BCA Maker space Mr. Barbetta's MakerSpace

"Measure Once, Cut Twice"

Autodesk Fusion API

BOM (Bill of Materials) and STL Creator

Many advanced software programs implement an API (Applications Programming Interface), which comprise a set of programming functions and objects, that allow one to write code in a pramming language to extend the functionality of the software. The Fusion API can be used with programs written in either C++ or Python, which Fusion refers to as a ***script***. One can download a script written by Autodesk or independent programmers or one can write their own. The script described in this document was written for missile systems to automate the process of creating a BOM (Bill of Materials) and generating the STL files for 3D printing components.

Notice of liability:

This script is intended for missile development. The author assumes full responsibility for any mishaps.

Below is the top of the script window when run for the classroom missile launcher. It shows that there are 431 parts in total and 192 different components and 70 3D printed parts.

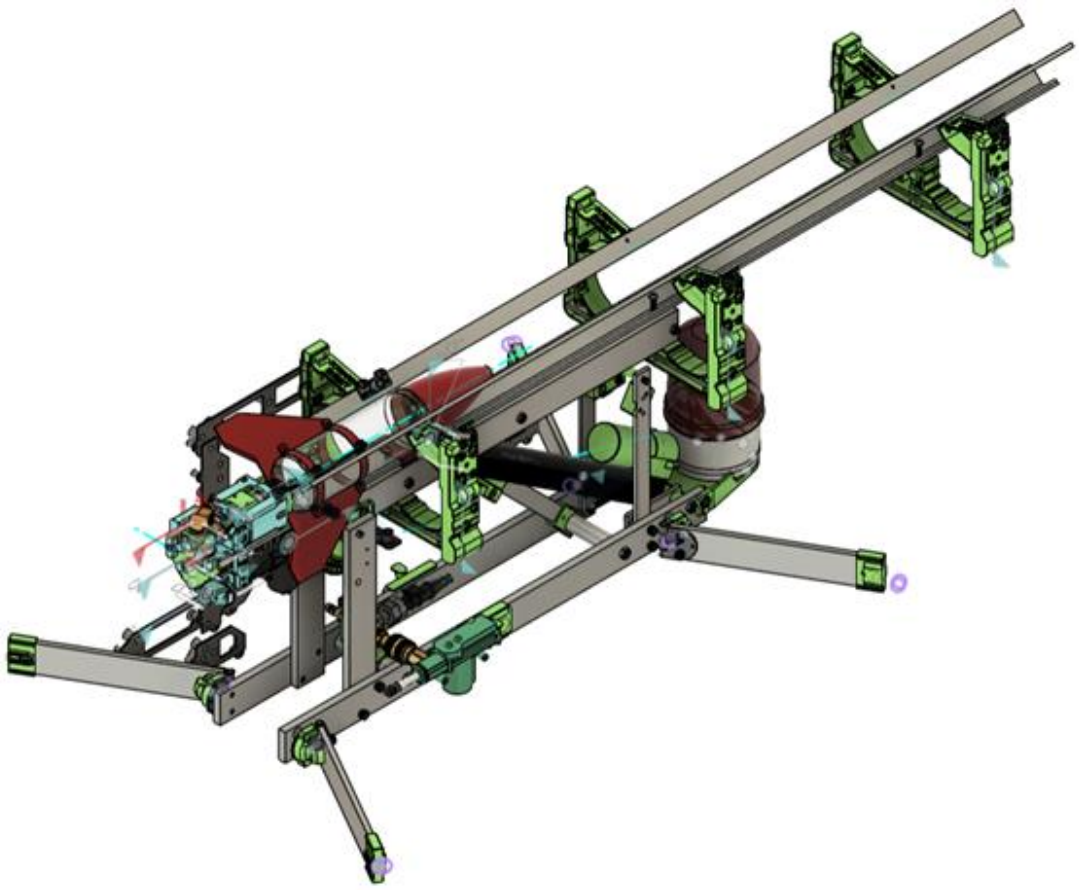
DESIGN BOM CREATOR

Create BOM | Save BOM | Export STL Files | Preferences | Help

Status: Finished: Component occurences=431 Components=192 Type matches=75 BOM items=192 Ignored=0 Prints=70

Click to Create BOM

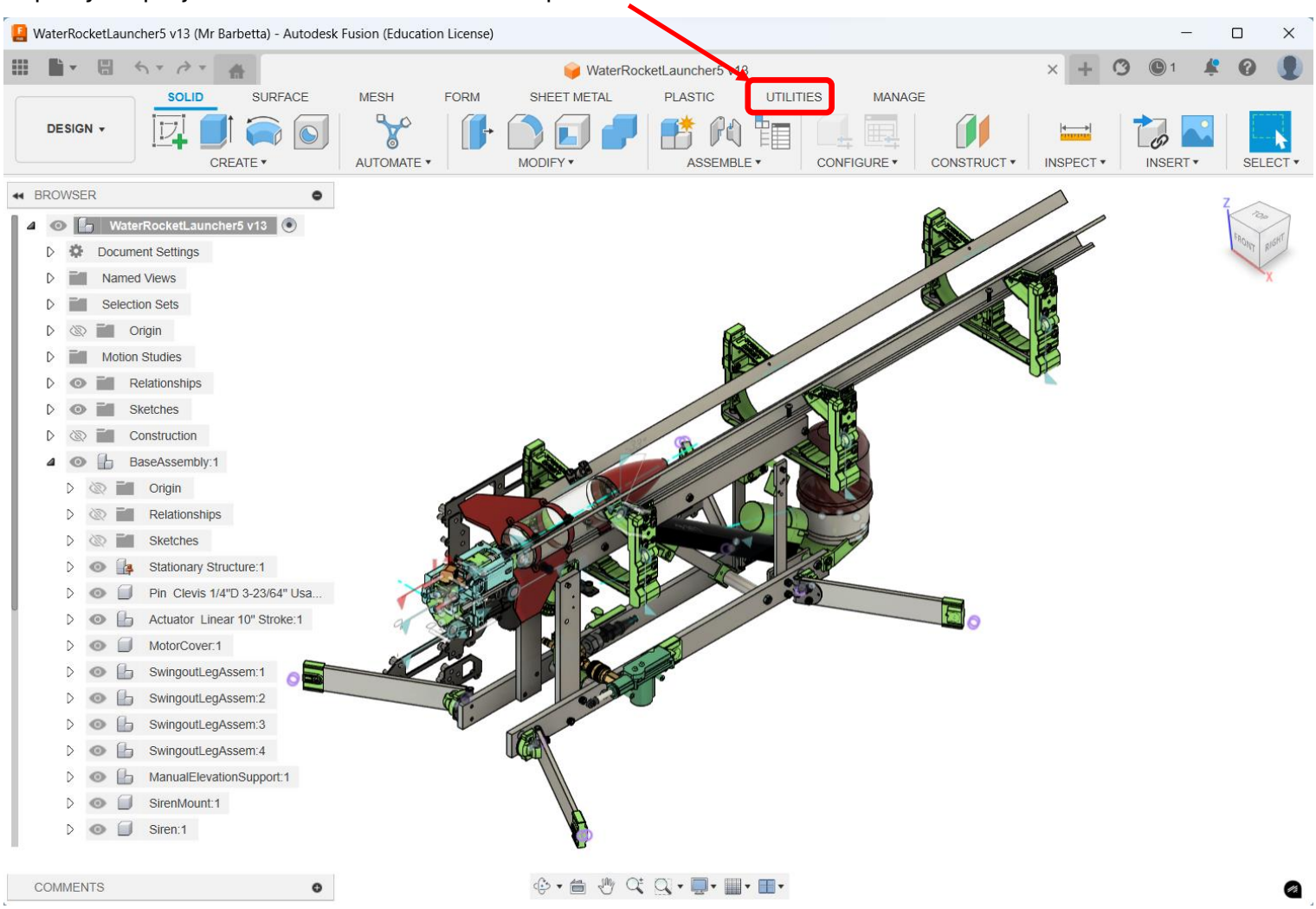
Line	Qty	Type	Description	Part Num	Price
001	14	screw	1/4"-20 3/4"L Socket-Head 18-8SS Black-Oxide	96006A706	$10.74(25)
002	02	screw	1/4"-20 1-1/4" SocketHead 316SS	92185A544	$4.81(10)
003	08	screw	1/4"-20 3/8"L Socket-Head 18-8SS Black-Oxide	96006A702	$7.52(25)



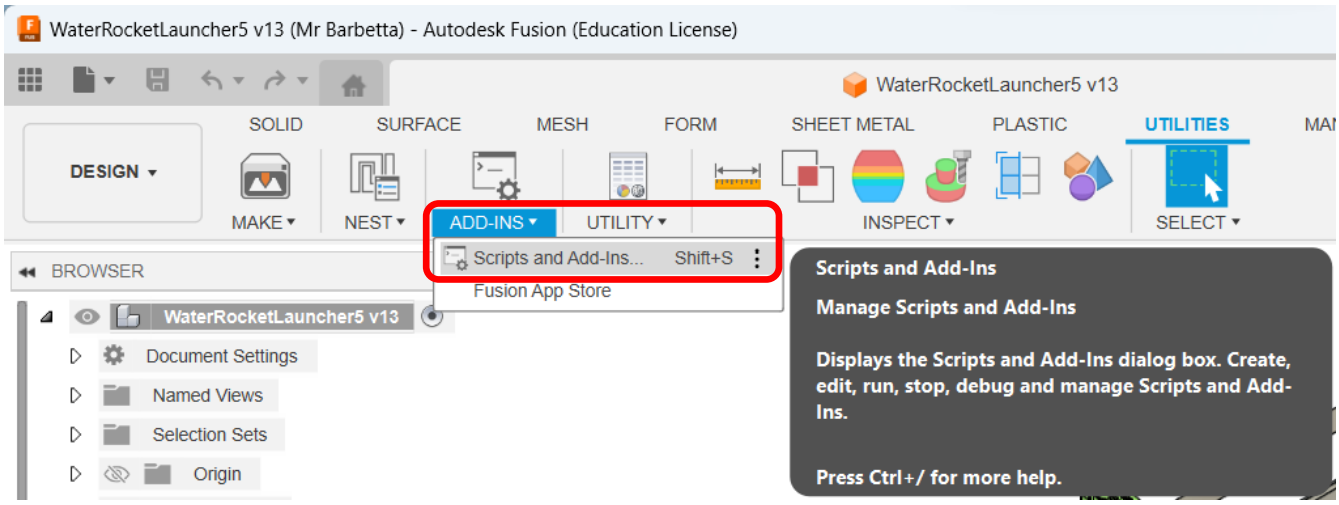
Contents

Acessing the Script

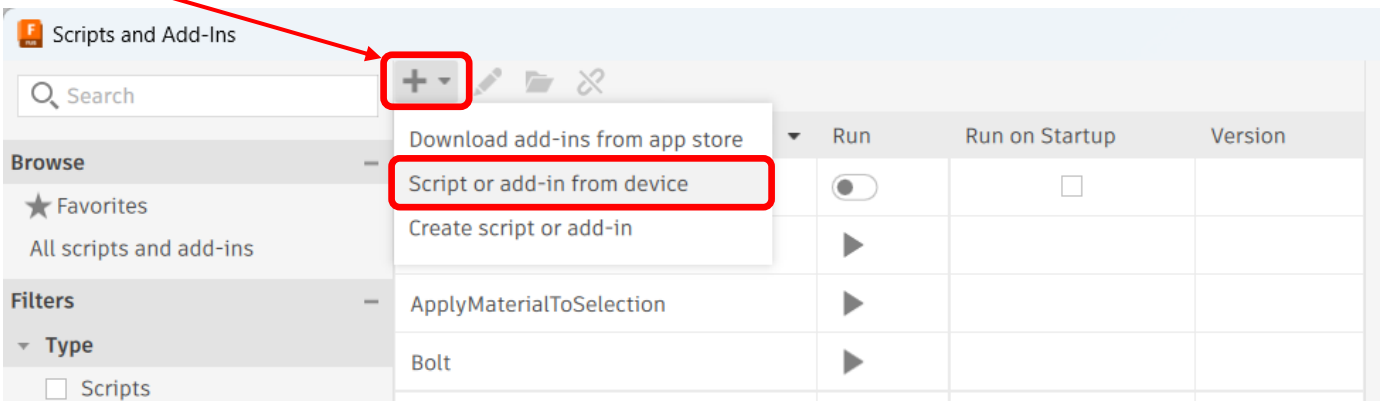
- open your project in Fusion and select the top **Utilities** item



- from the **ADD-INS** menu, select **Scripts and Add-Ins**

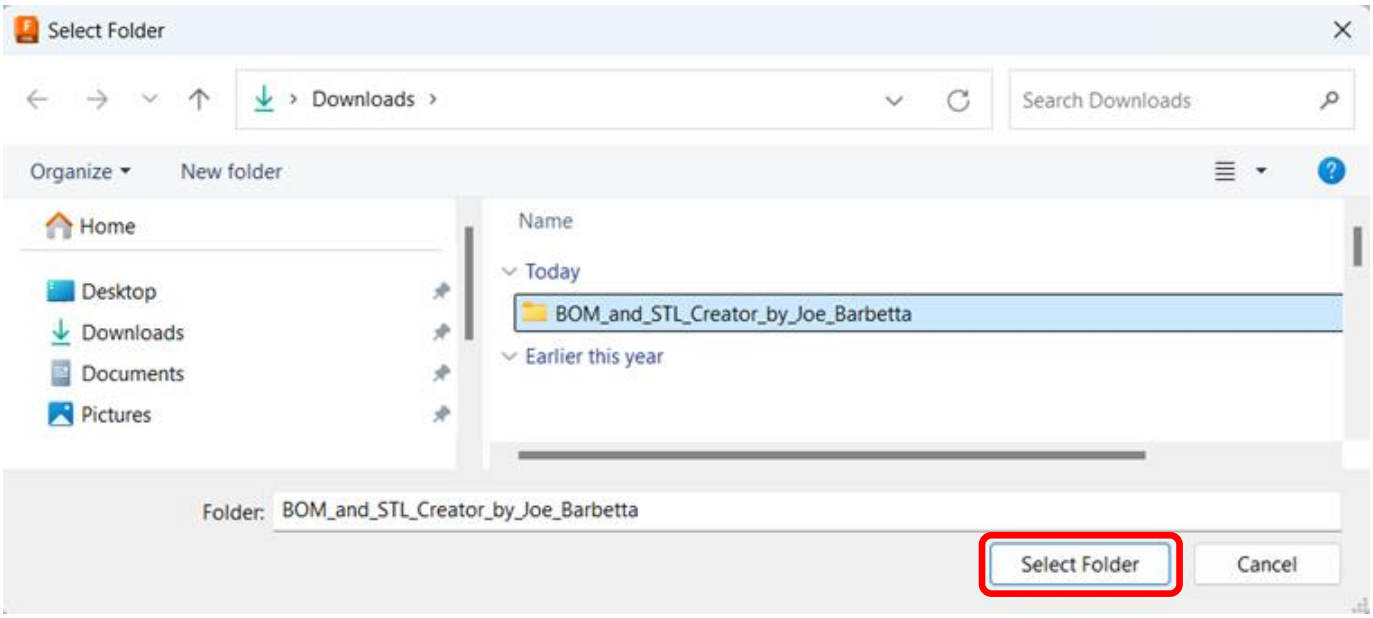


- from the **+** pull-down menu, select **Script or add-in from device**



- navigate to the location, select the Add-In folder, and click **Select Folder**

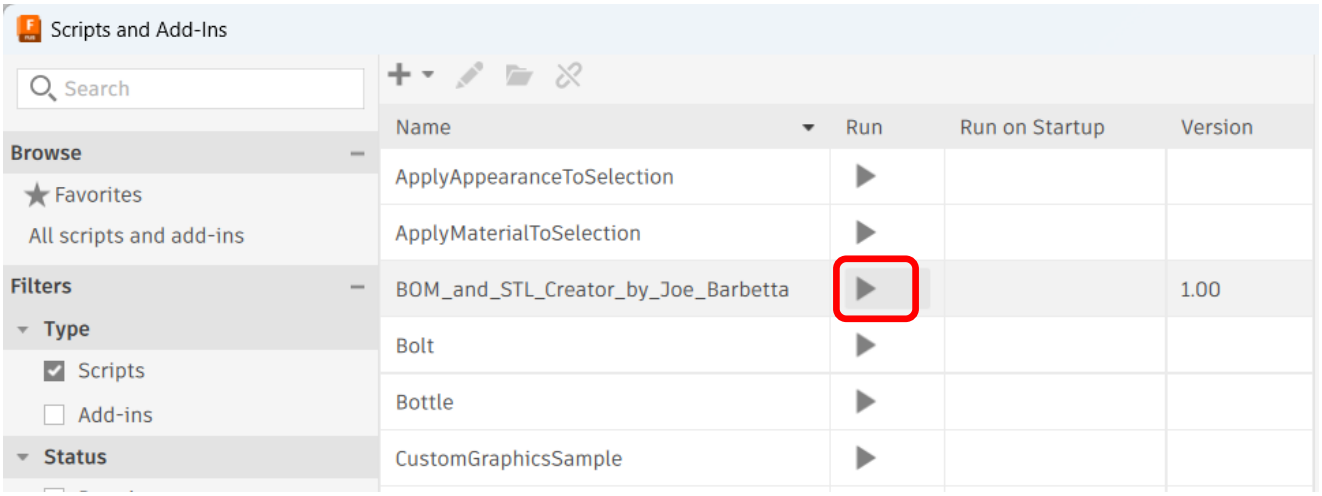
Below, the folder "**BOM_and_STL_Creator_by_Joe_Barbetta**" is shown in the Downloads folder.



- click on the **Play** icon next to the script name

For future use, the script should appear in the list upon opening the **Scripts and Add-Ins** window.

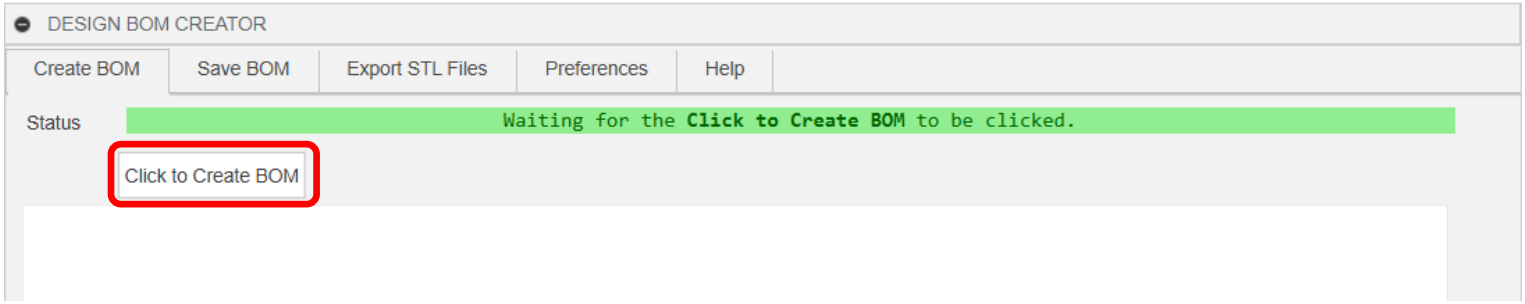
A Python program is often called a **Script** and hence the use of the term. When Play is clicked, the Python script, **BOM_and_STL_Creator_by_Joe_Barbetta.py**, will be run.



Creating the BOM

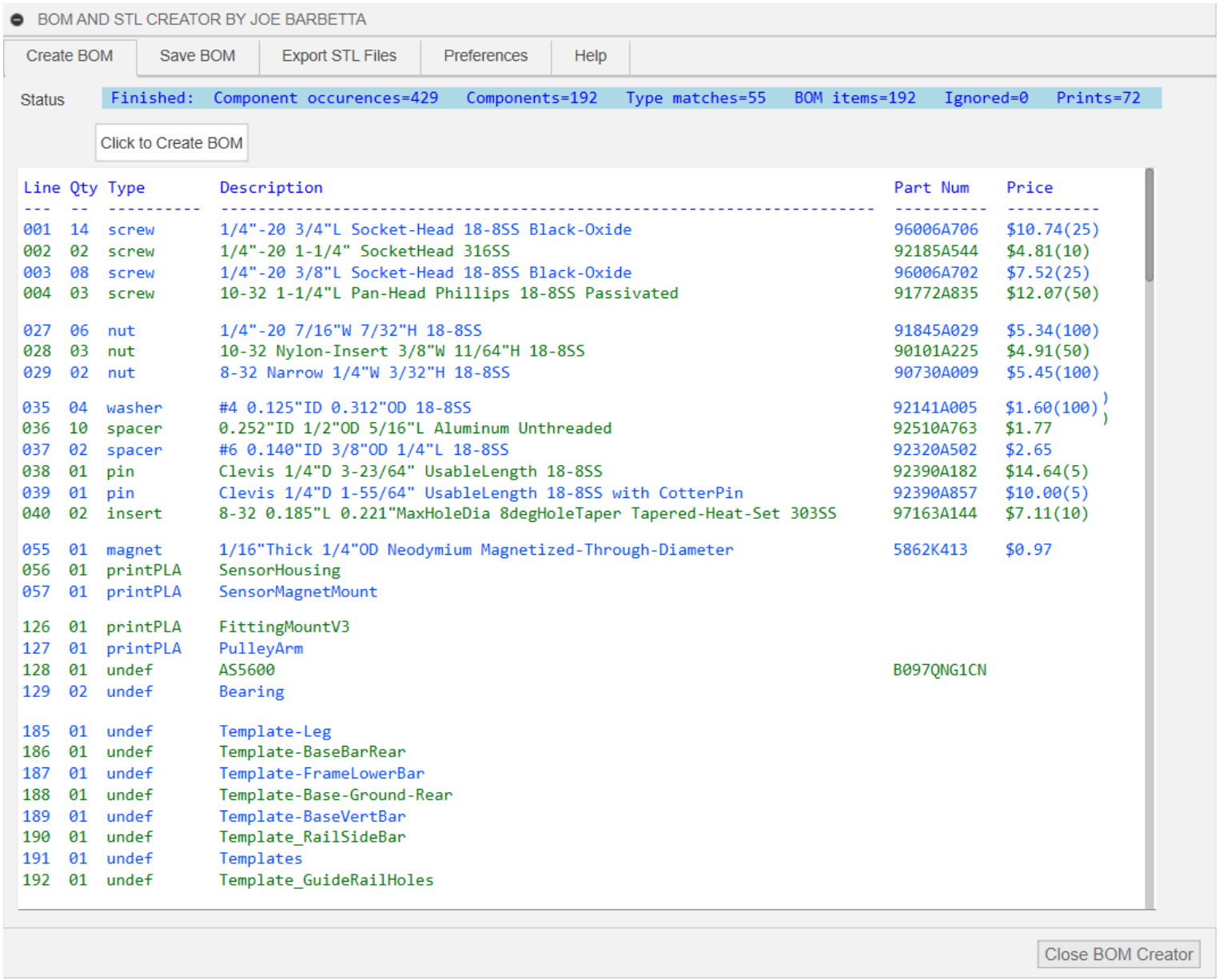
The Python script will show the below window.

- click on the **Click to Create BOM** button



This is the result from the rocket launcher design with **164 lines removed** to show how the script groups the parts according to **type**.

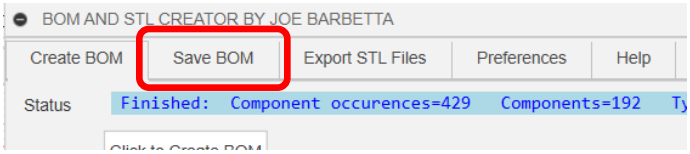
- One can click on the **Save BOM** tab to save the output to a file. See next page. There is also a tab to set **Preferences** and a tab for **Help.**



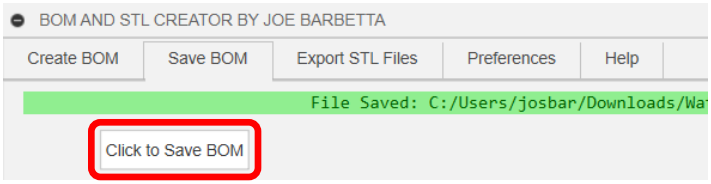
Line	Qty	Type	Description	Part Num	Price
001	14	screw	1/4"-20 3/4"L Socket-Head 18-8SS Black-Oxide	96006A706	$10.74(25)
002	02	screw	1/4"-20 1-1/4" SocketHead 316SS	92185A544	$4.81(10)
003	08	screw	1/4"-20 3/8"L Socket-Head 18-8SS Black-Oxide	96006A702	$7.52(25)
004	03	screw	10-32 1-1/4"L Pan-Head Phillips 18-8SS Passivated	91772A835	$12.07(50)
027	06	nut	1/4"-20 7/16"W 7/32"H 18-8SS	91845A029	$5.34(100)
028	03	nut	10-32 Nylon-Insert 3/8"W 11/64"H 18-8SS	90101A225	$4.91(50)
029	02	nut	8-32 Narrow 1/4"W 3/32"H 18-8SS	90730A009	$5.45(100)
035	04	washer	#4 0.125"ID 0.312"OD 18-8SS	92141A005	$1.60(100)
036	10	spacer	0.252"ID 1/2"OD 5/16"L Aluminum Unthreaded	92510A763	$1.77
037	02	spacer	#6 0.140"ID 3/8"OD 1/4"L 18-8SS	92320A502	$2.65
038	01	pin	Clevis 1/4"D 3-23/64" UsableLength 18-8SS	92390A182	$14.64(5)
039	01	pin	Clevis 1/4"D 1-55/64" UsableLength 18-8SS with CotterPin	92390A857	$10.00(5)
040	02	insert	8-32 0.185"L 0.221"MaxHoleDia 8degHoleTaper Tapered-Heat-Set 303SS	97163A144	$7.11(10)
055	01	magnet	1/16"Thick 1/4"OD Neodymium Magnetized-Through-Diameter	5862K413	$0.97
056	01	printPLA	SensorHousing		
057	01	printPLA	SensorMagnetMount		
126	01	printPLA	FittingMountV3		
127	01	printPLA	PulleyArm		
128	01	undef	AS5600	B097QNG1CN	
129	02	undef	Bearing		
185	01	undef	Template-Leg		
186	01	undef	Template-BaseBarRear		
187	01	undef	Template-FrameLowerBar		
188	01	undef	Template-Base-Ground-Rear		
189	01	undef	Template-BaseVertBar		
190	01	undef	Template_RailSideBar		
191	01	undef	Templates		
192	01	undef	Template_GuideRailHoles		

Saving the BOM

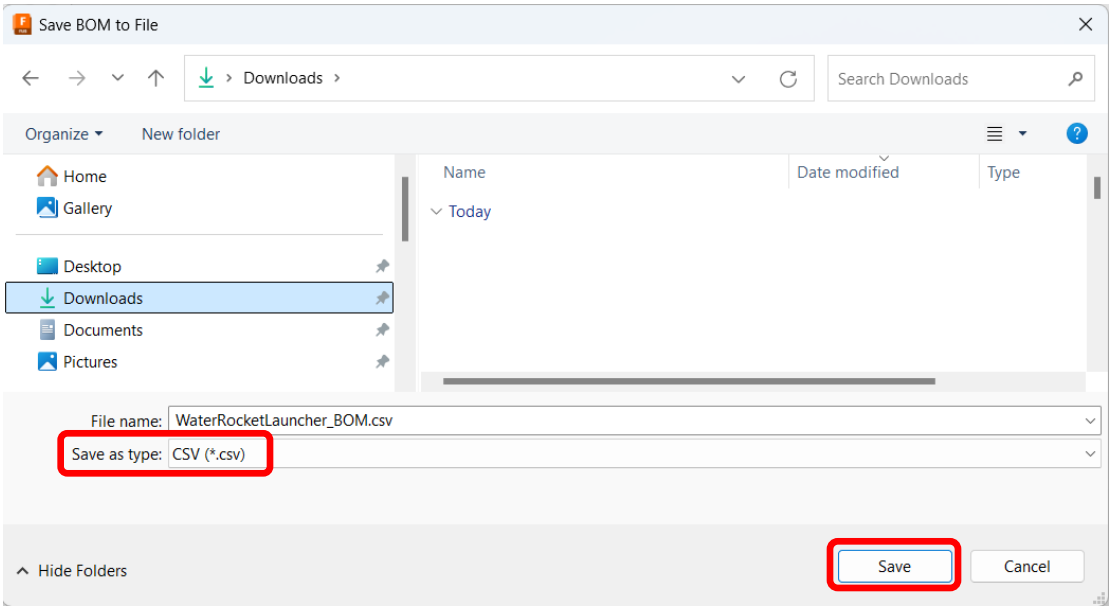
- select the **Save BOM** tab



- click the **Click to Save BOM** button



- select a **location** to save the file to. Here the **Downloads** folder was selected.
- keep **Save as type:** at the default of **CSV (*.csv)** or change it to **TXT (*.txt)** and click **Save**



Below are the 1st 5 lines shown in Notepad when the CSV option is selected. The different fields are separated by commas. This format may be desireable if the file will be opened in Excel.

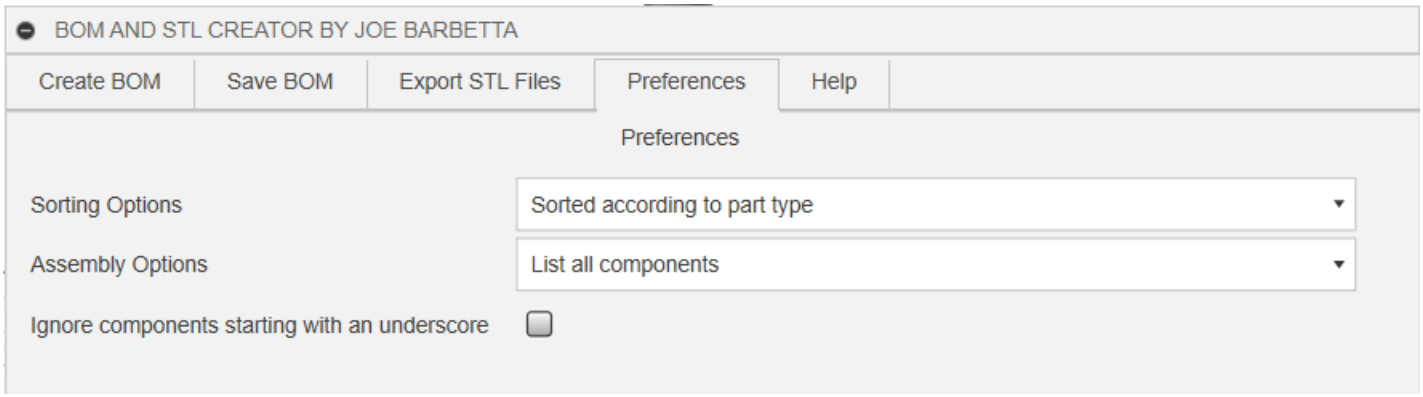
```
Line,Qty,Type,Description,Part Num,Price,
1,14,screw,1/4"-20 3/4"L Socket-Head 18-8SS Black-Oxide,96006A706,$10.74(25),
2,2,screw,1/4"-20 1-1/4" SocketHead 316SS,92185A544,$4.81(10),
3,8,screw,1/4"-20 3/8"L Socket-Head 18-8SS Black-Oxide,96006A702,$7.52(25),
4,3,screw,10-32 1-1/4"L Pan-Head Phillips 18-8SS Passivated,91772A835,$12.07(50)
5,3,screw,10-32 2"L Socket-Head 18-8SS Fully-Threaded 92196A226,$10.32(25),
```

Below are the 1st 5 lines shown in Notepad when the TXT option is selected. The different fields are aligned in columns, which provides a more readable format.

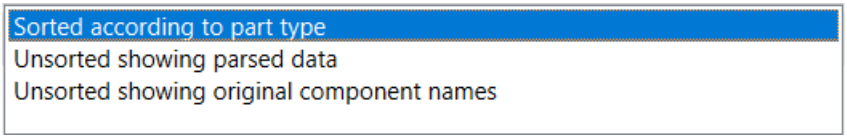
```
Line  Qty  Type        Description                                                    Part Num    Price
---   --   ----------  -------------------------------------------------------------  ----------  ----------
001   14   screw       1/4"-20 3/4"L Socket-Head 18-8SS Black-Oxide                   96006A706   $10.74(25)
002   02   screw       1/4"-20 1-1/4" SocketHead 316SS                                92185A544   $4.81(10)
003   08   screw       1/4"-20 3/8"L Socket-Head 18-8SS Black-Oxide                   96006A702   $7.52(25)
004   03   screw       10-32 1-1/4"L Pan-Head Phillips 18-8SS Passivated              91772A835   $12.07(50)
005   03   screw       10-32 2"L Socket-Head 18-8SS Fully-Threaded 92196A226          $10.32(25)
```

Preferences

One can click on the **Preferences** tab to set various options.



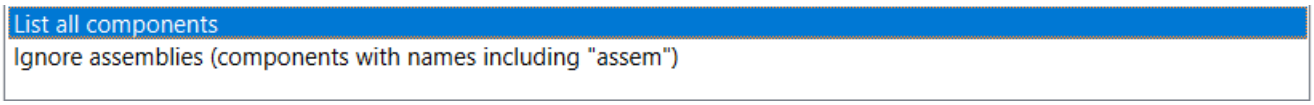
The three **Sorting Options** are as follows.



Below is an example of parsed data vs unparsed data. The **1st line** shows the **Component Name** parsed (split) in **Description**, **Part Number**, and **Price**. The **2nd line** shows the same part with the **Component Name** as it appears in Fusion.

```
038  02  screw      8-32 3/4"L PanHead Phillips 18-8SS                                  91772A197   $13.19(100)
038  02  screw      Screw  8-32 3/4"L PanHead Phillips 18-8SS  91772A197  $13.19(100)
```

The two **Assembly Options** are as follows.



It is good practice to name **Parent Components** comprised of multiple **Child Components** as an Assembly. If one **only wants the Child Components listed**, the **Ignore Assemblies** option can be selected. If a component include the text "**assem**" it will Not be listed.

The **Ignore components starting with an underscore** option allows one to exclude Components from being listed by starting the name with an underscore.

Setting Part Types, Screw Types, and 3D Print Materials

The script will group parts according to Part Types by finding a match for the first word in the Component Name within the Python list defined as **`g_partType`** below. Words can be added to the list if desired. As the comment states the checks are case-insensitive and thus all entries should be lowercase. If any word is longer that 10 characters, the **`g_partTypeLengthMax`** variable should be updated to that of the maximum word length.

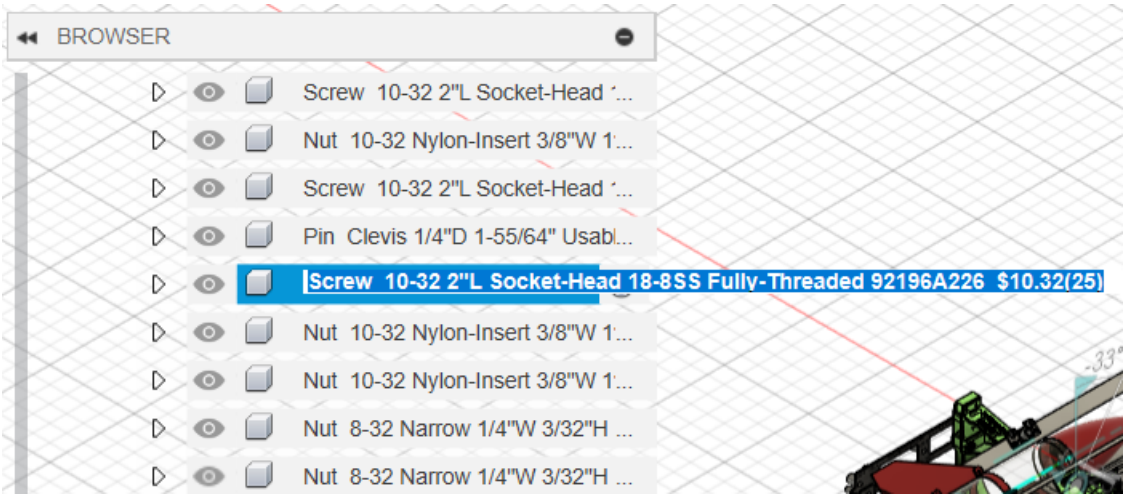
There is also a **`g_screwType`** list for screw threads, which the script will use to group screws by thread type.

```
# The matching method used to sort components according to part type is case-insensitive.
# For example, 'Screw' appearing in a component name will match 'screw' as included in
# list.
g_partType = ['screw', 'nut', 'washer', 'spacer', 'shaft', 'pin', 'insert', \
              'bar', 'tube', 'rod', 'angle', 'latch', 'gear', 'spring', 'coupling', \
              'fitting', 'hinge', 'motor', 'actuator',  'sensor', 'LED', 'laser', \
              'pulley', 'valve', 'pump', 'magnet', 'template']
g_partTypeLengthMax = 10

g_screwType = ['5/16"-18', '1/4"-20', '10-32', '10-24', '8-32', '6-32', '4-40', '2-56', \
               '#6', '#4', '#2']

# As with part types, the matching method used is case-insensitive.  It doesn't matter if
# materials here are upper or lowercase.
g_printedMaterials = ['pla', 'abs']
```

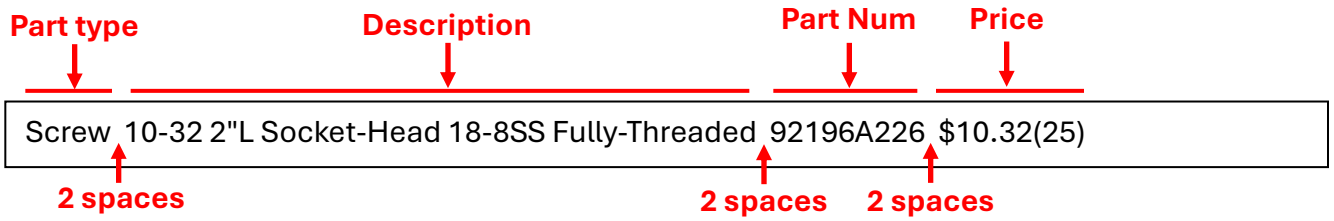
Format of Component Names

This is a partial view of the Fusion **BROWSER** section, which lists **Components** used in the design.

One can click on a **Component Name** to **select it** and then click on it again to **expand the** Component name with **all its text selected**. At this point the **ctrl+c** keys can be used to copy the text or the **ctrl+v** keys to paste in new text. When a Component Name is changed, the name of all other instances of the same Component will change as well.



It can be advantageous to maintain a naming convention and the convention specified in this document provides a balance of readability and ability to parse for a BOM listing.



When a part is imported from McMaster-Carr the Component Name can start with the McMaster-Carr part number followed by a description, as shown below.

91720A194_Super-Corrosion-Resistant 316 Stainless Steel Hex Head Screws

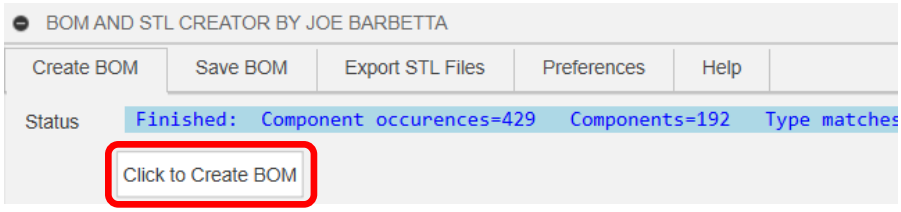
Using the formatting recommended here, the Component Name would be rewritten with the naming convention starting with a Part type and a Description that includes important parameters, such as the screw thread size and length.

Note that the **default deliminator** seperating the Part type, Description, Part Num, and Price is a **double space** and thus the Description must use single spaces between words. If a different deliminator is desired, the `g_partTypeDelim = '  '` variable can be changed from having 2 spaces between the quotes to another character(s), such as `g_partTypeDelim = '_'` for using an underscore instead.

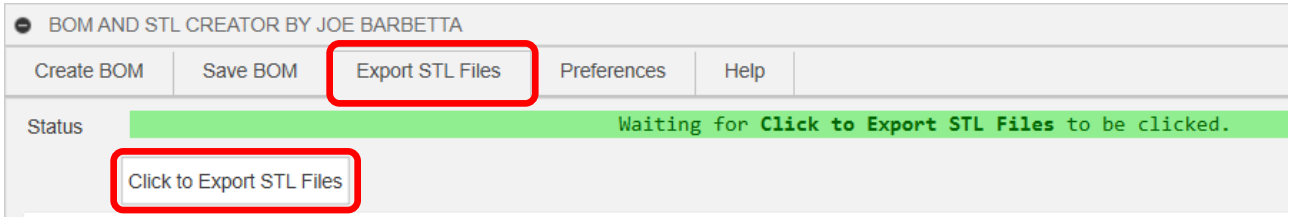
Generating STL Files

The script implements a feature to export all STL files in one operation, as opposed to doing so individually for each 3D printed component in Fusion.

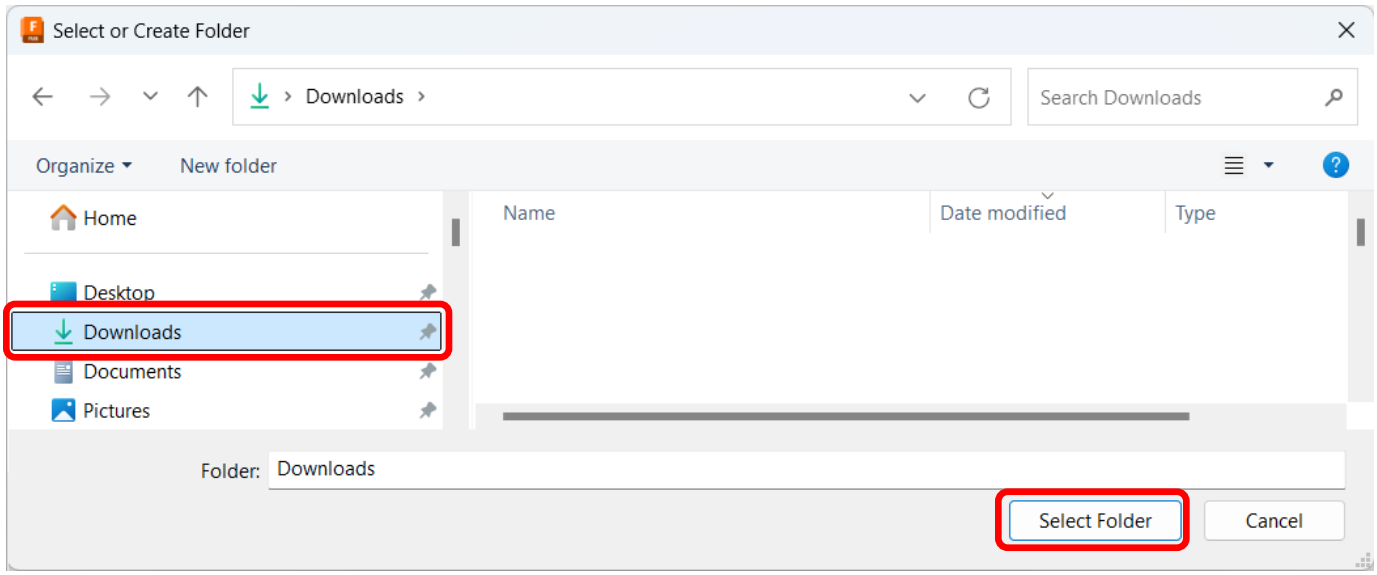
- if not done yet, click the **Click to Create BOM** button. The BOM does not need to be saved.



- select the **Export STL Files** tab and click on the **Click to Export STL Files** button

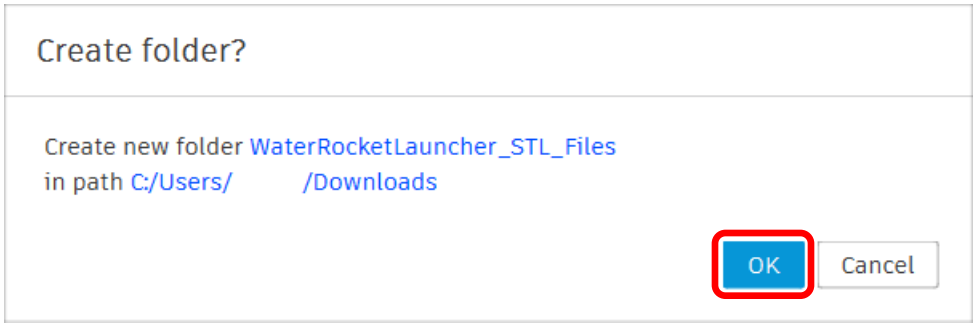


- select a folder for the STL file save location. Here the Downloads folder was selected.
- click **Select Folder**

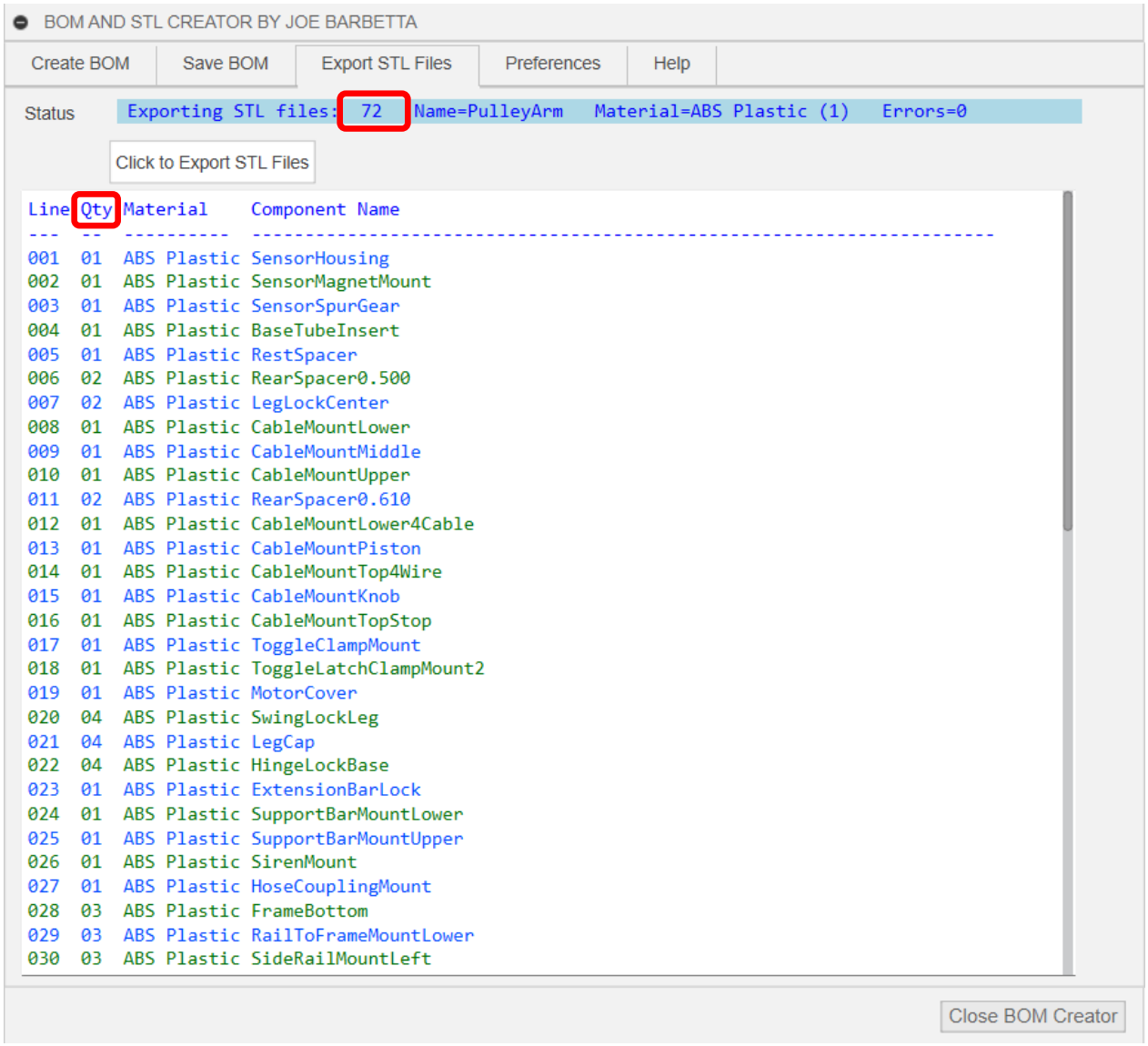


Especially if there are many STL files for a project, it is convenient to save them to their own folder. This script will automatically do so by creating a folder with a name matching the project name followed by "_STL_Files".

- click **OK**



This is the result from the Rocket Launcher.
The screen shows the number of STL files created and the list shows the Quantity of prints for each file. Most STL files will represent a single part, but some will have 2, 3, or 4 copies needed.



Python code listing

Much of the code in this script handles the user interface, such as top tabs, buttons, text boxes, etc. One will also see HTML code used in some strings to control the formatting of text in user interface elements. Much of the code uses the Autodesk Fusion API as imported as `import adsk.core, adsk.fusion`
The beginning of the script has many variables that control output formatting and can be adjusted if desired.

```
#============================================================================================
# Fusion API script "Design BOM(Bill of Materials) Creator" using Python
#
# Developed by Joe Barbetta
#
# ********** THIS SCRIPT IS ONLY FOR USE WITH NUCLEAR REACTOR DESIGN. **********
# ********** THE AUTHOR ASSUMES FULL LIABILITY FOR ANY CORE MELTDOWNS. **********
#
# Note that the term "Command", which used in many of the Fusion API objects and functions
# may cause confusion.  These objects and functions create a window, which appears when the
# script is run and handles the addition and functionality of controls, such as text boxes,
# buttons, dropdown selection lists, etc.  The API provides many additional controls.
# One will also see the term "Dialog" used, which also refers to this window.
# Autodesk has a page on its site that shows the available controls using the below
# Programming Interface>Fusion API User's Manual>User Interface Related Topics>Command Inputs
# Autodesk provides a Fusion API sample "Command Inputs API Sample" in both C++ and Python
# that demonstrates the use of controls.
#
# Programming Interface / Fusion API User's Manual / Python Specific Issues
# -Python API is auto-generated from the C++ API using SWIG, which is what actually
# interfaces directly with Fusion.

# ToDo:
# When the script runs, the Sketches and Features dissapear.  However, they return.
# If a message box is shown after the script completes deletes, the items will reappear
# after the message box is closed.
# It seems that if the Command Dialog remains open the items will reappear.  Many
# sample scripts have their command window close when the bottom OK is pressed, which
# then runs the script.
# DesignCleanerSimple works and the items don't reappear, however this version doesn't
# open a command dialog window.
# Fix:
# adding CommandExecuteHandler(), which is invoked when the user clicks the bottom "OK",
# and having the event firing call the taskRun(True) function seems to work.  Keeping
# the message box at the task completion allows the command dialog to remain open until
# the "OK" button on the message is clicked.  Otherwise the command dialog dissapears
# upon the task completing.  The only downside is that the textbox on the command dialog
# cannot be scrolled or its text selected and copied.
# It seems that invoking the task from a command button causes the changes to revert
# back.
# remove error counts if just listing ?
# have different text for msgbox if listing vs cleaning
#  it can state that the project shouldn't be saved if the results are not satisfactory
# remove cleaning button, show text instructing user to click OK to clean or cancel to not
# because textbox after cleaning cannot be accessed, copy to clipboard if possible or
# offer an option to have text saved to a file
# fix top text to include "listing only" as well
# widen window ?
```

```
# test with only root sketches and features or no root items and just of components
# handling of linked components ?
# handling of components without bodies, invisible ?
# feature can be dissolved or deleted, difference ?
# ignore components with children
# ExportSTLFiles  have text shown in Folder textbox

import adsk.core, adsk.fusion     # Autodesk and Fusion API libraries
import traceback                  # library to provide error information
import time                       # needed for time.sleep()
import datetime                   # needed for datetime.now() and datetime.strftime()
from pathlib import Path          # needed to get "Downloads" folder path
from dataclasses import dataclass # allows the use a C struct equivalent
from enum import Enum             # allows use of enumerated constants
import os                         # needed for os.path.exists() and os.mkdir()

scriptInput_var.author = 'Joe Barbetta'
scriptInput_var.version = '1.00'

g_scriptName = 'Design BOM Creator by Joe Barbetta'
g_scriptDescription = 'Creates a BOM(Bill of Materials)'
g_textTop = 'This script will create a BOM(Bill of Materials) \
             for the open design.'

g_textBoxLineNum = 30  # the number of text lines that will appear in the main text box.
                       # If the number of lines added to the text box exceeds this value,
                       # a vertical scroll bar will appear to allow the user to scroll
                       # through all the lines.
                       # This setting also determines the height of the form.  If the line
                       # count causes the text box to exceed its space on the window, a
                       # scroll bar will appear for the entire window. This is undesireable
                       # because all the controls on the form will be scrolled.



g_windowWidthInit = 1000  # initial width of dialog command window
g_windowHeightInit = 500  # initial height of dialog command window, note that the height
                          # may be overidden by call to setDialogSize(), which sizes the
                          # height to that needed by the controls on the window

g_windowWidthMin = 700  # width of dialog command window
g_windowHeightMin = 400  # width of dialog command window


# The matching method used to sort components according to part type is case-insensitive.
# For example, 'Screw' appearing in a component name will match 'screw' as included in
# list.
g_partType = ['screw', 'nut', 'washer', 'spacer', 'shaft', 'pin', 'insert', \
              'bar', 'tube', 'rod', 'angle', 'latch', 'gear', 'spring', 'coupling', \
              'fitting', 'hinge', 'motor', 'actuator',  'sensor', 'LED', 'laser', \
              'pulley', 'valve', 'pump', 'magnet', 'template']
g_partTypeLengthMax = 10

g_screwType = ['5/16"-18', '1/4"-20', '10-32', '10-24', '8-32', '6-32', '4-40', '2-56', \
               '#6', '#4', '#2']
```

```
# As with part types, the matching method used is case-insensitive.  It doesn't matter if
# materials here are upper or lowercase.
g_printedMaterials = ['pla', 'abs']

# The suggested formatting convention is to have 2 spaces between the first word, which
# specifies a type, eg 'Screw', 'Nut', 'Spacer'.  However, if the user wishes to use
# a different deliminator, such as an underscore, '_', that can be used here.
# 'Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded  92196A226  $10.32(25)'
# allow use of part type even if followed by single space?
# allow identification or part num and price without double spaces?
g_partTypeDelim = '  '

#  This class specifies the widths of the various columns for the BOM displayed in the
# text box and a saved text file.  These values also control the layout of the column
# titles, as the below example shows.
#  'Line Qty Type      Description                                Part Num  Price'
#  '---  --  --------  -----------------------------------------  --------  --------'
# The dataclass is essentially being used as a struct in C/C++ to group variables.
@dataclass
class ColWidths:
  space:   int = 2    # space used between each column
  lineCtr: int = 3
  qty:     int = 2
  type:    int = g_partTypeLengthMax
  descrip: int = 70
  partNum: int = 10
  price:   int = 10


# global variables
g_designName = ''  # will be set in code
g_textBoxMain = None
g_textBoxStatus = None

g_textBoxExportSTLsList = adsk.core.TextBoxCommandInput.cast(None)


g_text = ''  #keep ???

g_BOMasText = ''  # used for save as text file, updated in createBOM()
g_BOMasCsv = ''   # used for save as csv file

g_app = None
g_ui  = None
g_cmd = None
# Global set of event handlers to keep them referenced for the duration of the command
g_handlers = []


# this is essentially being used as a struct would in C/C++
class BomItem:
  def __init__(self, name: str, quantity: int, description: str, sortCtr: int,
               partType: str = '', material: str = '', component = None):
    self.name = name         # 'Screw  10-32 2"L Socket-Head 18-8SS  92196A226  $10.32(25)'
    self.qty = quantity        # quantity
    self.descrip = description # not used
```

```
    self.sortCtr = sortCtr     # 0=Not sorted, 1=sorted once, 2=sorted twice
    self.partType = partType   # 'screw', 'nut', ...
    self.material = material   # 'Steel', 'Aluminum 6061', 'ABS Plastic', ...
    self.comp = component      # component object
# lists of BOMItem objects are created locally in createBOM(), but this global list
# is used by other functions
g_BOMItemsToList: list[BomItem] = []


# this is essentially being used as a struct would in C/C++
# for components to be 3D Printed
class PrintedItem:
  def __init__(self, name: str, quantity: int, material: str = '', component = None):
    self.name = name
    self.qty = quantity
    self.material = material      # 'PLA', 'ABS', ...
    self.comp = component
# this list of 3D Printed items will be appended in createBOM() and used in
#  exportSTLsStart() and exportSTLFiles()
g_printedItems: list[PrintedItem] = []


#  Initializing variables to a cast of the command input provides the convenience of the
# viewing of properties of that command input in the IDE.
# For example, if a drop-down list is used for a control, then it can be initialized with
#  adsk.core.DropDownCommandInput.cast(None).
@dataclass
class Ctrls:
  sortOption = adsk.core.DropDownCommandInput.cast(None)    # drop-down list
  assemOption = adsk.core.DropDownCommandInput.cast(None)   # drop-down list
  tabCreateBOM = adsk.core.TabCommandInput.cast(None)       # top tab
  tabSaveBOM = adsk.core.TabCommandInput.cast(None)         # top tab
  tabSTLs = adsk.core.TabCommandInput.cast(None)            # top tab
  tabPrefs = adsk.core.TabCommandInput.cast(None)           # top tab
  tabHelp = adsk.core.TabCommandInput.cast(None)            # top tab
  chkBoxUnderscored = adsk.core.BoolValueCommandInput.cast(None)  # check box
  txtSaveBOMStatus = adsk.core.TextBoxCommandInput.cast(None)     # text box
  txtExportSTLsStatus = adsk.core.TextBoxCommandInput.cast(None)  # text box

# enumerated constants for sorting options
#  Prefixing by "e" provides the convenience of having the constants listed early in
# the IDE's autocomplete pop-up list.
class SortOption(int, Enum):
  eSORTED_BY_TYPE = 0
  eNO_SORTING = 1
  eNO_SORTING_ORG_NAMES = 2

class AssemOption(int, Enum):
  eLIST_ALL = 0
  eIGNORE_ASSEMS = 1

class FileType(int, Enum):
  eUNDEF = 0
  eCSV = 1
  eTXT = 2
```

```
@dataclass
class Prefs:
  # var name  data type    default using enumerated constant
  sortOption: SortOption = SortOption.eSORTED_BY_TYPE
  assemOption: AssemOption = AssemOption.eLIST_ALL
  ignoreUnderscored: bool = False


# don't we have this already ???
#=============================================================================================
# Event handler that reacts to any changes the user makes to any of the command inputs.
class CommandInputChangedHandler(adsk.core.InputChangedEventHandler):
  def __init__(self):
    super().__init__()
  def notify(self, args):
    # try and except prevents a program crash if a statement in its scope causes an error
    # if a statement causes an error the program execution will jump to except, which will
    # allow the program to provide feedback on the error.  Much nicer then just crashing.
    try:
      eventArgs = adsk.core.InputChangedEventArgs.cast(args)
      inputs = eventArgs.inputs
      cmdInput = eventArgs.input

    except:
      g_ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))


#=============================================================================================
# Event handler that reacts to when the command is destroyed. This terminates the script.
class CommandDestroyHandler(adsk.core.CommandEventHandler):
  def __init__(self):
    super().__init__()
  def notify(self, args):
    try:
      # When the command is done, terminate the script
      # This will release all globals which will remove all event handlers
      adsk.terminate()
    except:
      g_ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))


#=============================================================================================
#  Event handler that is invoked when the command definition is executed which
# results in the command dialog window being created and this event being fired.
#  This is the sub where objects, such as tabs, buttons, text boxs, list boxes, etc.
# are added to the command dialog window.
#  Objects cannot be added in the script startup run(context) because the dialog
# window has not yet been created.
#
# This event handler is created with the below code in def run(context):
#  onCommandCreated = CommandCreatedHandler()
#  cmdDef.commandCreated.add(onCommandCreated)
#  g_handlers.append(onCommandCreated)
#
class CommandCreatedHandler(adsk.core.CommandCreatedEventHandler):
```

```
def __init__(self):
  super().__init__()

# called when an event is triggered from any event that this handler has been added to.
def notify(self, args):

  try:
    # Get the command that was created.
    cmd = adsk.core.Command.cast(args.command)

    # sets minimum width and height of the Command Dialog. The user can drag the bottom
    # right corner of the command window to adjust the window width and height, but the
    # minimum allowable size is specified here
    # setDialogSize() can be called later anytime to override the size set here and if
    # its 2nd argument is 0, the size will be set to fit the items on the window.
    cmd.setDialogMinimumSize(g_windowWidthMin, g_windowHeightMin)

    # sets default size, which is used when the script is first run.  The size is then
    # determined by an entry starting with the text  <Area LayoutPattern  using the
    # script name in NULastDisplayedLayout.xml, which may be located in
    # C:\Users\<user>\AppData\Roaming\Autodesk\Neutron Platform\Options\<12 character id>
    # cmd.setDialogSize() is called lower down to size height to controls on window
    cmd.setDialogInitialSize(g_windowWidthInit, g_windowHeightInit)

    # Connect to the command related events, which is invoked when the
    # bottom "OK" button is clicked
    onExecute = CommandExecuteHandler()
    cmd.execute.add(onExecute)
    g_handlers.append(onExecute)

    # Connect to the command destroyed event.
    onDestroy = CommandDestroyHandler()
    cmd.destroy.add(onDestroy)
    g_handlers.append(onDestroy)

    # Connect to the input changed event.  These events occur when the user changes
    # a CommandInput object, such as Buttons, Radio Button, Dropdown List Boxes, etc.
    onInputChanged = CommandInputChangedHandler()
    cmd.inputChanged.add(onInputChanged)
    g_handlers.append(onInputChanged)

    #  The Validate event handler is not used now, but can be if one wishes the bottom
    # OK button to be disabled if desired using  eventArgs.areInputsValid = False.
    # This could be done if preferences set by the user are determined to be invalid.
    #  It seems that this is the only way to disable (gray out) the "OK" button and
    # that it can't disabled in the CommandInputChangedHandler() handler.
    #  Fusion API documentation states that this event may not always happen upon a
    # command input (control) event and that it can also fire at random times.  When
    # tested it seemed to fire multiple times upon every command input change.
    #onValidateInputs = CommandValidateInputsHandler()
    #cmd.validateInputs.add(onValidateInputs)
    #g_handlers.append(onValidateInputs)

    # By default the bottom of the command window has a "OK" and "Cancel" button.
    # setting isOKButtonVisible to false causes only a "Close" button to appear
    cmd.isOKButtonVisible = False
```

```
      # If the default of two buttons, "OK" and "Cancel", is maintained, the text of
      # the OK and Cancel buttons can be changed.  If isOKButtonVisible is set to false,
      # cancelButtonText will set the text of the single button.
      #cmd.cancelButtonText = 'New Cancel Text'
      #cmd.okButtonText = 'OK'
      cmd.cancelButtonText = 'Close BOM Creator'

      # Get the CommandInputs collection associated with the command window
      inputs = cmd.commandInputs

      # create top tabs
      # The object name created for a tab, must be added to  class Ctrls:
      #-------------------------------------------------------------------------------

      # create a top tab (Id, text)
      # spaces are added in tab text to widen tab
      Ctrls.tabCreateBOM = inputs.addTabCommandInput('tabCreateBOM', '   Create BOM   ')
      tab0ChildInputs =Ctrls.tabCreateBOM.children
      createCreateBOMControls(tab0ChildInputs)  # create the controls for this tab's window

      # create a top tab (Id, text)
      Ctrls.tabSaveBOM = inputs.addTabCommandInput('tabSaveBOM', '   Save BOM   ')
      tab1ChildInputs =Ctrls.tabSaveBOM.children
      createSaveBOMControls(tab1ChildInputs)    # create the controls for this tab's window

      # create a top tab (Id, text)
      Ctrls.tabSTLs = inputs.addTabCommandInput('tabExportSTLs', '   Export STL Files   ')
      tab2ChildInputs =Ctrls.tabSTLs.children
      createExportSTLControls(tab2ChildInputs)  # create the controls for this tab's window

      # create a top tab (Id, text)
      Ctrls.tabPrefs = inputs.addTabCommandInput('tabPrefs', '   Preferences   ')
      tab3ChildInputs = Ctrls.tabPrefs.children
      createPrefControls(tab3ChildInputs)       # create the controls for this tab's window

      # create a top tab (Id, text)
      Ctrls.tabHelp = inputs.addTabCommandInput('tabHelp', '   Help   ')
      tab4ChildInputs = Ctrls.tabHelp.children
      createHelpControls(tab4ChildInputs)      # create the controls for this tab's window

      # setDialogSize() can be called anytime and overrides other sizes.  If the height
      # is zero, the dialog will be sized to fit the command inputs currently displayed.
      cmd.setDialogSize(g_windowWidthInit, 0)

    except:
      g_ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))


#=============================================================================================
# Event handler that is invoked when the user clicks on the bottom "OK" button.
# It is Not invoked when controls, eg buttons, are clicked.
# Any operation that affects the design, eg deleting sketches and features, must be
# handled here, as opposed to the handler for buttons added to the dialog window.
#
class CommandExecuteHandler(adsk.core.CommandEventHandler):
```

```
  def __init__(self):
    super().__init__()
  def notify(self, args):
    try:
      eventArgs = adsk.core.CommandEventArgs.cast(args)
      #taskRun(True)

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



#=============================================================================================
# Event handler that fires when the user changes any added command window objects,
# including the pressing of a button.  It will not fire if a bottom OK or Cancel button
# is clicked.
class CommandInputChangedHandler(adsk.core.InputChangedEventHandler):
  def __init__(self):
    super().__init__()
  def notify(self, args):
    try:
      eventArgs = adsk.core.InputChangedEventArgs.cast(args)
      cmdInput = eventArgs.input

      #g_ui.messageBox('InputChangedEvent_' + cmdInput.id + '_', 'Event')
      # onInputChange when button is clicked
      if cmdInput.id == 'buttonList':
        createBOM()


      elif cmdInput.id == 'APITabBar':  # if a top tab is selected
        #  This event is not used now, but can be if one wishes to run code upon the
        # selection of a top tab.
        #  It seems that each tab cannot generate its own event and it was determined
        # through experimentation that an event with the id equal to 'APITabBar' is
        # generated when any tab is clicked.  It also seems that there is no way to
        # query the tab name, index, or id that was selected and the .isActive
        # property must be read for each.
        #g_ui.messageBox('tabMain='+ str(Ctrls.tabMain.isActive)  \
        #              + '  tabPrefs=' + str(Ctrls.tabPrefs.isActive) \
        #              + '  tabHelp=' + str(Ctrls.tabHelp.isActive), 'Event')
        pass


      elif cmdInput.id == 'buttonSaveBOM':
        # if BOM was not yet created, show message box.  Otherwise, show the Save File
        # dialog window to allow user to navigate to a folder to save the file to.
        saveBOMStart()


      elif cmdInput.id == 'button_exportSTLs':
        exportSTLsStart()


      # if preference for Sorting changed
```

```
      elif cmdInput.id == 'dropDown_prefsSortOptions':  # drop-down list
        #itemName = Ctrls.sortOption.selectedItem.name  # string of selected item
        itemIdx = Ctrls.sortOption.selectedItem.index  # index of selected item
        match itemIdx:
          case 0: Prefs.sortOption = SortOption.eSORTED_BY_TYPE
          case 1: Prefs.sortOption = SortOption.eNO_SORTING
          case 2: Prefs.sortOption = SortOption.eNO_SORTING_ORG_NAMES
        # Get the command that was created.

        #g_cmd.okButtonText = 'dfssgsdggds'
        #g_cmd.cancelButtonText = 'Close'        # uncomment for verifying event
        #adsk.doEvents()
        #txt = 'Prefs.sortOption=' + str(Prefs.sortOption)
        #g_ui.messageBox(txt, 'Event')

      # if preference to Ignore Assemblies(component name including "assem") changed
      elif cmdInput.id == 'dropDown_prefsAssemOptions':  # drop-down list
        itemIdx = Ctrls.assemOption.selectedItem.index  # index of selected item
        match itemIdx:
          case 0: Prefs.assemOption = AssemOption.eLIST_ALL
          case 1: Prefs.assemOption = AssemOption.eIGNORE_ASSEMS

      # if preference to ignore components with a name starting with an underscore changed
      elif cmdInput.id == 'checkbox_prefsIgnoreUnderscore':  # check box
        value = Ctrls.chkBoxUnderscored.value
        if value == True:
          Prefs.ignoreUnderscored = True
        else:
          Prefs.ignoreUnderscored = False


    except:
       g_ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))



#======================================================================================
# Called by:  CommandCreatedHandler()
# Creates the controls for the "Create BOM" tab's window.
#
# addTextBoxCommandInput(id, name, formattedText, numRows, isReadOnly)
# (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
# id: unique ID, ie textBoxBOM, must be unique with respect to other controls
# name: displayed name as seen in the dialog. If an empty string is provided then no name
# will be displayed and the text box will span the width of the command dialog.
#
# formattedText: Specifies the formatted text to display in the input. For example, one can
# use basic html formatting such as <code><b>Bold</b></code>, <code><i>Italic</i></code>,
# and <code><br /></code> for a line break. It also supports hyperlinks, which will open in
# the system's default browser.
# If you are using HTML formatting in your text, it's best to set the text box to be
# read-only. However, if you want to use the text box as a way to get input from the user,
# it's best to use simple text so not HTML formatting is assumed. To do this, use an empty
# string for this argument and then set the text using the text property after the input is
# created. When the text property is used any HTML formatting is ignored and the text is
# treated as basics text. This can be useful if you're using the text box to have the user
```

```
# enter HTML code so it's treated as a simple string.
#
# numRows: specifies the height of the text box as defined by the number of rows of text
#          that can be displayed. If the text is larger than will fit in the box a scroll
#          bar will automatically be displayed.
# isReadOnly: specifies if the text box is read-only or not. Returns the created
#      TextBoxCommandInput object or null if the creation failed.
def createCreateBOMControls(inputs):
  global g_textBoxMain, g_textBoxStatus, g_textTop

  textStatus = '<div style="font-family:consolas; background-color:lightgreen;'
  textStatus += 'text-align:center; font-size:12px; color:darkgreen; white-space:pre-wrap;">'
  textStatus += 'Waiting for the <b>Click to Create BOM</b> to be clicked.'
  textStatus += '</div>'

  # Create a read-only textbox input.  A read-only test box does not have a border.
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  g_textBoxStatus = inputs.addTextBoxCommandInput('textBox_status', 'Status', '', 1, True)
  g_textBoxStatus.formattedText = textStatus

  # Create bool value input with button style that can be clicked.
  # When a button is clicked CommandInputChangedHandler() will be invoked.
  # (id, name, isCheckBox, resourceFolder, initialValue)
  # isCheckBox: False = button
  # HTML formatting cannot be used for button text.  There is no .formattedText property
  #  by default text next to button and on button is set with 2nd argument. To change the
  #  button text its .text property can be set.  If no text is desired next to the button
  #  the 2nd argument must be set to a space.  If set to an empty string, the text next
  #  to the button will default to the text on the button.
  # Multiple buttons using addBoolValueInput(), as done here, cannot appear side by side
  # to make better use of space. There is a "selectable button row input" using
  # addButtonRowCommandInput(), however, it seems that they don't appear as traditional
  # buttons and must use icons from a resource folder, which would have to be distributed
  # with the script.
  btnList = inputs.addBoolValueInput('buttonList', ' ', False, '', False)
  btnList.text = 'Click to Create BOM'

  #btnHelp = inputs.addBoolValueInput('buttonSaveBOM', ' ', False, '', False)
  #btnHelp.text = 'Click to Save BOM'


  # https://forums.autodesk.com/t5/fusion-api-and-scripts/f360-api-defects-in-textboxcommandinput/td-
p/9331149

  # Create an editable textbox (not read-only) to show the list of components
  # There is no need for the user to be able to edit this text box, but a read-only
  # text box does not have a border, which results in an undesireable appearance.
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  g_textBoxMain = inputs.addTextBoxCommandInput( \
                  'textBox_main', '', '', g_textBoxLineNum, False)


#============================================================================================
# Called by: CommandCreatedHandler()
#  Called when the dialog window is created to add controls, such as text boxes,
# buttons, list boxes, etc. to show when the "Save BOM" Tab is selected.
```

```
#
def createSaveBOMControls(inputs):
  txt = '<div style="font-family:consolas; background-color:lightgreen;'
  txt += 'text-align:center; font-size:12px; color:darkgreen; white-space:pre-wrap;">'
  txt += 'Waiting for the <b>Click to Save BOM</b> to be clicked.'
  txt += '</div>'

  # Create a read-only textbox input.  A read-only test box does not have a border.
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  Ctrls.txtSaveBOMStatus = inputs.addTextBoxCommandInput('textBox_status', '', '', 1, True)
  Ctrls.txtSaveBOMStatus.formattedText = txt

  # create Button
  btnSaveBOM = inputs.addBoolValueInput('buttonSaveBOM', ' ', False, '', False)
  btnSaveBOM.text = 'Click to Save BOM'



#=============================================================================================
# Called by: CommandCreatedHandler()
#
def createExportSTLControls(inputs):
  global g_textBoxExportSTLsList

  txt = '<div style="font-family:consolas; background-color:lightgreen;'
  txt += 'text-align:center; font-size:12px; color:darkgreen; white-space:pre-wrap;">'
  txt += 'Waiting for <b>Click to Export STL Files</b> to be clicked.'
  txt += '</div>'

  Ctrls.txtExportSTLsStatus = inputs.addTextBoxCommandInput('textBox_exportSTLStatus', \
                                                            'Status', '', 1, True)
  Ctrls.txtExportSTLsStatus.formattedText = txt

  # create button to Export STL Files
  btnExport = inputs.addBoolValueInput('button_exportSTLs', ' ', False, '', False)
  btnExport.text = 'Click to Export STL Files'

  # Create an editable textbox (not read-only) to show the list of components
  # There is no need for the user to be able to edit this text box, but a read-only
  # text box does not have a border, which results in an undesireable appearance.
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  g_textBoxExportSTLsList = inputs.addTextBoxCommandInput( 'textBox_exportSTLsMain', \
                            '', '', g_textBoxLineNum, False)



#=============================================================================================
# Called by:  CommandCreatedHandler()
#  Called when the dialog window is created to add controls, such as text boxes,
# buttons, list boxes, etc. to show when the "Preferences" Tab is selected.
#
def createPrefControls(inputs):

  # Create a text box that spans the entire width of the dialog by setting the
  # 2nd argument, name, with an empty string.
  txt = '<div align="center">' + 'Preferences' + '</div>'
```

```
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  inputs.addTextBoxCommandInput('textBox_PrefsTop', '', txt, 1, True)


  # in future default values will be set from file

  # Create dropdown input with text list style.
  # When an option is selected CommandInputChangedHandler() will be invoked.
  #  class SortOption(int, Enum): eSORTED_BY_TYPE = 0, eNO_SORTING = 1,
  #   eNO_SORTING_ORG_NAMES = 2
  # (id: str, name: str, dropDownStyle: int)
  #  CheckBoxDropDownStyle     2  list of check boxes where multiple items can be checked
  #  LabeledIconDropDownStyle  0  list of items where each item has text and an icon. If
  #                               the icon of the list item is set to null, a radio button
  #                               will be displayed instead of the icon. A single item can
  #                               be selected at a time
  #  TextListDropDownStyle     1  scrollable list of text only items and one item can be
  #                               selected from the list
  Ctrls.sortOption = inputs.addDropDownCommandInput('dropDown_prefsSortOptions', \
                                                    'Sorting Options', 1)
  dropdownItems = Ctrls.sortOption.listItems
  dropdownItems.add('Sorted according to part type', True, '')
  dropdownItems.add('Unsorted showing parsed data', False, '')
  dropdownItems.add('Unsorted showing original component names', False, '')

  Ctrls.assemOption = inputs.addDropDownCommandInput('dropDown_prefsAssemOptions', \
                                                     'Assembly Options', 1)
  dropdownItems = Ctrls.assemOption.listItems
  dropdownItems.add('List all components', True, '')
  dropdownItems.add('Ignore assemblies (components with names including "assem")', \
                    False, '')

  # Example for creating a check box:
  # Create bool value input with checkbox style.
  # (id, name, isCheckBox, resourceFolder, initialValue)
  Ctrls.chkBoxUnderscored = inputs.addBoolValueInput('checkbox_prefsIgnoreUnderscore', \
                            'Ignore components starting with an underscore ', \
                            True, '', False)

  # Example for creating a text instruction under the controls:
  #txt = '<div style="text-align:center; font-size:16px; color:red; \
  #        white-space:pre-wrap;">'
  #txt += 'Select Main tab after changing any preferences.'
  #txt += '</div>'
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  #inputs.addTextBoxCommandInput('textBox_prefsBot', '', txt, 1, True)



#=============================================================================================
# Called by:  CommandCreatedHandler()
#  Called when the dialog window is created to add controls, such as text boxes,
# buttons, list boxes, etc. to show when the "Help" Tab is selected.
#
def createHelpControls(inputs):
  # Create a text box that spans the entire width of the dialog by setting the
```

```
# 2nd argument, name, with an empty string.
txt = '<div align="center">' + 'Help' + '</div>'
# (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
inputs.addTextBoxCommandInput('textBoxHelp_Top', '', txt, 1, True)

#  If adding text to a message box, the following font sizes allows the
# corresponding number of charcaters to fit on a line.
#  font-family:consolas; font-size:10px  80
#  font-family:consolas; font-size:11px  72
#  font-family:consolas; font-size:12px  66

# white-space:pre-wrap allows use of \n for line breaks in HTML
# font-family:courier seemed to be limited to a larger size
txt = '<div style="font-family:consolas; font-size:14px; color:darkgreen; \
        white-space:pre-wrap;">'
       #0000000001111111111222222222233333333334444444444555555555566666666667777777778
       #1234567890123456789012345678901234567890123456789012345678901234567890123456789
       #================================================================================
txt += ' This script generates a BOM (Bill of Materials) based on all the Components '
txt += 'of the present design.'
txt += '\n\n'
txt += '<span style="color:#006F08">'  # green
txt += ' Developed by Joe Barbetta\n'
txt += ' ****** THIS SCRIPT IS ONLY FOR USE WITH NUCLEAR REACTOR DESIGN. *******\n'
txt += ' ****** THE AUTHOR ASSUMES FULL LIABILITY FOR ANY CORE MELTDOWNS. ******\n'
txt += '</span>'
txt += '\n'
txt += '-Selecting the top <b>Preferences</b> tab provides custimization options.\n'
txt += '-The <b>Click to Create BOM</b> button will create the BOM.\n'
txt += '-The <b>Click to Save BOM</b> will allow the BOM shown to be saved in either '
txt += 'a text or CSV format'
txt += '\n\n'
txt += ' The parsing and sorting algorithm used is base on a recommended convention '
txt += 'for naming components, wherein a <b>double space</b> is used as a deliminator '
txt += 'to separate the various fields as shown below.\n'
txt += '<span style="color:#0049FF">'  # blue
txt += 'Type  Description  PartNumber  Price\n\n'
txt += '</span>'
txt += 'Some examples apear below.\n'
txt += '<span style="color:#0049FF">'  # blue
txt += 'Screw  1/2"-13 3"L SocketHead Titanium-Grade2 FullyThreaded  95435A965  $54.54\n'
txt += 'Nut  1/2"-13 3/4"W 19/32"H Titanium-Grade5  94528A121  $18.16\n\n'
txt += '</span>'
txt += '<b>No commas</b> should be used in the name.  Note that one can right-click on '
txt += 'a component name in the Fusion Browser and select Properties to set a part '
txt += 'number and description.  However, this is incovenient because the Properties '
txt += 'window is slow to open and it is easier to copy/paste names at once from '
txt += 'another source such as a master parts list.'
txt += '\n\n'
txt += ' The list of Part Types can be found near the start of the script source file '
txt += 'and can be ammended if desired.  By using a Part Type, eg. "Screw", at the '
txt += 'start of the component name allows the components to be sorted accordingly.'
txt += '\n\n'
txt += ' An underscore can be added at the start of a components name if it is desired '
txt += 'to have a component ignored.'
txt += '</div>'  # HTML div end
```

```
#_Screw  SocketHead 1/2"-13 3"L Titanium FullyThreaded  95435A965  $54.54
#Screw  1/4"-20 1-3/8" SocketHead PartiallyThreaded 316SS  92185A506  $6.49(10)
#Screw  1/4"-20 1-1/4" SocketHead 316SS  92185A544  $4.81(10)
#Control Rod Assembly


  # Create an editable textbox (not read-only) to show the list of components
  # There is no need for the user to be able to edit this text box, but a read-only
  # text box does not have a border, which results in an undesireable appearance.
  # (id: str, name: str, formattedText: str, numRows: int, isReadOnly: bool)
  g_textBoxStatus = inputs.addTextBoxCommandInput('textBox_Help', '', '', 30, False)
  g_textBoxStatus.formattedText = txt




# not used anymore
#============================================================================================
#
def showHelp():
  # font-family:consolas; font-size:10px  80
  # font-family:consolas; font-size:11px  72
  # font-family:consolas; font-size:12px  66

  # white-space:pre-wrap allows use of \n for line breaks in HTML
  # font-family:courier seemed to be limited to a larger size
  txt = '<div style="font-family:consolas; font-size:12px; color:darkgreen; \
          white-space:pre-wrap;">'
        #00000000011111111112222222222333333333344444444445555555556666666
        #12345678901234567890123456789012345678901234567890123456789012345678
        #================================================================
  txt += ' Clicking the button to <b>Generate BOM</b>'
  txt += 'will scan the design and list all the Components.'
  txt += '\n'
  txt += 'Developed by Joe Barbetta\n'
  txt += '**** THIS SCRIPT IS ONLY FOR USE WITH NUCLEAR REACTOR DESIGN. *****\n'
  txt += '**** THE AUTHOR ASSUMES FULL LIABILITY FOR ANY CORE MELTDOWNS. ****\n'
  txt += '</div>'  # HTML div end
  g_ui.messageBox(txt, g_scriptName + ' Help')



#============================================================================================
# called by: notify() in CommandInputChangedHandler()
# Called when the user clicks the "Create BOM" button.
# updates g_BOMItemsToList, g_BOMasText, and g_textBoxMain.formattedText
#
def createBOM():
  # do all these need to be specified here as global ???
  global g_app, g_text, g_designName, g_printedItems, g_BOMItemsToList

 #try:
  #  The default font is Not monospaced. Specifying consolas causes a monospaced font to
  # achieve alignment of column fields.  font-family:courier seemed to be limited to a
  # larger size.
```

```
#  white-space:pre-wrap allows use of \n for line breaks in HTML, otherwise <br> could
# likely be used
#  background-color:lightblue;  will change background color of text box, but a white
# border of a few pixels will remain inside the text box
statusHdr = '<div style="font-family:consolas; background-color:lightblue; \
             font-size:12px; color:blue; white-space:pre-wrap;">'

textHdr = '<div style="font-family:consolas; font-size:12px; color:blue; \
            white-space:pre-wrap;">'

app = adsk.core.Application.get()
product = app.activeProduct  # design data, toolpath data, ...
#product = g_app.activeProduct  # design data, toolpath data, ...
design = adsk.fusion.Design.cast(product)

if not design:
  textStatus = 'There is no design open to clean.'
  g_textBoxStatus.formattedText = statusHdr + textStatus + '</div>'
  adsk.doEvents()  # needed for text box update
  g_ui.messageBox('No active design', g_scriptName)
  return


#activeDesign = g_app.activeDocument  # get active design
activeDesign = app.activeDocument  # get active design
g_designName = activeDesign.name   # get name of active design


textStatus = 'Listing all Components'
g_textBoxStatus.formattedText = statusHdr + textStatus + '</div>'
adsk.doEvents()  # needed for text box update



# get the root component of the design, which is the top node in the Broswer
# every design has a single default Root Component
rootComp = design.rootComponent
occurrences = rootComp.allOccurrences  # get a list(array) of all component occurrences

occurrencesCnt = occurrences.count     # number of component occurrences

# error if No occurrences ???


dateTimeNow = datetime.datetime.now().strftime("%b %d, %Y") + '  '  # Ex: Jan 1 2024
dateTimeNow += datetime.datetime.now().strftime("%I:%M %p")         # Ex: 11:59 p


# create first 2 lines for design name, script run date and time, and component count
g_text = 'Design: ' + g_designName + '     Script Run: ' + dateTimeNow + '\n\n'
#g_text += 'Components:     ' + str(occurrencesCnt) + '\n' + '\n'

g_textBoxMain.formattedText = textHdr + g_text + '</div>'
adsk.doEvents()  # needed for text box update
time.sleep(1)    # 1 second delay
```

```
errorCtr = 0
compErrorCtr = 0

lineCtr = 0


#textStatus = 'dsfdasdfasf'
#textStatus = 'Line: ' + str(lineCtr).zfill(g_lineCtrWidth) + \
#             '   Sketches: ' + str(sketchCtr).zfill(3)  + \
#             '   Features: ' + str(featureCtr).zfill(3) + textOp

#g_textBoxStatus.formattedText = statusHdr + textStatus + '</div>'
#adsk.doEvents()  # needed for text box update


# create an empty list(array) of BomItem objects
bomItems: list[BomItem] = []

occurenceCtr = 0
componentCtr = 0
ignoreCtr = 0
childCnt = 0
printCtr = 0

# loop through all component occurrences
# because the a component can be copied, Autodesk uses the term "occurrence".
for i in range(occurrencesCnt):
  # example fullPathName returns:
  # MotorCover is printed, RailStructure is not, use material
  # RailAssembly:1+RailStructure:1+RailSupportFrame:1+FrameBottom:1
  # Base:1+Stationary Structure:1+Tube  3/4"OD 1/8"T  Aluminum 6061  9056K33  9.07(1ft):1
  # Base:1+Stationary Structure:1+LegLockCenter:1
  # Base:1+MotorCover:1
  # Base:1
  # RailAssembly:1+RailStructure:1+RailSupportFrame:1+FrameCapLeft:1
  # RailAssembly:1+RailStructure:1
  #pathName = occurrences[i].fullPathName
  # print if Component has a body ?
  # print if material = plastic ?
  comp = occurrences[i].component  # get Component

  if(comp.material):  # parent components (assemblies) will Not have a material ???
    materialName = comp.material.name  # Ex: 'Steel', 'Aluminum 6061', 'ABS Plastic'
  else:
    materialName = ''

  if Prefs.assemOption == AssemOption.eIGNORE_ASSEMS:
    # If 'assem', 'Assem', 'assembly', 'Assembly', or other case variations are found
    # in the component occurence name, then skip all following code and loop again.
    if comp.name.lower().find('assem') > 0:
      ignoreCtr += 1
      continue
  if Prefs.ignoreUnderscored == True:
    # if the first character is an underscore
    if comp.name[0] == '_':
      ignoreCtr += 1
```

```
      continue

  occurenceCtr += 1

  compMatch = False
  # check if this component occurance matches a component already added to the BOM
  for bomItem in bomItems:
    if bomItem.comp == comp:  # if this component has been added to the BOM already
                              # note that a object comparison is being done here
      bomItem.qty += 1
      compMatch = True
      break


  if compMatch == False:  # if there was No component match, Not a Part Type match
    componentCtr += 1
    # comp.name is the Component's name that appears in the Browser
    # A Component's description is often not used because to set it one must right-click
    # on the component's name in the Browser and select "Properties", which then after
    # an annoying delay shows a window to allow setting the Description.
    # Using .strip() will remove any leading or trailing spaces, but none between words.
    # This is mostly important to remove any spaces that happened to be entered before
    # the text.
    name = comp.name.strip()  # remove any leading or trailing spaces

    # append this Component to the BOM list
    # The constructor, as shown below, is used.
    # def __init__(self, name: str, quantity: int, description: str, sortCtr: int,
    #              partType: str = '', material: str = '', component = None):
    # Note that the last argument is the object instance, which will be used for
    # compares to determine matches.
    # Note that the description is often not set by the user.
    bomItems.append(BomItem(name, 1, comp.description, 0, '', materialName, comp))

    # updates Status text box
    createBOMStatusUpdate(' Scanning:  Component occurences=' + str(occurenceCtr) \
                          + '   Components=' + str(componentCtr))
    adsk.doEvents()  # needed for text box update
# end  for i in range(occurrencesCnt):


# Extract the part types from the component names.  If a component name doesn't specifiy
# a part type, which will be common for components to be printed, the
#  Below are some examples of Component names that were entered according to the format
# compatible with this script:
#  'Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded 92196A226  $10.32(25)'
#  'Nut  8-32 Narrow 1/4"W 3/32"H 18-8SS  90730A009  $5.45(100)'
#  'Spacer  #6 0.140"ID 3/8"OD 1/4"L 18-8SS  92320A502  $2.65'
# g_partType = ['screw', 'nut', 'washer',
for bomItem in bomItems:
  # getPartType() returns the first word of the Component name, which may be the
  # part type, eg 'screw', 'nut', 'washer'.  This part type will then be used to group
  # components according to part type.  Use of .lower() ensured that the part type is
  # all lowercase to help with matching of part types later on.
  # The 3rd argument specifies the maximum part type length.  If for some reason the
  # deliminator occurs far into the string, it will be assumed that there is no part type.
```

```
  # consider .caseFold() ???
  # if no part type is found an empty string is returned
  bomItem.partType = getPartType(bomItem.name, g_partTypeDelim, g_partTypeLengthMax).lower()
  #if len(bomItem.partType) == 0:
  #  if bomItem.material.upper().find('ABS') >= 0:  # not working ???
  #    bomItem.partType = 'printed'
  #  else:
  #    bomItem.partType = 'undef'


# In case any of the part types in the g_partType list has an uppercase letter, ensure
# they are all lowercase to aid matching.  Do the same for g_printedMaterials.
for partType in g_partType:
  partType = partType.lower()  # convert to lowercase
for material in g_printedMaterials:
  material = material.lower()  # convert to lowercase


# new empty list, which will be appended to include components sorted by part type
bomItemsSorted: list[BomItem] = []

typeMatchCtr = 0

# To group components according to part type this loop will iterate through the defined
# part types, eg. 'screw', 'nut', 'spacer' to collect the matching components.
for partType in g_partType:  # loop through part types
  for bomItem in bomItems:   # loop through all BOM items
    if bomItem.sortCtr == 0:  # if this component has Not been sorted already
      if bomItem.partType == partType:  # if the part type matches
        bomItem.sortCtr = 1
        bomItemsSorted.append(bomItem)
        typeMatchCtr += 1
        # updates Status text box
        createBOMStatusUpdate(' Sorting:  Component occurences=' + str(occurenceCtr) \
                              + '   Components=' + str(componentCtr) \
                              + '   Type matches=' + str(typeMatchCtr) \
                              + '   Ignored=' + str(ignoreCtr) \
                              + '   Prints=' + str(printCtr))
        adsk.doEvents()  # needed for text box update


# for components not yet sorted, determine which ones are to be printed by
# checking if the component's material matches that of the materials list
# best way to clear ???
#g_printedItems: list[PrintedItem] = []  # in case list was appended already, clear list
g_printedItems = []  # in case list was appended already, clear list
for bomItem in bomItems:
  #if len(bomItem.partType) == 0:
  if bomItem.sortCtr == 0:
    materialMatch = ''  # reset material match string
    # loop through printed materials list
    for material in g_printedMaterials:
      if bomItem.material.lower().find(material) >= 0:  # if material match
        materialMatch = material.upper()  # convert to uppercase
        break
```

```
    if materialMatch != '':  # if not empty string (if matched to a material)
      bomItem.sortCtr = 1
      bomItem.partType = 'print' + materialMatch
      bomItemsSorted.append(bomItem)  # append to list
      # append list, constructor shown below for reference
      # def __init__(self, name:str, quantity:int, material:str = '', component = None):
      g_printedItems.append(PrintedItem(bomItem.name, bomItem.qty, \
                                        bomItem.material, bomItem.comp))
      printCtr += 1
    else:
      bomItem.partType = 'undef'


# add all other components that did not have a matching part type or material
for bomItem in bomItems:
  if bomItem.sortCtr == 0:  # if this component has not been matched to any part type
    bomItemsSorted.append(bomItem)


# seems to work, more testing ???
# sorts Screws according to thread size
bomItems2ndSort: list[BomItem] = []
# create a list for each part type to be later combined ?
#g_screwType = ['5/16-18', '1/4-20', '10-32', '10-24', '8-32', '6-32', '4-40', '2-56', \
#             '#6', '#4', '#2']
# Screw  1/4"-20 3/4"L Socket-Head 18-8SS Black-Oxide  96006A706  $10.74(25)
# Screw  8-32 7/16" PanHead Phillips 18-8SS  91772A193  $9.55(100)
# Screw  1/4"-20 1-1/4" SocketHead 316SS  92185A544  $4.81(10)
for screwType in g_screwType:
  for bomItem in bomItemsSorted:
    # if this is a screw that hasn't been sorted by thread type
    if bomItem.partType == 'screw' and bomItem.sortCtr < 2:
      # check if '5/16-18' appears in component name
      # use other than 20
      index = bomItem.name.find(screwType, 0, 20)
      if index > 1:
        bomItem.sortCtr = 2
        bomItems2ndSort.append(bomItem)

for bomItem in bomItemsSorted:
  if bomItem.sortCtr < 2:
    bomItems2ndSort.append(bomItem)


showParsedNames = False
match Prefs.sortOption:
  case SortOption.eSORTED_BY_TYPE:
    showParsedNames = True
  case SortOption.eNO_SORTING:
    showParsedNames = True
  case SortOption.eNO_SORTING_ORG_NAMES:
    showParsedNames = False

# g_BOMItemsToList will be that to be listed and then saved
if Prefs.sortOption == SortOption.eSORTED_BY_TYPE:
  #g_BOMItemsToList = bomItemsSorted
```

```
  g_BOMItemsToList = bomItems2ndSort
  # use  bomItems = bomItems2ndSort  to write over a reuse bomItems ???
else:
  g_BOMItemsToList = bomItems

# use sortCtr as opposed to .sorted  =1 match to part type, =2 match to screw type
# before checking against this remove '"' from name ???
# g_screwType = ['5/16-18', '1/4-20', '10-32', '10-24', ...
# use list of lists with first element matching part type, ie ['screw', '5/16-18', ...
# for screwType in g_screwType:
# allow screwType text to be anywhere in name
# list of lists:
# partTypeSort[ \
#  ['screw', '5/16-18', ...], \
#  ['nut', '5/16-18', ...] \
#
# g_partSubType = 'screw, 5/16-18, ...'
# g_partSubType += 'nut, 5/16-18, ...'
#if Prefs.sortOption = SortOption.
#bomItemsSorted = bomItems


# this is used if showing original Component Names without parsing
compNameColWidth = ColWidths.descrip + ColWidths.space \
                 + ColWidths.partNum + ColWidths.space + ColWidths.price

# create Column Header text and undelines
# 'Line Qty Type      Description                                Part Num  Price'
# '---  --  --------  -----------------------------------------  --------  --------'
# or
# 'Line Qty Type      Part Name'
# '---  --  --------  ------------------------------------------------------------'
# create column Titles
lineText =  setStrSize('Line', ColWidths.lineCtr + ColWidths.space)
lineText += setStrSize('Qty', ColWidths.qty + ColWidths.space)
lineText += setStrSize('Type', ColWidths.type + ColWidths.space)
if showParsedNames == True:
  lineText += setStrSize('Description', ColWidths.descrip + ColWidths.space)
  lineText += setStrSize('Part Num', ColWidths.partNum + ColWidths.space)
  lineText += setStrSize('Price', ColWidths.price) + '\n'
else:
  lineText += setStrSize('Part Name', compNameColWidth) + '\n'

# create dashes under titles
lineText2 =  ('-' * ColWidths.lineCtr) + (' ' * + ColWidths.space)
lineText2 += ('-' * ColWidths.qty) + (' ' * + ColWidths.space)
lineText2 += ('-' * ColWidths.type) + (' ' * + ColWidths.space)
if showParsedNames == True:
  lineText2 += ('-' * ColWidths.descrip) + (' ' * + ColWidths.space)
  lineText2 += ('-' * ColWidths.partNum) + (' ' * + ColWidths.space)
  lineText2 += ('-' * ColWidths.price) + (' ' * + ColWidths.space) + '\n'
else:
  lineText2 += '-' * compNameColWidth + '\n'

# use += if other header info can get added ??
g_BOMasText =  lineText + lineText2
```

```
g_text += lineText + lineText2
g_textBoxMain.formattedText = textHdr + g_text + '</div>'
adsk.doEvents()  # needed for text box update
time.sleep(1)    # 1 second delay


# list the sorted components
#for bomItem in bomItemsSorted:
for bomItem in g_BOMItemsToList:
  lineCtr += 1

  # start line with HTML formatting to allow lines to have alternating colors
  # lower down the line must end with '</span>' to ensure the HTML style is used
  # for this line only
  if (lineCtr % 2) == 0:  # if even
    lineText = '<span style="color:#006F08">'  # green
  else:  # if odd
    lineText = '<span style="color:#0049FF">'  # blue

  # convert line counter and quantity to strings with leading zeros to ensure these
  # fields always have the same number of characters to ensure aligned columns
  lineCtrText = str(lineCtr).zfill(ColWidths.lineCtr)
  qtyText = str(bomItem.qty).zfill(ColWidths.qty)

  # use specified deliminator ???
  # other method to parse out Part Num and Price ???
  # Component Name example with double spaces separating Type, Name, Part Num, and Price
  #  'Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded  92196A226  $10.32(25)'
  # Parsed result:
  #  fields[0] = Type, ie 'Screw', 'Nut', 'Bar'
  #  fields[1] = Name, ie '10-32 2"L Socket-Head 18-8SS Fully-Threaded'
  #  fields[2] = Part Num, ie '92196A226'
  #  fields[3] = Price, ie '$10.32(25)'
  fields = bomItem.name.split('  ')

  # 'Line,Qty,Type,Part Description,Part Num,Price'
  # '000,00,Screw,10-32 2"L Socket-Head 18-8SS Fully-Threaded,92196A226,$10.32(25)

  # remove type column because type is repeated ???
  # or use undefined for other components ???

  # build line for part using specified column widths
  #  Line Qty Type   Part Description                              Part Num   Price
  # '000  00  Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded  92196A226  $10.32(25)
  lineText += setStrSize(lineCtrText, ColWidths.lineCtr + ColWidths.space)
  #if len(fields) < 2:
  # lineText += setStrSize(fields[0], ColWidths.descrip + ColWidths.space)
  #else:
  lineText += setStrSize(qtyText, ColWidths.qty + ColWidths.space)
  lineText += setStrSize(bomItem.partType, ColWidths.type + ColWidths.space)
  if showParsedNames == True:
    if len(fields) > 1:
      lineText += setStrSize(fields[1], ColWidths.descrip + ColWidths.space)
    else:
      lineText += setStrSize(bomItem.name, ColWidths.descrip + ColWidths.space)
```

```
      if len(fields) > 2:
        lineText += setStrSize(fields[2], ColWidths.partNum + ColWidths.space)
      if len(fields) > 3:
        lineText += setStrSize(fields[3], ColWidths.price + ColWidths.space)
    else:
      lineText += bomItem.name

    lineText += '</span>'

    g_text += lineText + '\n'
    g_textBoxMain.formattedText = textHdr + g_text + '</div>'

    # updates Status text box
    createBOMStatusUpdate(' Listing:  Component occurences=' + str(occurenceCtr) \
                          + '   Components=' + str(componentCtr) \
                          + '   Type matches=' + str(typeMatchCtr) \
                          + '   BOM items=' + str(lineCtr) \
                          + '   Ignored=' + str(ignoreCtr) \
                          + '   Prints=' + str(printCtr))
    adsk.doEvents()  # needed for text box update
  # end  for bomItem in g_BOMItemsToList:

  # updates Status text box
  createBOMStatusUpdate(' Finished:  Component occurences=' + str(occurenceCtr) \
                        + '   Components=' + str(componentCtr) \
                        + '   Type matches=' + str(typeMatchCtr) \
                        + '   BOM items=' + str(lineCtr) \
                        + '   Ignored=' + str(ignoreCtr) \
                        + '   Prints=' + str(printCtr))
  adsk.doEvents()  # needed for text box update


#============================================================================================
# called by: createBOM()
#
def createBOMStatusUpdate(text):
  hdr = '<div style="font-family:consolas; background-color:lightblue; \
         font-size:12px; color:blue; white-space:pre-wrap;">'

  g_textBoxStatus.formattedText = hdr + text + '</div>'


#============================================================================================
# called by: notify() in CommandInputChangedHandler()
# Called when the user clicks the "Save BOM to file" button.
#
def saveBOMStart():
  if len(g_textBoxMain.text) < 10:  #use other method

    msgText = 'BOM must be created first.'
    ret = showMsgBox(msgText, 'red', 'Error', adsk.core.MessageBoxButtonTypes.OKButtonType)

  else:
    # show a Save File Dialog window to allow user to select the save location.
    saveBOMFileDialog()
```

```
#============================================================================================
# called by:  saveBOMStart()
# Called when the user clicks the "Save BOM to file" button and the BOM had been created,
#
def saveBOMFileDialog():

  try:
    # this can be considered for use
    # returns string for path of Downloads folder  C:\Users\<user>\Downloads\
    #downloadsPath = str(Path.home() / "Downloads")

    fileNameSuffix = '_BOM'

    # http://help.autodesk.com/view/fusion360/ENU/?guid=GUID-69478fef-f96f-4e7c-b5af-766301072042
    fileDialog = g_ui.createFileDialog()
    fileDialog.isMultiSelectEnabled = False
    fileDialog.title = 'Save BOM to File'
    fileDialog.filter = 'CSV (*.csv);;TXT (*.txt);;All Files (*.*)'
    fileDialog.filterIndex = 0
    fileDialog.initialFilename = g_designName + fileNameSuffix
    #fileDialog.initialDirectory =

    #  The Save File dialog window has a lower "File name:" Combo Box and under it
    # is a "Save as type:" List Box.  A Combo Box allows one to either select drop-down
    # options as a List Box or the option to type or paste text.  A List Box only allows
    # selecting from drop-down options.
    #  When one clicks on a file shown in the navigation view, its name will show in the
    # "File name:" Combo Box.

    # If user selects an existing file, the operating system will display the message:
    #  "Confirm Save As"
    #  "Design1.csv already exists.  Do you want to replace it?"
    #   "Yes" "No" buttons
    dialogResult = fileDialog.showSave()
    if dialogResult == adsk.core.DialogResults.DialogOK:  # user clicked "OK"
      # check if valid file ???
      # returns file name with path
      # C:\Users\<user>\Downloads\ReactorV2_BOM.txt
      fileNameWithPath = fileDialog.filename

      # Ex filePath "C:\Users\<user>\Downloads\", Ex fileName "ReactorV2_BOM.txt"
      filePath, fileName = os.path.split(fileNameWithPath)
      # Ex fileNameWoExt ""ReactorV2_BOM", Ex fileExt ".txt"
      fileNameWoExt, fileExt = os.path.splitext(fileName)

      # uncomment for diagnostics
      #msgText =  'fileDialog.filterIndex= ' + str(fileDialog.filterIndex) + '\n'
      #msgText += 'fileNameWithPath= ' + str(fileNameWithPath) + '\n'
      #msgText += 'filePath= ' + str(filePath) + '\n'
      #msgText += 'fileName= ' + str(fileName) + '\n'
      #msgText += 'fileNameWoExt= ' + str(fileNameWoExt) + '\n'
      #msgText += 'fileExt= ' + str(fileExt)
      #showMsgBox(msgText, 'green', 'Diag', adsk.core.MessageBoxButtonTypes.OKButtonType)

      #  This commented code is kept here for reference to determine the filter option
```

```
# selected by the user.  It's not used now because the file extension is used
# instead because if All Files (*.*) is selected, the user can select a .csv or
# .txt file to write over.
#if fileDialog.filterIndex == 0:    # 1st filter option: CSV
  #fileType = 'csv'
#elif fileDialog.filterIndex == 1:  # 2nd filter option: TXT
  #fileType = 'txt'
#elif fileDialog.filterIndex == 3:  # 3rd filter option: All Files
  #fileType = 'txt'

#  Use the extension of the file selected using the file dialog window to determine
# the format to save as.  When the user selects All Files (*.*) as the filter, any
# file type can be selected.
#  Note that the first BOM save of a new design, will create a file that likely
# did not exist already because the file name will, by default, be set to the
# design name.  Later BOM saves for the same design will allow thu user to save
# over the same file, unless intentionally named otherwise.
if fileExt.lower() == '.csv':
  fileType = FileType.eCSV
elif fileExt.lower() == '.txt':
  fileType = FileType.eTXT
else:
  fileType = FileType.eUNDEF

#  If the user selects a file other than a .csv or .txt, a Message Box will be
# shown to alert the user.  The user does not have an option to continue. Clicking
# OK will close the message and abort the save.  The user will have to click
# Save again to select a different file.
if fileType == FileType.eUNDEF:
  msgText =  'Improper file type was selected.\n'
  msgText += 'Should be a <b>.csv</b> or <b>.txt</b> file.'
  ret = showMsgBox(msgText, 'red', 'Error', \
                   adsk.core.MessageBoxButtonTypes.OKButtonType)
  return  # the save will always be aborted here


#  Check if the file name (without the extension) equals the design name.  By
# default the file name will match the design name.  If the user, perhaps
# mistakingly, selects the BOM of another design to write over, alert the user.
#  By default, the file name used when the file is created is the design name
# followed by a '_BOM' suffix.  Before performing the name comparison, remove
# the suffix if it exists at the end of
# returns -1 if string not found, 0 if the string starts at the first character
index = fileNameWoExt.find(fileNameSuffix, 0)
if index > 0:
  # text[:index] extracts the text up to the index value.  The first character is at
  # index = 0. For example, if index = 2, the first two characters will be returned.
  fileNameWoExt = fileNameWoExt[:index]

if fileNameWoExt == g_designName:
  fileNameMatch = True
else:
  fileNameMatch = False

#  If names, as discussed above, don't match alert the user with a Message Box.
# The user can click OK to still write over the selected file or Cancel.
```

```
      if fileNameMatch == False:
        msgText = 'The file name selected <b>' + fileNameWoExt + '</b>\n'
        msgText += 'does not match the design name <b>' + g_designName + '</b>.\n\n'
        msgText += 'Continue anyway?'
        ret = showMsgBox(msgText, 'red', 'Error', \
                         adsk.core.MessageBoxButtonTypes.OKCancelButtonType)
        if ret == adsk.core.DialogResults.DialogCancel:  # if Cancel was clicked
          return

      # at this point fileType can only be FileType.eCSV or FileType.eTXT
      # if FileType.eCSV calls saveBOMasCsv()
      # if FileType.eTXT saves g_textBoxMain.text
      saveBOMtoFile(fileNameWithPath, fileType)

      # zzz
      # When All Files are selected, the user can select any type of file to write
      # over.  Only allow doing so with a .csv or .txt and use the extension to
      # determine which file type to save as.  If the file name does Not match the
      # present project name, show a message to the user to confirm the action.

      #saveBOMasCsv(

    else:  # user clicked "Cancel"
      return
    #g_ui.messageBox('filename=' + fileNameWithPath)

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


#============================================================================================
# called by: saveBOMFileDialog()
#
def saveBOMtoFile(fileNameWithPath, fileType: FileType):
  textToSave = ''

  if fileType == FileType.eCSV:
    # uses g_BOMItemsToList to create .csv
    textToSave = saveBOMasCsv()

  elif fileType == FileType.eTXT:
    # don't just save text box contents ???
    textToSave = g_textBoxMain.text


  try:
    #arguments:
    # 2nd  'r' = Opens a file for reading, error if the file does not exist
    #      'a' = Opens a file for appending, creates the file if it does not exist
    #      'w' = Opens a file for writing, creates the file if it does not exist
    #      'x' = Creates the specified file, returns an error if the file exists
    #       'b' can be appended to the above for binary data, text is default
    # 3rd  -1 = for Buffering use system default
    # 4th   utf-8-sig causes a "byte order mark" signature to be written to the start of
    #       the file to indicate UTF-8
```

```
    file = open(fileNameWithPath, 'w', -1, 'utf-8')

    file.write(textToSave)
    file.close()

    txt = 'File Saved: ' + fileNameWithPath
    saveBOMStatusUpdate(txt, 'lightgreen', 'darkgreen')

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



#============================================================================================
# called by: saveBOMtoFile()
#
def saveBOMStatusUpdate(text, backColor, fontColor):
  hdr = '<div style="font-family:consolas; font-size:12px; color:' + fontColor + '; '
  hdr += 'text-align:center; background-color:' + backColor + '; '
  hdr += 'white-space:pre-wrap;">'

  Ctrls.txtSaveBOMStatus.formattedText = hdr + text + '</div>'



#zzz
#============================================================================================
# called by: saveBOMtoFile()
#  Called if user chose for a .csv file to be created or selected an exisring .csv
# file to overwrite.
# Uses g_BOMItemsToList
# Ex:  textToSave = saveBOMasCsv()
#
def saveBOMasCsv():

  showParsedNames = False
  match Prefs.sortOption:
    case SortOption.eSORTED_BY_TYPE:
      showParsedNames = True
    case SortOption.eNO_SORTING:
      showParsedNames = True
    case SortOption.eNO_SORTING_ORG_NAMES:
      showParsedNames = False



  delim = ','

  # create Column Header text
  # 'Line,Qty,Type,Description,Part Num,Price'
  #  or
  # 'Line,Qty,Type,Part Name'
  lineText =  'Line' + delim
  lineText += 'Qty' + delim
  lineText += 'Type' + delim
  if showParsedNames == True:
    lineText += 'Description' + delim
```

```
    lineText += 'Part Num' + delim
    lineText += 'Price' + delim
  else:
    lineText += 'Part Name' + delim

  textCSV = lineText

  lineCtr = 0

  # list the sorted components
  #for bomItem in bomItemsSorted:
  for bomItem in g_BOMItemsToList:
    lineCtr += 1

    # 'Line,Qty,Type,Part Description,Part Num,Price'
    # '1,1,Screw,10-32 2"L Socket-Head 18-8SS Fully-Threaded,92196A226,$10.32(25)
    lineText = str(lineCtr) + delim
    lineText += str(bomItem.qty) + delim
    lineText += bomItem.partType + delim   # Ex: 'Screw', 'Nut', ...

    # use specified deliminator ???
    # other method to parse out Part Num and Price ???
    # Component Name example with double spaces separating Type, Name, Part Num, and Price
    #  'Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded  92196A226  $10.32(25)'
    # Parsed result:
    #  fields[0] = Type, ie 'Screw', 'Nut', 'Bar'
    #  fields[1] = Name, ie '10-32 2"L Socket-Head 18-8SS Fully-Threaded'
    #  fields[2] = Part Num, ie '92196A226'
    #  fields[3] = Price, ie '$10.32(25)'
    fields = bomItem.name.split('  ')

    if showParsedNames == True:
      if len(fields) > 1:
        lineText += fields[1] + delim     # Part Name without Type, Part Number, and Price
      else:
        lineText += bomItem.name + delim
      if len(fields) > 2:
        lineText += fields[2] + delim
      if len(fields) > 3:
        lineText += fields[3] + delim
    else:
      lineText += bomItem.name           # Part Name including Part Number and Price

    textCSV += '\n' + lineText

  return textCSV



#============================================================================================
# called by: notify() in CommandInputChangedHandler()
# Called when the user clicks the "Save BOM to file" button.
#
def exportSTLsStart():

  if len(g_printedItems) == 0:
```

```
  showMsgBox('Create BOM before exporting STLs.', 'red', 'Error', \
             adsk.core.MessageBoxButtonTypes.OKButtonType)
  return

# do these 3 at startup ???  can user select new design with script open ?
app = adsk.core.Application.get()
activeDesign = app.activeDocument  # get active design
g_designName = activeDesign.name   # get name of active design

try:
  # maintain global initial directory ???
  # returns string for path of Downloads folder  C:\Users\<user>\Downloads\
  downloadsPath = str(Path.home() / "Downloads")

  # Set styles of file dialog.
  folderDialog = g_ui.createFolderDialog()
  folderDialog.title = 'Select or Create Folder'
  folderDialog.initialDirectory = downloadsPath
  #folderDialog.folder = g_designName + 'STL Files'
  # Show folder dialog
  dialogResult = folderDialog.showDialog()
  if dialogResult == adsk.core.DialogResults.DialogOK:  # user clicked "OK"
    newFolderName = g_designName + '_STL_Files'
    newFolderPath = folderDialog.folder
    # <br> (break) is used instead of \n for line breaks in HTML
    txt = 'Create new folder ' \
        + '<span style="color:#0049FF">' + newFolderName + '</span><br>' \
        + ' in path ' \
        + '<span style="color:#0049FF">' + newFolderPath + '</span>'
    ret = g_ui.messageBox(txt, 'Create folder?', \
                    adsk.core.MessageBoxButtonTypes.OKCancelButtonType)
    if ret == adsk.core.DialogResults.DialogOK:  # if OK was clicked
      saveToPath = newFolderPath + '\\' + newFolderName  # note that \\ becomes \

      ret = createFolder(saveToPath)
      if ret == 1 or ret == 2:  # if the folder successfully created or exists already
        # check if STLs exist in folder
        fileCnt = checkIfFilesExist(saveToPath, 'stl')
        if fileCnt > 0:
          txt = 'Folder already contains ' + str(fileCnt) + ' .STL files,\n'
          txt += 'which may be overwritten.\n\n'
          txt += 'Click <b>OK</b> to continue or <b>Cancel</b>.'
          ret = showMsgBox(txt, 'orange', 'Warning', \
                           adsk.core.MessageBoxButtonTypes.OKCancelButtonType)
          if ret == adsk.core.DialogResults.DialogCancel:  # if Cancel was clicked
            return

        exportSTLFiles(saveToPath)

      else:
        showMsgBox('Error creating folder', 'red', 'Error', \
                    adsk.core.MessageBoxButtonTypes.OKButtonType)
    #os.mkdir()

    #ret = g_ui.messageBox('path=' + folderDialog.folder)
```

```
    else:   # user clicked Cancel
      pass

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


#============================================================================================
# called by: exportSTLsStart()
#
def checkIfFilesExist(folderPath, extension):
  if extension.find('.', 0, 1) == -1:  # if extension does not start with '.'
    extensionMatch = '.' + extension
  extensionMatch = extensionMatch.lower()  # ex '.stl'

  #g_ui.messageBox('path='+folderPath + '\n  ext='+extension, 'checkIfFilesExist')
  fileCtr = 0
  try:
    for file in os.listdir(folderPath):
      # issue if file has no extension ???
      # fileExtension will start with '.', ie '.stl'
      filename, fileExtension = os.path.splitext(file)
      #g_ui.messageBox('file='+filename + '\n  ext='+fileExtension, 'os.listdir')

      if fileExtension.lower() == extensionMatch:   # if match (case insensitive)
        fileCtr += 1
  except:
    if g_ui:
      g_ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))

  return fileCtr


#============================================================================================
# called by: exportSTLsStart()
# Ex: ret = showMsgBox('Error creating folder', 'red', 'Error', \
#                      adsk.core.MessageBoxButtonTypes.OKButtonType)
#
def showMsgBox(text, textColor, title, buttons: adsk.core.MessageBoxButtonTypes):
  # number of characters that will fit on a line for various font and size for
  # a message
  # font-family:consolas; font-size:10px  80
  # font-family:consolas; font-size:11px  72
  # font-family:consolas; font-size:12px  66
  # font-family:consolas; font-size:14px  58
  # white-space:pre-wrap allows use of \n for line breaks in HTML
  # font-family:courier seemed to be limited to a larger size
  txt = '<div style="font-family:consolas; font-size:14px; color:' + textColor +'; \
             white-space:pre-wrap;">'
        #00000000011111111112222222222333333333344444444445555555 5
        #12345678901234567890123456789012345678901234567890123456 8
        #============================================================
  txt += text + '\n'
  txt += '</div>'  # HTML div end
  # options provided by adsk.core.MessageBoxButtonTypes
```

```
  #  OKButtonType           0  message box contains an OK button (default)
  #  OKCancelButtonType     1  message box contains OK and Cancel buttons
  #  RetryCancelButtonType  2  message box contains Retry and Cancel buttons
  #  YesNoButtonType        3  message box contains Yes and No buttons
  #  YesNoCancelButtonType  4  message box contains Yes, No, and Cancel buttons
  # if return is used, adsk.core.DialogResults is an enumerated constant for values
  return g_ui.messageBox(txt, title, buttons)


#============================================================================================
# called by: exportSTLsStart()
# Ex: ret = createFolder(saveToPath)
#
def createFolder(path):
  # import os  is needed for the os functions
  # os.path.exists() returns True if there is either a folder or a regular file with
  #  the name.
  # os.path.isdir() will return True if the path exists and is a directory, or a
  #  symbolic link to a directory.
  ret = os.path.isdir(path)
  if ret == True:   # if the directory exists
    return 2

  try:
    os.mkdir(path)
  except OSError:  # if the directory creation failed
    return -1
    #print ("Creation of the directory %s failed" % path)
  else:
    return 1
    #print ("Successfully created the directory %s " % path)



#============================================================================================
# called by:  Not used now, but keep for possible future use
# Allows user to create a New Folder.
#
def createFolderDialog():
  try:
    # this can be considered for use
    # returns string for path of Downloads folder  C:\Users\<user>\Downloads\
    #downloadsPath = str(Path.home() / "Downloads")

    # http://help.autodesk.com/view/fusion360/ENU/?guid=GUID-69478fef-f96f-4e7c-b5af-766301072042
    fileDialog = g_ui.createFileDialog()
    fileDialog.isMultiSelectEnabled = False
    fileDialog.title = 'Select location of new folder for STL files.'
    fileDialog.filter = 'All Files (*.*)'
    fileDialog.filterIndex = 0
    fileDialog.initialFilename = g_designName + '_STL Files'
    #fileDialog.initialDirectory =
    dialogResult = fileDialog.showSave()
    if dialogResult == adsk.core.DialogResults.DialogOK:  # user clicked "OK"
      # check if valid file ???
      # returns file name with path
```

```
      # C:\Users\<user>\Downloads\ReactorV2_BOM.txt
      #fileNameWithPath = fileDialog.filename
      pass
    else:  # user clicked "Cancel"
      return
    #g_ui.messageBox('filename=' + fileNameWithPath)

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



#============================================================================================
# called by: exportSTLsStart()
#  Called to export STL files.  This is a different feature from saving the BOM file.
#
# Ex: exportSTLFiles(saveToPath)
# if folder exists use it and don't create new one ???
# append file names with material and qty ???
#
def exportSTLFiles(folder):
 try:
  # check if folder exists ???
  # use global app, product, design ???
  app = adsk.core.Application.get()
  product = app.activeProduct  # design data, toolpath data, ...
  design = adsk.fusion.Design.cast(product)

  exportSTLsStatusUpdate(' Exporting STL files:  ' + str(len(g_printedItems)))
  adsk.doEvents()  # needed for text box update

  # create a single exportManager instance
  exportMgr = design.exportManager

  textHdr = '<div style="font-family:consolas; font-size:12px; color:blue; \
             white-space:pre-wrap;">'

  # create Column Header text and undelines
  # 'Line Qty Material  Part Name'
  # '---  --  --------  ------------------------------------------------------------'
  # create column Titles
  lineText =  setStrSize('Line', ColWidths.lineCtr + ColWidths.space)
  lineText += setStrSize('Qty', ColWidths.qty + ColWidths.space)
  lineText += setStrSize('Material', ColWidths.type + ColWidths.space)
  lineText += setStrSize('Component Name', ColWidths.descrip + ColWidths.space) + '\n'

  # create dashes under titles
  lineText2 =  ('-' * ColWidths.lineCtr) + (' ' * + ColWidths.space)
  lineText2 += ('-' * ColWidths.qty) + (' ' * + ColWidths.space)
  lineText2 += ('-' * ColWidths.type) + (' ' * + ColWidths.space)
  lineText2 += ('-' * ColWidths.descrip) + (' ' * + ColWidths.space) + '\n'

  listText = lineText + lineText2
  g_textBoxExportSTLsList.formattedText = textHdr + listText + '</div>'
  adsk.doEvents()  # needed for text box update
```

```
lineCtr = 0
errorCtr = 0

# loop through components to be 3D Printed
for item in g_printedItems:
  lineCtr += 1
  comp = item.comp
  # Creates an STLExportOptions object that's used to export a design in STL format.
  #  Creation of the STLExportOptions object does not perform the export. You must
  # pass this object to the ExportManager.execute method to perform the export.
  #  Argument: geometry to export. Can be a BRepBody, Occurrence, or Component object.
  # A 2nd optiona argument can specify The filename of the STL file to be created
  # instead of later using the filename property.
  stlOptions = exportMgr.createSTLExportOptions(comp)

  # note that \\ becomes \
  fileNameWithPath = folder + '\\' + item.name + '(' + item.material + ').stl'
  stlOptions.filename = fileNameWithPath

  # allow preference option ???
  # set Mesh Refinement
  stlOptions.meshRefinement = adsk.fusion.MeshRefinementSettings.MeshRefinementMedium

  # create and save .stl file
  # returns True if successful
  ret = exportMgr.execute(stlOptions)
  if ret == False:
    errorCtr += 1

  # start line with HTML formatting to allow lines to have alternating colors
  # lower down the line must end with '</span>' to ensure the HTML style is used
  # for this line only
  if (lineCtr % 2) == 0:  # if even
    lineText = '<span style="color:#006F08">'  # green
  else:  # if odd
    lineText = '<span style="color:#0049FF">'  # blue

  # convert line counter and quantity to strings with leading zeros to ensure these
  # fields always have the same number of characters to ensure aligned columns
  lineCtrText = str(lineCtr).zfill(ColWidths.lineCtr)
  qtyText = str(item.qty).zfill(ColWidths.qty)

  # remove any text starting with '('
  # when materials are duplicated, the names are suffixed with '(1)', '(2)', ...
  materialName = cleanMaterialName(item.material)

  # build line for part using specified column widths
  #  Line Qty Material   Part Name
  # '000  00  ABS        Control Rod Spacer
  lineText += setStrSize(lineCtrText, ColWidths.lineCtr + ColWidths.space)
  lineText += setStrSize(qtyText, ColWidths.qty + ColWidths.space)
  lineText += setStrSize(materialName, ColWidths.type + ColWidths.space)
  lineText += setStrSize(item.name, ColWidths.descrip + ColWidths.space)
  # add column for error ???
```

```
    lineText += '</span>'

    listText += lineText + '\n'
    g_textBoxExportSTLsList.formattedText = textHdr + listText + '</div>'


    exportSTLsStatusUpdate(' Exporting STL files:  ' + str(lineCtr) \
                           + '   Name=' + str(item.name) \
                           + '   Material=' + str(item.material) \
                           + '   Errors=' + str(errorCtr))
    adsk.doEvents()  # needed for text box update

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


#=============================================================================================
# called by: exportSTLFiles()
#
# removes any text starting with '('
# when materials are duplicated, the names are suffixed with '(1)', '(2)', ...
def cleanMaterialName(materialName):
  # returns -1 if string not found, 0 if the string starts at the first character,
  # 1 if at 2nd character, ...
  ret = materialName.find('(')
  if ret > 0:
    return materialName[:ret]
  else:
    return materialName


#=============================================================================================
# called by: exportSTLFiles()
#
def exportSTLsStatusUpdate(text):
  hdr = '<div style="font-family:consolas; background-color:lightblue; \
         font-size:12px; color:blue; white-space:pre-wrap;">'
  Ctrls.txtExportSTLsStatus.formattedText = hdr + text + '</div>'




# not used ???
def getPartFields(text, deliminator):
  fields = text.split(deliminator)
  return fields

#=============================================================================================
# called by: taskRun()
#
# 'Screw  10-32 2"L Socket-Head 18-8SS Fully-Threaded  92196A226  $10.32(25)'
#
def getPartType(text, deliminator, lengthMax):
  #  Using .find() as done in this code, allows the optional use of a multi-character
```

```
  # deliminator, such as 2 spaces.
  # Below is a method that can be used if there is a single character deliminator:
  #  word = text.split(g_partTypeDelim, 1)[0]
  # If one wishes to handle multiple deliminator options a replace can be used to
  # replace one option with another.  Using text.replace('_', ' ') would replace
  # every '_' with a space and then split with using the space.

  # .find() returns the index of the start of the substring, which is the single or
  # multiple character deliminator that seperates the part type from the full name.
  # It returns -1 if the substring is Not found.
  # deliminator, start of string, last char to end search at
  index = text.find(deliminator, 0, lengthMax)
  if index > 0:  # if No deliminator is found
    # text[:index] extracts the text up to the index value.  The first character is at
    # index = 0.  For example, if the index = 2, the first two characters will be returned.
    return text[:index]
  else:
    return ''  # return an empty string


#not called ???
#================================================================================================
# called by: taskRun()
def showCompleteMsgBox(clean):

  if clean == True:  # if Cleaning design, otherwise just listing Sketches and Features
    msgTitle = 'Cleaning Complete'
  else:
    msgTitle = 'Listing Complete'

  # font-family:consolas; font-size:10px  80
  # font-family:consolas; font-size:11px  72
  # font-family:consolas; font-size:12px  66
  # font-family:consolas; font-size:14px  58
  # white-space:pre-wrap allows use of \n for line breaks in HTML
  # font-family:courier seemed to be limited to a larger size
  msgText = '<div style="font-family:consolas; font-size:14px; color:darkblue; \
             white-space:pre-wrap;">'
             #0000000001111111111222222222233333333334444444444555555555
             #1234567890123456789012345678901234567890123456789012345678
             #============================================================
  msgText += 'Save Report to File?' + '\n\n'
  msgText += 'This file will be saved to the Downloads folder.' + '\n'
  msgText += '</div>'  # HTML div end
  # OKButtonType           0  message box contains an OK button (default)
  # OKCancelButtonType     1  message box contains OK and Cancel buttons
  # RetryCancelButtonType  2  message box contains Retry and Cancel buttons
  # YesNoButtonType        3  message box contains Yes and No buttons
  # YesNoCancelButtonType  4  message box contains Yes, No, and Cancel buttons
  ret = g_ui.messageBox(msgText, msgTitle, 3)
  # DialogCancel 1  return value is Cancel (usually sent from a button labeled Cancel)
  # DialogError -1  An unexpected error occurred
  # DialogNo     3  return value is No (usually sent from a button labeled No)
  # DialogOK     0  return value is OK (usually sent from a button labeled OK)
  # DialogYes    2  return value is Yes (usually sent from a buttons labeled Yes and Retry)
  if ret == 2:
```

```
    saveReportFile(clean)


# not called ???
#==============================================================================================
# called by: showCompleteMsgBox()
def saveReportFile(clean):
  try:
    if clean == True:  # if Cleaning design, otherwise just listing Sketches and Features
      fileName = 'Cleaning Results for ' + g_designName + '.txt'
    else:
      fileName = 'Listing Results for ' + g_designName + '.txt'

    # returns string for path of Downloads folder  C:\Users\<user>\Downloads\
    downloadsPath = str(Path.home() / "Downloads")

    fileNameWPath = downloadsPath + '\\' + fileName  # note that \\ becomes \

    #arguments:
    # 2nd  'r' = Opens a file for reading, error if the file does not exist
    #      'a' = Opens a file for appending, creates the file if it does not exist
    #      'w' = Opens a file for writing, creates the file if it does not exist
    #      'x' = Creates the specified file, returns an error if the file exists
    #       'b' can be appended to the above for binary data, text is default
    # 3rd  -1 = for Buffering use system default
    # 4th   utf-8-sig causes a "byte order mark" signature to be written to the start of
    #       the file to indicate UTF-8
    file = open(fileNameWPath, 'w', -1, 'utf-8')
    file.write(g_text)
    file.close()

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


#==============================================================================================
# returns string with a length specified by the 2nd argument, which is the input string
# truncated or padded with trailing spaces.  This is useful to create column aligned
# tables
def setStrSize(text, size):
  if len(text) < size:
    return text + (size - len(text)) * ' '
  else:
    return text[:size]


#==============================================================================================
# starting point of the program
#  context is not used, but could be used to pass info, such as arguments as done
# with a Windows program.
#
def run(context):  # the program starts here
  global g_ui

  g_ui = None
```

```
  # try and except prevents a program crash if a statement in its scope causes an error
  # if a statement causes an error the program execution will jump to except, which will
  # allow the program to provide feedback on the error.  Much nicer then just crashing.
  try:

    app = adsk.core.Application.get()
    g_ui = app.userInterface  # user interface

    # Get the existing command definition or create it if it doesn't already exist.
    cmdDef = g_ui.commandDefinitions.itemById('cmdDialogCleaner')
    if not cmdDef:
      cmdDef = g_ui.commandDefinitions.addButtonDefinition( \
       'cmdDialogCleaner', g_scriptName, g_scriptDescription)

    # Connect to the command created event.
    onCommandCreated = CommandCreatedHandler()
    cmdDef.commandCreated.add(onCommandCreated)
    g_handlers.append(onCommandCreated)

    # Execute the command definition.
    cmdDef.execute()


    # Prevent this module from being terminated when the script returns, because we are waiting for
event handlers to fire.
    adsk.autoTerminate(False)

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


# if the program is closed some cleanup can be done
def stop(context):
  ui = None
  #try:
    #app = adsk.core.Application.get()
    #ui = app.userInterface
    #ui.messageBox('Stop addin')
  #except:
     #if ui:
       #ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```