LA-UR-14-25267

Approved for public release; distribution is unlimited.

Title: (U) MCNP to Abaqus Conversion Script Demonstration

Author(s): David L. Crane
Roger L. Martz

Intended for: MCNP web site

MCNP to Abaqus Conversion Script Demonstration

David L. Crane
Roger L. Martz

July 2014

Abstract

This document demonstrates how to write a python script that will take an MCNP 6 [1] eeout file and convert the contents into an Abaqus [2] database file. In addition, this document demonstrates how to write a PERL wrapper for the python script when only the basic functionality of generating the database file is required.

Discussion

Sometimes the edit output (eeout file) from an MCNP calculation with the unstructured mesh capability needs to be converted to an Abaqus odb database for use by the Abaqus CAE visualization software, etc. How to write the python code to do this using the Abaqus libraries is shown on the following pages. The reader should be aware that this is a demonstration and is not supported by LANL personnel. No guarantees are implied by the attached. Note, that in providing this information, some of the lines have been wrapped so that text wasn't truncated.

The following python code probably shows more features than what a user would need to generate a basic odb file for visualization in Abaqus CAE. The PERL code at the very end demonstrates how to write a PERL script with the minimum number of arguments that will execute the python script.

References

1. T. Goorley, et. al., "Initial MCNP6 Release Overview," Nuclear Technology, 180, 298 (Dec 2012).
2. Dessault Systemes Simulia, Inc., "ABAQUS USER MANUALS, Version 6.9," Providence, RI (2009).

Mcnp to Abaqus Conversion

```
# To compile this python script:
# >abaqus python
# >>import py_compile
# >>py_compile.compile('mcnp_to_abaqus.py')
# >>exit
# To execute:
#'>abaqus cae noGUI="C:\\InstallDir\\mcnp_to_abaqus.pyc"  -- path="C:\WorkDir\sample_mcnp_output_files"
#                          -- filenamein=name  -- asciifilenameout=name -- logfilename=name -- etafile=filename/[none]
#                          -- etafileformat=energy/[power] -- timeconversion=value/[1.0](default-shakes)
#                          -- lengthconversion=value/[1.0](default-cm) -- massconversion=value/[1.0](default-g)
#                          -- fluxconversion=value/[1.0](default-particles/cm^2)
#                          -- energydensityconversion=value/[1.0](default-MeV/g)
#                          -- fissiondensityconversion=value/[1.0](default-MeV/g)
#                          -- initialtimebinnegligible=yes/[no] -- initialtime=time/[none]
#                          -- calculatepowerdensities=yes/[no] -- calculatecumulativeenergydensities=yes/[no]
#                          -- movepowerdensitiestonodes=yes/[no] -- particlefornt11=1/2/[TOTAL_1_2]
#                          -- energybinfornt11=1/2/3/.../[TOTAL] -- input_file_percentage_inc=value/[10]
#                          -- detailedoutput=yes/[no]'
#
#  Note:  The data sets in the data output written to the mcnp output file have an extra term in the first position.
#         That term is the result associated with the background container used when particles are in gaps/voids.
#         Hence all element data begins in the second positon in the data of the data sets.

import os
import sys
import string
import struct
import time
import math
import copy
import time
import allAbaqusMacros

from abaqus import *
from abaqusConstants import *
import __main__
```

```
from odbAccess import *
from odbMaterial import *
from odbSection import *
from abaqusConstants import *

####################################################################################################################################

class readinputfile:

      def __init__(self,path='none',file_name_in='none',binary_file_name_in='none',ascii_file_name_in='none',
                   ascii_file_name_out='none',odb_file_name='none',log_file_name='none',detailedoutput='no',
                  input_file_percentage_inc=10.0):

          #initilize some variables
          self.path=path
          self.filenamein=file_name_in
          self.binaryfilename=binary_file_name_in
          self.asciifilenamein=ascii_file_name_in
          self.asciifilenameout=ascii_file_name_out
          self.logfilename=log_file_name+'.log'
          self.detailedoutput=detailedoutput
          self.odbfilename=odb_file_name
          self.internalasciifile=[]
          self.input_file_percentage_inc=float(input_file_percentage_inc)
          self.output_percentage=float(input_file_percentage_inc)
          self.binary_file_location=0
          self.binary_file_position_one=0
          self.binary_file_position_two=0
          self.binary_file_length=0
          self.binary_all_bytes=[]
          self.binary_all_lines=''
          self.ascii_file_line_count=0
          self.ascii_file_length=0

          #run some methods
          self.open_log_file()
          self.open_ascii_out_file()
          return

      def open_binary_file(self):
          if(os.path.isfile(self.binaryfilename)):
```

```
      try:
        self.binaryfile=open(self.path+self.binaryfilename,'rb')
      except:
        print 'Error: File '+str(self.path+self.binaryfilename)+' could not be opened as a binary file!'
        sys.exit()
    else:
      print 'Error: File '+str(self.path+self.binaryfilename)+' does not exist!'
      sys.exit()
    return

def close_binary_file(self):
    if(self.binaryfilename!='none'):
      self.binaryfile.close()
    return

def open_ascii_out_file(self):
    if(self.asciifilenameout!='none'):
      self.asciifileout=open(self.path+self.asciifilenameout,'w')
    return

def close_ascii_out_file(self):
    if(self.asciifilenameout!='none'):
      self.asciifileout.close()
    return

def open_ascii_in_file(self):
    if(os.path.isfile(self.asciifilenamein)):
      try:
        self.asciifilein=open(self.path+self.asciifilenamein,'r')
      except:
        self.output_to_log_file('exit','Error: File '+str(self.path+self.asciifilenamein)+' could not be opened as a ascii file!')
    else:
      self.output_to_log_file('exit','Error: File '+str(self.path+self.asciifilenamein)+' does not exist!')
    return

def close_ascii_in_file(self):
    self.asciifilein.close()
    return

def open_log_file(self):
    if(self.logfilename!='none'):
```

```
              self.logfile=open(self.path+self.logfilename,'w')
            return

        def close_log_file(self):
            self.logfile.close()
            return

        def output_percent_of_binary_file_complete(self):
            if(float(self.binary_file_location+self.binary_file_position_one)/
float(self.binary_file_length)*100.0>=float(self.output_percentage)):
              output_line='         Percentage complete:  %6.2f     Location:  %20d         File Length:  %20d         Date: %s Time: %s\n'
               % (float(self.binary_file_location)/float(self.binary_file_length)*100.0,
               int(self.binary_file_location),int(self.binary_file_length),
               time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
               self.output_to_log_file('pass',output_line)
              self.output_percentage=self.output_percentage+self.input_file_percentage_inc

        def output_percent_of_ascii_file_complete(self):
            if(float(self.ascii_file_line_count)/float(self.ascii_file_length)*100>=self.output_percentage):
              output_line='          Percentage complete:  %6.2f     Line:  %20d         Total Lines:  %20d         Date: %s Time: %s\n'
              % (float(self.ascii_file_line_count)/float(self.ascii_file_length)*100.0,int(self.ascii_file_line_count),
              int(self.ascii_file_length),time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
               self.output_to_log_file('pass',output_line)
              self.output_percentage=self.output_percentage+self.input_file_percentage_inc

        def output_to_ascii_file(self,exit_or_pass,output_lines):
            self.internalasciifile.append(output_lines)
            if(self.asciifilenameout!='none'):
              self.asciifileout.write(output_lines)
              self.asciifileout.flush()
              if(string.lower(exit_or_pass)=='exit'):
                 self.asciifileout.close()
                 print junk
                 sys.exit()
              else:
                 pass
            return

        def output_to_log_file(self,exit_or_pass_or_detailed,output_lines):

            if(self.logfilename!='none'):
```

```
          if(exit_or_pass_or_detailed=='exit' or exit_or_pass_or_detailed=='pass' ):
            self.logfile.write(output_lines)
           print output_lines
          elif(exit_or_pass_or_detailed=='detailed'):
           if(self.detailedoutput=='yes'):
              self.logfile.write(output_lines)
             print output_lines
          self.logfile.flush()
        if(string.lower(exit_or_pass_or_detailed)=='exit'):
           self.logfile.close()
           print junk
           sys.exit()
        else:
           pass
        return

    def join(self,list,sep):
        joined=''
        for entry in list:
            joined=joined+str(entry)
            if(sep!='none'):joined=joined+sep
        return(joined)

    def readbinaryfile(self):

        #         Open binary file and determine byte order for platform
        self.open_binary_file()
        self.output_to_log_file('detailed','Byte order for platform is "'+str(sys.byteorder)+' endian".\n')
        if(sys.byteorder=='little'):fmtprefix='<'
        elif(sys.byteorder=='big'):fmtprefix='>'
        self.output_to_log_file('detailed','Format prefix for "'+str(sys.byteorder)+' endian" is: '+str(fmtprefix)+'\n')
        self.output_to_log_file('detailed','size of c: '+str(struct.calcsize(fmtprefix+'c'))+'\n')
        self.output_to_log_file('detailed','size of i: '+str(struct.calcsize(fmtprefix+'i'))+'\n')
        self.output_to_log_file('detailed','size of l: '+str(struct.calcsize(fmtprefix+'l'))+'\n')
        self.output_to_log_file('detailed','size of q: '+str(struct.calcsize(fmtprefix+'q'))+'\n')
        self.output_to_log_file('detailed','size of f: '+str(struct.calcsize(fmtprefix+'f'))+'\n')
        self.output_to_log_file('detailed','size of d: '+str(struct.calcsize(fmtprefix+'d'))+'\n')
        self.output_to_log_file('detailed','size of h: '+str(struct.calcsize(fmtprefix+'h'))+'\n')

        #         Read lines from binary file
        self.output_to_log_file('detailed','Started: Reading lines from bindary file\n')
```

```
data_lines=self.binaryfile.readlines()
self.output_to_log_file('detailed','Finished: Reading lines from bindary file\n')

#         Combine lines and count lengths
self.output_to_log_file('detailed','Started: Combining lines and counting lengths\n')
self.binary_file_length=0
fmt=fmtprefix+'c'
for line in data_lines:
    self.binary_file_length=self.binary_file_length+len(line)
    self.binary_all_lines=self.binary_all_lines+line
    count=0
    while(count<len(line)):
      self.binary_all_bytes.append(struct.unpack(fmt,line[count:count+1])[0])
      count=count+1
self.output_to_log_file('detailed','Finished: Combining lines and counting lengths\n')
#         Output length of combined lines
output_line='          Number of locations in file: %i\n' % (len(self.binary_all_lines))
self.output_to_log_file('pass',output_line)

#         Output file information as characters for at most max locations
max=len(self.binary_all_lines)
if(max>26):max=26
self.output_to_log_file('detailed','Started: Output file information as characters for at most '+str(max)+' locations\n')
fmt=fmtprefix+'c'
self.output_to_log_file('detailed','\n\nformat: '+str(fmt)+'\n')
self.output_to_log_file('detailed','format size in bytes: '+str(struct.calcsize(fmt))+'\n')
count=0
while(count<max):
     self.output_to_log_file('detailed',str(struct.unpack(fmt,self.binary_all_lines[count:count+1]))+'\n')
     count=count+1
self.output_to_log_file('detailed','Finished: Output file information as characters for at most '+str(max)+' locations\n')

#         Determine file header size
self.output_to_log_file('detailed','Started: Determine file header size\n')
output_text='\n\nFile does not have 4 or 8 byte headers/trailers\n'
try:
    fmt4byteheader=fmtprefix+'i10ci'
    self.output_to_log_file('detailed','\n\ntrying 4 byte header/trailers:\n')
    self.output_to_log_file('detailed','format: '+str(fmt4byteheader)+'\n')
    self.output_to_log_file('detailed','format size in bytes: '+str(struct.calcsize(fmt4byteheader))+'\n')
```

```
                self.output_to_log_file('detailed','first 16 elements of file:
'+str(struct.unpack(fmt4byteheader,self.binary_all_lines[0:18]))+'\n')
                output_text='\n\nFile has 4 byte headers/trailers\n'
                fmtprefix=fmtprefix+'i'
                fmtsuffix='i'
                last_position=18
            except:
                self.output_to_log_file('detailed','\nFile does not have 4 byte headers/trailers\n')
                fmt8byteheader=fmtprefix+'q10cq'
                self.output_to_log_file('detailed','trying 8 byte header/trailers:\n')
                self.output_to_log_file('detailed','format: '+str(fmt8byteheader)+'\n')
                self.output_to_log_file('detailed','format size in bytes: '+str(struct.calcsize(fmt8byteheader))+'\n')
                self.output_to_log_file('detailed','first 26 elements of file:
'+str(struct.unpack(fmt8byteheader,self.binary_all_lines[0:26]))+'\n')
                output_text='File has 8 byte headers/trailers\n'
                fmtprefix=fmtprefix+'q'
                fmtsuffix='q'
                last_position=26

            self.output_to_log_file('detailed',output_text)
            self.output_to_log_file('detailed','length of headers: '+str(struct.calcsize(fmtprefix))+'\n')
            self.output_to_log_file('detailed','length of trailers: '+str(struct.calcsize(fmtsuffix))+'\n')
            self.output_to_log_file('detailed','Finised: Determine file header size\n')

            #         Read first line from binary file "MCNP EDITS A" or "MCNP EDITS B" 12 Characters where A-ASCII B-BINARY
            self.output_to_log_file('detailed','Started: Read first line of file\n')
            linecount=1
            self.binary_file_location=0
            fmt=fmtprefix+'12c'+fmtsuffix
            string=self.binary_all_lines[self.binary_file_location:self.binary_file_location+int(struct.calcsize(fmt))]
            if(self.binary_file_location+int(struct.calcsize(fmt))>(len(self.binary_all_lines)-1))
            :self.output_to_log_file('exit','Error: Assumed length of first record '+str(fmt)+' longer than length of binary file!')
            self.binary_file_location=self.binary_file_location+int(struct.calcsize(fmt))
            self.output_to_log_file('detailed','line: '+str(linecount)+'\n')
            self.output_to_log_file('detailed','fmt = '+str(fmt)+'\n')
            self.output_to_log_file('detailed','fmt size = '+str(struct.calcsize(fmt))+'\n')
            self.output_to_log_file('detailed','len(string) = '+str(len(string))+'\n')
            self.output_to_log_file('detailed',str(struct.unpack(fmt,string))+'\n')
            if(self.detailedoutput=='yes'):self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)[1:-1],'none'))+'\n')
            self.output_to_log_file('detailed','\n')
            self.output_to_log_file('detailed','Finished: Read first line of file\n')
```

```
 #         Read the rest of the lines in the file
 self.output_to_log_file('detailed','Started: Read the rest of the lines in file\n')
maxlines=1e30
readtitleline='no'
metadataline='yes'
reachedlastline='no'
while(linecount<maxlines and reachedlastline=='no'):
  if(metadataline=='yes'):
    linecount=linecount+1
    fmt=fmtprefix+'6q'+fmtsuffix
    string=self.binary_all_lines[self.binary_file_location:self.binary_file_location+int(struct.calcsize(fmt))]
    data_fmt=struct.unpack(fmt,string)[1:-1]
    if(self.binary_file_location+int(struct.calcsize(fmt))>(len(self.binary_all_lines)-1)):
      reachedlastline='yes'
      break
    self.binary_file_location=self.binary_file_location+int(struct.calcsize(fmt))
    self.output_to_log_file('detailed','line: '+str(linecount)+'\n')
    self.output_to_log_file('detailed','metadataline='+str(metadataline)+'\n')
    self.output_to_log_file('detailed','fmt = '+str(fmt)+'\n')
    self.output_to_log_file('detailed','fmt size = '+str(struct.calcsize(fmt))+'\n')
    self.output_to_log_file('detailed','len(string) = '+str(len(string))+'\n')
    self.output_to_log_file('detailed',str(struct.unpack(fmt,string))+'\n')
    self.output_to_log_file('detailed',str(struct.unpack(fmt,string)[1:-1])+'\n')
    self.output_to_log_file('detailed','\n')
    metadataline='no'
    readtitleline='no'

  elif(readtitleline=='no' and metadataline=='no' and data_fmt[0]!=0):
    linecount=linecount+1
    fmt=fmtprefix+str(data_fmt[0])+'c'+fmtsuffix
    string=self.binary_all_lines[self.binary_file_location:self.binary_file_location+int(struct.calcsize(fmt))]
    if(self.binary_file_location+int(struct.calcsize(fmt))>(len(self.binary_all_lines)-1)):
      reachedlastline='yes'
      break
    self.binary_file_location=self.binary_file_location+int(struct.calcsize(fmt))
    self.output_to_log_file('detailed','line: '+str(linecount)+'\n')
    self.output_to_log_file('detailed','metadataline='+str(metadataline)+'\n')
    self.output_to_log_file('detailed','fmt = '+str(fmt)+'\n')
    self.output_to_log_file('detailed','fmt size = '+str(struct.calcsize(fmt))+'\n')
    self.output_to_log_file('detailed','len(string) = '+str(len(string))+'\n')
  self.output_to_log_file('detailed',str(struct.unpack(fmt,string))+'\n')
  self.output_to_log_file('detailed',str(struct.unpack(fmt,string)[1:-1])+'\n')
  self.output_to_log_file('detailed','\n')
  self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)[1:-1],'none'))+'\n')
  if(data_fmt[1]==0):metadataline='yes'
  readtitleline='yes'

elif(metadataline=='no' and data_fmt[1]!=0):
  count2=0
  while(count2<data_fmt[1] and linecount<maxlines and reachedlastline=='no'):
    count2=count2+1
    fmt_character=''
    if(data_fmt[2]==0):
      pass
    elif(data_fmt[2]==1):
      fmt_character=str(data_fmt[3]*data_fmt[4])+'c'
    elif(data_fmt[2]==2):
      if(data_fmt[3]==4):fmt_character=str(data_fmt[4])+'i'
      elif(data_fmt[3]==8):fmt_character=str(data_fmt[4])+'q'
      else:pass
    elif(data_fmt[2]==3):
      if(data_fmt[3]==4):fmt_character=str(data_fmt[4])+'f'
      elif(data_fmt[3]==8):fmt_character=str(data_fmt[4])+'d'
      else:pass
    else:
      pass

    fmt=fmtprefix+fmt_character+fmtsuffix
    string=self.binary_all_lines[self.binary_file_location:self.binary_file_location+int(struct.calcsize(fmt))]


    if(data_fmt[5]<=1):
      linecount=linecount+1
      self.output_to_log_file('detailed','line: '+str(linecount)+'\n')
      self.output_to_log_file('detailed','metadataline='+str(metadataline)+'\n')
      self.output_to_log_file('detailed','fmt = '+str(fmt)+'\n')
      self.output_to_log_file('detailed','fmt size = '+str(struct.calcsize(fmt))+'\n')
      self.output_to_log_file('detailed','len(string) = '+str(len(string))+'\n')
      self.output_to_log_file('detailed',str(struct.unpack(fmt,string)[1:-1])+'\n')
      if(data_fmt[2]==1):self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)[1:-1],'none'))+'\n')
      else:self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)[1:-1],' '))+'\n')
```

```
                  self.output_to_log_file('detailed','\n')

                elif(data_fmt[5]>1):
                  self.binary_file_position_one=1
                  self.binary_file_position_two=self.binary_file_position_one+data_fmt[5]
                  cumlativelength=0
                  while(cumlativelength<len(struct.unpack(fmt,string)[1:-1]) and linecount<maxlines):
                     linecount=linecount+1
                     self.output_to_log_file('detailed','line: '+str(linecount)+'\n')
                     if(self.binary_file_position_two <= (len(struct.unpack(fmt,string))-1)):
                       self.output_to_log_file('detailed',str(self.join(struct.unpack(fmt,string)
                       [self.binary_file_position_one:self.binary_file_position_two],' '))+'\n')
                     else:
                       self.output_to_log_file('detailed',str(self.join(struct.unpack(fmt,string)[self.binary_file_position_one:-1],'
'))+'\n')
                     if(self.binary_file_position_two <= (len(struct.unpack(fmt,string))-1)):
                     self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)
                     [self.binary_file_position_one:self.binary_file_position_two],' '))+'\n')
                     else:self.output_to_ascii_file('pass',str(self.join(struct.unpack(fmt,string)[self.binary_file_position_one:-1],'
'))+'\n')
                     self.output_to_log_file('detailed','\n')
                     self.binary_file_position_one=self.binary_file_position_two
                     self.binary_file_position_two=self.binary_file_position_one+data_fmt[5]
                     cumlativelength=cumlativelength+data_fmt[5]
                      self.output_percent_of_binary_file_complete()
                  self.binary_file_position_one=0
                  self.binary_file_position_two=0

                 if(self.binary_file_location+int(struct.calcsize(fmt))>(len(self.binary_all_lines)-1)):
                   reachedlastline='yes'
                   break
                self.binary_file_location=self.binary_file_location+int(struct.calcsize(fmt))
                 self.output_percent_of_binary_file_complete()
              metadataline='yes'

            else:
              break
             self.output_percent_of_binary_file_complete()

          self.output_to_log_file('detailed','Finished: Read the rest of the lines in file\n')
```

```
    self.binary_file_location=self.binary_file_length
    self.output_percent_of_binary_file_complete()

    self.close_binary_file()
    self.close_ascii_out_file()

    return(self.internalasciifile)

def readasciifile(self):

    #         Open ascii file
    self.open_ascii_in_file()
    #         Read and output number of lines for ascii file
    lines=self.asciifilein.readlines()
    self.ascii_file_length=len(lines)
    output_line='          Number of lines in file: %i\n' % (self.ascii_file_length)
    self.output_to_log_file('pass',output_line)

    self.ascii_file_line_count=1
    self.output_to_ascii_file('pass',str(lines[self.ascii_file_line_count-1]))
    self.ascii_file_line_count=self.ascii_file_line_count+1
    while(self.ascii_file_line_count<=len(lines)):
         metadata=string.split(string.strip(lines[self.ascii_file_line_count-1]))
         self.ascii_file_line_count=self.ascii_file_line_count+1
         sub_line_count=1
         title_line=0
         if(int(metadata[0])>0):title_line=1
         if(float(metadata[5])>0.0):multiplier=(math.ceil(float(metadata[4])/float(metadata[5])))
         else:multiplier=1.0
         if(int(metadata[2])==1):multiplier=1.0
         total_sub_lines=(title_line+int(metadata[1])*multiplier)
         while(sub_line_count<=total_sub_lines):
            sub_line_count=sub_line_count+1
            self.output_to_ascii_file('pass',str(lines[self.ascii_file_line_count-1]))
            self.output_percent_of_ascii_file_complete()
            self.ascii_file_line_count=self.ascii_file_line_count+1

    self.close_ascii_out_file()

    return(self.internalasciifile)
```

```
def readfile(self,command_line_arguments):


    filetype=self.determine_file_type()

    if(filetype=='binary'):
      output_line='     Started reading binary input file: Date: %s Time: %s\n' %
      (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
      self.output_to_log_file('pass',output_line)
      input_file=self.readbinaryfile()
      output_line='     Finished reading binary input file: Date: %s Time: %s\n' %
      (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
      self.output_to_log_file('pass',output_line)
    elif(filetype=='ascii'):
      output_line='     Started reading ascii input file: Date: %s Time: %s\n' %
      (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
      self.output_to_log_file('pass',output_line)
      input_file=self.readasciifile()
      output_line='     Finished reading ascii input file: Date: %s Time: %s\n' %
      (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
      self.output_to_log_file('pass',output_line)
    self.input_file=input_file

    return(self.input_file)

def determine_file_type(self):

   self.output_to_log_file('detailed','\n\nDetermining file type:\n')

   blocksize=512
   self.fileintest=open(self.path+self.filenamein,'r').read(blocksize)

   filetype='none'
   if(self.fileintest[0:12]=='MCNP EDITS A'):
     self.output_to_log_file('detailed','\nFile type: ascii\n\n')
     filetype='ascii'
     self.asciifilename=self.filenamein
     return(filetype)
   else:

     #          Open binary file and determine byte order for platform
```

```
self.output_to_log_file('detailed','\n     Determining byte order for platform:\n')
if(sys.byteorder=='little'):fmtprefix='<'
elif(sys.byteorder=='big'):fmtprefix='>'
if(fmtprefix=='<'):self.output_to_log_file('detailed','\n          Byte order = little edian\n')
if(fmtprefix=='>'):self.output_to_log_file('detailed','\n          Byte order = big edian\n')

 #          Determine file header size
self.output_to_log_file('detailed','\n\n     Determining file header size:\n')
output_text='\n\n          File does not have 4 or 8 byte headers/trailers\n'
try:
   fmt4byteheader=fmtprefix+'i12ci'
   self.output_to_log_file('detailed','\n          Trying 4 byte header/trailers:\n')
   self.output_to_log_file('detailed','          Format: '+str(fmt4byteheader)+'\n')
   self.output_to_log_file('detailed','          Format size in bytes: '+str(struct.calcsize(fmt4byteheader))+'\n')
   self.output_to_log_file('detailed','          First 20 elements of file:
   '+str(struct.unpack(fmt4byteheader,self.fileintest[0:20]))+'\n')
   output_text='\n          File has 4 byte headers/trailers\n'
   fmtprefix=fmtprefix+'i'
   fmtsuffix='i'
   last_position=20
   filetype='binary'
   self.binaryfilename=self.filenamein
except:
   self.output_to_log_file('detailed','\n          File does not have 4 byte headers/trailers\n')
   fmt8byteheader=fmtprefix+'q12cq'
   self.output_to_log_file('detailed','          Trying 8 byte header/trailers:\n')
   self.output_to_log_file('detailed','          Format: '+str(fmt8byteheader)+'\n')
   self.output_to_log_file('detailed','          Format size in bytes: '+str(struct.calcsize(fmt8byteheader))+'\n')
   self.output_to_log_file('detailed','          First 28 elements of file:
   '+str(struct.unpack(fmt8byteheader,self.fileintest[0:28]))+'\n')
   output_text='\n          File has 8 byte headers/trailers\n'
   fmtprefix=fmtprefix+'q'
   fmtsuffix='q'
   last_position=28
   filetype='binary'
   self.binaryfilename=self.filenamein

 self.output_to_log_file('detailed',output_text)
 if(filetype=='binary'):
  self.output_to_log_file('detailed','          Length of headers: '+str(struct.calcsize(fmtprefix))+'\n')
  self.output_to_log_file('detailed','          Length of trailers: '+str(struct.calcsize(fmtsuffix))+'\n\n')
```

```
             self.output_to_log_file('detailed','File type: binary\n\n')

           if(filetype!='none'):return(filetype)
            else:
             output_line='Error:   Could not determine file type ascii/binary!'
              self.output_to_log_file('exit',output_line)

         return

##############################################################################################################

class createabaqusodb:

      def __init__(self):

          self.command_line_arguments={}
           self.command_line_arguments['command_line']='>abaqus cae noGUI="C:\\WorkDir\\mcnp_results_reader\\mcnp_to_abaqus.pyc"
          -- path="C:\WorkDir\sample_mcnp_output_files"
          -- filenamein=name  -- asciifilenameout=name -- logfilename=name -- etafile=filename/[none]
          -- etafileformat=energy/[power] -- timeconversion=value/[1.0](default-shakes)
          -- lengthconversion=value/[1.0](default-cm) -- massconversion=value/[1.0](default-g)
          -- fluxconversion=value/[1.0](default-particles/cm^2) -- energydensityconversion=value/[1.0](default-MeV/g)
          -- fissiondensityconversion=value/[1.0](default-MeV/g) -- initialtimebinnegligible=yes/[no] -- initialtime=time/[none]
          -- calculatepowerdensities=yes/[no] -- calculatecumulativeenergydensities=yes/[no]  -- movepowerdensitiestonodes=yes/[no]
          -- particlefornt11=1/2/[TOTAL_1_2] -- energybinfornt11=1/2/3/.../[TOTAL] -- input_file_percentage_inc=value/[10]
          -- detailedoutput=yes/[no]'
          self.command_line_processing()
          self.change_working_directory()

self.etafunctions=self.readetafunctionsfile(path=self.command_line_arguments['path'],etafilename=self.command_line_arguments['etafile']
)

self.input_info=readinputfile(path=self.command_line_arguments['path'],file_name_in=self.command_line_arguments['filenamein'],binary_fi
le_name_in=self.command_line_arguments['binaryfilenamein'],ascii_file_name_in=self.command_line_arguments['asciifilenamein'],ascii_file
_name_out=self.command_line_arguments['asciifilenameout'],odb_file_name=self.command_line_arguments['odbfilename'],log_file_name=self.c
ommand_line_arguments['logfilename'],detailedoutput=self.command_line_arguments['detailedoutput'],input_file_percentage_inc=self.comman
d_line_arguments['input_file_percentage_inc'])
          self.input_file=self.input_info.readfile(self.command_line_arguments)
          self.input_data={}
          self.output_line_keywords={}
          self.time_track={}
```

```
    self.execution_order=0
    return

def print_command_line(self,location):

    output_line='\n\nCommand Line:\n\n'
    if(location=='rpy'):print output_line
    elif(location=='log'):self.input_info.output_to_log_file('pass',output_line)

    output_line=self.command_line_arguments['command_line']
    if(location=='rpy'):print output_line
    elif(location=='log'):self.input_info.output_to_log_file('pass',output_line)

    output_line='\n\nYour Command Line Options:\n\n'
    if(location=='rpy'):print output_line
    elif(location=='log'):self.input_info.output_to_log_file('pass',output_line)

    for key in self.command_line_arguments.keys():
        if(key!='command_line'):
          output_line='     '+str(key)+'='+str(self.command_line_arguments[key])
          if(location=='rpy'):print output_line
          elif(location=='log'):self.input_info.output_to_log_file('pass',output_line+'\n')
    if(location=='rpy'):print ' '
    elif(location=='log'):self.input_info.output_to_log_file('pass','  \n')
    return

def command_line_processing(self):

    temporary_list=[]
    for arg in sys.argv:
        if(string.count(arg,'=')>=1):
          temporary_list=string.split(arg,'=')
          self.command_line_arguments[temporary_list[0]]=temporary_list[1]

    if(self.command_line_arguments.has_key('help')):
      print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
      print '\n\nUse this command line: ',self.command_line_arguments['command_line']
      print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
      sys.exit()

    if(not self.command_line_arguments.has_key('path')):
```

```
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  print '\n\n\nNo path was given on the command line! Default path is '+str(os.getcwd())+'!\n'
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  self.command_line_arguments['path']=os.getcwd()

if((not self.command_line_arguments.has_key('filenamein'))):
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  print '\n\n\nNo input file name was given on the command line!\n\n\n'
  print '\n  filenamein=None'
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  sys.exit()

if((not self.command_line_arguments.has_key('asciifilenameout'))):
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  print '\n\n\nNo ascii output file will be written.\n\n\n'
  print '\n  asciifilenameout=None'
  print '\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'
  self.command_line_arguments['asciifilenameout']='none'

if((not self.command_line_arguments.has_key('initialtimebinnegligible'))):
  self.command_line_arguments['initialtimebinnegligible']='no'

if((not self.command_line_arguments.has_key('initialtime'))):
  self.command_line_arguments['initialtime']='none'

if((not self.command_line_arguments.has_key('calculatepowerdensities'))):
  self.command_line_arguments['calculatepowerdensities']='no'

if((not self.command_line_arguments.has_key('calculatecumulativeenergydensities'))):
  self.command_line_arguments['calculatecumulativeenergydensities']='no'

if((not self.command_line_arguments.has_key('movepowerdensitiestonodes'))):
  self.command_line_arguments['movepowerdensitiestonodes']='no'

if((not self.command_line_arguments.has_key('asciifilenamein'))):
  self.command_line_arguments['asciifilenamein']=self.command_line_arguments['filenamein']
if((not self.command_line_arguments.has_key('binaryfilenamein'))):
  self.command_line_arguments['binaryfilenamein']=self.command_line_arguments['filenamein']

if((not self.command_line_arguments.has_key('logfilename'))):
  self.command_line_arguments['logfilename']='none'
```

```
if(string.count(self.command_line_arguments['path'],'\\')>=1):
  if(self.command_line_arguments['path'][-1]!='\\'):
    path=self.command_line_arguments['path']
    self.command_line_arguments['path']=path+'\\'
elif(string.count(self.command_line_arguments['path'],'/')>=1):
   if(self.command_line_arguments['path'][-1]!='/'):
    path=self.command_line_arguments['path']
    self.command_line_arguments['path']=path+'/'

if((not self.command_line_arguments.has_key('detailedoutput'))):
   self.command_line_arguments['detailedoutput']='no'

if((not self.command_line_arguments.has_key('particlefornt11'))):
   self.command_line_arguments['particlefornt11']='TOTAL_1_2'

if((not self.command_line_arguments.has_key('energybinfornt11'))):
   self.command_line_arguments['energybinfornt11']='TOTAL'

if((not self.command_line_arguments.has_key('input_file_percentage_inc'))):
   self.command_line_arguments['input_file_percentage_inc']=10.0
if((not self.command_line_arguments.has_key('lengthconversion'))):
   self.command_line_arguments['lengthconversion']=1.0
else:
  self.command_line_arguments['lengthconversion']=float(self.command_line_arguments['lengthconversion'])

if((not self.command_line_arguments.has_key('timeconversion'))):
   self.command_line_arguments['timeconversion']=1.0
else:
  self.command_line_arguments['timeconversion']=float(self.command_line_arguments['timeconversion'])

if((not self.command_line_arguments.has_key('massconversion'))):
   self.command_line_arguments['massconversion']=1.0
else:
  self.command_line_arguments['massconversion']=float(self.command_line_arguments['massconversion'])

if((not self.command_line_arguments.has_key('fluxconversion'))):
   self.command_line_arguments['fluxconversion']=1.0
else:
  self.command_line_arguments['fluxconversion']=float(self.command_line_arguments['fluxconversion'])
```

```
        if((not self.command_line_arguments.has_key('energydensityconversion'))):
          self.command_line_arguments['energydensityconversion']=1.0
        else:
          self.command_line_arguments['energydensityconversion']=float(self.command_line_arguments['energydensityconversion'])

        if((not self.command_line_arguments.has_key('fissiondensityconversion'))):
          self.command_line_arguments['fissiondensityconversion']=1.0
        else:
          self.command_line_arguments['fissiondensityconversion']=float(self.command_line_arguments['fissiondensityconversion'])

        if((not self.command_line_arguments.has_key('etafile'))):
          self.command_line_arguments['etafile']='none'

        if((not self.command_line_arguments.has_key('etafileformat'))):
          self.command_line_arguments['etafileformat']='power'


        if(self.command_line_arguments['asciifilenamein']!='none'):
        self.command_line_arguments['odbfilename']=self.command_line_arguments['asciifilenamein']+'_abaqus'
        if(self.command_line_arguments['binaryfilenamein']!='none'):
        self.command_line_arguments['odbfilename']=self.command_line_arguments['binaryfilenamein']+'_abaqus'
        self.print_command_line('rpy')
        return

    def run_all_macros(self):
        self.print_command_line('log')
        self.parse_input_file()
        self.create_odb()
        self.create_parts()
        self.add_nodes_to_parts()
        self.add_elements_to_parts()
        self.instance_parts()
        self.create_steps()
        self.create_frames_in_steps()
        self.create_fields_in_frames()
        self.add_data_to_fields()
        self.save_odb()
        self.close_odb()
        self.time_analysis()
        return
```

```
def change_working_directory(self):

    os.chdir(self.command_line_arguments['path'])
    return

def conditionetafile(self):
    # This method converst etafile into energy format if it was in power format
    if(self.command_line_arguments['etafileformat']=='power'):
      # This converts the eta file to units of energy if power was given
       data_line_count=0
      last_line_time_hold=0
      last_line_entries_hold=[]
      self.conditionedetafile=[]
       for line in self.etafile:
           if(line.strip()[0]=='#'):self.conditionedetafile.append(line)
          elif(line.strip().split()[0]=='time'):self.conditionedetafile.append(line)
          else:
            data_line_count=data_line_count+1
            column_number=0
            last_line_time=copy.deepcopy(last_line_time_hold)
            last_line_entries=copy.deepcopy(last_line_entries_hold)
            new_line=line.strip().split()[0]
            time=float(line.strip().split()[0])
            last_line_time_hold=copy.deepcopy(time)
            for entry in line.strip().split()[1:]:
                column_number=column_number+1
                last_line_entries_hold.append(entry)
                 if(data_line_count==1):new_line=new_line+str('  0.0000')
                 else:new_line=new_line+'     '+str((float(time)-float(last_line_time))*
                (float(entry)+float(last_line_entries[column_number-1]))/2.0)
             self.conditionedetafile.append(new_line)
    elif(self.command_line_arguments['etafileformat']=='energy'):
      # This script was originally written assuming values in the eta file were in units of energy
      self.conditionedetafile=copy.deepcopy(self.etafile)
   else:
       output_line='Error:  Eta file format ['+str(self.command_line_arguments['etafileformat'])+']
      not supported!\n        Should be one of [power/energy]'
       self.input_info.output_to_log_file('exit',output_line)
    return

def readetafunctionsfile(self,path='none',etafilename='none'):
```

```
          if(path!='none' and etafilename!='none'):
            input_eta_file=open(path+etafilename,'r')
            self.etafile=[]
            for line in input_eta_file:
                self.etafile.append(line)
            self.conditionetafile()
            eta_dict={}
            for line in self.conditionedetafile:
                if(line.strip()[0]!='#'):
                  if(string.lower(line.strip().split()[0])=='time'):
                    column_instance_map={}
                    column_number=-1
                    for entry in line.strip().split():
                        column_number=column_number+1
                        if(string.lower(entry)!='time'):
                          eta_dict[entry]=[]
                          column_instance_map[column_number]=entry
                  else:
                    column_number=0
                    time=self.unit_conversion(float(line.strip().split()[0]),'timeconversion')
                    for entry in line.strip().split()[1:]:
                        column_number=column_number+1
                        eta_dict[column_instance_map[column_number]].append([float(time),float(entry)])
            if(not eta_dict.has_key('all')):eta_dict['all']=eta_dict[column_instance_map[1]]
            return_value=eta_dict
          else:
            return_value='none'
          return(return_value)

      def parse_input_file(self):

          self.execution_order=self.execution_order+1
          output_line='     Started parse_input_file: Date: %s Time: %s\n' % (time.strftime("%m/%d/
%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
          self.input_info.output_to_log_file('pass',output_line)

          start_time=time.clock()
```

```
self.input_data={'materials':{},'instances':[],'nodes':{'individual_nodes':[]},'elements':{'individual_elements':[]},'output':{'mcnp
data output elements':[]}}
          self.output_line_keywords={
                                    'EEOUT:': 'none',
                                   'VERSION':'none',
                                    'NUMBER OF NODES':'none',
                                   'NUMBER OF MATERIALS':'none',
                                   'NUMBER OF INSTANCES':'none',
                                   'NUMBER OF 1st TETS':'none',
                                   'NUMBER OF 1st PENTS':'none',
                                   'NUMBER OF 1st HEXS':'none',
                                   'NUMBER OF 2nd TETS':'none',
                                   'NUMBER OF 2nd PENTS':'none',
                                   'NUMBER OF 2nd HEXS':'none',
                                   'NUMBER OF COMPOSITS':'none',
                                   'NUMBER OF HISTORIES':'none',
                                   'NUMBER OF REG EDITS':'none',
                                   'NUMBER OF COM EDITS':'none',
                                   'LENGTH CONVERSION':'none',
                                   'EDIT DESCRIPTION':'none',
                                   'COMPOSITE EDIT LIMITS':'none',
                                    'MATERIALS':'none',
                                    'INSTANCE CUMMULATIVE ELEMENT TOTALS':'none',
                                   'INSTANCE ELEMENT NAMES':'none',
                                    'INSTANCE ELEMENT TYPE TOTALS':'none',
                                   'NODES X':'none',
                                   'NODES Y':'none',
                                   'NODES Z':'none',
                                   'ELEMENT TYPE':'none',
                                   'ELEMENT MATERIAL':'none',
                                   'CONNECTIVITY DATA 1ST ORDER TETS':'none',
                                   'CONNECTIVITY DATA 1ST ORDER PENTS':'none',
                                   'CONNECTIVITY DATA 1ST ORDER HEXS':'none',
                                   'CONNECTIVITY DATA 2ND ORDER TETS':'none',
                                   'CONNECTIVITY DATA 2ND ORDER PENTS':'none',
                                   'CONNECTIVITY DATA 2ND ORDER HEXS':'none',
                                   'NEAREST NEIGHBOR DATA LIST FLAG':'none',
                                   'NEAREST NEIGHBOR DATA 1ST ORDER TETS':'none',
                                   'NEAREST NEIGHBOR DATA 1ST ORDER PENTS':'none',
                                   'NEAREST NEIGHBOR DATA 1ST ORDER HEXS':'none',
                         'NEAREST NEIGHBOR DATA 2ND ORDER TETS':'none',
                         'NEAREST NEIGHBOR DATA 2ND ORDER PENTS':'none',
                         'NEAREST NEIGHBOR DATA 2ND ORDER HEXS':'none',
                         'DATA OUTPUT PARTICLE':[],
                         'CENTROIDS X (cm)':'none',
                         'CENTROIDS Y (cm)':'none',
                         'CENTROIDS Z (cm)':'none',
                         'VOLUMES (cm^3)':'none',
                         'DENSITY (gm/cm^3)':'none',
                         }

nearest_neighbor_count=0
data_output_count=0
data_output_count_2=-1
data_sets_count=0
control_line='NONE'
control_line_change='no'
line_count=-1
number_of_lines_between_output=1000
line_count_for_output=0

location=self.input_info.logfile.tell()

for full_line in self.input_file:
    line=full_line.strip()
    line_count=line_count+1
    keyword_values=self.get_keyword_values(line,[';',':'])
    line_count_for_output=line_count_for_output+1

    #             process keyword lines
    if(keyword_values.has_key('EEOUT')):
      if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='EEOUT'
self.output_line_keywords['EEOUT']=line_count
    if(keyword_values.has_key('VERSION')):
      if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='VERSION'
self.output_line_keywords['VERSION']=line_count
        self.input_data['eeout_version']=line
    if(keyword_values.has_key('NUMBER OF NODES')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF NODES'
self.output_line_keywords['NUMBER OF NODES']=line_count
        self.input_data['nodes']['number_of_nodes']=int(keyword_values['NUMBER OF NODES'])
    if(keyword_values.has_key('NUMBER OF MATERIALS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF MATERIALS'
self.output_line_keywords['NUMBER OF MATERIALS']=line_count
    if(keyword_values.has_key('NUMBER OF INSTANCES')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF INSTANCES'
self.output_line_keywords['NUMBER OF INSTANCES']=line_count
    if(keyword_values.has_key('NUMBER OF 1st TETS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF 1st TETS'
self.output_line_keywords['NUMBER OF 1st TETS']=line_count
        self.input_data['elements']['number_1st_tets']=int(keyword_values['NUMBER OF 1st TETS'])
         self.input_data['elements']['number_of_nodes_in_1st_tets']=4
    if(keyword_values.has_key('NUMBER OF 1st PENTS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF 1st PENTS'
self.output_line_keywords['NUMBER OF 1st PENTS']=line_count
        self.input_data['elements']['number_1st_pents']=int(keyword_values['NUMBER OF 1st PENTS'])
         self.input_data['elements']['number_of_nodes_in_1st_pents']=6
    if(keyword_values.has_key('NUMBER OF 1st HEXS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF 1st HEXS'
self.output_line_keywords['NUMBER OF 1st HEXS']=line_count
        self.input_data['elements']['number_1st_hexs']=int(keyword_values['NUMBER OF 1st HEXS'])
         self.input_data['elements']['number_of_nodes_in_1st_hexs']=8
    if(keyword_values.has_key('NUMBER OF 2nd TETS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
control_line='NUMBER OF 2nd TETS'
```

```
self.output_line_keywords['NUMBER OF 2nd TETS']=line_count
       self.input_data['elements']['number_2nd_tets']=int(keyword_values['NUMBER OF 2nd TETS'])
        self.input_data['elements']['number_of_nodes_in_2nd_tets']=10
    if(keyword_values.has_key('NUMBER OF 2nd PENTS')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='NUMBER OF 2nd PENTS'
self.output_line_keywords['NUMBER OF 2nd PENTS']=line_count
       self.input_data['elements']['number_2nd_pents']=int(keyword_values['NUMBER OF 2nd PENTS'])
        self.input_data['elements']['number_of_nodes_in_2nd_pents']=15
    if(keyword_values.has_key('NUMBER OF 2nd HEXS')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='NUMBER OF 2nd HEXS'
self.output_line_keywords['NUMBER OF 2nd HEXS']=line_count
       self.input_data['elements']['number_2nd_hexs']=int(keyword_values['NUMBER OF 2nd HEXS'])
        self.input_data['elements']['number_of_nodes_in_2nd_hexs']=20
    if(keyword_values.has_key('NUMBER OF COMPOSITS')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='NUMBER OF COMPOSITS'
self.output_line_keywords['NUMBER OF COMPOSITS']=line_count
    if(keyword_values.has_key('NUMBER OF HISTORIES')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='NUMBER OF HISTORIES'
self.output_line_keywords['NUMBER OF HISTORIES']=line_count
    if(keyword_values.has_key('LENGTH CONVERSION')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='LENGTH CONVERSION'
self.output_line_keywords['LENGTH CONVERSION']=line_count
    if(keyword_values.has_key('EDIT DESCRIPTION')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='EDIT DESCRIPTION'
self.output_line_keywords['EDIT DESCRIPTION']=line_count
    if(keyword_values.has_key('COMPOSITE EDIT LIMITS')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
control_line='COMPOSITE EDIT LIMITS'
```

```
self.output_line_keywords['COMPOSITE EDIT LIMITS']=line_count
    if(keyword_values.has_key('MATERIALS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
        material_count=0
control_line='MATERIALS'
self.output_line_keywords['MATERIALS']=line_count
    if(keyword_values.has_key('INSTANCE ELEMENT BOUNDS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
        instance_count=0
control_line='INSTANCE CUMMULATIVE ELEMENT TOTALS'
self.output_line_keywords['INSTANCE CUMMULATIVE ELEMENT TOTALS']=line_count
    if(keyword_values.has_key('INSTANCE ELEMENT NAMES')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
        instance_count=0
control_line='INSTANCE ELEMENT NAMES'
self.output_line_keywords['INSTANCE ELEMENT NAMES']=line_count
    if(keyword_values.has_key('INSTANCE ELEMENT TYPE TOTALS')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
        instance_count=0
control_line='INSTANCE ELEMENT TYPE TOTALS'
self.output_line_keywords['INSTANCE ELEMENT TYPE TOTALS']=line_count
    if(keyword_values.has_key('NODES X (cm)')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
         node_count=0
control_line='NODES X (cm)'
self.output_line_keywords['NODES X (cm)']=line_count
    if(keyword_values.has_key('NODES Y (cm)')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
         node_count=0
control_line='NODES Y (cm)'
self.output_line_keywords['NODES Y (cm)']=line_count
    if(keyword_values.has_key('NODES Z (cm)')):
      if(len(keyword_values.keys())==1):
        control_line_change='yes'
         node_count=0
```

```
control_line='NODES Z (cm)'
self.output_line_keywords['NODES Z (cm)']=line_count
    if(keyword_values.has_key('ELEMENT TYPE')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
        element_count=0
control_line='ELEMENT TYPE'
self.output_line_keywords['ELEMENT TYPE']=line_count
    if(keyword_values.has_key('ELEMENT MATERIAL')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
        element_count=0
control_line='ELEMENT MATERIAL'
self.output_line_keywords['ELEMENT MATERIAL']=line_count

    if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER TETS NODE ORDERED')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
       element_begin=0
       element_count=0
        node_count_connectivity=0
control_line='CONNECTIVITY DATA 1ST ORDER TETS NODE ORDERED'
self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER TETS NODE ORDERED']=line_count
    if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
       element_begin=0
       element_count=0
        node_count_connectivity=0
control_line='CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED'
self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED']=line_count

    if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER PENTS NODE ORDERED')):
     if(len(keyword_values.keys())==1):
       control_line_change='yes'
       element_begin=self.input_data['elements']['number_1st_tets']
       element_count=0
        node_count_connectivity=0
control_line='CONNECTIVITY DATA 1ST ORDER PENTS NODE ORDERED'
self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER PENTS NODE ORDERED']=line_count
    if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER PENTS ELEMENT ORDERED')):
```

```
                if(len(keyword_values.keys())==1):
                  control_line_change='yes'
                  element_begin=self.input_data['elements']['number_1st_tets']
                  element_count=0
                   node_count_connectivity=0
          control_line='CONNECTIVITY DATA 1ST ORDER PENTS ELEMENT ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER PENTS ELEMENT ORDERED']=line_count

              if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER HEXS NODE ORDERED')):
                if(len(keyword_values.keys())==1):
                  control_line_change='yes'
                  element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']
                  element_count=0
                   node_count_connectivity=0
          control_line='CONNECTIVITY DATA 1ST ORDER HEXS NODE ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER HEXS NODE ORDERED']=line_count
              if(keyword_values.has_key('CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED')):
                if(len(keyword_values.keys())==1):
                  control_line_change='yes'
                  element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']
                  element_count=0
                   node_count_connectivity=0
          control_line='CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED']=line_count

              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER TETS NODE ORDERED')):
                if(len(keyword_values.keys())==1):
                  control_line_change='yes'

element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']
                  element_count=0
                   node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER TETS NODE ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER TETS NODE ORDERED']=line_count
              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER TETS ELEMENT ORDERED')):
                if(len(keyword_values.keys())==1):
                  control_line_change='yes'

element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']
```

```
                 element_count=0
                  node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER TETS ELEMENT ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER TETS ELEMENT ORDERED']=line_count

              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER PENTS NODE ORDERED')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'

element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']+self.input_data['elements']['number_2nd_tets']
                 element_count=0
                  node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER PENTS NODE ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER PENTS NODE ORDERED']=line_count
              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER PENTS ELEMENT ORDERED')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'

element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']+self.input_data['elements']['number_2nd_tets']
                 element_count=0
                  node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER PENTS ELEMENT ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER PENTS ELEMENT ORDERED']=line_count

              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER HEXS NODE ORDERED')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'

element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']+self.input_data['elements']['number_2nd_tets']+self.input_data['elements']['number_2nd_pents']
                 element_count=0
                  node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER HEXS NODE ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER HEXS NODE ORDERED']=line_count
              if(keyword_values.has_key('CONNECTIVITY DATA 2ND ORDER HEXS ELEMENT ORDERED')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
```

```
element_begin=self.input_data['elements']['number_1st_tets']+self.input_data['elements']['number_1st_pents']+self.input_data['elements'
]['number_1st_hexs']+self.input_data['elements']['number_2nd_tets']+self.input_data['elements']['number_2nd_pents']
                 element_count=0
                  node_count_connectivity=0
          control_line='CONNECTIVITY DATA 2ND ORDER HEXS ELEMENT ORDERED'
          self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER HEXS ELEMENT ORDERED']=line_count

              if(keyword_values.has_key('NEAREST NEIGHBOR DATA LIST FLAG')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA LIST FLAG'
          self.output_line_keywords['NEAREST NEIGHBOR DATA LIST FLAG']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 1ST ORDER TETS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA 1ST ORDER TETS'
          self.output_line_keywords['NEAREST NEIGHBOR DATA 1ST ORDER TETS']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 1ST ORDER PENTS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA 1ST ORDER PENTS'
          self.output_line_keywords['NEAREST NEIGHBOR DATA 1ST ORDER PENTS']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 1ST ORDER HEXS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA 1ST ORDER HEXS'
          self.output_line_keywords['NEAREST NEIGHBOR DATA 1ST ORDER HEXS']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 2ND ORDER TETS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA 2ND ORDER TETS'
          self.output_line_keywords['NEAREST NEIGHBOR DATA 2ND ORDER TETS']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 2ND ORDER PENTS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
          control_line='NEAREST NEIGHBOR DATA 2ND ORDER PENTS'
          self.output_line_keywords['NEAREST NEIGHBOR DATA 2ND ORDER PENTS']=line_count
              if(keyword_values.has_key('NEAREST NEIGHBOR DATA 2ND ORDER HEXS')):
                if(len(keyword_values.keys())==1):
                 control_line_change='yes'
```

```
control_line='NEAREST NEIGHBOR DATA 2ND ORDER HEXS'
self.output_line_keywords['NEAREST NEIGHBOR DATA 2ND ORDER HEXS']=line_count

    if(keyword_values.has_key('DATA OUTPUT PARTICLE')):
     data_output_count_2=data_output_count_2+1
     data_sets_count_2=-1
     control_line_change='yes'
     control_line='DATA OUTPUT PARTICLE'
     data_output_type=keyword_values['TYPE']
     self.output_line_keywords['DATA OUTPUT PARTICLE'].append({'DATA OUTPUT LINE':line_count,'type':
     keyword_values['TYPE'],'particle':keyword_values['DATA OUTPUT PARTICLE'],'DATA SET LINES':[]})
     self.input_data['output']['mcnp data output elements'].append({'type':
     self.output_line_keywords['DATA OUTPUT PARTICLE'][data_output_count_2]['type'],'particle':
     keyword_values['DATA OUTPUT PARTICLE'],'time bins':{},'energy bins':{},'data sets':[]})

    if(keyword_values.has_key('DATA SETS RESULT TIME BIN') or keyword_values.has_key('DATA SETS RESULT SQR TIME BIN')
      or keyword_values.has_key('DATA SETS REL ERROR TIME BIN')  ):
     data_sets_count_2=data_sets_count_2+1
     control_line_change='yes'
      if(keyword_values.has_key('DATA SETS RESULT TIME BIN')):
       control_line='DATA SETS RESULT TIME BIN'
      elif(keyword_values.has_key('DATA SETS RESULT SQR TIME BIN')):
       control_line='DATA SETS RESULT SQR TIME BIN'
      elif(keyword_values.has_key('DATA SETS REL ERROR TIME BIN')):
       control_line='DATA SETS REL ERROR TIME BIN'
     self.output_line_keywords['DATA OUTPUT PARTICLE'][data_output_count_2]['DATA SET LINES'].append(line_count)
     self.input_data['output']['mcnp data output elements'][data_output_count_2]['data sets'].append
     ({'name':'none', 'results_type':control_line, 'time bin': keyword_values[control_line], 'time value':
     self.float_or_total(keyword_values['TIME VALUE']) ,'type':
     self.output_line_keywords['DATA OUTPUT PARTICLE'][data_output_count_2]['type'] ,'energy bin':
     keyword_values['ENERGY BIN'],'energy value':self.float_or_total(keyword_values['ENERGY VALUE']),'emult':
     self.float_or_total(keyword_values['EMULT']), 'data':[]})
     if(not self.input_data['output']['mcnp data output elements'][data_output_count_2]
        ['time bins'].has_key(keyword_values[control_line])):
       if(type(self.float_or_total(keyword_values['TIME VALUE']))!=type('')):
         time_value=self.unit_conversion(self.float_or_total(keyword_values['TIME VALUE']),'timeconversion')
       else:
         time_value=self.float_or_total(keyword_values['TIME VALUE'])
       self.input_data['output']['mcnp data output elements'][data_output_count_2]['time bins']
                      [keyword_values[control_line]]=time_value
     if(not self.input_data['output']['mcnp data output elements'][data_output_count_2]
              ['energy bins'].has_key(keyword_values['ENERGY BIN'])):
              self.input_data['output']['mcnp data output elements'][data_output_count_2]['energy bins']
              [keyword_values['ENERGY BIN']]=self.float_or_total(keyword_values['ENERGY VALUE'])

        if(keyword_values.has_key('CENTROIDS X (cm)')):
          if(len(keyword_values.keys())==1):
            control_line_change='yes'
            centroids_count=0
control_line='CENTROIDS X (cm)'
self.output_line_keywords['CENTROIDS X (cm)']=line_count
        if(keyword_values.has_key('CENTROIDS Y (cm)')):
          if(len(keyword_values.keys())==1):
            control_line_change='yes'
            centroids_count=0
control_line='CENTROIDS Y (cm)'
self.output_line_keywords['CENTROIDS Y (cm)']=line_count
        if(keyword_values.has_key('CENTROIDS Z (cm)')):
          if(len(keyword_values.keys())==1):
            control_line_change='yes'
            centroids_count=0
control_line='CENTROIDS Z (cm)'
self.output_line_keywords['CENTROIDS Z (cm)']=line_count
        if(keyword_values.has_key('DENSITY (gm/cm^3)')):
          if(len(keyword_values.keys())==1):
            control_line_change='yes'
            density_count=0
control_line='DENSITY (gm/cm^3)'
self.output_line_keywords['DENSITY (gm/cm^3)']=line_count
        if(keyword_values.has_key('VOLUMES (cm^3)')):
          if(len(keyword_values.keys())==1):
            control_line_change='yes'
            volume_count=0
control_line='VOLUMES (cm^3)'
self.output_line_keywords['VOLUMES (cm^3)']=line_count

        #             process data lines
        if(len(line.strip()) != 0 and control_line_change=='no'):
         if(control_line=='MATERIALS' and self.output_line_keywords['MATERIALS'] < line_count):
           material_count=material_count+1
           self.input_data['materials'][str(line.strip())]={'number':material_count}
         if(control_line=='INSTANCE ELEMENT NAMES' and self.output_line_keywords['INSTANCE ELEMENT NAMES'] < line_count):
```

```
  instance_count=instance_count+1
  self.input_data['instances'].append({'name':str(line.strip())})
if(control_line=='INSTANCE ELEMENT TYPE TOTALS' and self.output_line_keywords['INSTANCE ELEMENT TYPE TOTALS']
  < line_count):
  entries=line.split()
  instance_count=instance_count+1
  self.input_data['instances'][instance_count-1]['1st tets']={'first element':int(entries[0]),
  'last element':int(entries[1])}
  self.input_data['instances'][instance_count-1]['2nd tets']={'first element':int(entries[2]), '
  last element':int(entries[3])}
  self.input_data['instances'][instance_count-1]['1st pents']={'first element':int(entries[4]),
  'last element':int(entries[5])}
  self.input_data['instances'][instance_count-1]['2nd pents']={'first element':int(entries[6]),
  'last element':int(entries[7])}
  self.input_data['instances'][instance_count-1]['1st hexs']={'first element':int(entries[8]),
  'last element':int(entries[9])}
  self.input_data['instances'][instance_count-1]['2nd hexs']={'first element':int(entries[10]),
  'last element':int(entries[11])}
  self.input_data['instances'][instance_count-1]['elements']=[]
  self.input_data['instances'][instance_count-1]['nodes']=[]
if(control_line=='NODES X (cm)' and self.output_line_keywords['NODES X (cm)'] < line_count):
  for entry in line.split():
    node_count=node_count+1
    self.input_data['nodes']['individual_nodes'].append({'number':node_count,'coords':
    [self.unit_conversion(float(entry.strip()),'lengthconversion'),float(0.0),float(0.0)],'nearest elements':set()})
if(control_line=='NODES Y (cm)' and self.output_line_keywords['NODES Y (cm)'] < line_count):
  for entry in line.split():
    node_count=node_count+1
    self.input_data['nodes']['individual_nodes'][node_count-1]['coords'][1]=
    self.unit_conversion(float(entry.strip()),'lengthconversion')
if(control_line=='NODES Z (cm)' and self.output_line_keywords['NODES Z (cm)'] < line_count):
  for entry in line.split():
    node_count=node_count+1
    self.input_data['nodes']['individual_nodes'][node_count-1]['coords'][2]=
    self.unit_conversion(float(entry.strip()),'lengthconversion')
if(control_line=='ELEMENT TYPE' and self.output_line_keywords['ELEMENT TYPE'] < line_count):
  for entry in line.split():
    element_count=element_count+1
    self.input_data['elements']['individual_elements'].append({'number':element_count,'type':
    int(entry.strip()),'material number':'none','density':'none','volume':'none','centroid':
    {},'nodal connectivity':[],'nearest neighbors':{'vertex':[],'edge':[],'face':[]}})
```

```
                if(control_line=='ELEMENT MATERIAL' and self.output_line_keywords['ELEMENT MATERIAL'] < line_count):
                  for entry in line.split():
                    element_count=element_count+1
                    self.input_data['elements']['individual_elements'][element_count-1]['material number']=int(entry.strip())

                if(control_line=='CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED' and self.output_line_keywords['CONNECTIVITY DATA 1ST
ORDER TETS ELEMENT ORDERED'] < line_count):

element_count,node_count_connectivity=self.read_element_connectivities_element_ordered(control_line,line,element_begin,element_count,no
de_count_connectivity,self.input_data['elements']['number_of_nodes_in_1st_tets'])
                 if(control_line=='CONNECTIVITY DATA 1ST ORDER TETS NODE ORDERED' and self.output_line_keywords['CONNECTIVITY DATA 1ST
ORDER TETS NODE ORDERED'] < line_count):

element_count=self.read_element_connectivities_node_ordered(control_line,line,element_begin,element_count,self.input_data['elements']['
number_1st_tets'])

                 if(control_line=='CONNECTIVITY DATA 1ST ORDER PENTS ELEMENT ORDERED' and
                self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER PENTS ELEMENT ORDERED'] < line_count):
                  element_count,node_count_connectivity=
                  self.read_element_connectivities_element_ordered(control_line,line,element_begin,element_count,
                  node_count_connectivity,self.input_data['elements']['number_of_nodes_in_1st_pents'])
                if(control_line=='CONNECTIVITY DATA 1ST ORDER PENTS NODE ORDERED' and
                self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER PENTS NODE ORDERED'] < line_count):
                  element_count=self.read_element_connectivities_node_ordered(control_line,line,element_begin,
                  element_count,self.input_data['elements']['number_1st_pents'])

                 if(control_line=='CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED' and
                self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER HEXS ELEMENT ORDERED'] < line_count):
                  element_count,node_count_connectivity=self.read_element_connectivities_element_ordered
                  (control_line,line,element_begin,element_count,node_count_connectivity,self.input_data
                  ['elements']['number_of_nodes_in_1st_hexs'])
                 if(control_line=='CONNECTIVITY DATA 1ST ORDER HEXS NODE ORDERED' and
                self.output_line_keywords['CONNECTIVITY DATA 1ST ORDER HEXS NODE ORDERED'] < line_count):
                  element_count=self.read_element_connectivities_node_ordered(control_line,line,element_begin,
                  element_count,self.input_data['elements']['number_1st_hexs'])

                 if(control_line=='CONNECTIVITY DATA 2ND ORDER TETS ELEMENT ORDERED' and
                self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER TETS ELEMENT ORDERED'] < line_count):
                  element_count,node_count_connectivity=self.read_element_connectivities_element_ordered
                  (control_line,line,element_begin,element_count,node_count_connectivity,self.input_data
                  ['elements']['number_of_nodes_in_2nd_tets'])
```

```
 if(control_line=='CONNECTIVITY DATA 2ND ORDER TETS NODE ORDERED' and self.output_line_keywords
['CONNECTIVITY DATA 2ND ORDER TETS NODE ORDERED'] < line_count):
  element_count=self.read_element_connectivities_node_ordered(control_line,line,
  element_begin,element_count,self.input_data['elements']['number_2nd_tets'])

 if(control_line=='CONNECTIVITY DATA 2ND ORDER PENTS ELEMENT ORDERED' and
self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER PENTS ELEMENT ORDERED'] < line_count):
  element_count,node_count_connectivity=self.read_element_connectivities_element_ordered
  (control_line,line,element_begin,element_count,node_count_connectivity,self.input_data
  ['elements']['number_of_nodes_in_2nd_pents'])
 if(control_line=='CONNECTIVITY DATA 2ND ORDER PENTS NODE ORDERED' and self.output_line_keywords
['CONNECTIVITY DATA 2ND ORDER PENTS NODE ORDERED'] < line_count):
  element_count=self.read_element_connectivities_node_ordered(control_line,line,
  element_begin,element_count,self.input_data['elements']['number_2nd_pents'])

 if(control_line=='CONNECTIVITY DATA 2ND ORDER HEXS ELEMENT ORDERED' and
self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER HEXS ELEMENT ORDERED'] < line_count):
  element_count,node_count_connectivity=self.read_element_connectivities_element_ordered
  (control_line,line,element_begin,element_count,node_count_connectivity,self.input_data
  ['elements']['number_of_nodes_in_2nd_hexs'])
 if(control_line=='CONNECTIVITY DATA 2ND ORDER HEXS NODE ORDERED' and
self.output_line_keywords['CONNECTIVITY DATA 2ND ORDER HEXS NODE ORDERED'] < line_count):
  element_count=self.read_element_connectivities_node_ordered(control_line,line,
  element_begin,element_count,self.input_data['elements']['number_2nd_hexs'])

if(control_line=='NEAREST NEIGHBOR DATA LIST FLAG' and self.output_line_keywords
['NEAREST NEIGHBOR DATA LIST FLAG'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 1ST ORDER TETS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 1ST ORDER TETS'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 1ST ORDER PENTS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 1ST ORDER PENTS'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 1ST ORDER HEXS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 1ST ORDER HEXS'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 2ND ORDER TETS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 2ND ORDER TETS'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 2ND ORDER PENTS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 2ND ORDER PENTS'] < line_count):
   pass
if(control_line=='NEAREST NEIGHBOR DATA 2ND ORDER HEXS' and self.output_line_keywords
['NEAREST NEIGHBOR DATA 2ND ORDER HEXS'] < line_count):
   pass

if(control_line=='DATA OUTPUT PARTICLE' and self.output_line_keywords['DATA OUTPUT PARTICLE']
[data_output_count_2]['DATA OUTPUT LINE'] < line_count):
  data_output_count=data_output_count+1
if(control_line=='DATA SETS RESULT TIME BIN' or control_line=='DATA SETS RESULT SQR TIME BIN' or
control_line=='DATA SETS REL ERROR TIME BIN'):
   if(self.output_line_keywords['DATA OUTPUT PARTICLE'][data_output_count_2]['DATA SET LINES']
  [data_sets_count_2] < line_count):
    data_sets_count=data_sets_count+1
    for entry in line.split():
      if(data_output_type[0:4]=='FLUX'):
         if(control_line=='DATA SETS RESULT TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'fluxconversion')
         elif(control_line=='DATA SETS RESULT SQR TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'fluxconversion')
          new_entry=self.unit_conversion(float(new_entry),'fluxconversion')
         elif(control_line=='DATA SETS REL ERROR TIME BIN'):
           new_entry=float(entry.strip())
      elif(data_output_type[0:6]=='ENERGY'):
         if(control_line=='DATA SETS RESULT TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'energydensityconversion')
         elif(control_line=='DATA SETS RESULT SQR TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'energydensityconversion')
          new_entry=self.unit_conversion(float(new_entry),'energydensityconversion')
         elif(control_line=='DATA SETS REL ERROR TIME BIN'):
           new_entry=float(entry.strip())
      elif(data_output_type[0:7]=='FISSION'):
         if(control_line=='DATA SETS RESULT TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'fissionconversion')
         elif(control_line=='DATA SETS RESULT SQR TIME BIN'):
          new_entry=self.unit_conversion(float(entry.strip()),'fissionconversion')
          new_entry=self.unit_conversion(float(new_entry),'fissionconversion')
         elif(control_line=='DATA SETS REL ERROR TIME BIN'):
           new_entry=float(entry.strip())
      else:
        output_line='Error:  The data type is not one of the following [ENERGY, FLUX, FISSION]!'
```

```
          self.input_info.output_to_log_file('exit',output_line)
        self.input_data['output']['mcnp data output elements'][data_output_count_2]['data sets']
        [data_sets_count_2]['data'].append(new_entry)


  if(control_line=='CENTROIDS X (cm)' and self.output_line_keywords['CENTROIDS X (cm)'] < line_count):
    for entry in line.split():
      centroids_count=centroids_count+1
      self.input_data['elements']['individual_elements'][centroids_count-1]['centroid']=
      {'x_coord':self.unit_conversion(float(entry.strip()),'lengthconversion'),'y_coord':
      float(0.0),'z_coord':float(0.0)}
  if(control_line=='CENTROIDS Y (cm)' and self.output_line_keywords['CENTROIDS Y (cm)'] < line_count):
    for entry in line.split():
      centroids_count=centroids_count+1
      self.input_data['elements']['individual_elements'][centroids_count-1]['centroid']['y_coord']=
      self.unit_conversion(float(entry.strip()),'lengthconversion')
  if(control_line=='CENTROIDS Z (cm)' and self.output_line_keywords['CENTROIDS Y (cm)'] < line_count):
    for entry in line.split():
      centroids_count=centroids_count+1
      self.input_data['elements']['individual_elements'][centroids_count-1]['centroid']['z_coord']=
      self.unit_conversion(float(entry.strip()),'lengthconversion')
  if(control_line=='DENSITY (gm/cm^3)' and self.output_line_keywords['DENSITY (gm/cm^3)'] < line_count):
    for entry in line.split():
      density_count=density_count+1
      self.input_data['elements']['individual_elements'][density_count-1]['density']=
      self.unit_conversion(self.unit_conversion(self.unit_conversion(self.unit_conversion
      (float(entry.strip()),'inverse_lengthconversion'),'inverse_lengthconversion'),'inverse_lengthconversion'),
      'massconversion')
  if(control_line=='VOLUMES (cm^3)' and self.output_line_keywords['VOLUMES (cm^3)'] < line_count):
    for entry in line.split():
      volume_count=volume_count+1
      self.input_data['elements']['individual_elements'][volume_count-1]['volume']=
      self.unit_conversion(self.unit_conversion(self.unit_conversion(float(entry.strip()),
      'lengthconversion'),'lengthconversion'),'lengthconversion')

if(line_count_for_output == number_of_lines_between_output):
  output_line='  Finished line %s: Date: %s Time: %s \n' %
  (str(line_count+1),time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
  self.input_info.logfile.seek(location)
  self.input_info.output_to_log_file('detailed',output_line)
  line_count_for_output=0
```

```
    if(line_count+1 == len(self.input_file)):
     output_line='  Finished last line %s: Date: %s Time: %s \n' %
     (str(len(self.input_file)),time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
     self.input_info.logfile.seek(location)
     self.input_info.output_to_log_file('detailed',output_line)

    control_line_change='no'

 #  elements associated with a node new
output_line='  Started determining nearest elements for a node: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

for element in self.input_data['elements']['individual_elements']:
    for node in element['nodal connectivity']:
        if(not self.input_data['nodes']['individual_nodes'][int(node-1)].has_key('nearest elements')):
         self.input_data['nodes']['individual_nodes'][int(node-1)]['nearest elements']=set()
        self.input_data['nodes']['individual_nodes'][int(node-1)]['nearest elements'].add(int(element['number']))
output_line='  Finished determining nearest elements for a node:: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

 #  elements in an instance
output_line='  Started putting elements in the instances: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)
self.input_data['instances'][0]['first element']=int(1)

element_types=['1st tets','1st pents','1st hexs','2nd tets','2nd pents','2nd hexs']
for instance in self.input_data['instances']:
    element_list=[]
    for element_type in element_types:
      element_number=instance[element_type]['first element']
     if(element_number!=0):
        while(element_number<=instance[element_type]['last element']):
          element_list.append(element_number)
          element_list.sort()
          element_number=element_number+1
        for element in set(element_list):
           instance['elements'].append(element)
        instance['elements'].sort()
```

```
output_line='  Finished putting elements in the instances: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

 #  nodes in an instance
output_line='  Started putting nodes in the instances: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)
for instance in self.input_data['instances']:
    node_list=set()
    for element in instance['elements']:
        for node in self.input_data['elements']['individual_elements'][element-1]['nodal connectivity']:
            node_list.add(node)
       final_node_list=list(node_list)
    final_node_list.sort()
    instance['nodes']=final_node_list

output_line='  Finished putting nodes in the instances: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

 #  add instance element is associated with to element information
for instance_entry in self.input_data['instances']:
    for element_number in instance_entry['elements']:
        self.input_data['elements']['individual_elements'][int(element_number)-1]['instance']=instance_entry['name']

 #  create ghost data by multiplying MCNP data by eta values from eta file
 if(self.command_line_arguments['etafile']!='none'):
   # move output data to new holding dictionary
   output_hold_dict=copy.deepcopy(self.input_data['output']['mcnp data output elements'])
   output_new_dict=[]
   for data_output in output_hold_dict:
       if(string.lower(data_output['particle'])=='total_1_2' and string.lower(data_output['type'][0:6])=='energy'):
          output_new_dict.append(copy.deepcopy(data_output))
          output_new_dict[len(output_new_dict)-1]['data sets']=[]
          output_new_dict[len(output_new_dict)-1]['time bins']={}
          energy_bin_needed='1'
          if(len(data_output['data sets'])>1):
             data_set_type_count=0
             for data_set in data_output['data sets']:
                 if(string.lower(str(data_set['results_type']))==string.lower('DATA SETS RESULT TIME BIN')):
```

```
              data_set_type_count=data_set_type_count+1
           if(data_set_type_count>1):energy_bin_needed='total'
        for data_set in data_output['data sets']:
             if(string.lower(str(data_set['energy bin']))==energy_bin_needed):
               if(string.lower(str(data_set['results_type']))==string.lower('DATA SETS RESULT TIME BIN')):
                time_bin_count=0
                for eta_entry in self.etafunctions['all']:
                    time_bin_count=time_bin_count+1
                    output_new_dict[len(output_new_dict)-1]['data sets'].append(copy.deepcopy(data_set))
                    output_new_dict[len(output_new_dict)-1]['data sets'][len(output_new_dict[len(output_new_dict)-1]
                    ['data sets'])-1]['time bin']=str(time_bin_count)
                    output_new_dict[len(output_new_dict)-1]['data sets'][len(output_new_dict[len(output_new_dict)-1]
                    ['data sets'])-1]['time value']=float(eta_entry[0])
                    output_new_dict[len(output_new_dict)-1]['time bins'][str(time_bin_count)]=eta_entry[0]
                    new_data=[]
                    element_count=-1
                    for element_data in data_set['data']:
                        element_count=element_count+1
                        # determine which instance an element is in
                        if(element_count==0):instance='all'
                        else:instance=self.input_data['elements']['individual_elements'][element_count-1]['instance']
                        multiplyer=1.0
                        if(self.etafunctions.has_key(instance)):multiplyer=float(self.etafunctions[instance]
                        [time_bin_count-1][1])
                        else:multiplyer=float(self.etafunctions['all'][time_bin_count-1][1])
                          if(string.lower(str(data_set['results_type']))==string.lower('DATA SETS RESULT TIME BIN')):
                        new_data.append(float(element_data)*multiplyer)
                    output_new_dict[len(output_new_dict)-1]['data sets'][len(output_new_dict[len(output_new_dict)-1]
                    ['data sets'])-1]['data']=copy.deepcopy(new_data)
  self.input_data['output']['mcnp data output elements']=copy.deepcopy(output_new_dict)

#  determining power density and total energy as a function of time and moving power densities to nodes
output_line='  Started power density and total energy as a function of time: Date: %s Time: %s\n'
% (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

for data_output in self.input_data['output']['mcnp data output elements']:
    data_output['energy bins']=self.convert_dic_keys_from_text_to_integers_and_text(data_output['energy bins'])
    data_output['time bins']=self.convert_dic_keys_from_text_to_integers_and_text(data_output['time bins'])
    data_output['time bin increments']=self.determine_time_increments_for_time_bins(data_output['time bins'])
```

```
        if(self.command_line_arguments['calculatepowerdensities']=='yes' or
        self.command_line_arguments['movepowerdensitiestonodes']=='yes'):
          self.create_incremental_power_density()
           if(self.command_line_arguments['movepowerdensitiestonodes']=='yes'):self.move_incremental_power_density_to_nodes()
        if(self.command_line_arguments['calculatecumulativeenergydensities']=='yes'):self.create_cumulative_energy_densities()

        output_line='  Finished power density and total energy as a function of time: Date: %s Time: %s\n' %
        (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
        self.input_info.output_to_log_file('pass',output_line)

         #  final output before leaving parsing method
        output_line='  Number of Nodes   : '+str(len(self.input_data['nodes']['individual_nodes']))+'\n'
        output_line=output_line+' Number of Elements   :
         '+str(len(self.input_data['elements']['individual_elements']))+'\n'
        output_line=output_line+' Number of Instances   : '+str(len(self.input_data['instances']))+'\n'
        output_line=output_line+' Number of Materials   : '+str(len(self.input_data['materials']))+'\n'
        output_line=output_line+' Number of Data Outputs    :
         '+str(len(self.input_data['output']['mcnp data output elements']))+'\n'
        self.input_info.output_to_log_file('pass',output_line)

        end_time=time.clock()
        self.time_track['parse_input_file']={'start_time':start_time,'end_time':end_time,'delta_time':end_time-
start_time,'execution_order':self.execution_order}
        output_line='     Finished parse_input_file in %s\n\n' % self.sec_to_days(end_time-start_time)
        self.input_info.output_to_log_file('pass',output_line)
         return

    def move_incremental_power_density_to_nodes(self):

         self.input_data['output']['calculated data output power densities nodes']=
        copy.deepcopy(self.input_data['output']['calculated data output power densities elements'])
         for data_output in self.input_data['output']['calculated data output power densities nodes']:
            if(data_output['type']=='POWER_DENSITY'):
              for data_set in data_output['data sets']:
                  node_data=[]
                   for node in self.input_data['nodes']['individual_nodes']:
                      node_value=0
                     mass_total=0
                      for element_number in node['nearest elements']:
                          density=self.input_data['elements']['individual_elements'][element_number-1]['density']
                         volume=self.input_data['elements']['individual_elements'][element_number-1]['volume']
```

```
                    mass=volume*density
                    mass_total=mass_total+mass
                     node_value=node_value+mass*data_set['data'][element_number]
                node_data.append(node_value/mass_total)
              data_set['data']=node_data
    return

def determine_time_increments_for_time_bins(self,time_bins):

    time_increments={}
    time_bin_keys=time_bins.keys()
    time_bin_keys.sort()
    time_bin_key_last='none'
    for time_bin_key in time_bin_keys:
        if(time_bin_key==1):
          if(self.command_line_arguments['initialtime']!='none'):
            time_increments[time_bin_key]=time_bins[time_bin_key]-float(self.command_line_arguments['initialtime'])
         else:
            time_increments[time_bin_key]=1e39
        elif(time_bin_key!=1 and time_bin_key!='TOTAL'):
        time_increments[time_bin_key]=time_bins[time_bin_key]-time_bins[time_bin_key-1]
        elif(time_bin_key=='TOTAL' and time_bin_key_last!='none'):time_increments[time_bin_key]=time_bins[time_bin_key_last]
        time_bin_key_last=time_bin_key

    return(time_increments)

def create_incremental_power_density(self):

    self.input_data['output']['calculated data output power densities elements']=
    copy.deepcopy(self.input_data['output']['mcnp data output elements'])
    items_to_remove=[]
    data_output_count=-1
    for data_output in self.input_data['output']['calculated data output power densities elements']:
        data_output_count=data_output_count+1
        if(data_output['type'][0:6]=='ENERGY' and data_output['type'][-1]=='6'):
          data_set_count=-1
         for data_set in data_output['data sets']:
             data_set_count=data_set_count+1
             data_count=-1
             new_data=[]
             max_data_value=-1e39
```

```
                for data in data_set['data']:
                    data_count=data_count+1
                   if(float(data)>max_data_value and data_count>0):max_data_value=float(data)
                   if(data_set['time bin']=='1' and max_data_value!=0 and data_count>0):
                     if(self.command_line_arguments['initialtimebinnegligible']==
                     'no' and self.command_line_arguments['initialtime']=='none'):
                        output_line='There is energy in the first time bin of data_output='
                        +str(data_output_count+1)+' data_set='+str(data_set_count+1)+' and\n'
                        output_line=output_line+'command_line_argument initialtimebinnegligible
                        was set or defaulted to no and command_line_argument initialtime\n'
                        output_line=output_line+'was not given!  Unable to calculate a
                        power density for the first time bin.\n'
                          self.input_info.output_to_log_file('exit',output_line)
                     if(data_set['time bin']=='1' and max_data_value!=0 and data_count>0 and
                   self.command_line_arguments['initialtimebinnegligible']=='yes'):
                     new_data.append(float(0))
                   elif(data_count!=0):
                     density=self.input_data['elements']['individual_elements'][data_count-1]['density']
                     if(data_set['time bin']=='TOTAL'):key=data_set['time bin']
                     else:key=int(data_set['time bin'])
                     time_inc=data_output['time bin increments'][key]
                     new_data.append(data*density/time_inc)
                   else:
                     new_data.append(data)
                data_set['data']=new_data
                data_set['type']='POWER_DENSITY'
            data_output['type']='POWER_DENSITY'
          else:
            items_to_remove.append(data_output_count)

      for location in items_to_remove:
            item_to_remove=self.input_data['output']['calculated data output power densities elements'][location]
            self.input_data['output']['calculated data output power densities elements'].remove(item_to_remove)

      return

  def create_cumulative_energy_densities(self):

      self.input_data['output']['calculated data output energy cumulative elements']=copy.deepcopy
      (self.input_data['output']['mcnp data output elements'])
      items_to_remove=[]
```

```
        data_output_count=-1
        for data_output in self.input_data['output']['calculated data output energy cumulative elements']:
            data_output_count=data_output_count+1
           if(data_output['type'][0:6]=='ENERGY' and data_output['type'][-1]=='6'):
             energy_bin_keys=data_output['energy bins'].keys()
             energy_bin_keys.sort()
             for energy_bin_key in energy_bin_keys:
                 energy_bin_value=data_output['energy bins'][energy_bin_key]
                 data_sets_with_energy_bin=[]
                 data_set_count=-1
                 for data_set in data_output['data sets']:
                     data_set_count=data_set_count+1
                     if(data_set['energy bin']==str(energy_bin_key)):
                       data_sets_with_energy_bin.append(data_set_count)
                 time_value_data_set_location={}
                 for location in data_sets_with_energy_bin:
                     time_value_data_set_location[data_output['data sets'][location]['time value']]=
                    {'location':location,'time bin':data_output['data sets'][location]['time bin']}
                 ordered_times=time_value_data_set_location.keys()
                 ordered_times.sort()
                 for time in ordered_times:
                     data_set_location=time_value_data_set_location[time]['location']
                     time_bin=time_value_data_set_location[time]['time bin']
                    if(time_bin!='1' and time_bin!='TOTAL'and time_bin!='total'):
                      data_output['data sets'][data_set_location]['data']=self.add_two_lists(data_output['data sets']
                      [data_set_location]['data'],data_output['data sets'][last_data_set_location]['data'])
                    last_data_set_location=data_set_location
                 data_set['type']=data_output['type']+'_CUMULATIVE'
             data_output['type']=data_output['type']+'_CUMULATIVE'
           else:
              items_to_remove.append(data_output_count)

       for location in items_to_remove:
              item_to_remove=self.input_data['output']['calculated data output energy cumulative elements'][location]
             self.input_data['output']['calculated data output energy cumulative elements'].remove(item_to_remove)
        return

    def add_two_lists(self,list_1,list_2):

        list_sum=[]
    if(len(list_1)>len(list_2)):
      long_list=list_1
      short_list=list_2
    elif(len(list_1)<len(list_2)):
      long_list=list_2
      short_list=list_1
    else:
      long_list=list_1
      short_list=list_2

    location=-1
    for item in long_list:
        location=location+1
        if(location<len(short_list)):new_item=item+short_list[location]
        else:new_item=item
        list_sum.append(new_item)

    return(list_sum)

def convert_dic_keys_from_text_to_integers_and_text(self,dict):

    new_dict={}
    for item in dict.keys():
        try:
           new_item=string.atoi(item)
        except:
           new_item=item
        new_dict[new_item]=dict[item]

    return(new_dict)

def read_element_connectivities_element_ordered(self,control_line,line,element_begin,
element_count,node_count_connectivity,number_of_nodes_in_element):
    for entry in line.split():
        node_count_connectivity=node_count_connectivity+1
        self.input_data['elements']['individual_elements'][element_begin+element_count]
        ['nodal connectivity'].append(int(entry.strip()))
        if(node_count_connectivity==int(number_of_nodes_in_element)):
          element_count=element_count+1
          node_count_connectivity=0
    return(element_count,node_count_connectivity)
```

```
def read_element_connectivities_node_ordered(self,control_line,line,element_begin,element_count,total_number_elements):

    for entry in line.split():
        element_count=element_count+1
        self.input_data['elements']['individual_elements'][element_begin+element_count-1]['nodal connectivity'].
        append(int(entry.strip()))
        if(element_count==int(total_number_elements)):element_count=0
    return(element_count)

def create_odb(self):

    self.execution_order=self.execution_order+1
    output_line='     Started create_odb: Date: %s Time: %s\n' % (time.strftime("%m/%d/%y",time.localtime()),
    time.strftime("%H:%M:%S",time.localtime()))
    self.input_info.output_to_log_file('pass',output_line)

    start_time=time.clock()
    self.odb = Odb(name=self.command_line_arguments['odbfilename'],analysisTitle='none',description='none',
    path=self.command_line_arguments['path']+self.command_line_arguments['odbfilename']+'.odb' )

    end_time=time.clock()
    self.time_track['create_odb']={'start_time':start_time,'end_time':end_time,'delta_time':
    end_time-start_time,'execution_order':self.execution_order}
    output_line='     Finished create_odb in %s\n\n' % self.sec_to_days(end_time-start_time)
    self.input_info.output_to_log_file('pass',output_line)
    return

def create_parts(self):

    self.execution_order=self.execution_order+1
    output_line='     Started create_parts: Date: %s Time: %s\n' %
    (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
    self.input_info.output_to_log_file('pass',output_line)

    start_time=time.clock()
    parts=[]

    for instance in self.input_data['instances']:
        parts.append({'name':instance['name'],instance['name']:self.odb.Part(name=instance['name'],
        embeddedSpace=THREE_D,type=DEFORMABLE_BODY)})
        self.parts=parts

        end_time=time.clock()
        self.time_track['create_parts']={'start_time':start_time,'end_time':end_time,'delta_time':
        end_time-start_time,'execution_order':self.execution_order}
        output_line='     Finished create_parts in %s\n\n' % self.sec_to_days(end_time-start_time)
        self.input_info.output_to_log_file('pass',output_line)
        return

    def add_nodes_to_parts(self):

        self.execution_order=self.execution_order+1
        output_line='     Started add_nodes_to_parts: Date: %s Time: %s\n' % (time.strftime("%m/%d/%y",time.localtime()),
        time.strftime("%H:%M:%S",time.localtime()))
        self.input_info.output_to_log_file('pass',output_line)

        start_time=time.clock()
        instance_count=-1
        for part in self.parts:
            instance_count=instance_count+1
            node_data=[]
            node_count=0
            for node_number in self.input_data['instances'][instance_count]['nodes']:
                node_count=node_count+1
                self.input_data['nodes']['individual_nodes'][node_number-1]['local_node_number']=node_count
                node_data.append((node_count,self.input_data['nodes']['individual_nodes'][node_number-1]['coords'][0],
                self.input_data['nodes']['individual_nodes'][node_number-1]['coords'][1],self.input_data['nodes']
                ['individual_nodes'][node_number-1]['coords'][2]))
            if(part['name']==self.input_data['instances'][instance_count]['name']):
              part[part['name']].addNodes(nodeData=node_data, nodeSetName=part['name'])
            else:
              output_line='Part name and instance name are not the same!'
              self.input_info.output_to_log_file('exit',output_line)

        end_time=time.clock()
        self.time_track['add_nodes_to_parts']={'start_time':start_time,'end_time':end_time,'delta_time':
        end_time-start_time,'execution_order':self.execution_order}
        output_line='     Finished add_nodes_to_parts in %s\n\n' % self.sec_to_days(end_time-start_time)
        self.input_info.output_to_log_file('pass',output_line)

        return
    def add_elements_to_parts(self):

        self.execution_order=self.execution_order+1
        output_line='     Started add_elements_to_parts: Date: %s Time: %s\n' %
        (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
        self.input_info.output_to_log_file('pass',output_line)

        start_time=time.clock()
        element_types={'6':'C3D8','5':'C3D6','4':'C3D4','16':'C3D20','15':'C3D15','14':'C3D10'}
        instance_count=-1
        for part in self.parts:
            instance_count=instance_count+1
            element_data_C3D8=[]
            element_data_C3D6=[]
            element_data_C3D4=[]
            element_data_C3D10=[]
            element_data_C3D15=[]
            element_data_C3D20=[]
            element_count=-1
            element_count_C3D8=-1
            element_count_C3D6=-1
            element_count_C3D4=-1
            element_count_C3D10=-1
            element_count_C3D15=-1
            element_count_C3D20=-1
            for element_number in self.input_data['instances'][instance_count]['elements']:
        element_count=element_count+1
        element_type=element_types[str(self.input_data['elements']['individual_elements'][element_number-1]['type'])]
               self.input_data['elements']['individual_elements'][element_number-1]['local_element_number']=element_count+1
        if(element_type=='C3D8'):
          element_count_C3D8=element_count_C3D8+1
          element_data_C3D8.append([element_count+1])
          for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
             element_data_C3D8[element_count_C3D8].append(self.input_data['nodes']['individual_nodes'][node_number-
1]['local_node_number'])
        if(element_type=='C3D6'):
          element_count_C3D6=element_count_C3D6+1
          element_data_C3D6.append([element_count+1])
          for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
```

```
       element_data_C3D6[element_count_C3D6].append(self.input_data['nodes']['individual_nodes']
               [node_number-1]['local_node_number'])
  if(element_type=='C3D4'):
    element_count_C3D4=element_count_C3D4+1
    element_data_C3D4.append([element_count+1])
    for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
       element_data_C3D4[element_count_C3D4].append(self.input_data['nodes']['individual_nodes']
               [node_number-1]['local_node_number'])
  if(element_type=='C3D20'):
    element_count_C3D20=element_count_C3D20+1
    element_data_C3D20.append([element_count+1])
    for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
       element_data_C3D20[element_count_C3D20].append(self.input_data['nodes']['individual_nodes']
               [node_number-1]['local_node_number'])
  if(element_type=='C3D15'):
    element_count_C3D15=element_count_C3D15+1
    element_data_C3D15.append([element_count+1])
    for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
       element_data_C3D15[element_count_C3D15].append(self.input_data['nodes']['individual_nodes']
               [node_number-1]['local_node_number'])
  if(element_type=='C3D10'):
    element_count_C3D10=element_count_C3D10+1
    element_data_C3D10.append([element_count+1])
    for node_number in self.input_data['elements']['individual_elements'][element_number-1]['nodal connectivity']:
       element_data_C3D10[element_count_C3D10].append(self.input_data['nodes']['individual_nodes']
               [node_number-1]['local_node_number'])

      if(part['name']==self.input_data['instances'][instance_count]['name']):
if(len(element_data_C3D8)>0):
  part[part['name']].addElements(elementData=element_data_C3D8, type='C3D8', elementSetName=part['name']+'_C3D8')
if(len(element_data_C3D6)>0):
  part[part['name']].addElements(elementData=element_data_C3D6, type='C3D6', elementSetName=part['name']+'_C3D6')
if(len(element_data_C3D4)>0):
  part[part['name']].addElements(elementData=element_data_C3D4, type='C3D4', elementSetName=part['name']+'_C3D4')

        if(len(element_data_C3D20)>0):
  part[part['name']].addElements(elementData=element_data_C3D20, type='C3D20', elementSetName=part['name']+'_C3D20')
if(len(element_data_C3D15)>0):
  part[part['name']].addElements(elementData=element_data_C3D15, type='C3D15', elementSetName=part['name']+'_C3D15')
if(len(element_data_C3D10)>0):
  part[part['name']].addElements(elementData=element_data_C3D10, type='C3D10', elementSetName=part['name']+'_C3D10')
```

```
                else:
          print 'Part name and instance name are not the same!'
          sys.exit()

            end_time=time.clock()
            self.time_track['add_elements_to_parts']={'start_time':start_time,'end_time':end_time,'delta_time':end_time-
start_time,'execution_order':self.execution_order}
            output_line='     Finished add_elements_to_parts in %s\n\n' % self.sec_to_days(end_time-start_time)
            self.input_info.output_to_log_file('pass',output_line)
            return

        def instance_parts(self):

            self.execution_order=self.execution_order+1
            output_line='     Started instance_parts: Date: %s Time: %s\n' % (time.strftime("%m/%d/%y",time.localtime()),
            time.strftime("%H:%M:%S",time.localtime()))
            self.input_info.output_to_log_file('pass',output_line)

            start_time=time.clock()
            instances=[]


            max_length_part_name=0
            for part in self.parts:
                if(len(part['name'])>max_length_part_name):max_length_part_name=len(part['name'])

            outputfile=open(self.command_line_arguments['path']+'instance_to_assembly_set_info.dat','w')
            outputfile.write('Instance Name'.ljust(max_length_part_name)+'     Assembly Node Set Name
            Assembly Element Set Name\n\n')

            part_count=0
            for part in self.parts:
                part_count=part_count+1
                outputfile.write(part['name'].upper().ljust(max_length_part_name)+'     '+str('node_set-'+str(part_count)).
                ljust(22)+'     '+'element_set-'+str(part_count))
                instance_name=string.upper(part['name'])
                instances.append({'name':instance_name,instance_name:self.odb.rootAssembly.Instance(name=instance_name,
                object=part[part['name']])})
                self.odb.rootAssembly.ElementSet(name='element_set-'+str(part_count),elements=(self.odb.rootAssembly.
                instances[instance_name].elements[0:],))
                self.odb.rootAssembly.NodeSet(name='node_set-'+str(part_count),nodes=(self.odb.rootAssembly.
                instances[instance_name].nodes[0:],))
            self.instances=instances
            outputfile.close()

            end_time=time.clock()
            self.time_track['instance_parts']={'start_time':start_time,'end_time':end_time,'delta_time':end_time-
start_time,'execution_order':self.execution_order}
            output_line='     Finished instance_parts in %s\n\n' % self.sec_to_days(end_time-start_time)
            self.input_info.output_to_log_file('pass',output_line)
            return

        def create_steps(self):

            self.execution_order=self.execution_order+1
            output_line='     Started create_steps: Date: %s Time: %s\n' % (time.strftime("%m/%d/%y",time.localtime()),
            time.strftime("%H:%M:%S",time.localtime()))
            self.input_info.output_to_log_file('pass',output_line)

            start_time=time.clock()
            self.steps_frames_fields=[]
            time_list=[]
            for data in self.input_data['output']['mcnp data output elements']:
                time_values=data['time bins'].values()
                for time1 in time_values:
            if(time1!='TOTAL'):
              time_list.append(time1)
            time_list.sort()
            time_list_unique=[]
            time_hold=-1e10
            for entry in time_list:
                if(entry!=time_hold):
          time_list_unique.append(entry)
          time_hold=entry

            if(len(time_list_unique)==1):
              step_time_period=time_list_unique[0]
            else:
              step_time_period=time_list_unique[-1]-time_list_unique[0]

            step_name='Step-1'
```

```
#---Begin: ABAQUS Bug workaround-----------------------------------------------------------------------------------------
    output_line='  Warning: Workaround for an abaqus visualization bug.\n'
    output_line=output_line+
    '                  The Period in the pull down window under Results-> Active Steps/Frames does not accept a time =
    1e+39 it will only accept a time < 1e+39.\n'
    output_line=output_line+
    '                  The bug is a problem only when one tries to deactivate a frame so it will not be displayed
    in a movie.\n'
    output_line=output_line+
    '                  Because 1e+39 is the default value for long times in MCNP when the user does not specify an
    end time this work around will reduce\n'
    output_line=output_line+
    '                  the time output to the window from (1.0e+39) to (1.0e+38).\n'
    output_line=output_line+
    '                  All values in the output database (*.odb) are correct except in the pull down window.\n'
    output_line=output_line+
    '                  This does not effect any results or movies.\n'

    self.input_info.output_to_log_file('pass',output_line)

    if(step_time_period>=1e+39):
      step_time_period_2=(1.0e+38)
    else:
      step_time_period_2=step_time_period
#---End: ABAQUS Bug workaround-------------------------------------------------------------------------------------------


    self.steps_frames_fields.append({'name':step_name,'time period':step_time_period,step_name:self.odb.
    Step(name=step_name, description='Time Period = '+str(step_time_period), domain=TIME, timePeriod=step_time_period_2,
    totalTime=0.0),'frames':[]})
    end_time=time.clock()
    self.time_track['create_steps']={'start_time':start_time,'end_time':end_time,'delta_time':end_time-start_time,
    'execution_order':self.execution_order}
    output_line='     Finished create_steps in %s\n\n' % self.sec_to_days(end_time-start_time)
    self.input_info.output_to_log_file('pass',output_line)
    return

def create_frames_in_steps(self):

    self.execution_order=self.execution_order+1
    output_line='     Started create_frames_in_steps: Date: %s Time: %s\n' %
    (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
    self.input_info.output_to_log_file('pass',output_line)

    start_time=time.clock()
    for step in self.steps_frames_fields:

        times_temporary=[]
        times=[]
        for output in self.input_data['output']['mcnp data output elements']:
    for time_value in output['time bins'].values():
        if(time_value!='TOTAL'):
         times_temporary.append(time_value)
        times_temporary.sort()

        count=0
        for time1 in times_temporary:
    count=count+1
    if(count==1):
      times.append(time1)
    else:
      if(time1!=time_hold):
        times.append(time1)
    time_hold=time1

        time_inc=0
        for time1 in times:
    time_inc=time_inc+1
    frame_name='frame_'+str(time_inc-1)
    step['frames'].append({'name':frame_name, 'increment':time_inc, 'time':time1, frame_name:
           step[step['name']].Frame(incrementNumber=time_inc, frameValue=time1, description=
           '     Time = '+str(time1)),'fields':[]})

    end_time=time.clock()
    self.time_track['create_frames_in_steps']={'start_time':start_time,'end_time':end_time,'delta_time':
    end_time-start_time,'execution_order':self.execution_order}
    output_line='     Finished create_frames_in_steps in %s\n\n' % self.sec_to_days(end_time-start_time)
    self.input_info.output_to_log_file('pass',output_line)
    return

def create_fields_in_frames(self):
 import abaqusConstants

self.execution_order=self.execution_order+1
 output_line='     Started create_fields_in_frames: Date: %s Time: %s\n' %
(time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
self.input_info.output_to_log_file('pass',output_line)

 self.NT11_location={}
 start_time=time.clock()
 step_count=-1
 for step in self.steps_frames_fields:
     step_count=step_count+1
     frame_count=-1
     for frame in step['frames']:
 frame_count=frame_count+1
         field_name_dic={}
        output_type_count=-1
        for key1 in self.input_data['output'].keys():
            output_type=self.input_data['output'][key1]
            output_type_count=output_type_count+1
     output_count=-1
     for output in output_type:
         output_count=output_count+1
         for set in output['data sets']:
                     if(set['results_type'].count('SQR')!=0):results_type='SQR'
                     elif(set['results_type'].count('REL ERROR')):results_type='REL_ERR'
                     else:results_type='RESULT'
            if(set['time value']!='TOTAL'):
              time_value=set['time value']
            else:
              times=output['time bins'].values()
              times.sort()
              time_value=times[-2]
            if(time_value==frame['time']):
              if(set['time bin']!='TOTAL'):
                name='p_' + str(output['particle']).replace('TOTAL_','') +
                       '-' + output['type'] +'-Ebin_' + set['energy bin'] +
                          '-' + results_type + '-'+string.strip(string.split(key1)[-1])
              else:
                name='p_' + str(output['particle']).replace('TOTAL_','') + '-' + output['type'] +
                             '-Tbin_' + set['time bin'] +'-Ebin_' + set['energy bin'] +  '-' +
                                 results_type + '-'+string.strip(string.split(key1)[-1])
                              description_name=' p_' + str(output['particle']).replace('TOTAL_','') + '-' +
                             output['type'] +'-Tbin_' + set['time bin'] +'-Ebin_' + set['energy bin'] +  '-' +
                              results_type + '-'+string.strip(string.split(key1)[-1])
                              description_name=''
                      set['name']=name
                             if(self.command_line_arguments['etafile']=='none' or
                             (self.command_line_arguments['etafile']!='none' and output['type']!='ENERGY_6')):
                               frame['fields'].append({'name':name,name:frame[frame['name']].
                               FieldOutput(name=name,description=description_name, type=SCALAR),'number':output_count})

                               #  Start:  Adding power densities at nodes as temperature NT11 for interpolation onto new mesh
                               field_name_dic[name]=name

                             if((not field_name_dic.has_key('NT11')) and output['particle']==
                             self.command_line_arguments['particlefornt11'] and  output['type']==
                             'POWER_DENSITY' and set['energy bin']==self.command_line_arguments['energybinfornt11']
                             and string.strip(string.split(key1)[-1])=='nodes'):
                               name_2='NT11'
                               description_name_2='Power Densities are saved at nodes as NT11 for interpolation to new mesh
                               integration points'
                               frame['fields'].append({'name':name_2,name_2:frame[frame['name']].
                               FieldOutput(name=name_2,description=description_name_2, type=SCALAR),'number':output_count})
                               field_name_dic[name_2]=name_2
                               self.NT11_location[frame_count]=len(frame['fields'])-1
                               #  End:  Adding power densities at nodes as temperature NT11 for interpolation onto new mesh

    end_time=time.clock()
    self.time_track['create_fields_in_frames']={'start_time':start_time,'end_time':end_time,'delta_time':
    end_time-start_time,'execution_order':self.execution_order}
    output_line='     Finished create_fields_in_frames in %s\n\n' % self.sec_to_days(end_time-start_time)
    self.input_info.output_to_log_file('pass',output_line)
    return

def add_data_to_fields(self):

    self.execution_order=self.execution_order+1
    output_line='     Started add_data_to_fields: Date: %s Time: %s\n' %
    (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
    self.input_info.output_to_log_file('pass',output_line)
```

```
start_time=time.clock()
instance_count=-1
for instance in self.instances:
    instance_count=instance_count+1
    step_count=-1
    for step in self.steps_frames_fields:
step_count=step_count+1
frame_count=-1
for frame in step['frames']:
    frame_count=frame_count+1
    field_count=-1
    for field in frame['fields']:
       field_count=field_count+1
               output_type_count=-1
               for key1 in self.input_data['output'].keys():
                   output_type_count=output_type_count+1
                   output_type=self.input_data['output'][key1]
           output_data_count=-1
           for output_data in output_type:
               output_data_count=output_data_count+1
               set_count=-1
               set_number=-1
               for set in output_data['data sets']:
                   set_count=set_count+1
                   if(set['time value']!='TOTAL'):
                      set_time_value=set['time value']
                   else:
                      times_data=output_data['time bins'].values()
                      times_data.sort()
                      set_time_value=times_data[-2]
                   if((set_time_value==frame['time']) and (field['name']==set['name'])):
                     set_number=set_count
               if(set_number!=-1):
                         if(string.strip(string.split(field['name'],'-')[-1])=='elements'):
                           object_type='elements'
                           object_position=SymbolicConstant('CENTROID')
                    object_labels=[]
                    object_data=[]
                    object_count=0
                         if(string.strip(string.split(field['name'],'-')[-1])=='nodes'):
                           object_type='nodes'
```

```
                                    object_position=SymbolicConstant('NODAL')
                              object_labels=[]
                              object_data=[]
                              object_count=0
                                   max=-1e39
                            for object in self.input_data['instances'][instance_count][object_type]:
                                object_count=object_count+1
                                object_labels.append(self.input_data[object_type]['individual_'+
                                       object_type][object-1]['local_'+object_type[0:-1]+'_number'])
                                       shift=0
                                       if(string.strip(string.split(field['name'],'-')[-1])=='nodes'):shift=1
                                object_data.append([output_data['data sets'][set_number]['data'][object-shift]])
                                       if(output_data['data sets'][set_number]['data'][object-shift]>max):
                                       max=output_data['data sets'][set_number]['data'][object-shift]
                                   if(self.command_line_arguments['etafile']=='none' or
                                   (self.command_line_arguments['etafile']!='none' and field['name'].count('ENERGY_6')==0)):
                              field[field['name']].addData(position=object_position, instance=instance[instance['name']],
                                     labels=object_labels, data=object_data)
                                     #  Start:  Adding power densities at nodes as temperature NT11 for interpolation onto new mesh
                                   if(field['name']=='p_' + str(self.command_line_arguments
                                   ['particlefornt11']).replace('TOTAL_','') +'-POWER_DENSITY-Ebin_'+
                                   str(self.command_line_arguments['energybinfornt11'])+'-RESULT-nodes'):
                                       frame['fields'][self.NT11_location[frame_count]]['NT11'].addData(position=object_position,
                                     instance=instance[instance['name']],labels=object_labels, data=object_data)
                                     #  End:  Adding power densities at nodes as temperature NT11 for interpolation onto new mesh

    end_time=time.clock()
    self.time_track['add_data_to_fields']={'start_time':start_time,'end_time':end_time,'delta_time':
    end_time-start_time,'execution_order':self.execution_order}
    output_line='     Finished add_data_to_fields in %s\n\n' % self.sec_to_days(end_time-start_time)
    self.input_info.output_to_log_file('pass',output_line)
    return

def save_odb(self):

    self.execution_order=self.execution_order+1
    output_line='     Started save_odb: Date: %s Time: %s\n' %
    (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
    self.input_info.output_to_log_file('pass',output_line)

    start_time=time.clock()
```

```
        self.odb.save()

        end_time=time.clock()
        self.time_track['save_odb']={'start_time':start_time,'end_time':
        end_time,'delta_time':end_time-start_time,'execution_order':self.execution_order}
        output_line='     Finished save_odb in %s\n\n' % self.sec_to_days(end_time-start_time)
        self.input_info.output_to_log_file('pass',output_line)
        return

    def close_odb(self):

        self.execution_order=self.execution_order+1
        output_line='     Started close_odb: Date: %s Time: %s\n' %
        (time.strftime("%m/%d/%y",time.localtime()),time.strftime("%H:%M:%S",time.localtime()))
        self.input_info.output_to_log_file('pass',output_line)
        start_time=time.clock()
        self.odb.close()

        end_time=time.clock()
        self.time_track['close_odb']={'start_time':start_time,'end_time':end_time,'delta_time':
        end_time-start_time,'execution_order':self.execution_order}
        output_line='     Finished close_odb in %s\n\n' % self.sec_to_days(end_time-start_time)
        self.input_info.output_to_log_file('pass',output_line)
        return

    def time_analysis(self):

        self.execution_order=self.execution_order+1
        output_line='Finished Macros: Date: %s Time: %s\n' % (time.strftime("%m/%d/%y",time.localtime()),time.
        strftime("%H:%M:%S",time.localtime()))
        self.input_info.output_to_log_file('pass',output_line)

        total=0
        for key in self.time_track.keys():
            total=total+self.time_track[key]['delta_time']

        output_line='\nTiming Analysis:\n\nTime(dd:hh:mm:ss.s)   Percent(%) Macro\n'
        self.input_info.output_to_log_file('pass',output_line)

        time_hold_dic={}
        percent_time_total=0.0
    for key in self.time_track.keys():
        entry=self.time_track[key]
        total_time='%.4f' % entry['delta_time']
        percent_time='%.2f' % (entry['delta_time']/total*100)
        percent_time_total=percent_time_total+string.atof(percent_time)
        time_hold_dic[int(entry['execution_order'])]='%20s   %8s       %s\n' %
        (self.sec_to_days(entry['delta_time']),percent_time,key)

    execution_order=time_hold_dic.keys()
    execution_order.sort()
    for order in execution_order:
        output_line=str(time_hold_dic[order])
        self.input_info.output_to_log_file('pass',output_line)

    output_line='\n   Totals:\n%20s   %8s\n\n\n' % (self.sec_to_days(total),str(percent_time_total))
    self.input_info.output_to_log_file('pass',output_line)
    return

def sec_to_days(self,value):

    total_days=value/60.0/60.0/24.0
    int_days=int(total_days)
    part_days=total_days-int_days

    total_hours=part_days*24.0
    int_hours=int(total_hours)
    part_hours=total_hours-int_hours

    total_minutes=part_hours*60.0
    int_minutes=int(total_minutes)
    part_minutes=total_minutes-int_minutes


    total_seconds=part_minutes*60.0
    int_seconds=int(total_seconds)
    part_seconds=total_seconds-int_seconds

    line='%3s:%2s:%2s:%6s' % (str(int_days).zfill(3),
    str(int_hours).zfill(2),str(int_minutes).zfill(2),string.strip(str('%6.3f' % total_seconds)).zfill(6))
    return(line)
```

```
def get_keyword_values(self,line,sep):

    keyword_values={}
    line_split=line.split(sep[0])
   for entry in line_split:
       if(entry.count(sep[1])==1):
         entry_split=entry.split(sep[1])
         keyword_values[entry_split[0].strip()]=entry_split[1].strip()
       elif(entry.count(sep[1])>1):
         output_line='More than one keyword value seperator '+str(sep[1])+' in keyword line! \n'
        +str(line)+'\n Entry: '+str(entry)+'\n'
        self.input_info.output_to_log_file('exit',output_line)
       else:
         keyword_values[entry.strip()]=entry.strip()
    return(keyword_values)

def float_or_total(self,value):

    if(value=='TOTAL'):
      newvalue=value
    elif(value=='N/A'):
      newvalue='TOTAL'
    else:
      newvalue=float(value)
    return (newvalue)

def unit_conversion(self,value,conversion):

    if(type(value)==type(1) or type(value)==type(float(1.0))):
     if(conversion=='timeconversion'):
        newvalue=float(value)*self.command_line_arguments['timeconversion']
     elif(conversion=='inverse_timeconversion'):
        newvalue=float(value)/self.command_line_arguments['timeconversion']
     elif(conversion=='massconversion'):
        newvalue=float(value)*self.command_line_arguments['massconversion']
     elif(conversion=='inverse_massconversion'):
        newvalue=float(value)/self.command_line_arguments['massconversion']
     elif(conversion=='lengthconversion'):
        newvalue=float(value)*self.command_line_arguments['lengthconversion']
     elif(conversion=='inverse_lengthconversion'):
        newvalue=float(value)/self.command_line_arguments['lengthconversion']
```

```
            elif(conversion=='fluxconversion'):
              newvalue=float(value)*self.command_line_arguments['fluxconversion']
            elif(conversion=='inverse_fluxconversion'):
              newvalue=float(value)/self.command_line_arguments['fluxconversion']
            elif(conversion=='energydensityconversion'):
              newvalue=float(value)*self.command_line_arguments['energydensityconversion']
            elif(conversion=='inverse_energydensityconversion'):
              newvalue=float(value)/self.command_line_arguments['energydensityconversion']
            elif(conversion=='fissiondensityconversion'):
              newvalue=float(value)*self.command_line_arguments['fissiondensityconversion']
            elif(conversion=='inverse_fissiondensityconversion'):
              newvalue=float(value)/self.command_line_arguments['fissiondensityconversion']
            else:
              newvalue=value
          else:
            newvalue=value

          return(newvalue)

createabaqusodb().run_all_macros()
```

PERL Wrapper

```
#!/usr/bin/perl

########################################################################
#  PERL wrapper script for mcnp_to_abaqus python script.               #
#  Converts MCNP eeout file to Abaqus/CAE odb file for visualization. #
#                                                                      #
########################################################################

  use Cwd;
```

```
$dir = cwd();

print "Current directory is $dir\n";

$codeDir = "/opt/local/codes/abaqus/Commands";
$script1 = "mcnp_to_abaqus.py";

if ($#ARGV > -1)
{
  if (($ARGV[0] eq '-help') || ($ARGV[0] eq '--help') || ($ARGV[0] eq '-man'))
  {
    &man;
    exit;
  }

  $a = $codeDir . "/abaqus cae ";
  $b = 'noGUI="' . $codeDir . '/mcnp_to_abaqus.py"';
  $c = ' -- path="' . $dir . '" ';
  $d = " -- filenamein=$ARGV[0]";
  $e = " -- asciifilenameout=asciifileout.txt -- logfilename=output";

  $f = $a . $b . $c . $d . $e;
  $h = $ARGV[0] . '.odb';

  print "Converting [$ARGV[0]] to [$h] in directory [$dir] using command:\n\n";
  print "$f\n\n";

  $z = `$f`;
}
else
{
  print "\n>>>>> ERROR: Must provide eeout file name.\n\n"
```

```
   }

###########################################################################################

sub man {

print <<MAN;
NAME
       mcnp2abaqus - perl wrapper for python conversion script.
                     Setup to do basic conversion of eeout file to Abaqus odb file.
                     eeout file is expected in current directory where this script is run.
                     Output files are stored in the current directory where this script is run.

SYNOPSIS
       mcnp2abaqus [SCRIPT OPTIONS]

       or

       mcnp2abaqus eeout_file_name

EXAMPLE
       mcnp2abaqus -man

         This will display this help message.

       mcnp2abaqus myfile.eeout

         Converts file "myfile.eeout" to "myfile.eeout.odb"


DESCRIPTION
       Runs the latest version of python conversion script.
       A basic conversion of the eeout file to the Abaqus odb format is performed.

       This script should be run from the directory where the eeout file resides.
       All output files are stored in this directory.
       There is a log file for the conversion utility that this script sets to "output.log";
       look here for clues if there is a problem with the conversion process.


SCRIPT OPTIONS

       -help, --help, -man
              Prints this help message.

MAN
1;
}
```