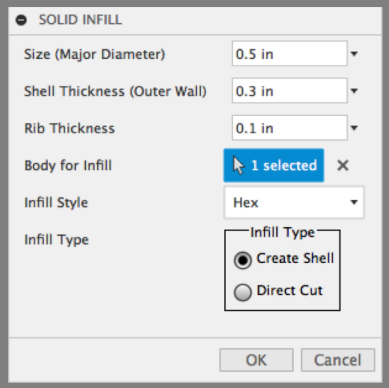
Create Core Body
- Shell Input Body
- Offset internal surfaces
- Boundary fill interior

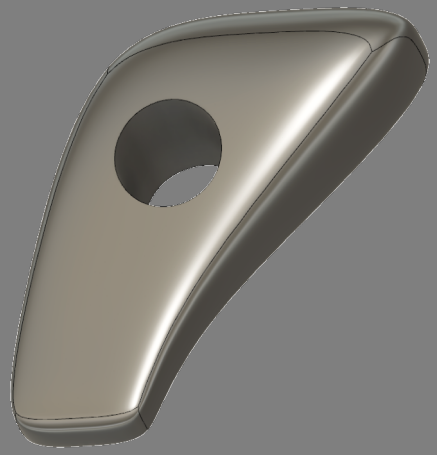
→

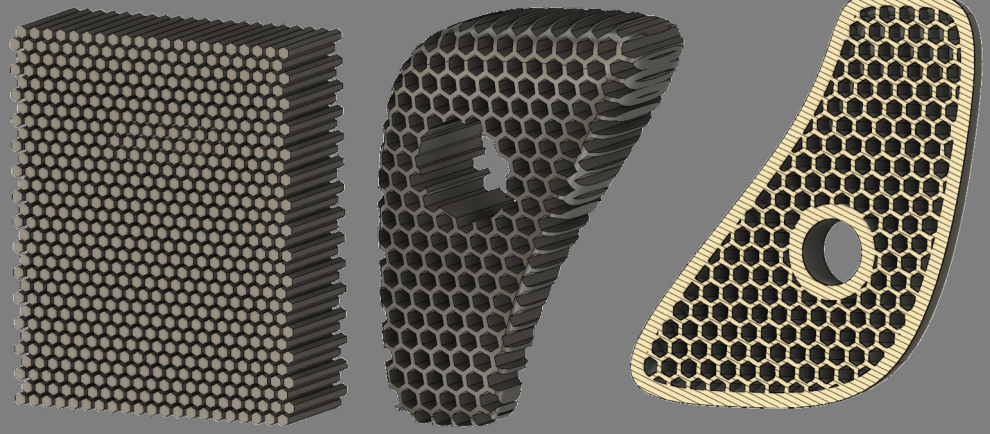
Cut Out Shapes
- Create tool shapes
- Create Pattern
- Boolean Cut

→

Combine With Original Body







Add-in to create a “solid in-fill” cut out in a solid part

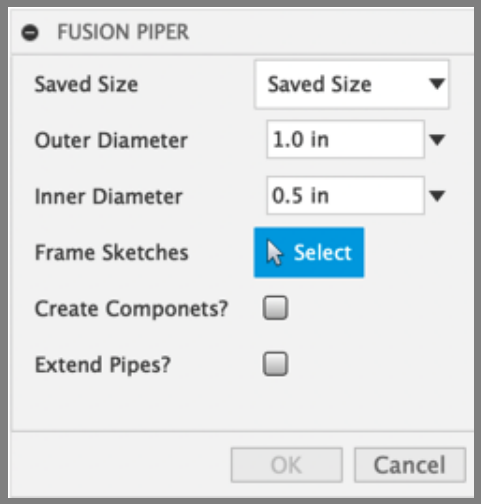
- Lots of calculations to determine position and spacing
- Analyze the input body to get inputs to calculations

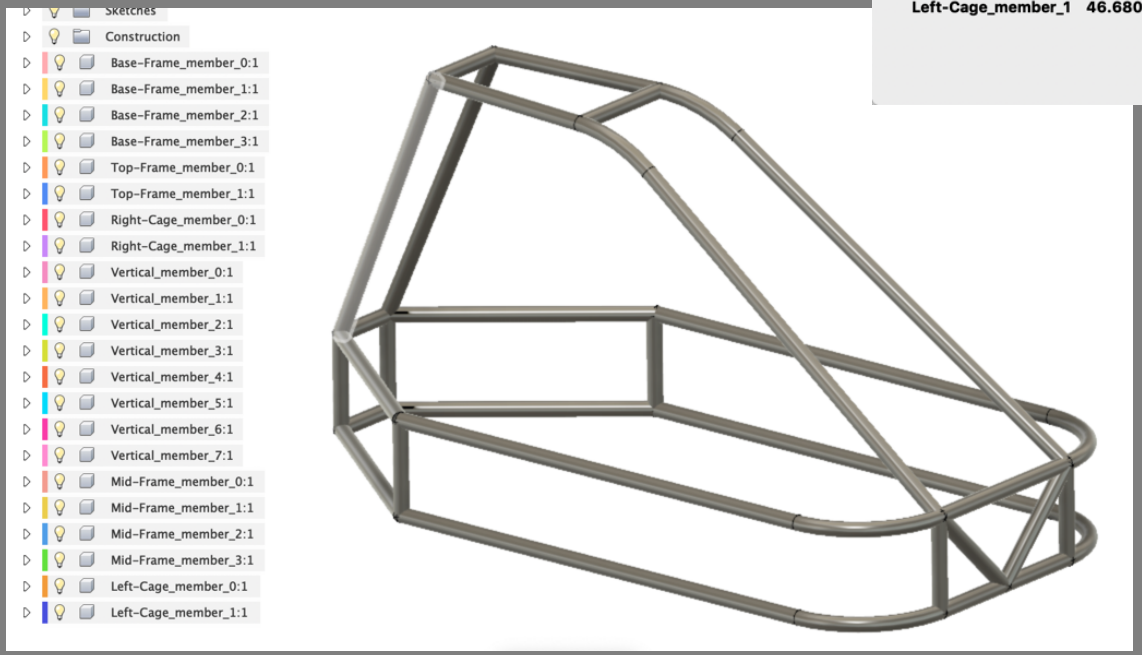
Piper

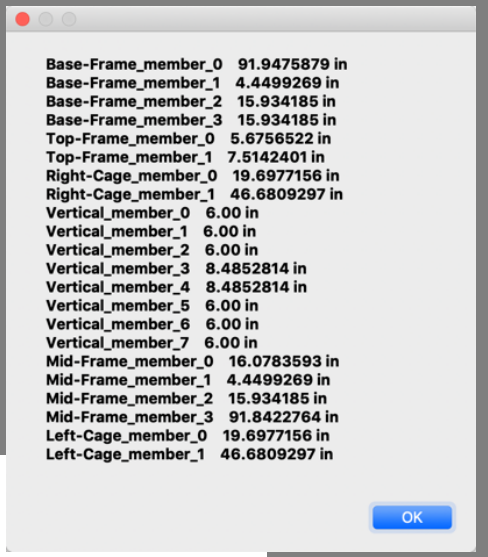
Overview:

- Automate creation of bent pipe
- Takes sketches as input
- Finds all individual pipe segments and adds pipe geometry

https://github.com/tapnair/FusionPiper







Fusion Bolter

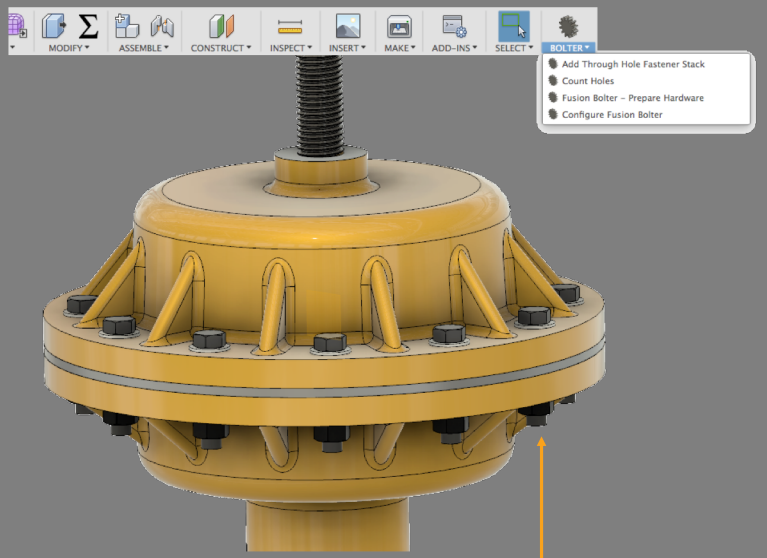
Hardware_Sizes

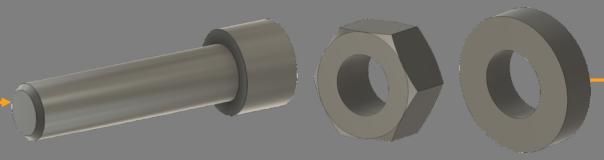
File Edit View Insert Format Data Tools Add-ons Help Accessibility Last edit was made on November 10, 2017 by Patrick Rainsberry

number

	A	B	C	D	E	F	G	H	I	J
1	number	screwsize	diameter	hexkey	headdiameter	headheight	min	max	radius	headradius
2	5267123523	No. 0	0.06000	0.05000	0.09600	0.06000	0.06000	0.07300	0.03	0.048
3	5267123524	No. 1	0.07300	0.06250	0.11800	0.07300	0.07300	0.08600	0.0365	0.059
4	5267123525	No. 2	0.08600	0.07813	0.14000	0.08600	0.08600	0.09900	0.043	0.07
5	5267123526	No. 3	0.09900	0.07813	0.16100	0.09900	0.09900	0.11200	0.0495	0.0805
6	5267123527	No. 4	0.11200	0.09375	0.18300	0.11200	0.11200	0.12500	0.056	0.0915
7	5267123528	No. 5	0.12500	0.09375	0.20500	0.12500	0.12500	0.13800	0.0625	0.1025
8	5267123529	No. 6	0.13800	0.10938	0.22600	0.13800	0.13800	0.16400	0.069	0.113
9	5267123530	No. 8	0.16400	0.14063	0.27000	0.16400	0.16400	0.19000	0.082	0.135
10	5267123531	No. 10	0.19000	0.15625	0.31200	0.19000	0.19000	0.21600	0.095	0.156
11	5267123532	No. 12	0.21600	0.15625	0.34500	0.21600	0.21600	0.25000	0.108	0.1725
12	5267123533	1/4"	0.25000	0.18750	0.37500	0.25000	0.25000	0.31300	0.125	0.1875
13	5267123534	5/16"	0.31300	0.25000	0.46900	0.31300	0.31300	0.37500	0.1565	0.2345
14	5267123535	3/8"	0.37500	0.31250	0.56300	0.37500	0.37500	0.43800	0.1875	0.2815
15	5267123536	7/16"	0.43800	0.37500	0.65600	0.43800	0.43800	0.50000	0.219	0.328
16	5267123537	1/2"	0.50000	0.37500	0.75000	0.50000	0.50000	0.56300	0.25	0.375
17	5267123538	9/16"	0.56300	0.43750	0.84400	0.56300	0.56300	0.62500	0.2815	0.422
18	5267123539	5/8"	0.62500	0.50000	0.93800	0.62500	0.62500	0.75000	0.3125	0.469
19	5267123540	3/4"	0.75000	0.62500	1.12500	0.75000	0.75000	0.87500	0.375	0.5625
20	5267123541	7/8"	0.87500	0.75000	1.31300	0.87500	0.87500	1.00000	0.4375	0.6565
21	5267123542	1"	1.00000	0.75000	1.50000	1.00000	1.00000	1.12500	0.5	0.75
22	5267123543	1-1/8"	1.12500	0.87500	1.68800	1.12500	1.12500	1.25000	0.5625	0.844
23	5267123544	1-1/4"	1.25000	0.87500	1.87500	1.25000	1.25000	1.50000	0.625	0.9375
24	5267123545	1-1/2"	1.50000	1.00000	2.25000	1.50000	1.50000	1.75000	0.75	1.125
25	5267123546	1-3/4"	1.75000	1.25000	2.62500	1.75000	1.75000	2.00000	0.875	1.3125
26	5267123547	2"	2.00000	1.50000	3.00000	2.00000	2.00000	2.25000	1	1.5
27										
28										

Master | SAE Washer W | SAE Washer N | SAE Washer R | Socket Head Cap Screw | Hex Nut - US | Hex Bolt | NEW SAE WASHER





Hardware Sizes stored in Google Sheets

Analyze model to find holes

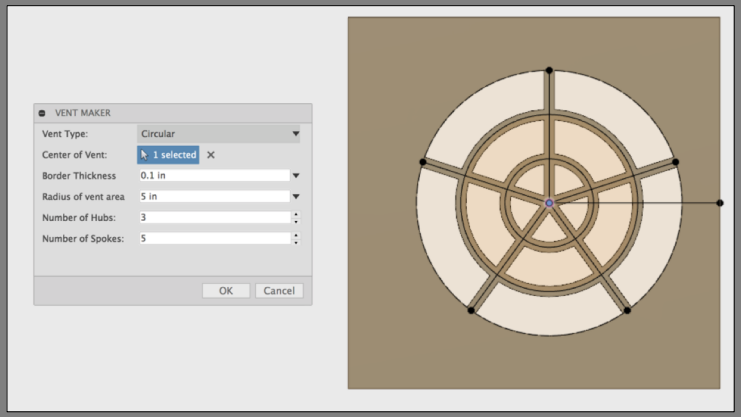
Place hardware and apply correct size

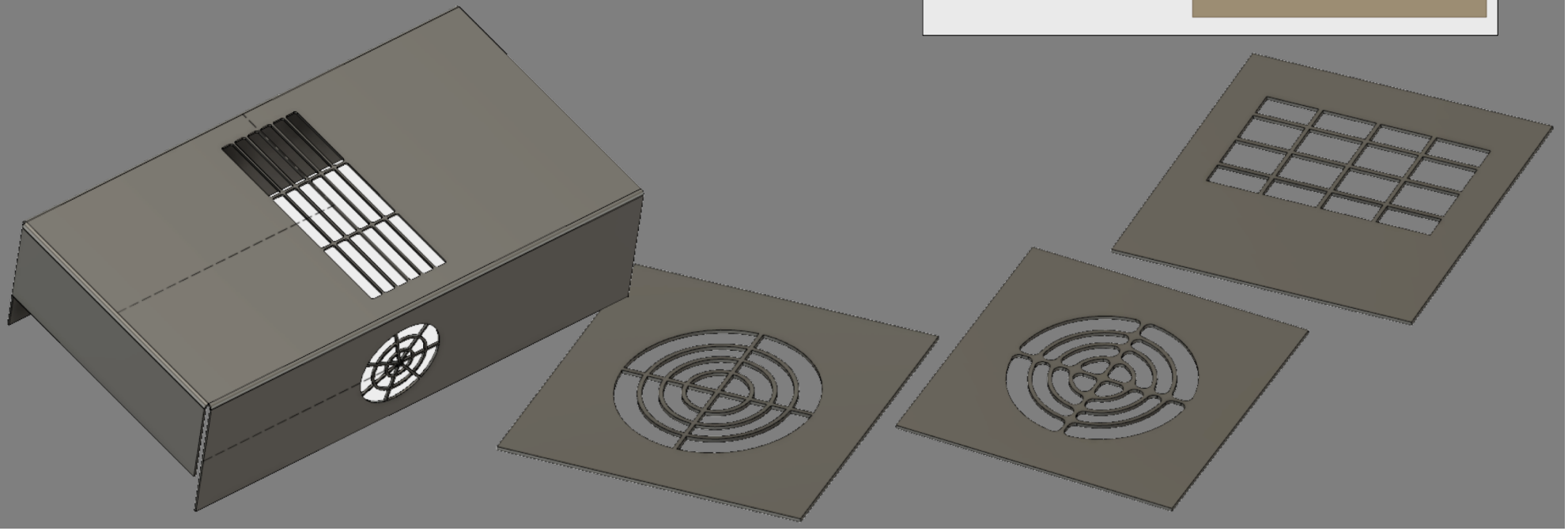
Vent Maker

Overview:

- Automate vent creation
- Take user inputs and create vent shapes

https://github.com/tapnair/ventMaker





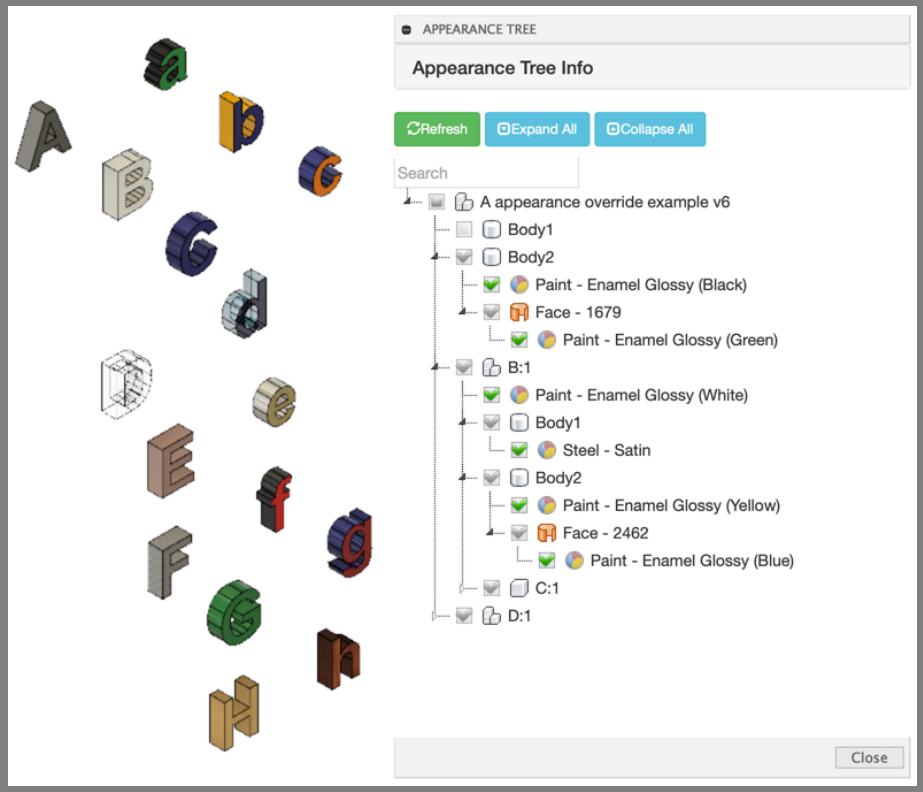
User Interfaces

Appearance Utility

Overview:

- Understand Appearance and Material overrides in a Fusion 360 Design
- Uses Fusion 360 Web Palettes
- Simple jquery structure on page

https://github.com/tapnair/AppearanceUtility



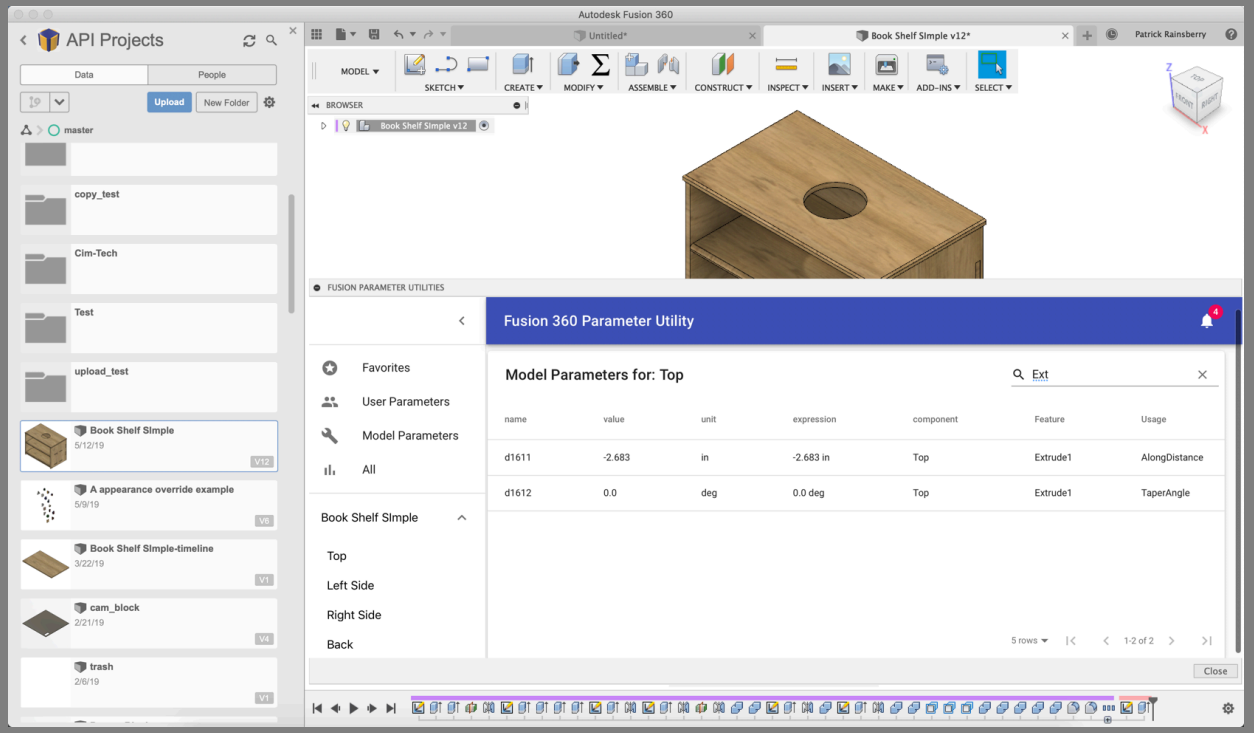
Parameter Utility

Overview:

Tools to view and edit parameters in a Fusion 360 Design

Uses React and material UI for to create a very elegant interface

As yet, unpublished



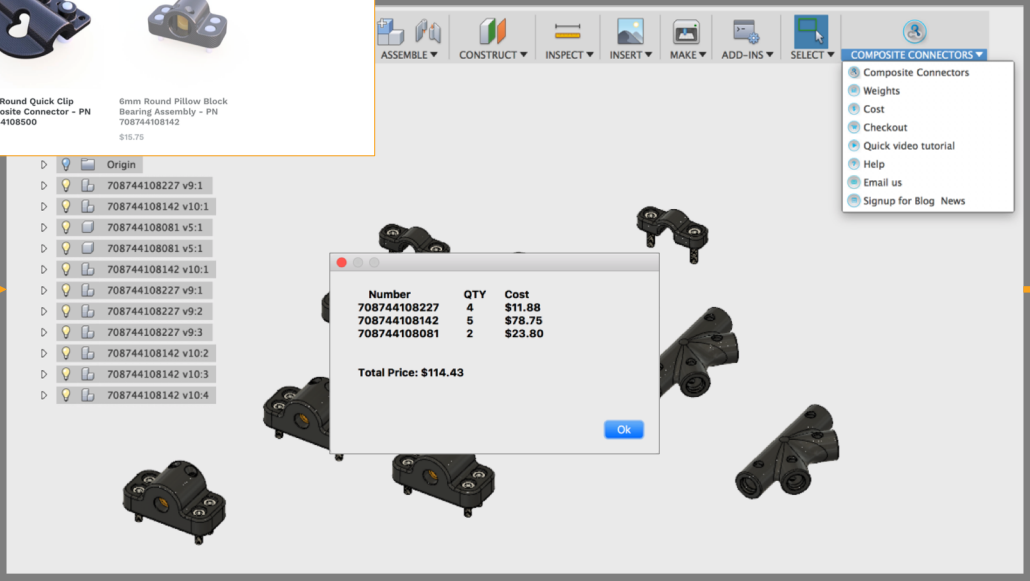
Connecting to External Systems

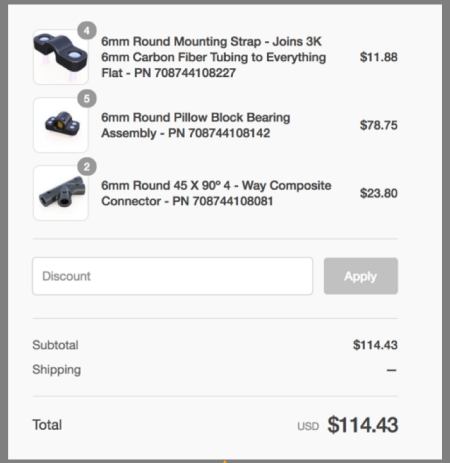
Overnight Composites - Design to Cart

Designers insert models directly from Overnight Composites web store

Fusion integration allows for cost calculation

Direct to cart button populates BOM in shopify shopping cart for purchase





Octoprint Integration

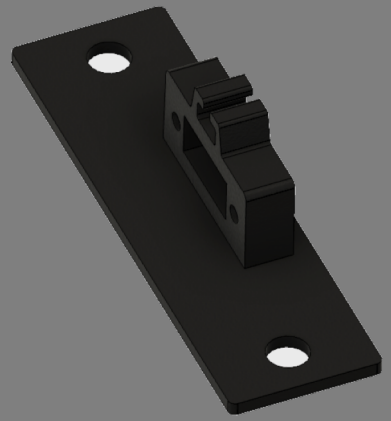
Exports model and pushes to Machine controller

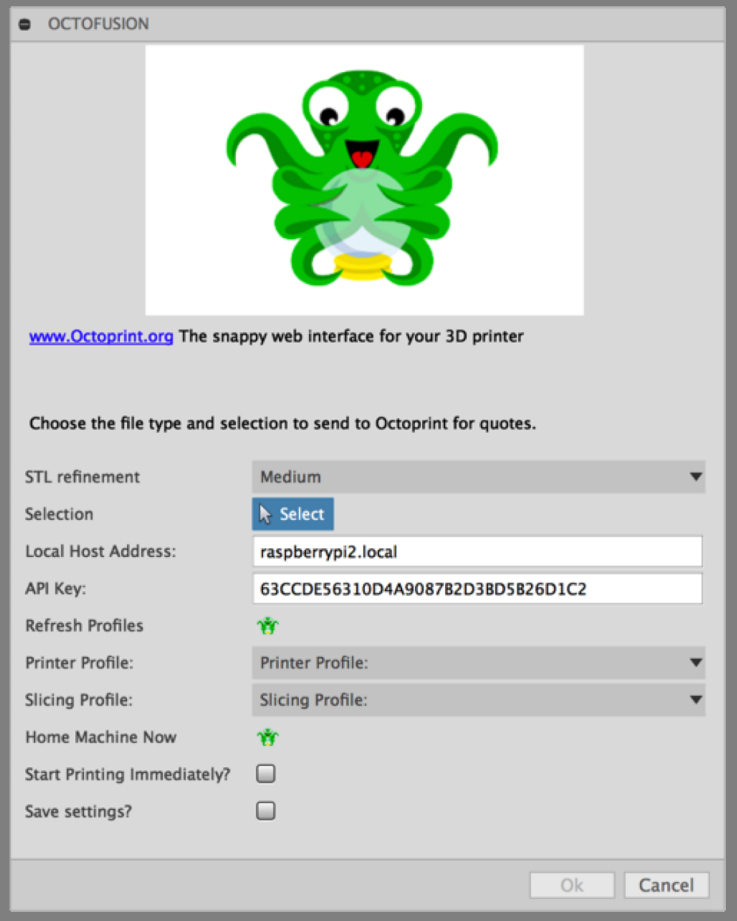
Leverages cloud slicing (g code generation)

Basic machine control within app

Machine configurations read on demand

Starts print from Fusion 360





Same principles could apply to more sophisticated controllers

Getting Help

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.

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

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

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

AUTODESK®
FUSION 360™
ACADEMY

Appendix

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

Dogbone - Create dog-bone fillets. Can create individual or automatically for entire assembly.

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

Command Inputs Samples

```
# Create a few inputs in the UI
inputs.addValueInput('value_input', '*Sample* Value Input', ao.units_manager.defaultLengthUnits,
                     default_value)

inputs.addBoolValueInput('bool_input', '*Sample* Check Box', True)
inputs.addStringValueInput('string_input', '*Sample* String Value', 'Some Default Value')
inputs.addSelectionInput('selection_input', '*Sample* Selection', 'Select Something')

# Create a Dropdown
drop_down_input = inputs.addDropDownCommandInput('drop_down_id', 'MY Dropdown',
                                                 adsk.core.DropDownStyles.TextListDropDownStyle);
drop_down_items = dropdownInput4.listItems
drop_down_items.add('VARIABLE_1, True, '')
drop_down_items.add(VARIABLE_2 False, '')

# Remove a specific item
for drop_item in drop_down_items:
    if drop_item.name = VARIABLE_1:
        drop_item.deleteMe()

# Remove all items:
drop_down_items.clear()
```