```
#!/usr/bin/env python

"""Filename: SrbRegisterUtil.py
A utility to ingest simulation run output and Copy important files into SRB.

IMPORTANT: Run this script from a directory where the simulation output is stored.
Example: SrbRegisterUtil.py is located in /home/head/Navy
         output is located in /home/head/Navy/20120522001/
         thus: %cd /home/head/Navy/20120522001
               %../SrbRegisterUtil.py -r 20120522001

IMPORTANT: Please run Sinit before running this script with your credentials


NOTE:
1. Only 32 attributes can be registered to SRB at this moment. The first 32 tags from Readme.xml is read
2. Unit consistency is dependent on the user writing correct and consistent Readme.xml
3. Only 256 characters value can be stored in metadata. This script copies the first 256 chars
4. Directory and file name with space character is not supported.

$Header: /cvs_repository/customers/HPCMP/testbed/NavyPilot/SrbRegisterUtil.py,v 1.5 2013/04/11 17:23:10 martin Exp $
"""
try:
        import os,glob,sys,getopt,pprint,subprocess,shlex
        from xml.etree.ElementTree import ElementTree
except:
        raise "Check you local Python Version. A minimum Python version required"


############# CONFIGURATION ##########

SCHEME="Name_Value"
COLLECTION_PATH=""
MAX_ATTRIBUTE=31            # start from 0 . E.g. val 19 means 20 attributes

KEY_MAXCHAR=15                          # Value of maximum minus 1. For example 31 for 32 keychars to compensate for ending null chars
VAL_MAXCHAR=255                         # Value of maximum minus 1. For example 255 for 255 valchars to compensate for ending null chars
class SrbRule:
        pass


### To add more rules, copy over the block to new line and append your rule to lrule_set list

ruleIn = SrbRule()
ruleIn.name = 'Input files'
ruleIn.rule = '*.in'
ruleIn.retention_period = 365
ruleIn.DR_behavior = "yes"

```

```
ruleRin = SrbRule()
ruleRin.name = 'restart files'
ruleRin.rule = '*rin*'
ruleRin.retention_period = 365
ruleRin.DR_behavior= "yes"


ruleXml = SrbRule()
ruleXml.name = 'Xml files'
ruleXml.rule = '*.xml'
ruleXml.retention_period = 365
ruleXml.DR_behavior = "yes"


ruleOcth = SrbRule()
ruleOcth.name = 'Output files for CTH'
ruleOcth.rule = 'oct*'
ruleOcth.retention_period = 365
ruleOcth.DR_behavior = "yes"

ruleHistory = SrbRule()
ruleHistory.name = 'History file'
ruleHistory.rule = ('hscth*')
ruleHistory.retention_period = 180
ruleHistory.DR_behavior = "no"

ruleHistory2 = SrbRule()
ruleHistory2.name = 'History file 2'
ruleHistory2.rule = ('hcth*')
ruleHistory2.retention_period = 180
ruleHistory2.DR_behavior = "no"


rulePlot = SrbRule()
rulePlot.name = 'Plot file'
rulePlot.rule = 'SPCTH/*'
rulePlot.retention_period = 180
rulePlot.DR_behavior = "no"

ruleRestart = SrbRule()
ruleRestart.name = 'Restart file'
ruleRestart.rule = 'RSCTH/*'
ruleRestart.retention_period = 45
ruleRestart.DR_behavior = "no"

##ALE 3D##


ruleRestartALE3D = SrbRule()
ruleRestartALE3D.name = 'Restart file for ALE3D'
ruleRestartALE3D.rule = '*_*_*'
ruleRestartALE3D.retention_period = 45
ruleRestartALE3D.DR_behavior = "no"

rulePlotALE3D = SrbRule()
rulePlotALE3D.name = 'Plot file for ALE3D'
rulePlotALE3D.rule = '*_*.*'
```

```
rulePlotALE3D.retention_period = 45
rulePlotALE3D.DR_behavior = "no"

ruleHistoryALE3D = SrbRule()
ruleHistoryALE3D.name = 'history file for ALE3D'
ruleHistoryALE3D.rule = 'timehist.*.*/'
ruleHistoryALE3D.retention_period = 180
ruleHistoryALE3D.DR_behavior = "no"

lruleSetCTH = 
[ruleIn,ruleXml,ruleOcth,ruleHistory,ruleHistory2,rulePlot,ruleRestart]
lruleSetALE3D = 
[ruleIn,ruleRin,ruleXml,ruleHistoryALE3D,rulePlotALE3D,ruleRestartALE3D]





########################################
def CheckPyVersion():
        """ Function to check Minimum Python Version """
        MIN_PYTHON_VER = 0x02060000
        major          = MIN_PYTHON_VER >> 24
        minor          = MIN_PYTHON_VER >> 16 & 0xff
        micro          = MIN_PYTHON_VER >> 8 & 0xffff

        if (sys.hexversion >= MIN_PYTHON_VER):
                return True
        else:
                print "Local Python is older than required Python 
{0}.{1}.{2}.".format(major,minor,micro)
                return False
#end function

def PyParseTree(rootElement):
        """ Function to parse tree elements (XML) """
        dElements = {}
        lOrderTag = []
        lElements = rootElement.getiterator()  #produces a list of elements
        for elem in lElements:
                key = elem.tag.replace(',', ' ') #strip commas
                val = elem.text.replace(',', ' ') #strip commas
                val = val.replace('\n', ' ') # strip linefeed and carr return

                if dElements.has_key(key[0:KEY_MAXCHAR]):
                        print "..Error. key ({0}) appears twice. Before 
truncation key value is ({1}). Check your Readme.xml".format(key[0:KEY_MAXCHAR],
key)
                        sys.exit(2)
                #end if

                if len(val)>VAL_MAXCHAR or len(key)>KEY_MAXCHAR:
                        if len(val)>VAL_MAXCHAR:
                                print "..Warning. Truncating value for key '{0}' 
to {1} characters.".format(key,VAL_MAXCHAR)
                        elif len(key)>KEY_MAXCHAR:
                                print "..Warning. Truncating key {0} to 
```

```
{1}".format(key,key[0:KEY_MAXCHAR])
                #end if
                        dElements[key[0:KEY_MAXCHAR]]=val[0:VAL_MAXCHAR]
                        lOrderTag.append(key[0:KEY_MAXCHAR])
                else:
                        dElements[key]=val
                        lOrderTag.append(key)
                #endif
        #endfor
        return dElements,lOrderTag

def version():
        print """$Header:
/cvs_repository/customers/HPCMP/testbed/NavyPilot/SrbRegisterUtil.py,v 1.5
2013/04/11 17:23:10 martin Exp $"""
#end function

def usage():
        print """SrbRegisterUtil.py
        usage: SrbRegisterUtil.py <arguments>
                -h : This help
                -r <RunID> : Unique simulation RunID
                --version : Print the script version

        Example
                SrbRegisterUtil.py -r 20120502101
        """
# end of usage function

def execMe(string, input_stderr=1):
        """ Helper function to execute a command line. Return rc, stdout,stderr
"""

        larg = shlex.split(string)
        ###DEBUG: print string
        ###DEBUG: pring larg

        if(not input_stderr == 1):
                #supress stderr
                fnull = open(os.devnull,'w')
                p = subprocess.Popen(larg, stdout=subprocess.PIPE, stderr=fnull)
        else:
                p = subprocess.Popen(larg, stdout=subprocess.PIPE)
        #endif

        out = p.communicate()[0]
        return (p.returncode,out)
# end function

def execSMe(string):
        """ Helper function to execute a command line using flag shell=True """
        ###DEBUG: print "execSMe: running {0}".format(string)
        p = subprocess.Popen(string, stdout=subprocess.PIPE, shell=True)
        out = p.communicate()[0]
        return (p.returncode,out)
# end function

```

```python
def PyIsCollectionExist(string):
        """ Helper function. Check if the relative path supplied by caller
        as string exists. If positive, return true, else return false
        """
        status = False
        print "Checking for existance of collection {0}...".format(string)
        (rc,out) = execMe('Sls {0} > /dev/null 2>&1'.format(string), 2) #suppress stderr output with second argument != 1

        if rc == 0:
                status = True

        return status
# end function

def PyIsSinit():
        """ Check if you have done an Sinit """
        status = False

        # If user is authenticated, Spwd (SRB print working directory) should work
        (rc,out) = execMe('Spwd')

        if rc == 0:
                status = True
        else:
                print"**Error: make sure that Sinit and Senv are in your path and rerun your Sinit command again."
        return status


def PySputAll(COLLECTION_PATH):
        """
        Takes care of copying recursively to SRB

        Example: current directory is /home/Navy/science
        and target collection is homecollection/New_science

        We want to avoid creating homecollection/New_science/science/<files>,
        instead we want to put the files to homecollection/New_science/<files>
        recursively.

        If a user uses wildcard characters (*), it will be expanded up to 1500 characters. There is a risk that not all files will be saved.
        More importantly, there could be an error because the last truncated file is considered target directory while it may be wrong directory or
        a regular file
        """

        status = False

        (rc,out) = execSMe("Sput -Rf {0} {1}".format( os.getcwd() ,COLLECTION_PATH))
        if (rc == 0):
                status = True

        return status
```

```


#end function



def PyConvertToFileList(a_rule):
        """ Convert a rule into list of files """

        lFiles = []
        if len(glob.glob(a_rule.rule)) == 0:
                pass # this rule match nothing in this dir
        else:
                lFiles.append(a_rule.rule)

        return  lFiles

#end function






def main():
        """ Main function. It takes one argument <RunID>

        This is where the program begins

        """
        lOrderTag = []
        if not CheckPyVersion():
                sys.exit(1)

        try:
                opts,args = getopt.getopt(sys.argv[1:], "hvr:", ['version'])
        except getopt.GetoptError, err:
                #print help info and exit:
                print str(err)
                usage()
                sys.exit(2)
        RunID = None
        for arg,val in opts:
                if arg == "-h":
                        usage()
                        sys.exit()
                elif arg == "--version" or arg == '-v':
                        version()
                        sys.exit()
                elif arg == "-r":
                        RunID = val
                else:
                        assert False, "unhandled option"
        #endfor


        # Try to open a file
        try:
                tree = ElementTree()
                root = tree.parse("Readme.xml")

                dElements,lOrderTag = PyParseTree(root)

                if dElements['RunID'] != RunID:
                        print "Error! Argument RunID ({0}) does not match XML file RunID ({1})".format(RunID,dElements['RunID'])
                        sys.exit()


        except IOError:
                print "I/O Error"
                sys.exit(1)
        else:
                print "File Readme.xml is opened successfully."
        #end of try except block


        #Have you run Sinit?
        if (PyIsSinit()):
                print "Sinit is OK..."
                status = 0
        #end of IsSInit block

        lfull_path = os.getcwd().split('/')
        COLLECTION_PATH=RunID

        if(PyIsCollectionExist(COLLECTION_PATH)):
                print "Collection {0} exists. Stop. Danger of overwriting older collection. Please remove collection manually before restarting this script.".format(COLLECTION_PATH)
                status = 1      # I have to stop here. Danger of overwriting existing collection
        else:
                print "Using new collection {0} ....".format(COLLECTION_PATH)

        #end if

        if status == 0 :

                #start Ingesting files to SRB here
                if(not PySputAll(COLLECTION_PATH)):
                        print "Sput call has failed"
                        status = 1
                #end if
        #end if

        ###
        #Set Retention_Period and DR_behavior based on lruleSet
        ####
        if dElements['AppCode'] == 'ALE3D':
                lruleSet = lruleSetALE3D
        elif dElements['AppCode'] == 'CTH':
                lruleSet = lruleSetCTH
        else:
```

```python
                print "Error. Unable to select ruleSet for application code {0}. Perhaps you should declare and set lruleSet{1}".format(dElements['AppCode'])
                sys.exit(1)
        #end if


        if status == 0 :
                for my_rule in lruleSet:
                        target = PyConvertToFileList(my_rule)
                        if len(target) > 0: #ignore empty targets (rules that doesn't match)
                                (rc,out) = execSMe("Sscheme -w -R -val Admin.Retention_Period::{0} {1}/{2}".format(my_rule.retention_period, COLLECTION_PATH, target[0]))
                                if rc != 0:
                                        print "Unable to set retention period for rule: {0} to: {1}".format(my_rule.name,my_rule.retention_period)
                                #end if

                                (rc,out) = execSMe("Sscheme -w -R -val Admin.DR_Behavior::{0} {1}/{2}".format(my_rule.DR_behavior,COLLECTION_PATH, target[0]))
                                if rc != 0:
                                        print "Unable to set DR_behavior for rule: {0} to: {1}".format(my_rule.name, my_rule.DR_behavior)
                                #end if
                        #end if


                # end of for loop

        #end if

        ###
        #Check the existence of Name_Value scheme set in SCHEME var
        ###

        if status == 0 :
                # Command to get the scheme listing. SgetS Name_Value
                (rc,out) = execMe("SgetS {0}".format(SCHEME))
                if rc != 0:
                        status = 1
                        print "**Unable to find Schema ({0}). Ask your admin to create it for you.".format(SCHEME)
                elif SCHEME not in out:
                        status = 1
                        print "**Unable to find Schema ({0}). Ask your admin to create it for you.".format(SCHEME)



        if status == 0:
        ###
        # Write Metadata to collection. Commandline equivalent is:
        # Scheme -w -scheme Name_Value -val Value[1]::"20120518001" PTW1800
        #
        # and so on... For brevity, you can chain up the name value pair as
```

```python
comma, separated list like so:
        # Name[0]::"AppCode",Value[0]::"ALE3D" and so on...
        ###
                del lOrderTag[0] # sentinel element
                lKeys = sorted(dElements.keys())

                sBeginCmd = "Sscheme -w -scheme {0} -val ".format(SCHEME)
                iCnt = 0

                # Let's read our dictionary from XML file and register
                # each name_value pair one by one
                #
                for key in lOrderTag:
                        if iCnt > MAX_ATTRIBUTE:
                                break
                        sCmd = sBeginCmd + "Name[{0}]::\"{1}\",Value[{2}]::\"{3}\"".format(iCnt,key,iCnt,dElements[key]) + " " + COLLECTION_PATH
                        (rc,out) = execMe(sCmd)
                        if (rc != 0):
                                print "**Unable to assign metadata {0}={1} to this collection".format(key,dElements[key])
                                status = 1
                        #end if

                        iCnt = iCnt+1
                #end of for loop

        #end of if

        if status == 0:
                print "Success. all metadata has been recorded"
        else:
                print "There is an error in this operation. See error message above and run again"

#end of main function


if __name__ == "__main__":
        main()

```