BCA Makerspace Mr. Barbetta's MakerSpace

"Measure Once, Cut Twice"

Autodesk Fusion API

Design Cleaner

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 programming 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 system designs to automate the process of deleting sketches and dissolving features.

Notice of liability:

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

Fusion can operate in two different modes: **Parametric Modeling** and **Direct Modeling**.

Parametric Modeling maintains a **timeline** and dimension **parameters** for design operations. This allows the convenience of editing operations further back in the timeline to change a design. **Model Parameters** are "automatically" saved during design, but it is also convenient for a user to create **User Parameters** that the user can define. One does need to exercise more discipline for Parametric Modeling. For example, a Component being edited should be activated to keep sketches and the timeline operations associated with that component.

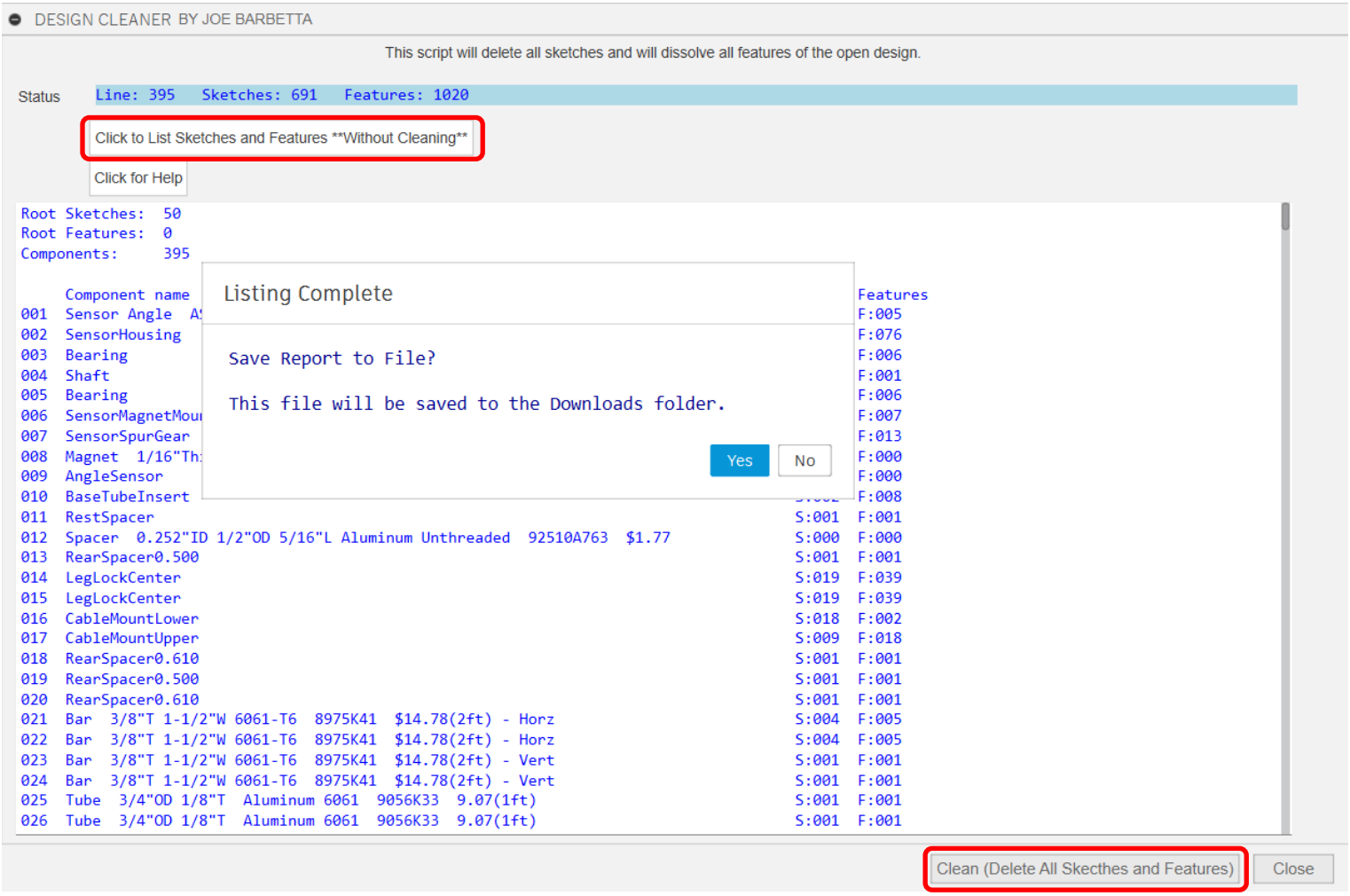
Direct Modeling does not maintain a timeline and does not use parameters. One still has a great amount of flexibility for design. There will be greater use of the Push-Pull and Move tool and new sketches can be created for adding or removing material.

There can still be a mix. For example, a design can be using Direct Modeling and Components can be designed separately using Parametric Modeling and then imported as a linked component. There is also the ability to create a Base Feature using Direct Modeling in a Parametric Modeling design.

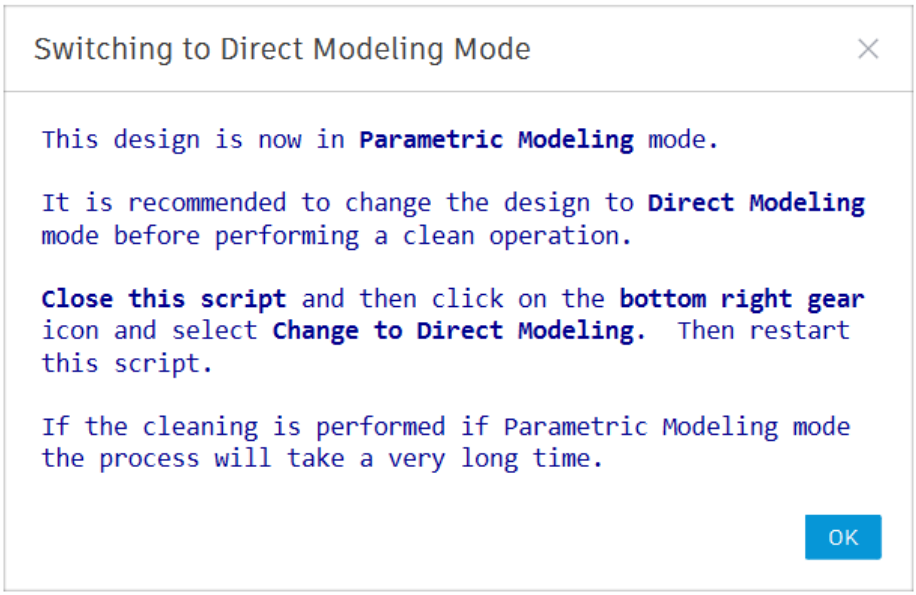
One advantage of using Direct Modeling is performance and exporting of complex designs. In Parametric Modeling, the overhead of the timeline can slow the responsiveness of the software. The file size will also be much larger for a Parametric Modeling design.

This script can be used to only list Sketches and Features (design operations, such as Extrudes, Fillets, etc.) without any "cleaning" and thus does not affect the design file. The other operation is the "Cleaning" operation, where the script will iterate through all Sketches and Features of the design to Delete them. Note that Fusion uses the term "Dissolve" for a feature. This deletes the Feature from the Browser, but the design change of the Feature remains. Fusion also allows a Feature to be "Deleted", which will also negate is effect on the design. This script will Dissolve Features, as opposed to Deleting them.

Below is the top of the script window when run for a design after the top button **Click to List Sketches and Features *Without Cleaning*** is pressed. Each Component is listed with a count of its Sketches and Features. The script also prompts the user to save a report to a file if desired.



The bottom **Clean (Delete All Sketches and Features)** button will start the Clean process. If the design is presently in the **Parametric Modeling** mode, the user will be prompted to **Close the Script** and **change the design to Direct Modeling mode**. The Clean operation can be used with a design in Parametric Modeling, but it will take a very long time, possibly 10 minutes for a complex design, as opposed to seconds for Direct Modeling mode.



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 Cleaner" 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
```

```
# allow ignoring of compoenents starting with '_' ?
# handling of linked components ?
# handling of components without bodies, invisible ?
# use tabs 'Main', 'Settings' ?


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

g_scriptName = 'Design Cleaner'
g_scriptDescription = 'Deletes all sketches and dissolves all features'
g_textTop = 'This script will delete all sketches and will dissolve all features \
             of 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_lineCtrWidth = 3     # number of characters used for line ctr, ie 3 for 000 to 999
g_compNameWidth = 80   # number of characters used for component names


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


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

g_app = None
g_ui  = None

# Global set of event handlers to keep them referenced for the duration of the command
g_handlers = []

#=============================================================================================
# 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 reacts when the command definition is executed which
# results in the command being created and this event being fired.
# created in def run(context) using the following
#  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.  "OK" button?
      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)

      # 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 = 'Clean (Delete All Sketches and Features)'
      cmd.cancelButtonText = 'Close'

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

      createTextBoxs(inputs)

      # 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 fires when the user clicks on the bottom "OK" button.
#
class CommandExecuteHandler(adsk.core.CommandEventHandler):
  def __init__(self):
    super().__init__()
  def notify(self, args):
    try:
      eventArgs = adsk.core.CommandEventArgs.cast(args)
      taskRun(True)                           # True for Cleaning (Not just listing)

    except:
      if g_ui:
```

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


#========================================================================================
# 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, you
# 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 createTextBoxs(inputs):
  global g_textBoxMain, g_textBoxStatus, g_textTop
  # g_text = '<div style="text-align:center; font-size:12px; color:blue">dfsfsffdsf</div>'

  # Create a message that spans the entire width of the dialog by setting the
  # 2nd argument, name, with an empty string.
  textTop = '<div align="center">' + g_textTop + '</div>'
  inputs.addTextBoxCommandInput('fullWidth_textBox', '', textTop, 1, True)

  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 top <b>Click to List... </b> or '
  textStatus += 'bottom <b>Clean (Delete All... </b> buttons 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('readonly_textBox', 'Status', '', 1, True)
  g_textBoxStatus.formattedText = textStatus

  # Create bool value input with button style that can be clicked.
  # (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 List Sketches and Features **Without Cleaning**'

  btnHelp = inputs.addBoolValueInput('buttonHelp', ' ', False, '', False)
  btnHelp.text = 'Click for Help'


  # 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( \
                  'writable_textBox', '', '', g_textBoxLineNum, False)


#=============================================================================================
# 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
      # onInputChange when button is clicked
      if cmdInput.id == 'buttonList':
        taskRun(False)                          # False for Listing only (Not Cleaning)
      elif cmdInput.id == 'buttonHelp':
        #taskRun(True)
        showHelp()

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


#=============================================================================================
#
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;">'
        #0000000001111111111222222222233333333334444444444555555555566666666
```

```
        #12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
        #================================================================
  txt += ' Clicking the button to <b>List Sketeches and Features...</b>'
  txt += 'will scan the design and list the items that it finds, but will'
  txt += 'Not delete them.  Clicking the bottom <b>Close</b> will end the script.\n'
  txt += ' There can be Sketeches and Features for the "root" component, but'
  txt += 'most will be associated with the Components of the design.\n'
  txt += ' Clicking on the bottom <b>Clean (Delete All Skecthes and Features)</b> '
  txt += 'button will cause all the Sketches and Features to be deleted.  To'
  txt += 'delete Features, the Dissolve command is invoked for each.\n'
  txt += ' All this can be done manually, by right-clicking on Sketches and '
  txt += 'selecting <b>Delete</b> and right-clicking on Features and selecting '
  txt += '<b>Dissolve</b>.  This script just performs these operations using'
  txt += 'the Fusion API.\n'
  txt += ' After cleaning the design, one can then save the design if happy '
  txt += 'with the cleaning results.\n'
  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() or CommandExecuteHandler()
# Called when the user clicks the top "List Only" button or
#        the bottom "Clean Design" button.
# If clean = False this function will only list items and Not delete them
#
def taskRun(clean):
  global g_app, g_text, g_compNameWidth, g_designName

 #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


if design.designType == adsk.fusion.DesignTypes.DirectDesignType:
  bDirectModeling = True       # Direct Modeling mode (No History)
else:
  bDirectModeling = False      # Parametric Modeling mode (Maintains History)

if bDirectModeling == False and clean == True:
  showDirectModelingMsgBox()
  return                                    # > > > > > > > > > > > > > > > > > > > > > > > >


if clean == True:
  textStatus = 'Cleaning all Sketches and Features'
else:
  textStatus = 'No Cleaning, only listing Sketches and Features'
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

rootSketches = rootComp.sketches   # read all Root Sketches into a list/array
rootFeatures = rootComp.features   # read all Root Features into a list/array


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

if (g_compNameWidth < 15):
  g_compNameWidth = 15
# create first 4 lines
g_text = 'Design: ' + g_designName + '     Script Run: ' + dateTimeNow + '\n'
g_text += 'Root Sketches:  ' + str(rootSketches.count) + '\n'
g_text += 'Root Features:  ' + str(rootFeatures.count) + '\n'
g_text += 'Components:     ' + str(occurrencesCnt) + '\n' + '\n'
g_text += '     Component name' + ((g_compNameWidth - 14) * ' ') + 'Sketches Features\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
featureCnt = 0
sketchCntPerComp = 0
featureCntPerComp = 0

lineCtr = 0
sketchCtr = rootSketches.count
featureCtr = rootFeatures.count

sketchDeleteCtr = 0
featureDissolveCtr = 0

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

if clean == True:  # if Cleaning design, otherwise just listing Sketches and Features
  # reversed keyword is used to loop from the last element to the first, it was found
  # that looping from first to last was causing only half of the sketches to be deleted.
  # This may be due to the method by which Fusion "reorders" sketches when one is deleted.
  for sketch in reversed(rootSketches):  # loop through all sketches of root component
    ret = sketch.deleteMe()
    if ret == False:
      errorCtr += 1
    sketchDeleteCtr += 1
    # error check if delete fails ???


# loop through all component occurrences
# because the a component can be copied, Autodesk uses the term "occurrence".
for i in range(occurrencesCnt):
  comp = occurrences[i].component

  compSketches = comp.sketches           # read all Sketches into a list/array
  compFeatures = comp.features           # read all features(operations) into a list/array

  sketchCntPerComp = compSketches.count
  featureCntPerComp = compFeatures.count

  sketchCtr += sketchCntPerComp
  featureCtr += featureCntPerComp
  compErrorCtr = 0

  # in the lower for loops, the reversed keyword is used to loop from the last element to
  # the first.  It was found that looping from first to last was causing only half of the
  # sketches to be deleted.  This may be due to the method by which Fusion "reorders"
```

```
    # sketches and features when one is deleted.  Dissolving a feature deletes it.
    if clean == True:  # if Cleaning design, otherwise just listing Sketches and Features
      for sketch in reversed(compSketches):  # loop through all sketches of root component
        ret = sketch.deleteMe()           # delete sketch, check for errors ???
        if ret == False:
          errorCtr += 1
          compErrorCtr += 1
        sketchDeleteCtr += 1

      for feature in reversed(compFeatures):  # loop thru features(operations) of component
        ret = feature.dissolve()          # dissolve feature, this will delete its showing
        if ret == False:
          errorCtr += 1
          compErrorCtr += 1
        featureDissolveCtr += 1


    lineCtr += 1

    # limit the length of the component name
    compName = setStrSize(comp.name, g_compNameWidth)

    g_text += str(lineCtr).zfill(g_lineCtrWidth) + '  ' + compName \
             + '  S:' + str(sketchCntPerComp).zfill(3) \
             + '  F:' + str(featureCntPerComp).zfill(3) + '\n'
#             + '  E:' + str(compErrorCtr).zfill(3) + '\n'
    g_textBoxMain.formattedText = textHdr + g_text + '</div>'

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

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

  # no effect
  #Forces the view to refresh.  Can be useful to refresh to see edits made using the API
  #app.activeViewport.refresh()

  showCompleteMsgBox(clean)


#============================================================================================
# 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;">'
             #000000000111111111122222222233333333334444444445555555555
             #123456789012345678901234567890123456789012345678901234567
             #============================================================
  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)


#=============================================================================================
# 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()))



#============================================================================================
# called by: taskRun()  if the user starts a Clean operation and the design is in
#                       Parametric mode
#
def showDirectModelingMsgBox():

  msgTitle = 'Switching to Direct Modeling Mode'

  # 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 += 'This design is now in <b>Parametric Modeling</b> mode.' + '\n\n'
  msgText += 'It is recommended to change the design to <b>Direct Modeling</b>' + '\n'
  msgText += 'mode before performing a clean operation.' + '\n\n'
  msgText += '<b>Close this script</b> and then click on the <b>bottom right gear</b>' + '\n'
  msgText += 'icon and select <b>Change to Direct Modeling</b>.  Then restart' + '\n'
  msgText += 'this script.' + '\n\n'
  msgText += 'If the cleaning is performed if Parametric Modeling mode' + '\n'
  msgText += 'the process will take a very long time.' + '\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, 0)
  # 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:
  #  do something
```