# Core Logic for Load Step Generation Tool

Servicing Manual

**Language:**  Python 3, TCL, HTML

**Modules** **used:**  xlrd, xlsxwriter, sys, pyinstaller, PyQt5

**Supporting utilities:**  pip, pyuic, pyinstaller, Qt Designer

# Manual:

1. Package installation for servicing
2. Install Python 3 from python.org and set Python to your PATH environment variable
3. Install PyCharm (IDE) from JetBrains
4. It is recommended to create a new virtual environment to work with a project while configuring the Project Interpreter in PyCharm
5. Run the following commands in PyCharm Terminal to setup the required packages on the computer
   1. pip install pyqt5
   2. pip install pyqt5-tools
   3. pip install pyqt5-stubs
   4. pip install xlrd
   5. pip install xlsxwriter
6. TCL commands to export the entity and path data from HyperMesh
7. Start
8. Export the names and the IDs of the entities, LOADs, SPCs, SPCADDs and the directory of HyperMesh installation to a text file by running the enclosed TCL script in HyperMesh
9. The ***hm\_entitylist*** command returns the name and ID of of all entities as a list (loadcol and loadstep details are exported)
10. The ***hm\_info -appinfo ALTAIR\_HOME***command returns the directory where HyperMesh is installed on the computer
11. The ***hm\_getcardimagenamemark*** command returns the image of all entities
12. The ***hm\_getconfigtypeincol*** command returns the config codes for the entity under study; for SPCs, the command returns ‘3 1’ whereas ‘1 1’ for LOADs
13. The image of each entity is examined to separate LOADADD and SPCADD cards
14. All the information fetched by executing this command is dumped into “*D:\collector.txt*”
15. GUI updation
16. The GUI skeleton is designed with the Qt Designer
17. The Qt Designer is a part of the pyqt5-tools packages installed using the pip utility
18. The Designer can be found at *“<PYTHON\_INSTALLATION\_DIRECTORY>\Python37-32\Lib\site-packages\pyqt5\_tools\designer.exe”*
19. No programming is required to setup the basic UI layout and to add widgets to the window
20. All components in the Qt GUI are called as Widgets (all components inherit the QWidget class)
21. The main window in this application is based on the QDialog class
22. Widgets can be added or removed like working with any photo editor
23. ***Note****: It is important to lay all the widgets in a proper layout, failing which will result in goofy arrangement of widgets inside the QDialog class*
24. The right side of the Qt Designer has the “Property Editor” pane where the properties of each widget can be changed
25. Use the “Edit Tab Order” from ***Edit -> Edit Tab Order*** to change the order in which the focus shifts from widget to widget on pressing the *Tab* key
26. Though Qt Designer has the feature to map buttons to actions (signals to slots in Qt words), it is recommended to do this manually via code for flexibility and to write custom actions
27. Once UI changes have been done, save the file in .ui format. This must be converted to .py using *pyuic* tool in Terminal

pyuic <PATH/FILE\_NAME.ui> -o <PATH/FILE\_NAME.py>

1. Find more on the Qt Documentation from <https://doc.qt.io/qt-5/qtdesigner-manual.html>

1. Driver logic
2. Start
3. The application execution begins from

app = QApplication(sys.argv)

ux = UX()

ux.show()

app.exec\_()

This will create a new instance of the QApplication class. This instance is a container for the GUI to show. A new instance of the UX class is created. It is the actual window that we are intending to display to the user. The ***app.exec\_()*** is the terminal point for the application.

1. The *UX* class inherits the *Ui\_load* class and the QDialog class. The *Ui\_load* class is the result of compiling the GUI (.ui file) designed in Qt Designer to .py file. The QDialog class is the super class which refers to the main window rendered on the screen.
2. The UX class has a constructor (the \_\_init\_\_() method)

QDialog.\_\_init\_\_(self)

Ui\_load.\_\_init\_\_(self)

The above statements will invoke the constructor of the inherited classes namely QDialog and *Ui\_load*.

self.setupUi(self)

The *setupUi()* method is used to lay all the widgets on the window and render the UI.

self.browseExcel.clicked.connect(self.setPathExcel)

self.browseTCL.clicked.connect(self.setPathTCL)

self.generate.clicked.connect(self.gen\_excel)

self.continue\_2.clicked.connect(self.main)

self.resetButton.clicked.connect(self.reset)

self.aboutButton.clicked.connect(self.about)

The above statements are used to connect each signal to the respective slot. In Qt terminology, any event is called as a ***signal*** and the handler/catcher for the signal is called as a ***slot***.

The above statements can be generalized as

self.WIDGET\_NAME.SIGNAL.connect(self.SLOT)

In the context of clicked signal for buttons

self.BUTTON\_NAME.clicked.connect(self.SLOT)

A slot is a method that doesn’t take any parameters. Parenthesis ‘()’ should not be added at the end of the slot name. The slot method will not take any parameter other than *self*.

***Right syntax:*** self.generate.clicked.connect(**self.gen\_excel**)

*Wrong syntax:* self.generate.clicked.connect(**~~self.gen\_excel()~~**)

1. On successful rendering of the window, the application will wait for a signal (from the ***‘Generate Excel Workbook’*** button). The tool expects two paths, one for the Excel sheet and another for the TCL file. Two QLineEdits are provided for the same. One more QLineEdit is provided to input the SPC for Pretension, if any. A QCheckBox is provided to check the NLPARM-LGDISP option for non-linear analysis.
2. The button name for “*Generate Excel Workbook*” button is “***generate***” which can be referenced as ***self.generate*** inside the *UX* class. The generate button is linked to the slot ***gen\_excel*** by the statement

self.generate.clicked.connect(self.gen\_excel)

which implies that, on the triggering of ‘*clicked*’ signal from the generate button, the method *gen\_excel()* will be executed.

It is very important to understand the logic of the *gen\_excel()* method as it is wholly responsible for the generation of the Excel Workbook where the user fills in data. Any incorrect formatting in the workbook can lead to unwanted errors in the rest of the app’s lifecycle.

**Algorithm for *gen\_excel()*:**

1. Refer the required global variables
2. Initialize *nlparm* and *nlparm\_id*
3. Open the collector.txt file from the Documents directory and assign it to the variable *raw\_data* with the statement below

raw\_data = open("D:/collector.txt", 'r')

***Note****: The ‘~’ character refers to the home directory of the present user.*

1. Get the path to store the Excel sheet with the below statement

wkbook = self.excel\_file.text()

where *excel\_file* is the name of the QLineEdit which accepts the path for generating the Excel sheet

1. Perform a sequential read on the file associated with the *raw\_data* variable
   1. The first line in the file consists of the names of all load collectors present in the HyperMesh model
   2. The second line consists of all the IDs of the load collectors present in the model
   3. The third line contains the path to the directory where is installed
   4. The fourth line consists of the IDs of all the existing load steps in the model
   5. The fifth line consists of the names of all the existing load steps in the model
   6. The sixth line contains the IDs of all the SPCs defined in the model
   7. The seventh line contains the names of all the SPCs defined in the model
   8. The eighth line contains the IDs of all the LOADADDs present in the model
   9. The ninth line contains the names of all the LOADADDs present in the model
   10. The tenth line contains the IDs of all the SPCADDs defined in the model
   11. The eleventh line contains the names of all the SPCADDs defined in the model
2. Read the first line and store it in *entity\_name* as a list
3. Read the second line and store it in *entity\_id* as a list
4. Initialize *lc\_id* as an empty list and copy all the items in *entity\_id* as integers
5. Read the third line and initialize *hyper\_path* with the HyperMesh directory
6. Read the fourth line and initialize *ls\_id\_list* with the list of IDs of all load steps
7. Iterate through all the items in *ls\_id\_list* and convert them to integers
8. The following loop is to find the maximum ID of load steps in the model

if not len(ls\_id\_list) == 0:  
 check = max(ls\_id\_list)  
 if check == **''** or check == **' '** or check == 0:  
 check = 0  
 max\_LS = int(check)  
else:  
 max\_LS = 0

1. Similarly read the rest of the lines and initialize the capture variables with the corresponding data
2. Capture the Pretension ID from the *ls\_id\_list*, if any, as this must be appended at the end of the LOADADD sheet
3. Check if NLPARM is already defined in the model by checking if NLPARM is present in the *entity\_name* variable
   1. If present, set *nlparm* to False
   2. Remove NLPARM and its ID from *entity\_name* and *entity\_id*
4. Iterate through the list *spcadd\_name* to check if SPCADD cards starting with the name “SPCADD\_” are already present; if present, set *max\_SPC\_add\_count* accordingly
5. Create a new instance of *xlsxwriter.Workbook()* with the path to xlsx sheet as the parameter

wkbk = xlsxwriter.Workbook(wkbook)

*wkbook* is the string containing the path to the xlsx book.

1. Create two sheets – one for LOADADD generation and one for LOADSTEP generation with the following statements

ldcase = wkbk.add\_worksheet('CREATE\_LOADADD')

ldstep = wkbk.add\_worksheet('CREATE\_LOADSTEP')

1. The *set\_column()* method is used to set the width of a column in a excel sheet; its syntax goes as follows

*WORKSHEET\_NAME*.set\_column(*START\_COL, END\_COL, WIDTH*)

1. Use the *add\_format()* method to add styling to the sheet
2. Initialize *lc\_entity\_fin* and *lc\_id\_fin* as empty lists
3. Run the below loop to fill *lc\_entity\_fin* and *lc\_id\_fin* with the items that must be written into the sheet

for i in range(len(entity\_name)):

if entity\_name[i] in spcname or entity\_name[i] in loadadd\_name or entity\_name[i] in spcadd\_name:

continue

else:

lc\_entity\_fin.append(entity\_name[i])

lc\_id\_fin.append(lc\_id[i])

This loop will add all the entities into the lists *lc\_entity\_fin* and *lc\_id\_fin* which aren’t SPCs, SPCADDs or LOADADDs.

1. Write the names and IDs of all the entities in *lc\_entity\_fin* and *lc\_id\_fin* to the *CREATE\_LOADADD* sheet with the below statements in a for loop

**All information in the sheet is written in a vertical fashion. The first column will contain the IDs of the load collectors; the following column will contain the names.**

ldcase.write(i + 2, 1, str(x))

ldcase.write(i + 2, 0, int(lc\_id\_fin[lc\_entity\_fin.index(x)]))

The write() method takes three parameters

* + 1. Row number
    2. Column number
    3. Data to be written

***Note****: Row and column number are absolute indices – they start from 0.*

1. Add Pretension and its name at the end of the sheet, if any
2. Fill in the cells which provide instructions to the user
3. Use the below loop to write the SPC names and their IDs to the *CREATE\_LOADSTEP* sheet from the lists *spcname* and *spcid*

i = 0

for x in spcname:

ldstep.write(1, i + 3, str(x))

ldstep.write(2, i + 3, int(spcid[spcname.index(x)]))

i += 1

1. Use the below conditional statements to merge the cells above the SPC names and name it as ‘SPC’

if not len(spcname) == 1:

ldstep.merge\_range('D1:' + chr(68 + len(spcname) - 1) + '1', 'SPC', merge\_format)

else:

ldstep.write(0, 3, 'SPC', merge\_format)

The *merge\_format* variable consists of the styling information which can be defined with the *add\_format()* method.

1. The *data\_validation()* method is used to write a excel macro for drop-down lists to choose the type of analysis and for enabling Pretension
2. Similarly, another conditional statement is written on all the cells along the second row of *CREATE\_LOADSTEP* sheet to mirror all the LOADADD names from the *CREATE\_LOADADD* sheet

A sample formula looks like

=IF(CREATE\_LOADADD!D2 ="","",CREATE\_LOADADD!D2)

which means that if the source cell is empty, then leave the destination cell empty, else copy the source cell.

1. Close the workbook with the *close()* method along with the name of the workbook as the parameter
2. On the emission of the clicked signal, the *gen\_excel()* method is fired and the Excel workbook is created at the specified location
3. A QLabel stating that the Excel workbook has been successfully generated and the “*continue*” button are made visible on the screen
4. The application now waits for the next signal to be emitted and it is expected from the *continue* button
5. After filling the Excel sheets with the needed data, the *continue* button must be clicked
6. The *continue* button is connected to the slot *main()* with the following statement

self.continue\_2.clicked.connect(self.main)

1. The *main()* method is fired when the button is clicked

**Algorithm for *main()*:**

1. Retrieve the path to the workbook from the QLineEdit
2. Call *loadcase()* with the path to the workbook
3. Call *loadstep()* with the path to the workbook
4. The total workflow is split into two dependent methods namely *loadcase()* and *loadstep()*
5. The *loadcase()* method has to be executed first followed by *loadstep()* and vice versa is not permitted as certain data required by *loadstep()* is set/generated/retrieved by the *loadcase()* method
6. The *loadcase()* method fires the *spcAdd()* method which is used to create SPCADD cards for the load steps where more than one SPC is selected

**Algorithm for *spcAdd(<PATH\_TO\_WORKBOOK><ID>):***

* 1. Refer the required global variables
  2. Initialize *spcList, spcAddDict, selSPCArray and selSPCNoDu*pl as empty lists; the *spcList* will contain the names of the SPCs, the *spcAddDict* will contain a map of the generated SPCADDs to their corresponding IDs, the *selSPCArray* will capture the SPC selection for each load step, the *selSPCNoDupl* will contain distinct combinations of SPC selections – i.e. the same selection of SPCs on different load steps will be absent
  3. Use the *xlrd* package to open the *CREATE\_LOADSTEP* sheet

wkbk = xlrd.open\_workbook(wkbook)

sheet = wkbk.sheet\_by\_index(1)

* 1. Initialize *nLoadsteps* with the number of load steps

nLoadsteps = sheet.nrows

* 1. Iterate through the *CREATE\_LOADSTEP* sheet to capture the SPC selections and append it as a list to *selSPCArray*
  2. Remove duplicates from *selSPCArray* and append it to *selSPCNoDupl*
  3. Get the path to the TCL file and store it in ‘*path’*
  4. Open the TCL file in append mode
  5. Set *iter* as *max\_SPC\_add\_count*; this is to prevent ID numbering conflict if SPCADDs are already present in the model
  6. For each list in *selSPCNoDupl*, write TCL script in the file to generate SPCADD card
  7. Append the names of all SPCADD cards along with their IDs to *l1* and *spcDictHelper* as a tuple

if (tupl not in l1):

l1.append(tupl)

* 1. Create a dictionary from *spcDictHelper* list and assign it to *spcAddDict*
  2. Close the file
  3. Return the ID of the SPCADD card generated at the last

1. On the completion of the execution of *spcAdd()* method, the application will resume the execution of *loadcase()*
2. The data inside the ‘About’ dialog can be edited with any HTML editor. This might be needed to add more information/update existing information/change version numbering and add revisions. HTML is used for rich text formatting

The below code must be changed

def about(self):

msg = QMessageBox()

msg.setIcon(QMessageBox.Information)

msg.setText(‘<***WRITE HTML CONTENT HERE***>’)

msg.setWindowTitle("About")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

1. Main Logic

The core of the load step generator tool lies in two methods namely loadcase() and loadstep().

1. **Algorithm for loadcase(<PATH\_TO\_WORKBOOK>):**
2. Refer all required global variables
3. Initialize l1 as an empty list
4. Use the xlrd package to read xlsx files
5. Create a new instance of a workbook using the xlrd.open\_workbook() method with the path to the workbook as a parameter
6. Open the first sheet with the sheet\_by\_index() method, passing the index of the first sheet (0) as the parameter
7. Count the number of LOADADDs and store it in nlcase with the statement

nlcase = sheet.ncols – 2

1. Set the number of load collectors based on the presence of Pretension
   1. If Pretension is present, set **nlcase = sheet.nrows – 3**
   2. Else, set **nlcase = sheet.nrows – 2**
2. Set the path to the TCL file in ‘path’ variable by accessing the ‘text’ property in the tcl\_file object
3. Open a text file in write mode on the path retrieved from the TCL\_FILE QLineEdit
4. Initialize LCount as an empty list; LCount is a two-dimensional array which will hold the scaling factors for each LOADADD card as a list
5. Run the below loop to read data from the Excel sheet and append it to LCount

for i in range(2, nlcase + 2):

x = []

for j in range(2, nlcount + 2):

x.append(sheet.cell\_value(j, i))

LCount.append(x)

1. Fill the blank values in LCount read from blank cells in the sheet by running the following loop

for m in range(len(LCount)):

for n in range(len(LCount[m])):

if LCount[m][n] == '':

LCount[m][n] = float(0)

1. Run the below loop to transpose LCount; the TCL file will take a series of lists which contain the scaling factors for each load collector (horizontal); since we have captured the scaling factors for each LOADADD (vertical), transposing is necessary

tLCount = [[LCount[j][i] for j in range(len(LCount))] for i in range(len(LCount[0]))]

1. Initialize ctr as 0
2. Iterate through each element in tLCount (the transposed matrix) and write its contents to the TCL file with the statement

new\_tcl.write("set lst" + str(ctr) + ' [split "' + lstdata + '" ","]\n')

1. Find the maximum of lc\_id with the max() function and assign it to highestIDVal; since HyperMesh assigns IDs to the new LOADADD cards based on the existing IDs, it is important to make note of the highest ID in the load collectors section and begin the iteration from the value after highest ID
2. Append all the missed items from lc\_entity\_fin to l1 along with their IDs
3. Begin iteration to write TCL code for each LOADADD entity
   1. Get the name of the LOADADD card from lcase\_name
   2. Append lcase\_name[i] and {highestIDVal + 1 + i} (i.e, name, ID) to l1
   3. The variable k is the number of non-zero scaling factors for each LOADADD; it can be calculated as

k = nlcount - LCount[i].count(0) - LCount[i].count(str(0.0))

* 1. Use the hyper\_path global variable to refer to the HyperMesh installation directory
  2. Form a list of all indices which have non-zero scaling factors with the following loop

nonzerolistindices = [x for x in range(len(LCount[i])) if not (LCount[i][x] == 0 or str(LCount[i][x])

* 1. Increment all values in the nonzerolistindices list by 1
  2. Append each value in nonzerolistindices upto k to loadNumStr which must be later written to the TCL file
  3. Initialize RefIDList as an empty list which will contain all the IDs of the load collectors which have a non-zero scaling factor
  4. Iterate through all the indices in nonzerolistindices and fetch the corresponding ID from the CREATE\_LOADADD sheet with the cell\_value() method; append the retrieved value to RefIDList
  5. Follow the code to write the required commands in the TCL file based on the data captured above
  6. End loop

1. Close the TCL file
2. This application will also generate the NLPARM card if it doesn’t exist in the model
3. The ID for NLPARM ‘nlparmID’ is set as the next available ID after running the above loop
4. Call the spcAdd() method to generate TCL script for creating SPCADD cards in case where more than one SPC is selected for a load step; this method will return the maximum occupied ID
5. Check if the global variable nlparm is true
   1. If true:
      1. Open the TCL file using the path captured earlier
      2. Write script to TCL file for creating the NLPARM card with the ID returned from the spcAdd() method
      3. Append (‘nlparmID’, NLPARM\_ID) to l1
   2. If false:
      1. Append (‘nlparmID’, nlparm\_id) to l1 where nlparm\_id is the global variable containing the NLPARM ID fetched from the collector.txt file in the gen\_excel() method
6. Set errorLabel QLabel that TCL has been generated for LOADADD
7. Make a dictionary with the list l1 containing tuples of names and IDs with the below statement

global\_dict = dict(l1)

This dictionary will contain all the names of the LOADADDs, LOADs, Pretension, NLPARM and other entities other than SPCs and SPCADDs along with their corresponding IDs.

**Dictionary format:**

{([“key”] : “value”)}

1. **Algorithm for loadstep(<PATH\_TO\_WORKBOOK>):**
2. Refer all the global variables required
3. Initialize errorCount as 0
4. Capture the checked state of the QCheckBox to set NLPARM-LGDISP as enabled or disabled correspondingly
5. Open the Excel sheet CREATE\_LOADSTEP and store its reference in loadstep\_sheet
6. Open the Excel sheet CREATE\_LOADADD and store its reference in pretsheet (the details of the Pretension and its ID will be retrieved from this sheet)
7. Initialize the variable nLoadsteps containing the number of load steps as

nLoadsteps = loadstep\_sheet.nrows – 3

1. Initialize nls\_attributes as 0 (the sum of SPC + LOADADD columns in the sheet)
2. Run the following loop to count the number of attributes

i = 3

while (i < 16384):

if not (loadstep\_sheet.cell\_value(1, i) in [0, '']):

nls\_attributes += 1

i += 1

else:

break

The idea is to increment nls\_attributes until a 0 or an empty space is encountered. The number 16384 refers to the maximum number of columns allowed in a xlsx book

1. Initialize loadstep\_matrix as an empty list; this will store the selected items for each load step in the form of a two-dimensional list
2. Run the below loop to iterate through the Excel sheet and capture the selected items

for p in range(3, nLoadsteps + 3):

x = []

for q in range(3, nls\_attributes + 3):

t = loadstep\_sheet.cell\_value(p, q)

if t == 1:

x.append(1)

else:

x.append(0)

loadstep\_matrix.append(x)

This loop will read the data from each cell using the cell\_value() method and evaluate them. If the value is 1, 1 will be appended to the list, else 0 will be appended.

1. Check if all the rows in loadstep\_matrix have atleast one 1; if it doesn’t contain, return an error that at least one SPC must be selected
2. Initialize pretension as an empty list
3. Iterate through the third column in the CREATE\_LOADSTEP sheet.
   1. If the cell’s value equals to “Yes”, append 1 to pretension
   2. Else if cell’s value equals to “No”, append 0 to pretension
   3. Throw an error if an invalid value is found
4. Capture the path to the TCL file in tcl\_file\_name
5. Open the TCL file using the captured path in append mode
6. Initialize pretensionEnabled as False
7. Set pret\_ID as 1; this will change based on the number of load steps already present in the model as well as based on the presence of a “Pretension” load step in the model
8. Set y = -1; this will act as a flag variable to note if “Pretension” is already present in the model
9. Run the below code if Pretension has at least one 1, else skip this block (this block creates the “Pretension” load step)
   1. Set pretensionEnabled as True
   2. If “Pretension” is present in ls\_list, **set pret\_ID as pret\_ls\_id** <global variable set in gen\_excel() method> and **set y as 0** <flag variable changed to imply that “Pretension” load step is already present>
   3. Else,
      1. Read the Pretension ID from pretsheet; if no value is found, use 0 as the ID
      2. Run code to write TCL script
      3. If NLPARM-LGDISP is enabled, branch and write a different set of script in the file
10. Get all the SPCs and their IDs from spcname and spcid and add them as a tuple to the list l1 containing all the names along with their IDs
11. Rebuild global\_dict from l1
12. Iterate through loadstep\_matrix to write script for generating load steps (iter variable: i)
    1. If pretension is enabled, set ID as max\_LS + i + 2
    2. If pretension load step is already present in the model and in all other cases, set ID as max\_LS + i + 1
    3. The names and IDs can be fetched either from the global dictionary, the global variables initialized in the gen\_excel() method or the sheet itself directly based on the need; the current code accesses them mostly from the dictionary and the sheet, based on their availability
    4. Use the index() method to return the first occurrence of 1; in the cases where you need the last index, use

len(loadstep\_matrix[i]) - loadstep\_matrix[i][::-1].index(1)

* 1. A branching is done to direct the program flow based on the type of analysis; different set of statements are executed based on the program’s direction of execution – the logic for both the branches remain the same – TCL codes referring to the type of analysis and corresponding attribute codes change at multiple locations; the overview for both the branches is described below
  2. Check the count of 1’s in loadstep\_matrix[i]; i corresponds to the index of the load step being worked with currently
     1. If only LOAD is selected, return an error and abort execution
     2. If only SPC is selected, throw a warning and resume program execution
     3. If LOAD+SPC count is greater than 1, split loadstep\_matrix[i] into spcArray[] and loadArray[]; spcArray will contain the selections made on SPCs and loadArray will contain the selections made on LOADs
     4. If more than one LOAD is selected, return an error and abort execution

if (loadArray.count(1) > 1):

self.errorLabel.setText(

self.errorLabel.text() + '**Error: More than one load selected** in step ' + str(

i + 1) + '\n')

**raise Exception**

* + 1. If no SPCs are selected, return an error and abort execution

if (spcArray.count(1) == 0):

self.errorLabel.setText(

self.errorLabel.text() + **'Error: No SPC selected** in step ' + str(

i + 1) + '\n')

**raise Exception**

* + 1. If more than one SPC is selected, throw a warning and resume program execution; the SPCADD card generated in the spcAdd() method corresponding to the selection made is linked to the load step whose information is available in global\_dict

line = '\*attributeupdateentity loadsteps ' + id + ' 4147 1 1 0 loadcols ' + str(**global\_dict[loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -loadstep\_matrix[i][::-1].index(1) - 1 + 3)]**) + '\n'

* + 1. In normal cases (one SPC and one LOAD), resume execution
  1. Check if Pretension is enabled for the current load step under interpretation
  2. Write TCL script to the file based on the status of Pretension; there are differences in the number of attributes and their respective codes in each case (Pretension: enabled or disabled)
  3. In case of non-linear analysis, attach the NLPARM card to the load step; there is a conditional branch based on the checked state of the QCheckBox in the UI; NLPARM-LGDISP has different attribute codes
  4. Add provisions to increment errorCount in exception handling block so that they can be detected and required actions can be taken

1. Close the TCL file
2. If errorCount equals to 0, show a QMessageBox with completion information

msg = QMessageBox()

msg.setIcon(QMessageBox.Information)

msg.setText("TCL has been successfully generated!")

msg.setWindowTitle("Completed")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

1. Else show a QMessageBox with error information; also update the error information in the error/activity log managed by the errorLabel object

msg = QMessageBox()

msg.setIcon(QMessageBox.Warning)

msg.setText("Failed to generate TCL!")

msg.setWindowTitle("Error")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

1. Reset errorCount to 0 for the next execution
2. Miscellaneous Logic
3. **setPathExcel()**
4. The **QFileDialog.getSaveFileName()** method shows the Save dialog box. It uses the operating system’s native dialog provider for storage access, therefore advanced code is not needed for this part
5. The above-mentioned function returns a list. The name of the file is the first element of the list which is accessed with the [0]th index
6. If the returned file name doesn’t end with “.xlsx”, append “.xlsx” to the string
7. Set the text inside the excel\_file QLineEdit with the captured string
8. **setPathTCL()**
9. The **QFileDialog.getSaveFileName()** method shows the Save dialog box. It uses the operating system’s native dialog provider for storage access, therefore advanced code is not needed for this part
10. The above-mentioned function returns a list. The name of the file is the first element of the list which is accessed with the [0]th index
11. If the returned file name doesn’t end with “.tcl”, append “.tcl” to the string
12. Set the text inside the tcl\_file QLineEdit with the captured string
13. **reset()**
14. Set the text in all QLineEdits and errorLabel QLabel to ‘’ by using the setText() method on the respective objects
15. Hide continueLabel QLabel and continue\_2 QPushButton with hide() method called on the respective objects
16. Most variables are reinitialized at their parent methods when the application is rerun

**Note**: Code has been written for the generation of GLOBAL\_OUTPUT\_REQUEST and OUTPUT cards – it has been disabled.

To make it functional, uncomment the code block and remove the pass statement at the beginning of the block – this required all the cards to be deleted from the model.

This limitation can be fixed by capturing the list of the IDs of all the cards in the model. The maximum value from this list can be used to generate the cards.

An error would still arise if GLOBAL\_OUTPUT\_REQUEST and OUTPUT cards are already present in the model; this cannot be overcome as no HyperMesh commands are available to return the names of the cards. An exception handling block can be constructed as shown below

if [ **catch** {\*cardcreate "GLOBAL\_OUTPUT\_REQUEST"} ] {

set answer [tk\_messageBox -title "Warning" -message " GLOBAL\_OUTPUT\_REQUEST card not created!" -type ok]

}

if [ **catch** {\*cardcreate "OUTPUT"} ] {

set answer [tk\_messageBox -title "Warning" -message "OUTPUT card not created!" -type ok]

}

This will show a message box with a notification that the respective cards are not created.

# Appendix A:

# TCL Code to extract information from HyperMesh

set file [open **"~/Documents/collector.txt"** w]  
set name [hm\_entitylist loadcol name]  
set id [hm\_entitylist loadcol id]  
set l1 [hm\_entitylist loadstep id]  
set l2 [hm\_entitylist loadstep name]  
  
\*createmark loadcols 1 all  
set images [hm\_getcardimagenamemark loadcol 1]  
set id [hm\_entitylist loadcol id]  
set l1 [hm\_entitylist loadstep id]  
set name [hm\_entitylist loadcol name]  
set reqid **""**set reqname **""**set len [llength $images]  
**for** {set i 0} {$i < $len} {incr i} {  
 set val [lindex $images $i]  
 **if** {$val == **""**} {  
 set t [lindex $id $i]  
 set r [lindex $name $i]  
 set reqid [concat $reqid $t]  
 set reqname [concat $reqname $r]  
 }  
}  
  
set fspcid **""**set fspcname **""**set len [llength $reqid]  
**for** {set i 0} {$i < $len} {incr i} {  
 set val1 [lindex $reqid $i]  
 set val2 [lindex $reqname $i]  
 set test [hm\_getconfigtypeincol loadcols loads $val1 -byid]  
 set first\_val [lindex $test 0]  
 **if** {$first\_val == 3} {  
 set fspcid [concat $fspcid $val1]  
 set fspcname [concat $fspcname $val2]  
 }  
}  
  
puts $fspcid  
puts $fspcname  
  
set loadaddnames **""**set loadaddids **""**set name [hm\_entitylist loadcol name]  
set id [hm\_entitylist loadcol id]  
set l1 [hm\_entitylist loadstep id]  
  
set len [llength $images]  
**for** {set i 0} {$i < $len} {incr i} {  
 set val1 [lindex $id $i]  
 set val2 [lindex $name $i]  
 set test [lindex $images $i]  
 **if** {$test == **"LOADADD"**} {  
 set loadaddids [concat $loadaddids $val1]  
 set loadaddnames [concat $loadaddnames $val2]  
 }  
}  
  
puts $loadaddids  
puts $loadaddnames  
  
set spcaddnames **""**set spcaddids **""**set len [llength $images]  
set name [hm\_entitylist loadcol name]  
set id [hm\_entitylist loadcol id]  
set l1 [hm\_entitylist loadstep id]  
  
**for** {set i 0} {$i < $len} {incr i} {  
 set val1 [lindex $id $i]  
 set val2 [lindex $name $i]  
 set test [lindex $images $i]  
 **if** {$test == **"SPCADD"**} {  
 set spcaddids [concat $spcaddids $val1]  
 set spcaddnames [concat $spcaddnames $val2]  
 }  
}  
  
puts $spcaddids  
puts $spcaddnames  
  
set file [open **"~/Documents/collector.txt"** w]  
set name [hm\_entitylist loadcol name]  
set id [hm\_entitylist loadcol id]  
set l1 [hm\_entitylist loadstep id]  
set l2 [hm\_entitylist loadstep name]  
set PATH\_Hyper [hm\_info -appinfo ALTAIR\_HOME]  
puts $file $name  
puts $file $id  
puts $file $PATH\_Hyper  
puts $file $l1  
puts $file $l2  
puts $file $fspcid  
puts $file $fspcname  
puts $file $loadaddids  
puts $file $loadaddnames  
puts $file $spcaddids  
puts $file $spcaddnames  
  
puts **"Entity export OK!"**set answer [tk\_messageBox -title **"Information"** -message **"All entity information has been exported successfully!"** -type ok]  
close $file

# Appendix B:

# Driver and main program – Python 3

**import** xlrd, sys, xlsxwriter, os  
**from** xlsxwriter.utility **import** xl\_rowcol\_to\_cell  
**from** PyQt5 **import** QtCore, QtWidgets  
**from** PyQt5.QtWidgets **import** \*  
  
  
**class** Ui\_load(object):  
 **def** setupUi(self, load):  
 load.setObjectName(**"load"**)  
 load.resize(645, 583)  
 self.gridLayout = QtWidgets.QGridLayout(load)  
 self.gridLayout.setObjectName(**"gridLayout"**)  
 self.continueLabel = QtWidgets.QLabel(load)  
 self.continueLabel.setEnabled(**True**)  
 self.continueLabel.setObjectName(**"continueLabel"**)  
 self.gridLayout.addWidget(self.continueLabel, 6, 1, 1, 1)  
 self.browseExcel = QtWidgets.QPushButton(load)  
 self.browseExcel.setFocusPolicy(QtCore.Qt.NoFocus)  
 self.browseExcel.setObjectName(**"browseExcel"**)  
 self.gridLayout.addWidget(self.browseExcel, 0, 2, 1, 1)  
 self.excel\_file = QtWidgets.QLineEdit(load)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.excel\_file.sizePolicy().hasHeightForWidth())  
 self.excel\_file.setSizePolicy(sizePolicy)  
 self.excel\_file.setMinimumSize(QtCore.QSize(400, 0))  
 self.excel\_file.setFocusPolicy(QtCore.Qt.StrongFocus)  
 self.excel\_file.setObjectName(**"excel\_file"**)  
 self.gridLayout.addWidget(self.excel\_file, 0, 1, 1, 1)  
 self.browseTCL = QtWidgets.QPushButton(load)  
 self.browseTCL.setFocusPolicy(QtCore.Qt.NoFocus)  
 self.browseTCL.setObjectName(**"browseTCL"**)  
 self.gridLayout.addWidget(self.browseTCL, 1, 2, 1, 1)  
 self.label\_2 = QtWidgets.QLabel(load)  
 self.label\_2.setObjectName(**"label\_2"**)  
 self.gridLayout.addWidget(self.label\_2, 1, 0, 1, 1)  
 self.pretSPC = QtWidgets.QLineEdit(load)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.pretSPC.sizePolicy().hasHeightForWidth())  
 self.pretSPC.setSizePolicy(sizePolicy)  
 self.pretSPC.setMinimumSize(QtCore.QSize(400, 0))  
 self.pretSPC.setFocusPolicy(QtCore.Qt.StrongFocus)  
 self.pretSPC.setText(**""**)  
 self.pretSPC.setObjectName(**"pretSPC"**)  
 self.gridLayout.addWidget(self.pretSPC, 2, 1, 1, 1)  
 self.label\_6 = QtWidgets.QLabel(load)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.label\_6.sizePolicy().hasHeightForWidth())  
 self.label\_6.setSizePolicy(sizePolicy)  
 self.label\_6.setMinimumSize(QtCore.QSize(0, 150))  
 self.label\_6.setMaximumSize(QtCore.QSize(16777215, 40))  
 self.label\_6.setObjectName(**"label\_6"**)  
 self.gridLayout.addWidget(self.label\_6, 8, 0, 1, 3)  
 spacerItem = QtWidgets.QSpacerItem(0, 0, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)  
 self.gridLayout.addItem(spacerItem, 10, 0, 1, 1)  
 spacerItem1 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)  
 self.gridLayout.addItem(spacerItem1, 4, 1, 1, 1)  
 self.generate = QtWidgets.QPushButton(load)  
 self.generate.setFocusPolicy(QtCore.Qt.StrongFocus)  
 self.generate.setObjectName(**"generate"**)  
 self.gridLayout.addWidget(self.generate, 5, 1, 1, 1)  
 self.label\_5 = QtWidgets.QLabel(load)  
 self.label\_5.setObjectName(**"label\_5"**)  
 self.gridLayout.addWidget(self.label\_5, 2, 0, 1, 1)  
 self.label = QtWidgets.QLabel(load)  
 self.label.setObjectName(**"label"**)  
 self.gridLayout.addWidget(self.label, 0, 0, 1, 1)  
 self.errorLabel = QtWidgets.QLabel(load)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.MinimumExpanding)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.errorLabel.sizePolicy().hasHeightForWidth())  
 self.errorLabel.setSizePolicy(sizePolicy)  
 self.errorLabel.setMinimumSize(QtCore.QSize(50, 300))  
 self.errorLabel.setMaximumSize(QtCore.QSize(16777215, 16777215))  
 self.errorLabel.setText(**""**)  
 self.errorLabel.setObjectName(**"errorLabel"**)  
 self.gridLayout.addWidget(self.errorLabel, 9, 0, 1, 3)  
 self.tcl\_file = QtWidgets.QLineEdit(load)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.tcl\_file.sizePolicy().hasHeightForWidth())  
 self.tcl\_file.setSizePolicy(sizePolicy)  
 self.tcl\_file.setMinimumSize(QtCore.QSize(400, 0))  
 self.tcl\_file.setFocusPolicy(QtCore.Qt.StrongFocus)  
 self.tcl\_file.setObjectName(**"tcl\_file"**)  
 self.gridLayout.addWidget(self.tcl\_file, 1, 1, 1, 1)  
 self.checkBox = QtWidgets.QCheckBox(load)  
 self.checkBox.setChecked(**False**)  
 self.checkBox.setObjectName(**"checkBox"**)  
 self.gridLayout.addWidget(self.checkBox, 3, 1, 1, 1)  
 self.continue\_2 = QtWidgets.QPushButton(load)  
 self.continue\_2.setEnabled(**True**)  
 self.continue\_2.setCheckable(**False**)  
 self.continue\_2.setObjectName(**"continue\_2"**)  
 self.gridLayout.addWidget(self.continue\_2, 7, 1, 1, 1)  
 self.resetButton = QtWidgets.QPushButton(load)  
 self.resetButton.setObjectName(**"resetButton"**)  
 self.gridLayout.addWidget(self.resetButton, 10, 2, 1, 1)  
 self.aboutButton = QtWidgets.QPushButton(load)  
 self.aboutButton.setObjectName(**"aboutButton"**)  
 self.gridLayout.addWidget(self.aboutButton, 10, 0, 1, 1)  
  
 self.retranslateUi(load)  
 QtCore.QMetaObject.connectSlotsByName(load)  
 load.setTabOrder(self.excel\_file, self.tcl\_file)  
 load.setTabOrder(self.tcl\_file, self.pretSPC)  
 load.setTabOrder(self.pretSPC, self.checkBox)  
 load.setTabOrder(self.checkBox, self.generate)  
 load.setTabOrder(self.generate, self.continue\_2)  
 load.setTabOrder(self.continue\_2, self.resetButton)  
  
 **def** retranslateUi(self, load):  
 \_translate = QtCore.QCoreApplication.translate  
 load.setWindowTitle(\_translate(**"load"**, **"Load Step Generator for HyperMesh"**))  
 self.continueLabel.setText(\_translate(**"load"**, **"<html><head/><body><p align=\"center\">Fill the generated Excel sheet and click &quot;Continue&quot; to generate TCL</p></body></html>"**))  
 self.browseExcel.setText(\_translate(**"load"**, **"Browse"**))  
 self.browseTCL.setText(\_translate(**"load"**, **"Browse"**))  
 self.label\_2.setText(\_translate(**"load"**, **"Path to save TCL file:"**))  
 self.pretSPC.setPlaceholderText(\_translate(**"load"**, **"Empty if no Pretension"**))  
 self.label\_6.setText(\_translate(**"load"**, **"<html><head/><body><p><span style=' font-weight:600; color:#e70000;'>Check the following in HyperMesh before using the tool:</span></p><p><span style=' font-weight:600;'>1. TCL script to extract SPC and LOAD details is run in HyperMesh.</span></p><p><span style=' font-style:italic;'>---the extracted info will be stored in the Documents folder;</span></p><p><span style=' font-style:italic;'>---will be automatically retieved by this tool.</span></p><p><span style=' font-weight:600;'>2. All load collectors and load steps don't have spaces in their names (underscores are allowed).</span></p><p align='center'><span style=' font-weight:600; font-style:italic; color:#de0003;'>Error/Activity Log</span></p></body></html>"**))  
 self.generate.setText(\_translate(**"load"**, **"Generate Excel Workbook"**))  
 self.label\_5.setText(\_translate(**"load"**, **"SPC ID for Pretension:"**))  
 self.label.setText(\_translate(**"load"**, **"Path to generate Excel file:"**))  
 self.checkBox.setText(\_translate(**"load"**, **"NLPARM (LGDISP)"**))  
 self.continue\_2.setText(\_translate(**"load"**, **"Continue"**))  
 self.resetButton.setText(\_translate(**"load"**, **"Reset"**))  
 self.aboutButton.setText(\_translate(**"load"**, **"About"**))  
  
global\_dict = {}  
l1 = []  
hyper\_path = **''**spcList = []  
selSPCNoDupl = []  
ls\_id\_list = []  
spcAddDict = {}  
max\_LS = 0  
pret\_ls\_id = 1  
ls\_list = []  
lc\_id = []  
loadadd\_name = []  
loadadd\_id = []  
spcadd\_name = []  
spcadd\_id = []  
spcname = []  
spcid = []  
lc\_entity\_fin = []  
lc\_id\_fin = []  
max\_SPC\_add\_count = 1  
nlparm = **True**nlparm\_id = 10000  
  
**class** UX(QDialog, Ui\_load):  
 **def** \_\_init\_\_(self):  
 QDialog.\_\_init\_\_(self)  
 Ui\_load.\_\_init\_\_(self)  
 self.setupUi(self)  
 self.reset()  
 self.browseExcel.clicked.connect(self.setPathExcel)  
 self.browseTCL.clicked.connect(self.setPathTCL)  
 self.generate.clicked.connect(self.gen\_excel)  
 self.continue\_2.clicked.connect(self.main)  
 self.resetButton.clicked.connect(self.reset)  
 self.aboutButton.clicked.connect(self.about)  
  
 **def** setPathExcel(self):  
 excelfname = QFileDialog.getSaveFileName()[0]  
 **if not** excelfname.endswith(**'.xlsx'**):  
 excelfname += **'.xlsx'** self.excel\_file.setText(excelfname)  
  
 **def** setPathTCL(self):  
 tclfname = QFileDialog.getSaveFileName()[0]  
 **if not** tclfname.endswith(**'.tcl'**):  
 tclfname += **'.tcl'** self.tcl\_file.setText(tclfname)  
  
 **def** reset(self):  
 self.excel\_file.setText(**''**)  
 self.tcl\_file.setText(**''**)  
 self.errorLabel.setText(**''**)  
 self.pretSPC.setText(**''**)  
 self.continueLabel.hide()  
 self.continue\_2.hide()  
  
 **def** about(self):  
 msg = QMessageBox()  
 msg.setIcon(QMessageBox.Information)  
 msg.setText(**'<html><head/><body><p><span style=" font-weight:600;">Load Step Generator for HyperMesh</span></p><p><span style=" font-weight:600;">Version 1.0.0 </span>Stable</p><p>Build 20190719<br/></p><p><span style=" font-weight:600;">Tool compatibility:</span></p><p>Tested to work on HyperMesh v2014.x and v2017.2</p><p><span style=" font-weight:600;">Usage:</span></p><p>1. Generation of LOADADD entities</p><p>2. Loadstep generation - for linear/non-linear analysis with and without Pretension</p></body></html>'**)  
 msg.setWindowTitle(**"About"**)  
 msg.setStandardButtons(QMessageBox.Ok)  
 msg.exec\_()  
  
 **def** gen\_excel(self):  
 **global** hyper\_path, max\_LS, nlparm, nlparm\_id, ls\_list, pret\_ls\_id, ls\_id\_list, lc\_id, max\_SPC\_add\_count, loadadd\_id, loadadd\_name, spcid, spcname, spcadd\_id, spcadd\_name, lc\_entity\_fin, lc\_id\_fin  
 nlparm = **True** nlparm\_id = 10000  
 **try**:  
 **try**:  
 raw\_data = open(os.path.expanduser(**"~"**).replace(repr(**'\n'**).strip(**"'"**).strip(**'n'**), **"/"**)+**"/Documents/collector.txt"**, **'r'**)  
 **except**:  
 self.errorLabel.setText(self.errorLabel.text() + **'Error: Run TCL first to export entity information\n'**)  
 **try**:  
 **if** self.excel\_file.text()==**'' or** self.tcl\_file.text()==**''**:  
 **raise** Exception  
 **except**:  
 self.errorLabel.setText(self.errorLabel.text() + **'Error: Have you entered the correct path?\n'**)  
 wkbook = self.excel\_file.text()  
 entity\_name = raw\_data.readline().split()  
 entity\_id = raw\_data.readline().split()  
 lc\_id = []  
 **for** item **in** entity\_id:  
 lc\_id.append(item)  
 **for** i **in** range(len(lc\_id)):  
 lc\_id[i] = int(lc\_id[i])  
 hyper\_path = raw\_data.readline().rstrip(**'\n'**)  
 ls\_id\_list = raw\_data.readline().rstrip(**'\n'**).split()  
  
 **for** i **in** range(len(ls\_id\_list)):  
 ls\_id\_list[i] = int(ls\_id\_list[i])  
  
 **if not** len(ls\_id\_list) == 0:  
 check = max(ls\_id\_list)  
 **if** check == **'' or** check == **' ' or** check == 0:  
 check = 0  
 max\_LS = int(check)  
 **else**:  
 max\_LS = 0  
  
 ls\_list = raw\_data.readline().split()  
 spcid = raw\_data.readline().split()  
 **for** m **in** range(len(spcid)):  
 spcid[m] = int(spcid[m])  
  
 spcname = raw\_data.readline().split()  
  
 loadadd\_id = raw\_data.readline().split()  
 **for** m **in** range(len(loadadd\_id)):  
 loadadd\_id[m] = int(loadadd\_id[m])  
  
 loadadd\_name = raw\_data.readline().split()  
  
 spcadd\_id = raw\_data.readline().split()  
 **for** m **in** range(len(spcadd\_id)):  
 spcadd\_id[m] = int(spcadd\_id[m])  
  
 spcadd\_name = raw\_data.readline().split()  
  
 pret\_ls\_id = 1  
 **if 'Pretension' in** ls\_list:  
 pret\_ls\_id = int(ls\_id\_list[ls\_list.index(**'Pretension'**)])  
  
 temp\_list = []  
 **for** i **in** range(0, len(entity\_name)):  
 temp\_list.append((entity\_name[i], entity\_id[i]))  
 **if "NLPARM" in** entity\_name:  
 nlparm = **False** x = entity\_name.index(**"NLPARM"**)  
 nlparm\_id = int(entity\_id[x])  
 temp\_list.remove((**"NLPARM"**, str(nlparm\_id)))  
 entity\_name.remove(**"NLPARM"**)  
 entity\_id.remove(str(nlparm\_id))  
 temp\_dict = dict(temp\_list)  
  
 *# marker 1* max\_SPC\_add\_count = 1  
 **for** item **in** spcadd\_name:  
 **if** item.startswith(**"SPCADD\_"**):  
 r = item[7:]  
 **if** r.isnumeric():  
 **if** int(r) >= max\_SPC\_add\_count:  
 max\_SPC\_add\_count = int(r) + 1  
  
  
 wkbk = xlsxwriter.Workbook(wkbook)  
 ldcase = wkbk.add\_worksheet(**'CREATE\_LOADADD'**)  
 ldstep = wkbk.add\_worksheet(**'CREATE\_LOADSTEP'**)  
  
 ldstep.set\_column(0, 2, 15)  
  
 cell\_format = wkbk.add\_format({**'bold'**: **True**, **'font\_color'**: **'red'**})  
  
 i = 0  
  
 lc\_entity\_fin = []  
 lc\_id\_fin = []  
 **for** i **in** range(len(entity\_name)):  
 **if** entity\_name[i] **in** spcname **or** entity\_name[i] **in** loadadd\_name **or** entity\_name[i] **in** spcadd\_name:  
 **continue  
 else**:  
 lc\_entity\_fin.append(entity\_name[i])  
 lc\_id\_fin.append(lc\_id[i])  
  
 print(lc\_entity\_fin, lc\_id\_fin)  
  
 i = 0  
 **for** x **in** lc\_entity\_fin:  
 **if** x == **"Pretension"**:  
 **continue** ldcase.write(i + 2, 1, str(x))  
 ldcase.write(i + 2, 0, int(lc\_id\_fin[lc\_entity\_fin.index(x)]))  
 i += 1  
 **if "Pretension" in** entity\_name:  
 ldcase.write(i + 2, 1, **"Pretension"**)  
 ldcase.write(i + 2, 0, int(temp\_dict[**"Pretension"**]))  
  
 ldcase.write(1, 0, **'Enter LC names ->'**, cell\_format)  
 ldcase.write(0, 2, **'Enter scaling factors for required loads against each load case'**, cell\_format)  
  
 i = 0  
 **for** x **in** spcname:  
 ldstep.write(1, i + 3, str(x))  
 ldstep.write(2, i + 3, int(spcid[spcname.index(x)]))  
 i += 1  
 ldstep.write(2, 0, **'Loadstep name'**)  
 ldstep.write(2, 1, **'Loadstep type'**)  
 ldstep.write(2, 2, **'Pretension'**)  
 merge\_format = wkbk.add\_format({  
 **'bold'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'yellow'**})  
 **if not** len(spcname) == 1:  
 ldstep.merge\_range(**'D1:'**+chr(68+len(spcname)-1)+**'1'**, **'SPC'**, merge\_format)  
 **else**:  
 ldstep.write(0, 3, **'SPC'**, merge\_format)  
  
  
 ldstep.data\_validation(**'B3:B1048576'**, {**'validate'**: **'list'**,  
 **'source'**: [**'linear static'**, **'non-linear quasi-static'**]})  
 ldstep.data\_validation(**'C3:C1048576'**, {**'validate'**: **'list'**,  
 **'source'**: [**'Yes'**, **'No'**]})  
  
  
 **for** j **in** range(0, 16370):  
 cell = xl\_rowcol\_to\_cell(1, j + 2)  
 print(cell)  
 ldstep.write(1, j + i + 3,  
 **'=IF(CREATE\_LOADADD!'**+cell+**'="","",CREATE\_LOADADD!'**+cell+**')'**)  
 wkbk.close()  
 self.continue\_2.show()  
 self.continueLabel.show()  
 self.errorLabel.setText(**''**)  
 **except** Exception **as** e:  
 self.errorLabel.setText(self.errorLabel.text()+**'Error: Workbook is possibly open!\nPlease close the workbook and try again!\n'**)  
 print(e)  
  
 **def** spcAdd(self, wkbook, id):  
 **global** hyper\_path, spcList, selSPCNoDupl, spcAddDict, l1, ls\_list, ls\_id\_list, max\_SPC\_add\_count, spcname  
 spcList = []  
 spcAddDict = []  
 selSPCNoDupl = []  
 wkbk = xlrd.open\_workbook(wkbook)  
 sheet = wkbk.sheet\_by\_index(1)  
 nLoadsteps = sheet.nrows  
  
 spcList = []  
 **for** item **in** spcname:  
 spcList.append(item)  
  
 selSPCArray = []  
  
 **for** i **in** range(3, nLoadsteps):  
 checked = []  
 **for** j **in** range(3, len(spcList)+3):  
 **if** str(sheet.cell\_value(i, j)).split(**'.'**)[0] == str(1):  
 checked.append(1)  
 **else**:  
 checked.append(0)  
 **if** checked.count(1)>1:  
 selSPCArray.append(checked)  
  
 **for** x **in** selSPCArray:  
 **if** x **not in** selSPCNoDupl:  
 selSPCNoDupl.append(x)  
  
 path = self.tcl\_file.text()  
 new\_tcl = open(path, **'a'**)  
  
 iter = max\_SPC\_add\_count  
 spcDictHelper = []  
 **for** item **in** selSPCNoDupl:  
 new\_tcl.write(**'\*collectorcreate loadcols "SPCADD\_'**+str(iter)+**'" "" 11\n'**)  
 new\_tcl.write(**'\*createmark loadcols 2 "SPCADD\_'**+str(iter)+**'"\n'**)  
 new\_tcl.write(**'\*dictionaryload loadcols 2 "'**+hyper\_path+**'/templates/feoutput/optistruct/optistruct" "SPCADD"\n'**)  
 new\_tcl.write(**'\*startnotehistorystate {Attached attributes to loadcol "SPCADD\_'**+str(iter)+**'"}\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '**+str(id)+**' 3240 1 2 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '**+str(id)+**' 3235 1 0 0 '**+str(item.count(1))+**'\n'**)  
 text = **'\*createarray '**+str(item.count(1))+**' '  
 for** i **in** range(0, len(spcList)):  
 **if** item[i] == 1:  
 text += str(int(sheet.cell\_value(2, i+3))) + **' '** text += **'\n'** new\_tcl.write(text)  
 new\_tcl.write(**'\*attributeupdateentityidarray loadcols '**+str(id)+**' 370 1 2 0 loadcols 1 '** + str(item.count(1))+**'\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Attached attributes to loadcol "SPCADD\_'**+str(iter)+**'""}\n'**)  
 tupl = (str(item), **"SPCADD\_"** + str(iter))  
 spcDictHelper.append(tupl)  
 tupl = (**"SPCADD\_"** + str(iter), str(id))  
 **if** (tupl **not in** l1):  
 l1.append(tupl)  
 iter += 1  
 id += 1  
  
  
 spcAddDict = dict(spcDictHelper)  
 new\_tcl.close()  
 **return** id  
  
  
 **def** main(self):  
 wkbook = self.excel\_file.text()  
 self.loadcase(wkbook)  
 self.loadstep(wkbook)  
  
 **def** loadcase(self, wkbook):  
 **global** global\_dict  
 **global** l1  
 **global** hyper\_path  
 **global** nlparm, nlparm\_id, lc\_id  
 **global** lc\_entity\_fin, lc\_id\_fin  
  
 l1 = []  
  
 **try**:  
 loadcasesheet = xlrd.open\_workbook(wkbook)  
 sheet = loadcasesheet.sheet\_by\_index(0)  
 nlcase = sheet.ncols - 2 *# no of load cases* **if** (sheet.cell\_value(sheet.nrows - 1, 1) == **'Pretension'**):  
 nlcount = sheet.nrows - 3 *# no of load counter* **else**:  
 nlcount = sheet.nrows - 2 *# no of load counter* path = self.tcl\_file.text()  
 new\_tcl = open(path, **'w'**)  
  
 LCount = []  
 **for** i **in** range(2, nlcase + 2):  
 x = []  
 **for** j **in** range(2, nlcount + 2):  
 x.append(sheet.cell\_value(j, i))  
 LCount.append(x)  
  
 **for** m **in** range(len(LCount)):  
 **for** n **in** range(len(LCount[m])):  
 **if** LCount[m][n] == **''**:  
 LCount[m][n] = float(0)  
  
 **if** (len(LCount) == 0):  
 self.errorLabel.setText(self.errorLabel.text() + **'Error: Workbook is possibly empty!\n'**)  
 **raise** Exception  
  
 *# transpose LCount* tLCount = [[LCount[j][i] **for** j **in** range(len(LCount))] **for** i **in** range(len(LCount[0]))]  
 print(tLCount)  
 **for** q **in** range(0, len(tLCount)):  
 **for** w **in** range(0, len(tLCount[q])):  
 **if** tLCount[q][w] == **''**:  
 tLCount[q][w] = float(0)  
 print(type(tLCount[q][w]))  
  
 print(tLCount)  
  
 *# iter to write lst data to tcl file* ctr = 0  
 **for** x **in** tLCount:  
 ctr = ctr + 1  
 lstdata = str(x)[1:-1]  
 new\_tcl.write(**"set lst"** + str(ctr) + **' [split "'** + lstdata + **'" ","]\n'**)  
  
 *# iter to get loadcase names* lcase\_name = []  
 **for** i **in** range(nlcase):  
 lcase\_name.append(sheet.cell\_value(1, i + 2))  
  
 highestIDVal = max(lc\_id)  
  
 *# append all collector and IDs  
 # marker 2* **for** item **in** lc\_entity\_fin:  
 **if** ((item, int(lc\_id\_fin[lc\_entity\_fin.index(item)])) **not in** l1):  
 l1.append((item, int(lc\_id\_fin[lc\_entity\_fin.index(item)])))  
  
 *# iter to write tcl script for each case* **for** i **in** range(0, nlcase):  
 new\_tcl.write(**"set name "** + lcase\_name[i] + **"\n"**)  
 new\_tcl.write(**"set j [expr "** + str(i + 1) + **"+"** + str(  
 highestIDVal) + **"]\n"**) *# tcl: j is the ID of the loadcase being created (not nlcount+2)* l1.append((lcase\_name[i], i + highestIDVal + 1))  
 k = nlcount - LCount[i].count(0) - LCount[i].count(  
 str(0.0)) *# tcl: k is no. of non-zero scaling factors in a loadcase (non-zero cells in a excel row)* new\_tcl.write(**"set k "** + str(k) + **"\n"**)  
 new\_tcl.write(**'\*startnotehistorystate {Created loadcollector $name}\n'**)  
 new\_tcl.write(**'\*collectorcreate loadcols $name "" 11\n'**)  
 new\_tcl.write(**'\*createmark loadcols 2 $name\n'**)  
 new\_tcl.write(  
 **'\*dictionaryload loadcols 2 "'** + hyper\_path + **'/templates/feoutput/optistruct/optistruct" "LOADADD"\n'**)  
 new\_tcl.write(**'\*startnotehistorystate {Attached attributes to loadcol $name}\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols $j 3240 1 2 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols $j 379 1 2 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols $j 3236 1 0 0 1\n'**)  
 new\_tcl.write(**'\*createdoublearray 1 0\n'**)  
 new\_tcl.write(**'\*attributeupdatedoublearray loadcols $j 380 1 2 0 1 1\n'**)  
 new\_tcl.write(**'\*createarray 1 0\n'**)  
 new\_tcl.write(**'\*attributeupdateentityidarray loadcols $j 383 1 2 0 loadcols 1 1\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Attached attributes to loadcol $name}\n'**)  
 new\_tcl.write(**'\*startnotehistorystate {Attached attributes to loadcol $name}\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols $j 3236 1 0 0 $k\n'**)  
  
 *# iter to get all non-zero scaling factor indices* nonzerolistindices = [x **for** x **in** range(len(LCount[i])) **if not** (LCount[i][x] == 0  
 **or** str(LCount[i][x]) == **'0.0'**)]  
 **for** x **in** range(0, len(nonzerolistindices)):  
 nonzerolistindices[x] += 1  
 loadNumStr = **""  
 for** j **in** range(1, k + 1):  
 new\_tcl.write(  
 **"set load"** + str(j) + **" [lindex $lst"** + str(nonzerolistindices[j - 1]) + **" [expr "** + str(  
 i + 1) + **"-1]]\n"**)  
 loadNumStr += **"$load"** + str(j) + **" "** *# iter to form IDList (non-zero ones)* RefIDList = []  
 **for** t **in** nonzerolistindices:  
 RefIDList.append(int(sheet.cell\_value(t + 1, 0)))  
  
 new\_tcl.write(**'\*createdoublearray $k '** + loadNumStr[:-1] + **'\n'**)  
 new\_tcl.write(**'\*attributeupdatedoublearray loadcols $j 380 1 2 0 1 $k\n'**)  
 new\_tcl.write(**"\*createarray $k "** + str(RefIDList)[1:-1].replace(**","**, **""**) + **"\n"**)  
 new\_tcl.write(**'\*attributeupdateentityidarray loadcols $j 383 1 2 0 loadcols 1 $k\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Attached attributes to loadcol $name}\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Created loadcollector $name}\n'**)  
 new\_tcl.write(**'#iteration '** + str(i + 1) + **' ends\n'**)  
  
 new\_tcl.close()  
  
  
 nlparmID = str(int(i) + int(highestIDVal) + 1)  
  
 **try**:  
 id = self.spcAdd(wkbook, int(nlparmID)+1)  
 **except** Exception **as** e:  
 self.errorLabel.setText(self.errorLabel.text()+**"Error: Failed to generate SPCADD!\n"**)  
 print(e)  
  
 *# NLPARM gen* **if** nlparm == **True**:  
 new\_tcl = open(path, **'a'**)  
 nlparmID = str(id)  
 new\_tcl.write(**'\*startnotehistorystate {Created loadcollector "NLPARM"}\n'**)  
 new\_tcl.write(**'\*collectorcreate loadcols "NLPARM" "" 11\n'**)  
 new\_tcl.write(**'\*createmark loadcols 2 "NLPARM"\n'**)  
 new\_tcl.write(  
 **'\*dictionaryload loadcols 2 "'** + hyper\_path + **'/templates/feoutput/optistruct/optistruct" "NLPARM"\n'**)  
 new\_tcl.write(**'\*startnotehistorystate {Attached attributes to loadcol "NLPARM"}\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 3240 1 2 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 4113 1 0 0 10\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 4232 1 2 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 4234 1 0 0 6\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 4088 1 0 0 25\n'**)  
 new\_tcl.write(**'\*attributeupdatestring loadcols '** + nlparmID + **' 4089 1 0 0 "UPW"\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 4090 1 0 0 0.001\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 4091 1 0 0 0.001\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 4092 1 0 0 1e-007\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 4238 1 0 0 20\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 4240 1 0 0 0.001\n'**)  
 new\_tcl.write(**'\*attributeupdatedouble loadcols '** + nlparmID + **' 10201 1 0 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 10614 1 2 0 1\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Attached attributes to loadcol "NLPARM"}\n'**)  
 new\_tcl.write(**'\*startnotehistorystate {Attached attributes to loadcol "NLPARM"}\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadcols '** + nlparmID + **' 4113 1 1 0 10\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Attached attributes to loadcol "NLPARM"}\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {Created loadcollector "NLPARM"}\n'**)  
 l1.append((**'nlparmID'**, int(nlparmID)))  
 new\_tcl.close()  
 **else**:  
 l1.append((**'nlparmID'**, int(nlparm\_id)))  
  
 self.errorLabel.setText(**"TCL generated for LOADADD!\n"**)  
 new\_tcl.close()  
 global\_dict = dict(l1)  
 *# print(global\_dict)* **except** Exception **as** e:  
 self.errorLabel.setText(**"Error generating TCL file! (Unhandled Exception)\n"**)  
 msg = QMessageBox()  
 msg.setIcon(QMessageBox.Warning)  
 msg.setText(**"Failed to generate TCL!"**)  
 msg.setWindowTitle(**"Error"**)  
 msg.setStandardButtons(QMessageBox.Ok)  
 msg.exec\_()  
 print(e)  
  
 **def** loadstep(self, wkbook):  
 **global** global\_dict, max\_LS, ls\_list, pret\_ls\_id, spcname, spcid  
 **global** l1, selSPCNoDupl, spcAddDict, spcList  
 errorCount = 0  
 lgdisp = int(self.checkBox.checkState())  
 **try**:  
 excel\_input = xlrd.open\_workbook(wkbook)  
 loadstep\_sheet = excel\_input.sheet\_by\_index(1)  
 pretsheet = excel\_input.sheet\_by\_index(0)  
 nLoadsteps = loadstep\_sheet.nrows - 3  
 nls\_attributes = 0  
 i = 3  
 **while** (i < 16384):  
 **if not** (loadstep\_sheet.cell\_value(1, i) **in** [0, **''**]):  
 nls\_attributes += 1  
 i += 1  
 **else**:  
 **break** loadstep\_matrix = []  
 print(nLoadsteps, nls\_attributes)  
 **for** p **in** range(3, nLoadsteps+3):  
 x = []  
 **for** q **in** range(3, nls\_attributes+3):  
 t = loadstep\_sheet.cell\_value(p, q)  
 **if** t == 1:  
 x.append(1)  
 **else**:  
 x.append(0)  
 loadstep\_matrix.append(x)  
  
 **for** x **in** loadstep\_matrix:  
 **if** 1 **not in** x:  
 print(**'here'**)  
 self.errorLabel.setText(self.errorLabel.text() + **'Error: No SPC or Load selected in step '** + str(  
 loadstep\_matrix.index(x) + 1) + **'!\n'**)  
 errorCount += 1  
  
 pretension = []  
 **try**:  
 **for** i **in** range(0, nLoadsteps):  
 **if** (loadstep\_sheet.cell\_value(i + 3, 2) == **'Yes'**):  
 pretension.append(1)  
 **elif** (loadstep\_sheet.cell\_value(i + 3, 2) == **'No'**):  
 pretension.append(0)  
 **else**:  
 **raise** Exception  
 **except** Exception **as** e:  
 self.errorLabel.setText(self.errorLabel.text() + **"Invalid Pretension Value!\n"**)  
 print(e)  
 errorCount += 1  
  
 tcl\_file\_name = self.tcl\_file.text()  
 new\_tcl = open(tcl\_file\_name, **'a'**)  
  
 print(pretension)  
 pretensionEnabled = **False** pret\_ID = str(1)  
 y = -1  
 **try**:  
 **if** (1 **in** pretension):  
 pretensionEnabled = **True  
 if 'Pretension' in** ls\_list:  
 pret\_ID = str(pret\_ls\_id)  
 y = 0  
 **else**:  
 pret\_ID = str(max\_LS + 1)  
 **if**(self.pretSPC.text() == **''**):  
 **raise** Exception  
 pretspcid = int(self.pretSPC.text())  
 **if** (pretsheet.cell\_value(pretsheet.nrows - 1, 1) == **'Pretension'**):  
 pretloadid = int(pretsheet.cell\_value(pretsheet.nrows - 1, 0))  
 **else**:  
 self.errorLabel.setText(self.errorLabel.text()+**"Warning: No Pretension found, will use 0 as Pretension ID.\n"**)  
 pretloadid = 0  
 new\_tcl.write(**'\*startnotehistorystate {LoadSteps Creation}\n'**)  
 new\_tcl.write(**'\*createmark loadcols 1\n'**)  
 new\_tcl.write(**'\*createmark outputblocks 1\n'**)  
 new\_tcl.write(**'\*createmark groups 1\n'**)  
 new\_tcl.write(**'\*loadstepscreate "Pretension" 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 4143 1 1 0 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 4709 1 1 0 9\n'**)  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '**+pret\_ID+**' 4145 1 1 0 loadcols '** + str(int(pretspcid)) + **'\n'**)  
 **if** lgdisp == 0:  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '**+pret\_ID+**' 4187 1 1 0 loadcols '** + str(global\_dict[**'nlparmID'**]) + **'\n'**)  
 **else**:  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '**+pret\_ID+**' 9931 1 1 0 loadcols '** + str(global\_dict[**'nlparmID'**]) + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 3800 1 1 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 707 1 1 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 2396 1 1 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 8134 1 1 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '**+pret\_ID+**' 2159 1 1 0 loadcols '** + str(int(pretloadid)) + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 2160 1 1 0 0\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '**+pret\_ID+**' 10212 1 1 0 0\n'**)  
 new\_tcl.write(**'\*endnotehistorystate {LoadSteps Creation}\n'**)  
 new\_tcl.write(**'#Pretension Created!\n'**)  
 **except**:  
 self.errorLabel.setText(self.errorLabel.text() + **'Error: SPC ID for Pretension is not defined!\n'**)  
 errorCount += 1  
  
 *# store SPC name and ID to dict* SPC\_count = 0  
 **for** item **in** spcname:  
 SPC\_count += 1  
 l1.append((item, str(int(spcid[spcname.index(item)]))))  
 global\_dict = dict(l1)  
 print(global\_dict)  
  
 **for** i **in** range(0, len(loadstep\_matrix)):  
 **if** (pretensionEnabled):  
 id = str(i + 2 + max\_LS)  
 **if** y == 0:  
 id = str(i + 1 + max\_LS)  
 **else**:  
 id = str(i + 1 + max\_LS)  
 new\_tcl.write(**'\*startnotehistorystate {LoadSteps Creation}\n'**)  
  
 **if** (loadstep\_matrix[i].count(1) == 2):  
 line = **'\*createmark loadcols 1 "'** + str(  
 loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)) + **'" "'** line += str(loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) - loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3)) + **'"'** + **'\n'  
 elif** loadstep\_matrix[i].count(1) == 1:  
 line = **'\*createmark loadcols 1 "'** + str(  
 loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)) + **'"'** + **'\n'  
 else**:  
 line = **''  
 pass** new\_tcl.write(line)  
 new\_tcl.write(**'\*createmark outputblocks 1\n'**)  
 new\_tcl.write(**'\*createmark groups 1\n'**)  
 new\_tcl.write(**'\*loadstepscreate "'** + str(loadstep\_sheet.cell\_value(i + 3, 0)) + **'" 1'** + **'\n'**)  
  
 *# check for linear static* **if** (loadstep\_sheet.cell\_value(i + 3, 1) == **'linear static'**):  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 4143 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 4709 1 1 0 1'** + **'\n'**)  
  
 print(loadstep\_matrix[i])  
  
 *# run if only SPC or Load available* **if** (loadstep\_matrix[i].count(1) == 1):  
 *# run if only Load is available* **if not** (str(loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)) **in** spcname):  
 *# line = '\*attributeupdateentity loadsteps ' + id + ' 4147 1 1 0 loadcols ' + str(  
 # global\_dict[loadstep\_sheet.cell\_value(0, loadstep\_matrix[i].index(1) + 4)]) + '\n'  
 # new\_tcl.write(line)* self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: Only Load is selected in step '** + str(i + 1) + **'!\nPlease select atleast one SPC\n'**)  
 errorCount += 1  
 *# run if only SPC is available* **else**:  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)]) + **'\n'** new\_tcl.write(line)  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Warning: Only SPC is selected! in step '** + str(i + 1) + **'\n'**)  
  
  
 *# run if SPC&Load is available* **elif** (loadstep\_matrix[i].count(1) > 1):  
 **try**:  
 spcArray = loadstep\_matrix[i][0:len(spcList)]  
 **for** z **in** range(0, len(spcArray)):  
 **if** spcArray[z] == **''**:  
 spcArray[z] = 0  
  
 loadArray = loadstep\_matrix[i][len(spcList):]  
 print(spcList)  
 print(loadstep\_matrix[i])  
 print(spcArray, loadArray)  
 print(spcAddDict)  
  
 **if**(loadArray.count(1)>1):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: More than one load selected in step '** + str(  
 i + 1)+**'\n'**)  
 **raise** Exception  
  
 **if**(spcArray.count(1) == 0):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: No SPC selected in step '** + str(  
 i + 1)+**'\n'**)  
 **raise** Exception  
  
  
 **if**(spcArray.count(1)>1):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Warning: More than one SPC selected in step '** + str(  
 i + 1)+**'\n'**)  
 print(spcAddDict[str(spcArray)])  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[spcAddDict[str(spcArray)]]) + **'\n'** new\_tcl.write(line)  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4147 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -  
 loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3)]) + **'\n'** new\_tcl.write(line)  
  
 **if** spcArray.count(1) == 1:  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)]) + **'\n'** new\_tcl.write(line)  
 print(loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -  
 loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3))  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4147 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -  
 loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3)]) + **'\n'** new\_tcl.write(line)  
  
 **except** Exception **as** e:  
 self.errorLabel.setText(  
 self.errorLabel.text() + **"Error in Excel sheet, check whether SPC and Loadcases\nare selected properly for step "** + str(  
 i + 2) + **"! \nFailed to generate TCL for Loadstep!\n"**)  
 *# os.system("pause")* errorCount += 1  
 **continue** *# check pretension==yes* **if** (pretension[i] == 0):  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 3800 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 707 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2396 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 8134 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2160 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 10212 1 1 0 1'** + **'\n'**)  
 **else**:  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 3800 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 707 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2396 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 8134 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2160 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*createarray 1 '** + pret\_ID + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateentityidarray loadsteps '** + id + **' 2161 1 1 0 loadsteps 1 1\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 10212 1 1 0 0\n'**)  
  
 *# non-linear quasi-static* **elif** (loadstep\_sheet.cell\_value(i + 3, 1) == **'non-linear quasi-static'**):  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 4143 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 4709 1 1 0 9'** + **'\n'**)  
  
 *# run if only SPC or Load available* **if** (loadstep\_matrix[i].count(1) == 1):  
 *# run if only load is available* **if not** (str(loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)) **in** spcname):  
 *# line = '\*attributeupdateentity loadsteps ' + id + ' 4147 1 1 0 loadcols ' + str(  
 # global\_dict[loadstep\_sheet.cell\_value(0, loadstep\_matrix[i].index(1) + 4)]) + '\n'  
 # new\_tcl.write(line)* self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: Only Load is selected in step '** + str(i + 1) + **'1\nPlease select atleast one SPC.\n'**)  
 errorCount += 1  
 *# run if only SPC is available* **else**:  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)]) + **'\n'** new\_tcl.write(line)  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Warning: Only SPC is selected! in step '** + str(i + 1) + **'\n'**)  
  
 **if** (loadstep\_matrix[i].count(1) == 0):  
 errorCount += 1  
 **continue  
  
 elif** (loadstep\_matrix[i].count(1) > 1):  
 **try**:  
 spcArray = loadstep\_matrix[i][0:len(spcList)]  
 **for** z **in** range(0, len(spcArray)):  
 **if** spcArray[z] == **''**:  
 spcArray[z] = 0  
 loadArray = loadstep\_matrix[i][len(spcList):]  
  
 **if** (loadArray.count(1) > 1):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: More than one load selected in step '** + str(  
 i + 1)+**'\n'**)  
 **raise** Exception  
  
 **if** (spcArray.count(1) == 0):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: No SPC selected in step '** + str(  
 i + 1)+**'\n'**)  
 **raise** Exception  
  
 **if** (spcArray.count(1) > 1):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Warning: More than one SPC selected in step '** + str(  
 i + 1)+**'\n'**)  
  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[spcAddDict[str(spcArray)]]) + **'\n'** new\_tcl.write(line)  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4147 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -  
 loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3)]) + **'\n'** new\_tcl.write(line)  
  
 **if** spcArray.count(1) == 1:  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4145 1 1 0 loadcols '** + str(  
 global\_dict[  
 loadstep\_sheet.cell\_value(1, loadstep\_matrix[i].index(1) + 3)]) + **'\n'** new\_tcl.write(line)  
 line = **'\*attributeupdateentity loadsteps '** + id + **' 4147 1 1 0 loadcols '** + str(  
 global\_dict[loadstep\_sheet.cell\_value(1, len(loadstep\_matrix[i]) -  
 loadstep\_matrix[i][::-1].index(  
 1) - 1 + 3)]) + **'\n'** new\_tcl.write(line)  
 **except** Exception **as** e:  
 self.errorLabel.setText(  
 self.errorLabel.text() + **"Error in Excel sheet, check whether SPC and Loadcases\nare selected properly for step "** + str(  
 i + 2) + **"!\nFailed to generate TCL for Loadstep!\n"**)  
 *# os.system("pause")* errorCount += 1  
 **continue  
 elif** (loadstep\_matrix[i].count(1) > 2):  
 self.errorLabel.setText(  
 self.errorLabel.text() + **"Error: Load+SPC count is more than 2 in step "** + id + **"!\nPlease select only one load and one SPC\n"**)  
 **continue  
 if** lgdisp == 0:  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '** + id + **' 4187 1 1 0 loadcols '** + str(  
 global\_dict[**'nlparmID'**]) + **'\n'**)  
 **else**:  
 new\_tcl.write(**'\*attributeupdateentity loadsteps '** + id + **' 9931 1 1 0 loadcols '** + str(  
 global\_dict[**'nlparmID'**]) + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 3800 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 707 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2396 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 8134 1 1 0 0'** + **'\n'**)  
 **if** (pretension[i] == 1):  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2160 1 1 0 1'** + **'\n'**)  
 new\_tcl.write(**'\*createarray 1 '** + pret\_ID + **'\n'**)  
 new\_tcl.write(  
 **'\*attributeupdateentityidarray loadsteps '** + id + **' 2161 1 1 0 loadsteps loadsteps 1 1\n'**)  
 **elif** pretension[i] == 0:  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 2160 1 1 0 0'** + **'\n'**)  
 new\_tcl.write(**'\*attributeupdateint loadsteps '** + id + **' 10212 1 1 0 0'** + **'\n'**)  
  
 **else**:  
 self.errorLabel.setText(self.errorLabel.text() + **"Check Loadstep Type!"**)  
 new\_tcl.write(**'\*endnotehistorystate {LoadSteps Creation}\n'**)  
 new\_tcl.write(**"#end of iter "** + str(i + 1) + **'\n'**)  
 **if** (errorCount == 0):  
 **pass** *# cards  
 # new\_tcl.write('\*startnotehistorystate {Modified control card}\n')  
 # new\_tcl.write('if [ catch {\*cardcreate "GLOBAL\_OUTPUT\_REQUEST"} ] {\n')  
 # new\_tcl.write('set answer [tk\_messageBox -title "Warning" -message "GLOBAL\_OUTPUT\_REQUEST card not created!" -type ok]\n')  
 # new\_tcl.write('}\n')  
 # new\_tcl.write('\*startnotehistorystate {Attached attributes to card}\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3321 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9630 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9307 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9317 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9327 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3880 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 4119 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 4114 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7121 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2938 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 10688 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 523 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2385 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 4052 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3712 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3885 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 274 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3057 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7113 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8500 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2419 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9709 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3809 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7125 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 4877 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9337 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9347 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9357 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3325 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7093 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3333 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2423 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 4047 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 9275 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 5463 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8949 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 10440 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7329 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7333 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2427 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8153 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8150 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8144 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3642 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 2431 1 2 0 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7337 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 7117 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3891 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 3329 1 2 0 0\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1902 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3881 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3882 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3883 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1901 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 "SORT1"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4871 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "H3D"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4315 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4008 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4876 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2174 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2287 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2175 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9621 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 10026 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 10027 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2939 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1906 1 0 0 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2177 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " "OPTI"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4316 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3336 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 10996 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2176 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2290 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8137 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 "ALL" "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3713 1 2 0 1 2\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1910 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4318 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4867 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2292 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2420 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1916 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4321 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3318 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2424 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 8950 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8951 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "YES"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8952 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1921 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9609 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4323 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3342 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3343 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2294 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2428 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1922 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 "SORT1"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4872 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "H3D"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4324 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3338 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3339 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "CORNER"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9603 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 696 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9606 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9997 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9933 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3643 1 2 0 1 1\n')  
 # new\_tcl.write('\*attributeupdateint cards 1 1923 1 0 0 1\n')  
 # new\_tcl.write('\*createstringarray 1 "SORT1"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4873 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "H3D"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4325 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3386 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 3387 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "CORNER"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 4839 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 1221 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2295 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8136 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8430 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 9932 1 2 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 " "\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 8429 1 2 0 1 1\n')  
 # new\_tcl.write('\*createdoublearray 1 0\n')  
 # new\_tcl.write('\*attributeupdatedoublearray cards 1 9254 1 0 0 1 1\n')  
 # new\_tcl.write('\*createdoublearray 1 0\n')  
 # new\_tcl.write('\*attributeupdatedoublearray cards 1 9255 1 0 0 1 1\n')  
 # new\_tcl.write('\*createarray 1 0\n')  
 # new\_tcl.write('\*attributeupdateintarray cards 1 9280 1 0 0 1 1\n')  
 # new\_tcl.write('\*createdoublearray 1 0\n')  
 # new\_tcl.write('\*attributeupdatedoublearray cards 1 9281 1 0 0 1 1\n')  
 # new\_tcl.write('\*createstringarray 1 "YES"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 1 2432 1 2 0 1 1\n')  
 # new\_tcl.write('\*endnotehistorystate {Attached attributes to card}\n')  
 # new\_tcl.write('\*endnotehistorystate {Modified control card}\n')  
 # new\_tcl.write('\*startnotehistorystate {Modified control card}\n')  
 # new\_tcl.write('if [ catch {\*cardcreate "OUTPUT"} ] {\n')  
 # new\_tcl.write(  
 # 'set answer [tk\_messageBox -title "Warning" -message "OUTPUT card not created!" -type ok]\n')  
 # new\_tcl.write('}\n')  
 # new\_tcl.write('\*startnotehistorystate {Attached attributes to card}\n')  
 # new\_tcl.write('\*attributeupdateint cards 2 3850 1 0 0 2\n')  
 # new\_tcl.write('\*attributeupdatestring cards 2 130 1 0 0 "0"\n')  
 # new\_tcl.write('\*createstringarray 2 "H3D" "OP2"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 2 3851 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 " " "MODEL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 2 3854 1 2 0 1 2\n')  
 # new\_tcl.write('\*createstringarray 2 "ALL" "ALL"\n')  
 # new\_tcl.write('\*attributeupdatestringarray cards 2 3852 1 2 0 1 2\n')  
 # new\_tcl.write('\*endnotehistorystate {Attached attributes to card}\n')  
 # new\_tcl.write('\*endnotehistorystate {Modified control card}\n')  
 # new\_tcl.write('\*startnotehistorystate {Modified control card}\n')  
 # new\_tcl.write('\*endnotehistorystate {Modified control card}\n')* **else**:  
 self.errorLabel.setText(self.errorLabel.text() +  
 **'Error: TCL file is not generated! Please check the excel file!\n'**)  
 msg = QMessageBox()  
 msg.setIcon(QMessageBox.Warning)  
 msg.setText(**"Failed to generate TCL!"**)  
 msg.setWindowTitle(**"Error"**)  
 msg.setStandardButtons(QMessageBox.Ok)  
 msg.exec\_()  
 new\_tcl.close()  
  
 **if** errorCount == 0:  
 self.errorLabel.setText(self.errorLabel.text() + **'TCL generated for loadstep!\n'**)  
 self.errorLabel.setText(self.errorLabel.text() + **'All OK!\n'**)  
 msg = QMessageBox()  
 msg.setIcon(QMessageBox.Information)  
 msg.setText(**"TCL has been successfully generated!"**)  
 msg.setWindowTitle(**"Completed"**)  
 msg.setStandardButtons(QMessageBox.Ok)  
 msg.exec\_()  
 **except** Exception **as** e:  
 self.errorLabel.setText(  
 self.errorLabel.text() + **'Error: TCL generation failed for loadstep! (Unhandled Exception)\n'** + **"\n"**)  
 msg = QMessageBox()  
 msg.setIcon(QMessageBox.Warning)  
 msg.setText(**"Failed to generate TCL!"**)  
 msg.setWindowTitle(**"Error"**)  
 msg.setStandardButtons(QMessageBox.Ok)  
 msg.exec\_()  
 print(e)  
 errorCount = 0  
  
  
app = QApplication(sys.argv)  
ux = UX()  
ux.show()  
app.exec\_()