import datetime,itertools,logging,os,sys, msatools, sqlite3,re, glob
import numpy as np
import random as rnd

APPLOGGER=logging.getLogger("ngsphy-refselector")
class NRSException(Exception):
def **init**(self, expression, message, time):
self.expression = expression
self.message = message
self.time= time

class ReferenceSelection:
startTime=None
endTime=None
path=""
projectName=""
output=""
inputprefix=""
outputprefix=""
seqDescriptionFile=""
method=0
nsize=-1
numLociPerReplicate=[]
numLociPerReplicateDigits=[]
numReplicates=0
numReplicatesDigits=0
ploidy=1
db="" # Path of the SimPhy DB

```
def __init__(self, args):
    self.startTime=datetime.datetime.now()
    self.endTime=None
    APPLOGGER.info(\
        "{0}".format(\
        "RefSelector started",\
    ))
    # Variable initialization
    self.method=args.method
    self.inputprefix=args.input_prefix
    self.outputprefix=args.output_prefix
    self.nsize=args.nsize
    if args.seq_desc_file:
        self.seqDescriptionFile=os.path.abspath(args.seq_desc_file)

    ################################################################################
```

```
    # Checking correctness of the given paths
    if (args.simphy_path[-1]=="/"):
        self.projectName=os.path.basename(args.simphy_path[0:-1])
    else:
        self.projectName=os.path.basename(args.simphy_path)
    self.path=os.path.abspath(args.simphy_path)
    output=os.path.abspath(args.output)
    outputFolderName=""
    if (args.output[-1]=="/"):
        outputFolderName=os.path.basename(args.output[0:-1])
    else:
        outputFolderName=os.path.basename(output)
    if (os.path.exists(output)):
        listdir=os.listdir("{}".format(os.path.dirname(output)))
        counter=0
        for item in listdir:
            if outputFolderName in item:
                counter+=1
        if not counter == 0: outputFolderName+="_{0}".format(counter+1)
    self.output=os.path.join(os.path.dirname(output),outputFolderName)
    if args.ploidy in [1,2]:
        self.ploidy=args.ploidy
    else:
        parserMessageWrong="\n\t{0}\n\t{1}\n\t{2}".format(\
            "Ploidy value is out of range.",\
            "Value must be in [1,2].",\
            "Please verify. Exiting."\
        )
        raise NRSException(False, message, datetime.datetime.now()-self.startTime)

    ###########################################################################
    # Generation of the output folder
    try:
        os.mkdir(self.output)
        APPLOGGER.info("Generating output folder:\t{}".format(self.output))
    except:
        APPLOGGER.info("Output folder ({0}) exists. ".format(self.output))
    if  self.nsize > -1:
        try:
            os.mkdir(os.path.join(self.output, "bed"))
            APPLOGGER.info("Generating output folder:\t{}".format(os.path.join(self
        except:
            APPLOGGER.info("Output folder ({0}) exists. ".format(os.path.join(self.(

def checkArgs(self):
    APPLOGGER.info("Checking arguments...")
    APPLOGGER.info("\tSimPhy...")
    simphydir=os.path.exists(self.path)
    ###########################################################################
    if simphydir:
        APPLOGGER.info("SimPhy folder exists:\t{0}".format(simphydir))
    else:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        ex="SimPhy folder does not exist."
        message="{1} | File: {2} - Line:{3}\n\t{0}".format(\
```

```
            ex,exc_type,fname, exc_tb.tb_lineno,\
            "Please verify. Exiting.")
        raise NRSException(False, message, datetime.datetime.now()-self.startTime)
    fileList=os.listdir(os.path.abspath(self.path))
    for index in range(0,len(fileList)):
        fileList[index]=os.path.abspath(os.path.join(self.path,fileList[index]))
    self.db = os.path.join(self.path,"{0}.db".format(self.projectName))
    if not self.db in fileList:
        ex="SimPhy required file do not exist."
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        message="{1} | File: {2} - Line:{3}\n\t{0}".format(\
            ex,exc_type,fname, exc_tb.tb_lineno,\
            "Please verify. Exiting."\
        )
        raise NRSException(False, message, datetime.datetime.now()-self.startTime)
    APPLOGGER.info("SimPhy data base exist:\t{0} ({1})".format(\
        os.path.basename(self.db),self.db in fileList)
    )
    APPLOGGER.info("\tIdentifying replicates...")
    # check how many of them are dirs
    for item in fileList:
        baseitem=os.path.basename(item)
        if (os.path.isdir(os.path.abspath(item)) and  baseitem.isdigit()):
            self.numReplicates=self.numReplicates+1
    self.numReplicatesDigits=len(str(self.numReplicates))
    ###########################################################################
    # check if at least one
    if not (self.numReplicates>0):
        ex="Number of replicates/folders:\t{0} [Required at least 1]".format(self.nu
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        message="{1} | File: {2} - Line:{3}\n\t{0}".format(\
            ex,exc_type,fname, exc_tb.tb_lineno,\
            "Please verify. Exiting."\
        )
        raise NRSException(False, message, datetime.datetime.now()-self.startTime)
    APPLOGGER.info("\tDone!")
    APPLOGGER.info("Number of replicates:\t{0}".format(self.numReplicates))
    ###########################################################################
    if self.nsize > -1:
        APPLOGGER.info("Reference sequences will be concatenated.")
    ###########################################################################
    if self.method==1:
        if not (os.path.exists(self.seqDescriptionFile) and os.path.isfile(self.seqI
            message="{0}\n\t{1}".format(\
                "Sequence description file does not exist.",\
                "Please verify. Exiting."\
            )
            raise NRSException(False, message, datetime.datetime.now()-self.startTi
    ###########################################################################
    self.getNumLociPerReplicate()
    ###########################################################################
    ###########################################################################
    # sequence list initialization
    ###########################################################################
```

```
        replicateIndexes=[self.numLociPerReplicate[item]*[item+1] for item in range(0, s
        replicateIndexes=[item for sublist in replicateIndexes for item in sublist]
        lociIndexes=[range(1,self.numLociPerReplicate[item]+1) for item in range(0, self
        lociIndexes=[item for sublist in lociIndexes for item in sublist]
        seqs=["1_0_0"]*len(lociIndexes)
        self.sequenceList=[ (replicateIndexes[index], lociIndexes[index], seqs[index]) f

    def getNumLociPerReplicate(self):
        query="select N_Loci from Species_Trees"
        con = sqlite3.connect(self.db)
        res=con.execute(query).fetchall()
        con.close()
        self.numLociPerReplicate=[item for sublist in res for item in sublist]
        for item in range(0,len(self.numLociPerReplicate)):
            self.numLociPerReplicateDigits+=[len(str(self.numLociPerReplicate[item]))]

    def getBEDfile(self, index):
        repID=index+1
        control=os.path.join(\
            self.path,\
            "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
            "control.txt"\
        )
        f=open(control,"r")
        lines=f.readlines()
        f.close()
        sizes=[ int(line.replace("[","").replace("]","").strip().split()[4]) for line in
        startpos=0
        bedfile=os.path.join(\
            self.output,\
            "bed",\
            "{0}_{1:0{2}d}.bed".format(\
                self.outputprefix,\
                repID,self.numReplicatesDigits\
            )
        )
        outfile=open(bedfile,'a')
        totalConcatSize=len(str(sum(sizes)+(self.nsize*len(sizes))))
        sequenceListIndex=sum(self.numLociPerReplicate[0:repID-1])
        if repID==1: sequenceListIndex=0
        for locID in range(1,self.numLociPerReplicate[repID-1]+1):
            endpos=startpos+sizes[locID-1]
            chromField="{chrom:{align}0{fillChrom}d}".format(\
                align="<",\
                chrom=repID,\
                fillChrom=self.numReplicatesDigits)
            positions="{startPOS:{align}{posSIZE}}\t{endPOS:{align}{posSIZE}}".format(\
                align=">",\
                startPOS=startpos,\
                endPOS=(endpos-1),\
                posSIZE=int(str(totalConcatSize))\
            )

            nameField="{name:0{fillName}d}.{seqDescription}".format(\
                name=locID,\
                fillName=self.numLociPerReplicateDigits[repID-1],\
```

```
            seqDescription=self.sequenceList[sequenceListIndex][2]
        )
        outfile.write("Replicate{0}\t{1}\tLocus{2}\n".format(\
            chromField,\
            positions,\
            nameField\
        ))
        startpos=endpos+(self.nsize-1)
        sequenceListIndex+=1
    outfile.close()


def filterReplicatesMatchingIndPerSpeciesAndPloidy(self, ploidy):
    """
    Identifies and filters the species tree replicates that  SimPhy
    database.
    ---------------------------------------------------------------------
    Returns: a list with the number of loci per species tree replicate
    """
    query="select SID from Species_Trees"
    if not (ploidy == 1):
        query="select SID from Species_Trees WHERE Ind_per_sp % {0} = 0".format(plo
    con = sqlite3.connect(self.db)
    res=con.execute(query).fetchall()
    con.close()
    res=[item for sublist in res for item in sublist]
    return res

def iterateOverReplicate(self):
    APPLOGGER.debug("IterateOverReplicate")
    filtered=self.filterReplicatesMatchingIndPerSpeciesAndPloidy(self.ploidy)
    for index in range(0, self.numReplicates):
        repID=index+1
        if repID in filtered:
            APPLOGGER.debug("Replicate {0}/{1} ".format(repID, self.numReplicates))
            curReplicatePath=os.path.join(\
                self.path,\
                "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
            )
            fileList=glob.glob("{0}/{1}_*.fasta".format(curReplicatePath,self.input
            prefixLoci=len(fileList)
            APPLOGGER.info("Method chosen: {0}".format(self.method))
            print (self.numLociPerReplicateDigits)
            ##########################################################################
            if self.method==0:
                self.methodOutgroup(index)
            ##########################################################################
            if self.method==1:
                if not self.seqDescriptionFile == "":
                    self.seqPerLocus(index)
                else:
                    message="{0}\n\t{1}".format(\
                        "Required file for method 1.",\
                        "Please verify. Exiting."
                    )
                    raise NRSException(False, message, datetime.datetime.now()-self
```

```
            ########################################################################
            if self.method==2:
                self.methodRandomIngroup(index)
            ########################################################################
            if self.method==3:
                self.methodConsensusRandomSpecies(index)
            ########################################################################
            if self.method==4:
                self.methodConsensusAll(index)

            if (self.nsize>-1): # only way I know sequences are concatenated
                self.getBEDfile(index)


def writeLocus(self, index,locID,description,sequence):
    APPLOGGER.debug("Write Loci()")
    if self.nsize > -1:
        APPLOGGER.debug("CONCAT")
        self.concatSelectedLoci(index,locID,description,sequence)
    else:
        APPLOGGER.debug("SEPARATE SEQUENCE")
        self.writeSelectedLoci(index,locID,description,sequence)

def methodOutgroup(self,index):
    """
    Method 0 for the selection of reference loci.
    This method selects the outgroup as a reference locus.
    ----------------------------------------------------------------------
    attributes: repID: Index of the species tree that is being used.
    returns: Nothing
    """
    APPLOGGER.debug("method outgroup")
    description="0_0_0"
    repID=index+1
    sequenceListIndex=sum(self.numLociPerReplicate[0:index])
    if index==0: sequenceListIndex=0
    for locID in range(1,self.numLociPerReplicate[index]+1):
        APPLOGGER.info("Locus {0}/{1} | Table Index: {2}".format(locID,self.numLociI
        mytuple=list(self.sequenceList[sequenceListIndex])
        mytuple[2]=description
        self.sequenceList[sequenceListIndex]=tuple(mytuple)
        fastapath=os.path.join(\
            self.path,\
            "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
            "{0}_{1:0{2}d}.fasta".format(self.inputprefix,locID, self.numLociPerRep
        )
        lociData=msatools.parseMSAFileWithDescriptions(fastapath)
        selectedSequence=lociData[description]
        self.writeLocus(index,locID,description,selectedSequence)
        sequenceListIndex+=1
    APPLOGGER.info("Done outgroup sequence")

def seqPerLocus(self,index):
    """
    Method 1 for the selection of reference loci.
    ----------------------------------------------------------------------
```

```
        This method selects a sequence per locus as indicated in the seq_desc_file.
        Args: repID: Index of the species tree that is being used.
        Returns: Nothing
        """
        entries=self.parseReferenceLociFile(self.seqDescriptionFile)
        for entry in entries:
            repID=entry[0]
            locID=entry[1]
            seqID=entry[2]
            APPLOGGER.info("Locus {0}/{1}".format(locID,self.numLociPerReplicate[index]
            fastapath=os.path.join(
                self.path,\
                "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
                "{0}_{1:0{2}d}_TRUE.fasta".format(self.inputprefix,locID,self.numLociPe
            )
            fastaFile=msatools.parseMSAFileWithDescriptions(fastapath)
            sequence=""
            try:
                sequence=fastaFile[seqID]
            except:
                message="{0}\n\t{1}".format(\
                    "One of the selected sequences (description) has not been found on
                    "Please verify. Exiting"
                )
                raise NRSException(False, message, datetime.datetime.now()-self.startTi

            self.writeLocus(index,locID,seqID,sequence)
        APPLOGGER.info("Done Seq Per locus")

    def methodRandomIngroup(self,index):
        """
        Method 1 for the selection of reference loci.
        ------------------------------------------------------------------------
        This method selects a random sequence from the ingroup species as
        a reference and from all the loci.
        Args: repID: Index of the species tree that is being used.
        Returns: Nothing
        """
        repID=index+1
        sequenceListIndex=sum(self.numLociPerReplicate[0:index])
        if index==0: sequenceListIndex=0
        for locID in range(1,self.numLociPerReplicate[index]+1):
            APPLOGGER.info("Locus {0}/{1} | Table Index: {2}".format(locID,self.numLoci
            fastapath=os.path.join(\
                self.path,\
                "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
                "{0}_{1:0{2}d}.fasta".format(self.inputprefix,locID, self.numLociPerRep
            )
            lociData=msatools.parseMSAFile(fastapath)
            keys=lociData.keys()
            rndKey1="0"; rndKey2="0"
            try:
                rndKey1=rnd.sample(set(keys)-set("0_0"),1)[0]
            except:
                rndKey1="0"
            subkeys=lociData[rndKey1]
```

```
            try:
                rndKey2=rnd.sample(len(subkeys),1)[0]
            except:
                rndKey2="0"
            selected=lociData[rndKey1][rndKey2]
            # print(self.sequenceList[sequenceListIndex])
            mytuple=list(self.sequenceList[sequenceListIndex])
            mytuple[2]="{0}_{1}".format(rndKey1,rndKey2)
            self.sequenceList[sequenceListIndex]=tuple(mytuple)
            self.writeLocus(index,locID,selected["description"],selected["sequence"])
            sequenceListIndex+=1
        APPLOGGER.info("Done random ingroup sequence")

    def methodConsensusRandomSpecies(self,index):
        """
        Method 2 for the selection of reference loci.
        --------------------------------------------------------------------
        This method selects a consensus sequence, obtained from the random selection
        of a species, and then computing a consensus from all the sequences
        within the selected species.
        Args: repID: Index of the species tree that is being used.
        Returns: Nothing
        """
        repID=index+1
        sequenceListIndex=sum(self.numLociPerReplicate[0:index])
        if repID==1: sequenceListIndex=0
        for locID in range(1,self.numLociPerReplicate[index]+1):
            APPLOGGER.info("Locus {0}/{1} | Table Index: {2}".format(locID,self.numLociP
            fastapath=os.path.join(\
                self.path,\
                "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
                "{0}_{1:0{2}d}.fasta".format(self.inputprefix,locID, self.numLociPerRep
            )
            lociData=msatools.parseMSAFile(fastapath)
            keys=lociData.keys()
            rndKey1=0; rndKey2=0
            try:
                rndKey1=rnd.sample(set(keys)-set("0_0"),1)[0]
            except:
                rndKey1=0
            subkeys=lociData[rndKey1]
            sequences=[]
            self.sequenceList[sequenceListIndex]
            for sk in subkeys:
                sequences+=[lociData[rndKey1][sk]["sequence"]]
            mytuple=list(self.sequenceList[sequenceListIndex])
            mytuple[2]="{0}_CONSENSUS_RND_SP".format(rndKey1)
            self.sequenceList[sequenceListIndex]=tuple(mytuple)
            selected=self.computeConsensus(sequences)
            selectedDes=">consensus_sp_{0}".format(rndKey1)
            self.writeLocus(index,locID,selectedDes,selected)
            sequenceListIndex+=1
        APPLOGGER.info("Done random ingroup consensus")

    def methodConsensusAll(self,index):
        """
```

```
    Method 3 for the selection of reference loci.
    ------------------------------------------------------------------------
    Computes the consensus from all the sequences of a gene tree file,
    and uses this sequence as reference loci.
    Args: repID: Index of the species tree that is being used.
    Returns: Nothing
    """
    repID=index+1
    sequenceListIndex=sum(self.numLociPerReplicate[0:index])
    if repID==1: sequenceListIndex=0
    for locID in range(1,self.numLociPerReplicate[index]+1):
        APPLOGGER.info("Locus {0}/{1} | Table Index: {2}".format(locID,self.numLociP
        fastapath=os.path.join(\
            self.path,\
            "{0:0{1}d}".format(repID, self.numReplicatesDigits),\
            "{0}_{1:0{2}d}.fasta".format(self.inputprefix,locID, self.numLociPerRepl
        )
        lociData=msatools.parseMSAFileWithDescriptions(fastapath)
        keys=set(lociData.keys())
        sequences=[]
        for mk in keys:
            sequences+=[lociData[mk]]
        selected=self.computeConsensus(sequences)
        self.writeLocus(index,locID,">consensus_all",selected)
        mytuple=list(self.sequenceList[sequenceListIndex])
        mytuple[2]="{0}_CONSENSUS_ALL".format(rndKey1)
        self.sequenceList[sequenceListIndex]=tuple(mytuple)
        sequenceListIndex+=1
    APPLOGGER.info("Done all ingroups consensus")


def computeConsensus(self, sequences):
    """
    Method for the computation of a consensus sequence from a set
    of sequences.
    ------------------------------------------------------------------------
    Args: sequences: list of the sequences
    Returns: Single consensus sequence
    """
    conseq=""
    # Assume that all the sequences in a file have the same length
    seqSize=len(sequences[0])
    # Need to know how many sequences I have
    numSeqs=len(sequences)
    # Seqs for all nucleotides
    A=np.zeros(seqSize);
    C=np.zeros(seqSize);
    G=np.zeros(seqSize);
    T=np.zeros(seqSize);
    N=np.zeros(seqSize);
    for indexCol in range(0, seqSize):
        for indexRow in range(0,numSeqs):
            if sequences[indexRow][indexCol]=="A": A[indexCol]+=1
            if sequences[indexRow][indexCol]=="C": C[indexCol]+=1
            if sequences[indexRow][indexCol]=="G": G[indexCol]+=1
            if sequences[indexRow][indexCol]=="T": T[indexCol]+=1
```

```
            if sequences[indexRow][indexCol]=="N": N[indexCol]+=1

    for indexCol in range(0,seqSize):
        if A[indexCol] > C[indexCol] and A[indexCol] > G[indexCol] and A[indexCol] >
            conseq+="A"
        elif C[indexCol] > A[indexCol] and C[indexCol] > G[indexCol] and C[indexCol]
            conseq+="C"
        elif G[indexCol] > A[indexCol] and G[indexCol] > C[indexCol] and G[indexCol]
            conseq+="G"
        elif T[indexCol] > A[indexCol] and T[indexCol] > C[indexCol] and T[indexCol]
            conseq+="T"
        else: conseq+="N"

    APPLOGGER.info("Consensus computed")
    return conseq


def writeSelectedLociMultipleSpecies(self,index,locID,seqs):
    """
    Writes multiple sequences in a single file.
    ----------------------------------------------------------------------
    Args:
    - repID: Index of the species tree that is being used.
    - locID: Index of the locus being used.
    - seqs: sequence of the locus to be written.
    ----------------------------------------------------------------------
    Returns: Nothing
    ----------------------------------------------------------------------
    Generates a file for all the selected loci.
    """
    repID=index+1
    # seqs=[(description,seq), ..., (description,seq)]
    APPLOGGER.info("Writing selected loci {1} from ST: {0}", repID,locID)
    outname=os.path.join(\
        self.output,\
        "{0}_{1:0{2}d}_{3:0{4}d}.fasta".format(\
            self.outputprefix,\
            repID,self.numReplicatesDigits,\
            locID,self.numLociPerReplicateDigits[index]\
        )\
    )
    outfile=open(outname,'a')
    for item in range(0,len(seqs)):
        des=seqs[item][0]
        nucSeq=seqs[item][1]
        newDes=">{0}:{1:0{2}d}:REF:{7}:{6}:{3:0{4}d}:{5}".format(\
            self.projectName,\
            repID,\
            self.numReplicatesDigits,\
            locID,\
            self.numLociPerReplicateDigits[index],\
            des[1:len(des)],\
            self.inputprefix,\
            self.outputprefix
        )
        outfile.write("{0}\n{1}\n".format(newDes,nucSeq))
```

```
        outfile.close()

    def parseReferenceLociFile(self, filename):
        """
        Used to parse sequenceList, file with format: STID,LOCID,TIPLABEL
        ------------------------------------------------------------------------
        Parameters:
        - filename: path of the sequenceList file.
        There's only ONE file with the relation of the ref_alleles
        If "None" inputted (file is missing) then reference by default is 1_0_0
        for all species tree replicates.
        Returns:
        - output: list. each element of the list is a triplet
                (REPLICATEID,sequence_tip_label )
                replicate_ID    locus_ID    sequence_description_locus
        """
        APPLOGGER.info("Retrieving identifiers of the reference alleles...")
        filepath=os.path.abspath(filename)
        if os.path.exists(filepath):
            # There's a file
            lines=None
            with open(filepath) as f:
                lines=[line.strip().split() for line in f if (not line.strip()=="") and
            # print lines
            lines=sorted(lines, key=lambda x:(x[1],x[2]))
            skipped=False
            message="Parsing reference list. "+\
                "A default reference has been introduced.\n"+\
                "Replicate index:"

            for i in range(0,len(self.sequenceList)):
                for j in range(0,len(lines)):
                    if  (not lines[j][0] == "") and \
                        (not lines[j][1] == "") and \
                        (not lines[j][2] == "") and \
                        (lines[i][0]==self.sequenceList[j][0] and lines[i][1]==self.seq
                        bool(re.match("^([1-9]+_[0-9]+_[0-9]+){1}",lines[i][3])):
                        self.sequenceList[j][3]=lines[i][3]
        return self.sequenceList

    def concatSelectedLoci(self,index,locID,description,sequence):
        """
        BEDFILE: replicateID startPOS endPOS locID
        """
        repID=index+1
        APPLOGGER.info("Writing selected loci {1} from ST: {0}".format(repID,locID))
        outname=os.path.join(\
            self.output,\
            "{0}_{1:0{2}d}.fasta".format(\
                self.outputprefix,\
                repID,self.numReplicatesDigits
            )
        )
        newDes=">{0}:{1}:{2:0{3}d}".format(\
            self.projectName,\
            self.outputprefix,\
```

```
            repID, self.numReplicatesDigits
        )
        nsequence="".join("N" for item in range(0,self.nsize))
        # I'm assuming that if the file does not exist it will be created
        fullseq="{0}{1}".format(sequence,str(nsequence))
        if os.path.exists(outname):
            with open(outname, 'a+') as f:
                f.seek(-1,2)
                if locID == self.numLociPerReplicate[repID-1]:
                    f.write('\n'.encode())
                else:
                    f.write(fullseq.encode())
        else:
            f=open(outname,"a+")
            f.write(">{0}\n{1}".format(description, fullseq))
            f.close()

    def writeSelectedLoci(self,index,locID,des,seq):
        """
        Writes a single sequence per file.
        ------------------------------------------------------------------------
        Input:
            - repID: Index of the species tree that is being used.
            - locID: Index of the locus being used.
            - des: description of the sequence to be written.
            - seq: sequence of the locus to be written.
        Returns: Nothing
            - Generates a file per selected locus.
        """
        repID=index+1
        APPLOGGER.info("Writing selected loci {1} from ST: {0}", repID,locID)
        outname=os.path.join(\
            self.output,\
            "{0}_{1:0{2}d}_{3:0{4}d}.fasta".format(\
                self.outputprefix,\
                repID,self.numReplicatesDigits,\
                locID, self.numLociPerReplicateDigits[repID-1]
            )
        )
        newDes=">{0}:{1:0{2}d}:REF:{7}:{6}:{3:0{4}d}:{5}".format(\
            self.projectName,\
            repID,\
            self.numReplicatesDigits,\
            locID,\
            self.numLociPerReplicateDigits[repID-1],\
            des[1:len(des)],\
            self.inputprefix,\
            self.outputprefix
        )
        # I'm assuming that if the file does not exist it will be created
        outfile=open(outname,'a')
        outfile.write("{0}\n{1}\n".format(newDes,seq))
        outfile.close()


    def run(self):
```

```
        """
        Run process of the program.
        """
        self.checkArgs()
        self.iterateOverReplicate()
        raise NRSException(True,"",datetime.datetime.now()-self.startTime)
```