

# Fusion 360 API – программный интерфейс для прикладных задач

А.Ю. Стремнев, к.т.н. (БГТУ им. В.Г. Шухова)

nml2351@yandex.ru

Autodesk Fusion 360 – современная, динамично развивающаяся платформа для представления конструкторских идей в электронном формате. Система предоставляет среды для трехмерного геометрического моделирования, визуализации, инженерного анализа, программирования обработки на станках с ЧПУ, подготовки документации и обеспечения коллективной работы. Всё удобно, практично, функционально. Чего же еще пожелать!? Может быть, более полного контроля? Иногда, кажется, программа сама делает то, что только зарождается в мыслях проектировщика...

Интригует, правда, кнопка  (*Scripts and Add-Ins*), скромно расположившаяся справа на панели инструментов.

Скрипты, дополнения, программные коды ... Нужно ли в них “копаться”, когда всё и так работает неплохо? Попробуем разобраться. А для этого поставим несложную прикладную задачу: автоматически строить трехмерную модель цветка (рис. 1) по заданному максимальному диаметру (*pDF*).

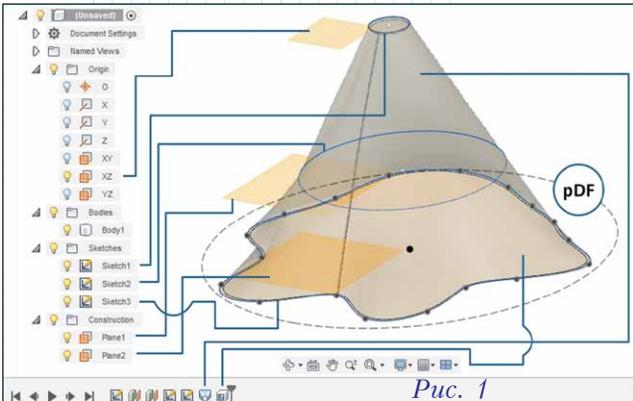


Рис. 1

Начинаем с запуска упомянутой команды *Scripts and Add-Ins*. В открывшемся окне на вкладке *Scripts* щелкаем по кнопке *Create* (рис. 2).

В следующем окне (рис. 3) выбираем язык программирования (в нашем случае *Python*), даем скрипту название (пусть будет *Flower*), вводим его краткое описание (*Flower Generator*)

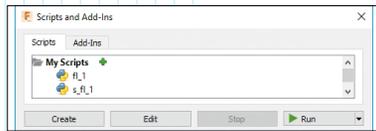


Рис. 2

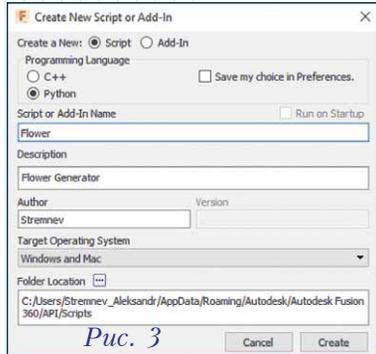


Рис. 3

и указываем автора. Особое внимание следует уделить полю *Folder Location*, где указывается расположение не только самого скрипта, но и всех необходимых для проекта папок и файлов.

Помимо файла собственно скрипта (*Flower.py*) в нашем проекте понадобятся изображение для текстурирования модели, а также текстовый документ для записи и считывания главного параметра – диаметра цветка. Таким образом, структура каталога проекта (*Flower*) будет иметь вид, представленный на рис. 4. В папке *s* находится текстовый файл для хранения текущего значения диаметра бутона (в сантиметрах), а каталог *m* предназначен для графического файла текстуры.

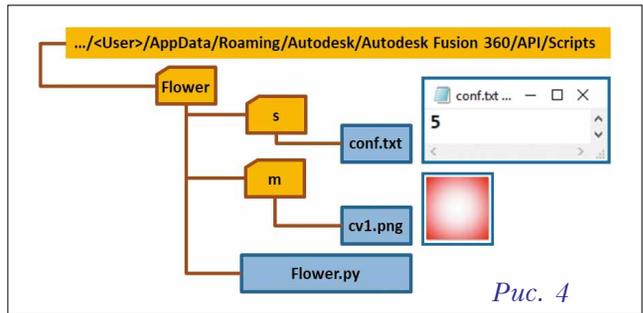


Рис. 4

Далее в окне создания скрипта щелкаем по кнопке *Create*. Найдем в общем списке созданный скрипт (рис. 5), выделим его и нажмем кнопку *Edit* (готовый скрипт можно будет запускать командой *Run*).

При первом использовании скриптов на языке *Python* система *Fusion 360* предложит загрузить бесплатную среду разработки – *Spyder*. В дальнейшем (после установки) эта среда будет автоматически запускаться при открытии скриптов на редактирование (рис. 6).

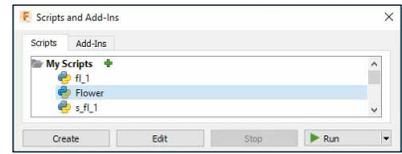


Рис. 5

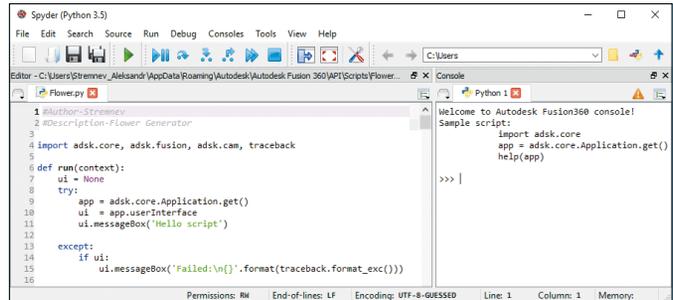


Рис. 6

Инструментарий *Spyder* включает в себя все необходимые возможности для отладки кода. Сохранение текущего модуля производится командой *Save File* (📁), а запуск на выполнение – командой *Run File* (▶).

При создании нового скрипта автоматически создается шаблон кода, включающий справочные метаданные, блок ссылок (*import*) на необходимые библиотеки и заготовку стартовой процедуры (*run*).

Теперь внесем изменения в шаблон кода (см. **фрагмент 1**).

Следует обратить внимание на структуру *try ... except*, которая позволяет контролировать появление ошибочных ситуаций в блоке *try*, перенаправляя выполнение в *except*.

Далее в процедуре *run* необходимо описать и вызвать команду для генерации цветка (см. **фрагмент 2**).

### Фрагмент 1

```
#Author-Stremnev
#Description-Flower Generator
# Добавлена ссылка на библиотеку математических функций,
# Добавлена ссылка на файловые функции операционной системы
import adsk.core, adsk.fusion, traceback, math, os.path
# Добавлен импорт функции генерации случайных чисел
from random import random
def run(context):
    try:
        # Объявлены переменные, доступные во всем проекте
        global app, ui, mpath, spath
        app = adsk.core.Application.get()
        ui = app.userInterface
        # Переменная - путь к файлу параметров модели
        spath=os.path.join(os.path.dirname(os.path.realpath(__file__)), 's\conf.txt')
        # Переменная - путь к графическому файлу текстуры цветка
        mpath=os.path.join(os.path.dirname(os.path.realpath(__file__)), 'm\cv1.png')
    except:
        if ui:
            ui.messageBox('Ошибка:\n{}'.format(traceback.format_exc()))
```

### Фрагмент 2

```
# Создание переменной cmdDef для новой команды
cmdDef = ui.commandDefinitions.itemById('Flower')
if not cmdDef:
    # Задание окна команды с указанным названием
    cmdDef = ui.commandDefinitions.addButtonDefinition('Flower','Flower Generator',")
# Создание обработчика flowerHandler события выполнения команды cmdDef
onCommandCreated = flowerHandler()
cmdDef.commandCreated.add(onCommandCreated)
handlers.append(onCommandCreated)
# Вызов команды на выполнение
cmdDef.execute()
# Задание ожидания действий пользователя до завершения модуля
adsk.autoTerminate(False)
```

### Фрагмент 3

```
class flowerHandler(adsk.core.CommandCreatedEventHandler):
    def __init__(self):
        super().__init__()
    def notify(self, args):
        try:
            # Обращение к команде, связанной с обработчиком
            cmd = adsk.core.Command.cast(args.command)
            # Обращение к коллекции элементов управления, связанных с командой
            inputs = cmd.commandInputs
            # Чтение в переменную v1 диаметра бутона из файла данных
            try:
                f = open(spath)
                v1=int(f.readline())
                f.close()
            except:
                # Инициализация переменной при ошибке чтения файла
                v1=int(6)
            # Создание в окне команды 'слайдера' для выбора диаметра бутона
            inputs.addIntegerSliderCommandInput ('slider_1', 'Диаметр бутона, см', 2, 9)
            # Инициализация переменной для обращения к элементу-'слайдеру'
            slider = inputs.itemById('slider_1')
            # Задание шага 'слайдера'
            slider.spinStep=1
            # Инициализация 'слайдера' значением из файла данных
            slider.valueOne=v1
            # Создание обработчика нажатия кнопки ОК в окне генератора цветка
            onExecute = flowersDialogOKEventHandler()
            cmd.execute.add(onExecute)
            handlers.append(onExecute)
        except:
            ui.messageBox('Ошибка:\n{}'.format(traceback.format_exc()))
```

#### Фрагмент 4

```
class flowersDialogOKEventHandler(adsk.core.CommandEventHandler):
    def __init__(self):
        super().__init__()
    def notify(self, args):
        try:
            # Получение доступа к элементам управления окна генерации цветка
            command = args.firingEvent.sender
            slider = command.commandInputs.itemById('slider_1')
            # Чтение значения из 'слайдера' в переменную для диаметра бутона
            pDF = slider.valueOne
            # Инициализация переменной - высоты бутона цветка
            pHF = 4

            # Создание нового документа в Fusion 360
            app.documents.add(0)
            # Получение доступа к корневому компоненту модели
            design=app.activeProduct
            design.designType=adsk.fusion.DesignTypes.ParametricDesignType
            rootComp = design.rootComponent

            # Создание переменной для коллекции эскизов модели
            sketches = rootComp.sketches

            # Создание эскиза на плоскости XZ
            Sketch1 = sketches.add(rootComp.xZConstructionPlane)
            # Получение доступа к эскизным объектам типа 'окружность'
            circles = Sketch1.sketchCurves.sketchCircles
            # Создание эскизной окружности для основания бутона
            circles.addByCenterRadius(adsk.core.Point3D.create(0,0,0),pDF/20)
            # Сохранение эскизной окружности как первого профиля
            profile1 = Sketch1.profiles.item(0)

            # Получение доступа к рабочим плоскостям модели
            planes=rootComp.constructionPlanes
            # Подготовка переменной для параметров задания плоскости
            planeInput=planes.createInput()
            # Определение плоскости на расстоянии от существующей (XZ)
            offset= adsk.core.ValueInput.createByReal(0.65*pHF)
            planeInput.setByOffset(rootComp.xZConstructionPlane,offset)
            # Завершение создания плоскости для промежуточного эскиза
            plane1=planes.add(planeInput)
            # Создание плоскости для эскиза вершины бутона
            offset= adsk.core.ValueInput.createByReal(1*pHF)
            planeInput.setByOffset(rootComp.xZConstructionPlane,offset)
            plane2=planes.add(planeInput)

            # Создание эскизной окружности для промежуточного сечения бутона
            Sketch2 = sketches.add(plane1)
            circles2 = Sketch2.sketchCurves.sketchCircles
            circles2.addByCenterRadius(adsk.core.Point3D.create(0,0,0),pDF/4)
            profile2 = Sketch2.profiles.item(0)

            # Создание эскиза для верхнего контура бутона
            Sketch3 = sketches.add(plane2)
            # Создание коллекции точек
            points = adsk.core.ObjectCollection.create()
            n=18 # Задание количества точек верхнего контура
            # Цикл формирования точек эскизного контура
            for number in range(n):
                # Задание координат текущей точки со случайной вариацией
                RF = (0.4*random()+0.6)*pDF/2
                u=number*360/n
                c_x = RF * math.cos(u*math.pi/180)
                c_y = RF * math.sin(u*math.pi/180)
                # Добавление точки с найденными координатами в коллекцию
                points.add(adsk.core.Point3D.create(c_x, c_y, 0))
                # Добавление в коллекцию еще одной 'первой' точки как замыкающей
                points.add(points.item(0))
            # Создание контура в виде сплайна на основе коллекции точек
            Sketch3.sketchCurves.sketchFittedSplines.add(points)
            profile3 = Sketch3.profiles.item(0)
```

```

→ # Создание элемента 'лофта' по заданным эскизным профилям
loftFeats = rootComp.features.loftFeatures
loftVariant=adsk.fusion.FeatureOperations.NewBodyFeatureOperation
loftInput = loftFeats.createInput(loftVariant)
loftSectionsObj = loftInput.loftSections
loftSectionsObj.add(profile1)
loftSectionsObj.add(profile2)
loftSectionsObj.add(profile3)
loftInput.isSolid = True
# Контроль возможности реализации 'лофта'
try:
    loft1=loftFeats.add(loftInput)
except:
    ui.messageBox('Попробуйте перезапустить построение')

# Создание элемента 'оболочки'
entities1 = adsk.core.ObjectCollection.create()
# Удаление верхней грани 'лофта' бутона
entities1.add(loft1.endFace)
shellFeats = rootComp.features.shellFeatures
isTangentChain = True
shellFeatureInput = shellFeats.createInput(entities1, isTangentChain)
# Задание толщины стенок бутона
thickness = adsk.core.ValueInput.createByReal(0.04)
shellFeatureInput.insideThickness = thickness
# Контроль возможности реализации 'оболочки'
try:
    shellFeats.add(shellFeatureInput)
except:
    ui.messageBox('Попробуйте перезапустить построение')

# Доступ к объекту для окрашивания - 'первому' телу из структуры модели
body = rootComp.bRepBodies.item(0)
# Обращение к стандартной библиотеке материалов Fusion 360
matLibName='Fusion 360 Appearance Library'
fusionMaterials = app.materialLibraries.itemByName(matLibName)
# Чтение образца материала из библиотеки в переменную
sColor = fusionMaterials.appearances.itemByName('Oak')
# Создание копии материала
newColor = design.appearances.addByCopy(sColor, 'MyRedFlowerColor')
# Назначение материала телу (элементу модели) - бутону цветка
body.appearance = newColor
# Обращение к визуальным свойствам материала
prop = newColor.appearanceProperties.itemById("opaque_albedo")
tex = prop.connectedTexture.properties
# Связывание графического файла с текстурой
tex.itemById("unifiedbitmap_Bitmap").value = mpath
# Задание масштабирования текстуры
tex.itemById("texture_ScaleLock").value = False
tex.itemById("texture_RealWorldScaleX").value = 2
tex.itemById("texture_RealWorldScaleY").value = 2
# Задание 'мощения' (повторения) текстуры по модели
tex.itemById("texture_URepeat").value = True
tex.itemById("texture_VRepeat").value = True

# Скрытие всех эскизов модели в рабочей области
for each_sketch in sketches:
    each_sketch.isVisible=False
# Отображение модели цветка на весь размер рабочей области
app.activeViewport.fit()

# Сохранение текущего значения диаметра бутона в файле данных
try:
    f = open(spath, 'w')
    f.write(str(pDF) + '\n')
    f.close()
except:
    ui.messageBox('Ошибка записи в файл')

except:
    if ui:
        ui.messageBox('Ошибка выполнения команды')

```

Перед процедурой *run* нужно не забыть проинициализировать пустым значением переменную для обработчиков событий.

```
handlers = []
```

Созданный в процедуре *run* обработчик *flowerHandler* необходимо описать. Обработчик должен формировать окно команды для генерации цветка с необходимыми элементами управления. Для этого после процедуры *run* создадим соответствующий класс (см. фрагмент 3).

Теперь следует описать обработчик нажатия кнопки *OK* в окне генератора. Это будет основная программная часть нашего проекта. Именно здесь будет формироваться геометрия модели цветка по выбранному диаметру бутона, и здесь же мы определим параметры отображаемого материала (см. фрагмент 4).

Далее мы сохраняем скрипт и запускаем его одним из описанных выше способов. В открывшемся диалоговом окне *Flower Generator* задаем диаметр бутона, нажимаем *OK* и любуемся результатом в рабочем поле *Fusion 360* (рис. 7).

При следующем запуске скрипта диаметр бутона из предыдущего сеанса будет загружаться в слайдер. В дальнейшем пользователь сможет легко генерировать необходимые ему конфигурации модели (рис. 8).

Итак, поставленная задача решена – скрипт автоматически генерирует управляемую параметрами модель цветка, в форму которого вносятся элементы случайной трансформации. При этом нам не приходится вручную выполнять множество рутинных операций по созданию вспомогательных плоскостей, эскизов и конструктивных элементов. Всё это особенно актуально, если, например, необходимо смоделировать букет цветов или небольшую клумбу. В такой ситуации можно использовать сохраненные модели отдельных цветков или, если вы уже “почувствовали” код, модернизировать

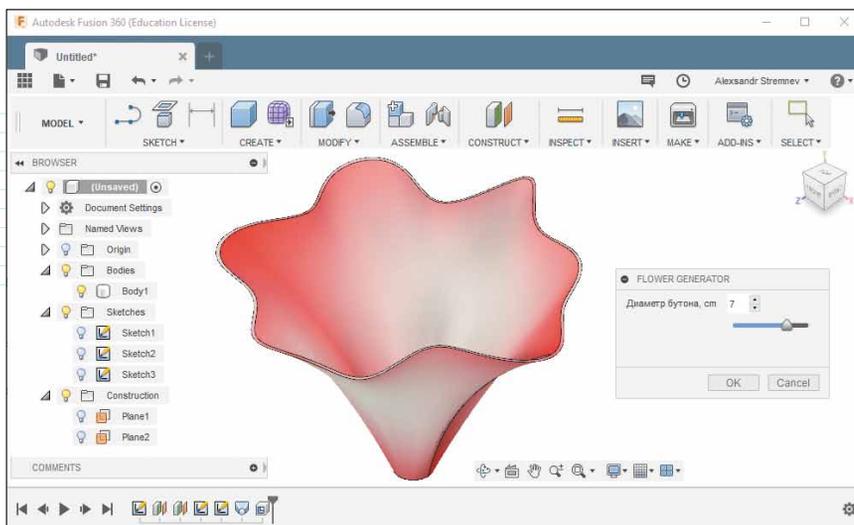


Рис. 7

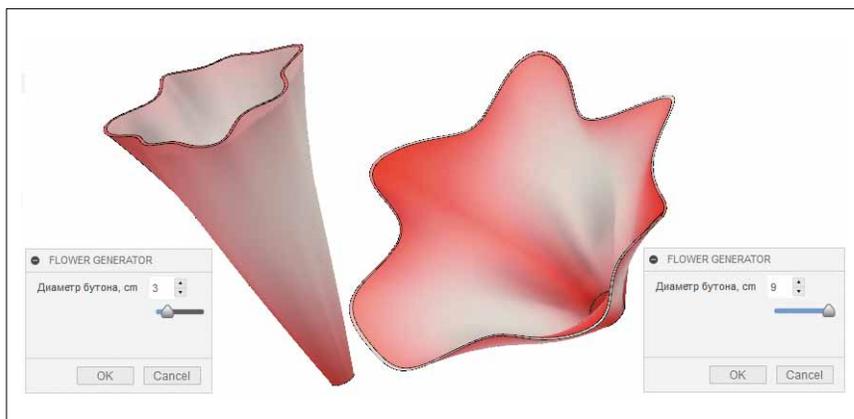


Рис. 8

описанный выше скрипт для генерации многокомпонентной сборки. Найти его можно в магазине приложений *Autodesk* [6].

### Полезные ссылки

1. Официальная справочная система Fusion 360 // <http://help.autodesk.com/view/fusion360/ENU/?guid=GUID-A92A4B10-3781-4925-94C6-47DA85A4F65A>
2. Объектная модель Fusion 360 // <http://help.autodesk.com/cloudhelp/ENU/Fusion-360-API/images/Fusion.pdf>
3. Fusion 360 API and Scripts (официальный форум) // <https://forums.autodesk.com/t5/fusion-360-api-and-scripts/bd-p/22>
4. Сообщество программистов Autodesk в СНГ // <http://adn-cis.org/fusion-360-api>
5. Разработка приложений для 3D-проектирования (учебный курс) // <https://academy.autodesk.com/course/119472/develop-an-app-for-3d-design-automation>
6. <https://apps.autodesk.com/FUSION/en/Detail/Index?id=5294598418314539609>

### Об авторе

**Александр Юрьевич Стремнев** – канд. техн. наук, доцент кафедры информационных технологий Белгородского государственного технологического университета им. В.Г. Шухова.