Appendix B: Satsophsi.py documentation and computer code

Satsophsi.py Program File Documentation

The program that performs the HSI/HEP calculations is satophsi.py. It was developed using the Python programming language. Large amounts of data are used by the program and must be stored in memory as variables, lists, and dictionaries for access as the program runs. Several individual functions within the program are called at various points to make calculations using data contained in the input tables. The majority of these were written specifically for satsophsi.py with the remaining having been borrowed from other programs or being part of a library of functions that come with LMS.

Data storage

Data used by satsophsi.py are in the form of variables, lists and dictionaries. Each type has a place in the program. Variables are individual values used repeatedly throughout the program or taken from lists or dictionaries of data stored in memory. These can be thought of as single-dimensional data. Lists contain several data of a related type. This may be a list of the HSI models that will be run by the program or a list of conifer tree species codes. Dictionaries are multi-dimensional data, with each datum being stored with keys that point to its location in the dictionary.

Proc_args(args)

Satsophsi.py is run be calling the Python interpreter and sending a command line with all necessary arguments and options for running the program. Proc_args(), which was developed by Jim McCarter (Silviculture Laboratory, College of Forest Resources, University of Washington) and modified to satsophsi.py, processes the command line and returns a list of arguments and options from the command line. Input for proc_args() is the command line arguments that are called by the sys.argv[] command.

ProcessHSIINI(inifile)

ProcessHSIINI() was originally developed by Jim McCarter to take needed information from the hsi.ini configuration file. It was greatly expanded as the configuration file grew in size and scope. Input for ProcessHSIINI() is the hsi.ini configuration file. ProcessHSIINI() then works through the file, taking all the data and storing them as global variables, lists or dictionaries.

CheckSpecies(sppcode)

CheckSpecies() was developed by Jim McCarter to create lists of conifer and deciduous species codes. Input for CheckSpecies() is a species code from the list of conifer species created by ProcessHSIINI(). Each time the a species code is fed to CheckSpecies(), it is checked against the list of conifer species. If the species code is found, the function then returns to the main program. If the species code is not found in the list of conifer species, it is added to the list of deciduous species that is created by CheckSpecies().

IsConifer(sppcode)

IsConifer() was developed by Jim McCarter to check species codes against the list of conifer species. Input is a species code. If the code is contained, IsConifer() will return a variable value of “TRUE”. If not it will return “FALSE”.

CalcLMSData(y, s)

CalcLMSData() calculates all needed tree-based values from inventory data that are stored in memory as the main program processes the inventory file fed to satsophsi.py by LMS; it then stores those values in a data dictionary. Input for CalcLMSData() are the year and stand from the inventory file being processed. When CalcLMSData() is run, most of the calculations are performed by functions from an LMS library. These include total TPA, conifer TPA, big trees

per acre, average overstory height, average DBH, overstory DBH, canopy layers, total basal are, conifer basal area, canopy closure, conifer canopy closure, and percentages of conifer and deciduous trees.

Trees per acre values are calculated using an algorithm that sums all expansion factors for each tree record in memory. "Total trees pre acre" sums all expansion factors while, for conifer TPA, a list of conifer species codes is fed to the algorithm and the sum is limited to those species only. To calculate the number of big trees, a minimum DBH of 20.9" is fed to the algorithm and the sum limited to the tree records having a DBH above that threshold.

Average overstory height is calculated by an algorithm that performs an arithmetic average of the height of all tree records in memory. To get an average of only the overstory a limit of the tallest 40 trees per acre is given to the algorithm; and the average is calculated using only those 40 trees.

Average diameters are calculated for the entire stand as well as only the overstory. Arithmetic averages are used here since that was used for the original HEP calculations. Overstory average diameters are calculated using the largest 40 tree records per acre stored in memory.

Canopy layers are calculated using an algorithm developed by Baker and Wilson (Baker and Wilson 2000). This algorithm keeps a running average of heights to live crown base, beginning with the tallest tree in the tree records stored in memory and working to the shortest. If a break in the vertical structure of the canopy is found, it is counted and the process is started over again with the remaining tree records. This process is repeated until all tree records have been exhausted.

Basal area is calculated for the entire stand for all species and for conifer species only. The algorithm first calculates basal area for all species, then for each species in the conifer species list. The conifer basal areas are then summed for a value of total conifer basal area.

Canopy closure is calculated using an algorithm published by Crookston and Stage (Crookston and Stage 1999). This is performed for all species and for conifer species only. Total crown area for all species and conifer species only are fed into the algorithm. These are calculated and stored in memory as the inventory file is processed when the main program is executed. The values returned are closure values assuming that there is crown overlap and will not exceed 100% closure.

For cover typing the stands, it is necessary to calculated the percent of the stand in conifer and deciduous species. This calculation is done with either percent of total trees per acre or total basal area as defined in the configuration file. Since conifer trees per acre and basal area have already been calculated, they are used to calculate the percentages, with the remaining percentage being the deciduous percentage.

After these data are calculated, they are stored in a data dictionary with keys of year and stand name. These can then be recalled by other functions of the program by just using the variable name, year and stand.

HSI Models

Satsophsi.py contains four HSI models that were originally used for the Satsop HEP (WPPSS 1994a): Cooper's hawk (USDI 1980c), pileated woodpecker (Schroeder 1983), southern red-backed vole (Allen 1983), and spotted towhee (USDI 1978). Each model is a codified version of the original graphical model previously presented in the Background section. Each model uses inputs of year and stand name to recall the necessary input data from data dictionaries. These values are then fed into piecewise equations that calculate the HSI value for each of the life requisites for each species. These are then used to calculate an overall HSI for each stand for each species.

All of the models other than Cooper's hawk use some non-tree-based habitat attributes. These are snag and coarse woody debris as well as understory attributes of grass and shrub cover.

Since these attributes are not currently modeled in LMS, the original HEP data and process is used. Each stand is given a cover type during the cover typing process. With a cover type, non-tree-based values can then be related to the cover type. These values are originally stored in memory by ProcessHSIINI() and are then retrieved by the HSI models when they are run. Given the year and stand input, the stands' cover type is called from the data dictionary; and then the cover type is used to call the needed non-tree-based data from another data dictionary.

Habitat units (HU) are then calculated based on the acreage of the stand. Each of the models then stores the resultant HSI variable values; overall HSI and HU values are then stored in a data dictionary for future use.

StoreHUData(y, ct, m, h, ha, hu)

StoreHUData() summarizes and stores habitat unit data for future use in data dictionaries. Input values are year, cover type, HSI model, total habitat units, habitat acreage, cover type habitat units.

TypeCode(typecode)

Satsophis.py will calculate HSI values for non-timbered cover types if data exists for those cover types in the configuration file. These stands must be part of the LSM portfolio that is being analyzed, with a designation for non-timbered cover type. The SDB file for LMS portfolios contains a column for latitude. Since this field is very rarely used for growth models, it is used in this analysis to designate timbered or non-timbered cover types. Options for these values are: 0 for timbered, 1 for brush, 2 for grass, 3 for palustrine forest, and 4 for palustrine emergent cove types. TypeCode() takes these numeric values and converts them to a alphabetic type code that is then returned as a variable to the main program.

CoverType(y, s)

The CoverType() function calculates the cover type for each stand. Inputs are year and stand name. These are then used to call all the necessary data from the data dictionaries for comparisons to threshold values for each cover type, as defined in the configuration file. Threshold values are called from a data dictionary that was created by ProcessHSININ(). A series of "if" and "and" Boolean statements are then used to classify each stand into a certain cover type. These are a codified set derived from the original cover typing rules (Table XX). The cover type is then returned to the main program as a variable.

Upon running the first version of the cover typing code, it was noticed that some stands failed to classify because of the maximum height on C1 stands. The original cover typing rules place a maximum average height of 15 feet with a DBH range of one to four inches. Since stands with an average DBH of four inches can have a height of more than fifteen feet, that criterion was dropped from the classification rules.

Program Operation

With all functions defined, the main program can then be run. First proc_args() is called to parse the command line and return lists of arguments and options from the command line. Arguments are the files being used for the run. The first is the configuration file, second is the SDB file, third is the inventory file, and fourth is the output file. These give variable names to be used later in the program. ProcessHSININ() is then called to process the configuration file and store all data. Now the SDB file is opened, processed to create dictionaries of stand acreage and alphabetic type codes, and then closed. Processing the inventory file is next. The file is opened, and all lines are read into memory individually. As each line is read into memory, the year, stand name, species code, expansion factor, mean crown width, and height are taken as variables. As data is being accumulated, crown area for all species and conifer species are calculated and summed for all records for the stand being analyzed. When all records for a stand have been read

into memory CalcLMSData() is called and all necessary data for future calculations is calculated and stored in data dictionaries.

Once all data from LMS inventory has been processed, stands are given a cover type; and acreage for each cover type is calculated. This is performed by looping through all years in the list of years created while processing the inventory file and looping through all stands in the list of stands created while processing the SDB file for each year. Acreage and cover type code are called from data dictionaries using year as the input. Then if the cover type code is "T" for "timbered", the CoverType() function is then called; and the stand is given a cover type classification. Acreage for each cover type is then summed as the loop of stands progresses for each year. When the loop of stands is completed, the data are stored in a data dictionary for future use.

If the habitat models are being used, the HSI models are then run. These runs are performed by looping through the list of years and stands as above, but running each model in the list of models defined in the configuration file. When the HSI models have been run for all stands for all years, HSI values, acreages of habitat for each species, and habitat units are then summarized and stored using the StoreHUData() function. Calculating the average amount of habitat for each species for the entire analysis period is then done. The growth period is needed for calculating an annual average, and is calculated from the list of years by taking the difference between the first and second entry in the list. If there is no first element in the list, such as when no stands have been projected, the growth period is assumed to be 0. Habitat units are then summed for each species model run and averaged for an annual average figure.

When all calculations have been made, the output file is written. These all follow the formats covered in the configuration file documentation (appendix C).

Satsophsi.py Computer Code

```
#
# satsophsi.py v1.2
# 30-04-2001
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Satsop Forest Habitat Suitability
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Created by Kevin "thujaman" Ceder, with assistance and guidance
#    from Jim McCarter at the Silviculture Lab, College of
#    Forest Resources, University of Washington, Seattle, WA, USA
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# This program impliments the calculation process of a Habitat
#    Evaluation Procedure (HEP) that was originally performed
#    on the Satsop Nuclear Site (now Satsop Forest) to develop
#    a wildlife mitigation agreement with the Washington
#    Department of Fish and Wildlife (WDFW).  To modernize the
#    the process and allow for analysis of a larger set of
#    potential management alternatives the processes are
#    implemented as an extension for the Landscape Management
#    System (LMS).
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# HEP processes:
#    Cover typing: Determine the cover type for each stand as
#       set forth but the cover typing rules in the original
#       HEP.
#    Cover types used and the tree-based attributes for
#       determining the cover type for timbered polygons are
#       contained in the hsi.ini file in the LMS root directory
#       under "Cover Types", "Timber Types", and in the
#       "Attribute Thresholds" sections.  Thresholds for each
#       cover type can be modified in the hsi.ini BUT DO SO AT
#       YOUR OWN PERIL!!
# Habitat Suitability Index (HSI) calculations:
#    HSI values are calculated for the following species on a
#       per stand basis:
#       - Cooper's hawk
#       - Southern red-backed vole
```

```
#        - Pileated woodpecker
#        - Spotted towhee
#     Single species or multiple species can be run by editing
#        the "ModelList" under "HSI Models" in the hsi.ini file
#        in the LMS root directory.
# Habitat Unit (HU) calculations:
#     HU's are HSI * stand acreage which gives a relative measure
#        of available habitat for each species.  These are
#        by species for the entire landscape contained in the
#        LMS portfolio for each projection period.
# Annual Average Habitat Units (AAHU) calculations:
#     AAHU values are calculated for each species for the life
#        proposed management alternative (LMS scenario run).
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# LMS hooks:
#     HSI calculations (uses -h option):
#        [LMSTable: SATSOPHSI]
#        Execute=MULTIPLE
#        Input1=LandscapeAttributeTable, $foliodir\$cache\$folioname.att
#        Input2=ScenarioTable, $stand, tre, $foliodir\$cache\$folioname.txt
#        F1Line1=cd $foliodir\$cache
#        F1Line2=$lmsdir\python.exe $lmsdir\python\satsophsi.py -h
#           $lmsdir\hsi.ini $folioname.att $folioname.txt $filename
#    HEP cover typing (uses -c option):
#        [LMSTable: SATSOPHEP]
#        Execute=MULTIPLE
#        Input1=LandscapeAttributeTable, $foliodir\$cache\$folioname.att
#        Input2=ScenarioTable, $stand, tre, $foliodir\$cache\$folioname.txt
#        F1Line1=cd $foliodir\$cache
#        F1Line2=$lmsdir\python.exe $lmsdir\python\satsophep.py -c
#           $lmsdir\hsi.ini $folioname.att $folioname.txt $filename
#
# Table definition lines for methods.ini:
#     TableXX=SATSOPHSI, Habitat - Satsop HSI Table, 0, 0, 0
#     TableXX=SATSOPHEP, Habitat - Satsop HEP Cover Type Table, 0, 0, 0
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Files needed for implimentation in LMS:
#     hsi.ini - Configuration file located in the LMS root directory
#        containing:
#        - List of conifer species codes
#        - Length of portfolio growth period
#        - List of cover types
```

```
#       - List of timbered cover types
#       - List of timbered cover type attributes for cover type
#          determination
#       - List of habitat attributes needed for HSI calculations
#       - Habitat attribute data, both tree-based and non-tree-based
#          collected on Satsop Foresty in 1991 for the original
#          HEP and HSI calculations.  These are per cover type
#          averages and are assumed to NOT CHANGE through time.
#       - Type of HSI run
#       - List of HSI models used for the HSI run
#       - List of applicable cover types for model runs by species
#       - Cover typing output table type
#       - HSI output table type
#    FOLIONAME.att - Landscape attribute table for the LMS portfolio.
#       Contains all landscape attributes contained in FOLIONAME.sdb.
#       Created by LMS upon exection of "Satsop HSI Table" or
#          "Satsop HEP Cover Type Table" and deposited in the
#          FOLIONAME/Cache directory
#    FOLIONAME.txt - Landscape inventory table for the LMS propfolio.
#       Contains all timber inventory for initial year and all projected
#          years for all stands.
#       Created by LMS upon exection of "Satsop HSI Table" or
#          "Satsop HEP Cover Type Table" and deposited in the
#          FOLIONAME/Cache directory.
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Data used for cover type determination:
#    These are all calculated from LMS portfolio initial and projected
#       inventory data:
#    - % conifer
#       Determined by TPA or BA by changing "PM" entry under
#          "Percent Method" in hsi.ini.  Conifer species list is
#          "ConiferSpecies" in hsi.ini
#    - % deciduous (hardwood)
#       Determined as TPA or BA NOT conifer
#    - Canopy layers
#    - % total canopy closure (all species)
#       Assumes crown overlap
#    - Average DBH (arithmetic average)
#    - Big trees (DBH > 21") per acre
#    - Dominant height (tallest 40 TPA)
#    - Totat trees per acre (all species)
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
```

```
# Data used for HSI calculations:
#    Tree-based - These are all calculated from LMS:
#       - % total canopy closure (all species)
#       - % conifer canopy closure
#       - Overstory average DBH (inches)
#       - Number of trees per acre with DBH >= 21"
#    Non-tree-based - These were collected at or near Satsop Forest in 1991
#       during the original HEP:
#       ***NOTE*** These are all per cover type averages and
#          assumed to be the same across all stands having that cover type and
#          to not change with time or forest management.
#       - % total ground cover
#       - % grass cover
#       - % ground cover of downfall litter
#       - Shrub Suitability Index (used for spotted towhee calculations)
#       ***NOTE*** The following attributes for stnading dead and down wood
#          are currently from HEP data but will be calcuated from LMS
#          portfolio initial and projected snag invnetory data once the
#          snag model is fully up and running
#       - Number of stumps per acre
#       - Number of logs per acre > 7" diameter
#       - Number of snags per acre >= 21" DBH and 30' tall
#       - Average diameter of snags >= 21" DBH and 30' tall
#
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Available output table types:
#    Determined by "HEPOut" and "HSIOut" entries in hsi.ini file
#    Cover type table options:
#       AV - Table formatted for importing into ESRI ArcView to be
#          joined to the stands shapefile for mapping purposes.
#          Formatted as:
#             Stand, Year_CoverType, NextYear_CoverType,...
#       STD - Table formatted with the following columns:
#          Year, Stand, Acres, CoverType
#       SUM - Summary of cover type acreage by year formatted as:
#          Year, CoverType_Acreage, NextCoverType_Acreage, ...
#       ALL - STD output followed by SUM output with seperator
#       DEBUG - STD output with all attribute values for
#          debugging purposes
#    HSI table options:
#       AV - Table formatted for importing into ESRI ArcView to be
#          joined to the stands shapefile for mapping purposes.
#          Formatted as:
#             Stand, Year_SpeciesHSI, Year_NextSpeciesHSI, ...,
```

```
#                 NextYear_SpeciesHSI, NextYear_NextSpeciesHSI, ...
#       STD - Table formatted with the following columns:
#          Year, Stand, Acres, SpeciesHSI, NextSpeciesHSI, ...
#       HU - Table of habitat units for each species by year formatted as:
#          Year, SpeciesAvgHSI, SpeciesAcreage, SpeciesHU,
#             NextSpeciesAvgHSI, NextSpeciesAcreage, NextSpeciesHU, ...
#       AAHU - Table of annual average habitat units formatted as:
#          SpeciesAAHU, NextSpeciesAAHU, ...
#       ALL - STD, HU, and AAHU outputs seperated by seperators
#       DEBUG - STD output with all habitat attributed.  Used for
#          debugging purposes.
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Functions contained in this program:
#     proc_args( args ) - Devloped by Jim McCarter and tweaked by thujaman.
#        Returns a list of files and a list of options from the command line:
#           "F1Line2" line of LMS hooks
#           - Arguments are necessary files for implimentation as described above
#           - Options are:
#              "-h" for HSI calculations
#              "-c" for cover typing only
#
#    ProcessHSIINI( hsiinifile ) - Initial development by Jim McCarter with
#       scads o' additions by thujaman.
#       Reads hsi.ini file and returns variables, lists, and dictionaries of configuration data
#       All configuration data documented in the hsi.ini file
#
#    CheckSpecies( species code) - Developed by Jim McCarter
#       Checks species codes against list of conifer species returned by ProcessHSIINI().
#       Appends lsit of deciduous species with non-conifer species codes
#
#    IsConifer( species code ) - Developed by Jim McCarter
#       Checks species code to determine if it truly is conifer.
#       Returns TRUE or FALSE
#
#    ComputeStats( year, stand ) - Initial devlopment by Jim McCarter with
#       several tweaks by thujaman.
#    *** CREATE NEW FUNCTION HERE ***
#
#    CHawk( year, stand ) - Devloped by thujaman.
#       Implimentation of Cooper's hawk HSI models (USDI, 1980)
#       Reads data dictionaries for habitat attributes
#       Calculates and returns to dictionary:
#          - Each HSI varialbe value
#          - HSI by year and stand
```

```
#           - HU by year and stand
#           - Habitat acreage by year
#
#    SRVole( year, stand ) - Developed by thujaman
#       Implimentation of southern red-backed vole HSI models (Allen, 1983)
#       Reads data dictionaries for habitat attributes
#       Calculates and returns to dictionary:
#           - Each HSI varialbe value
#           - HSI by year and stand
#           - HU by year and stand
#           - Habitat acreage by year
#
#    PWoodpecker( year, stand ) - Developed by thujaman
#       Implimentation of Pileated woodpecker HSI models (Schroeder, 1983)
#       Reads data dictionaries for habitat attributes
#       Calculates and returns to dictionary:
#           - Each HSI varialbe value
#           - HSI by year and stand
#           - HU by year and stand
#           - Habitat acreage by year
#
#    STowhee( year, stand ) - Devloped by thujaman
#       Implimentation of spotted towhee HSI models (USDI, 1978)
#       Reads data dictionaries for habitat attributes
#       Calculates and returns to dictionary:
#           - Each HSI varialbe value
#           - HSI by year and stand
#           - HU by year and stand
#           - Habitat acreage by year
#
#    HEPCode( year, stand ) - Developed by thujaman
#       Reads data dictionaries for timber cover typing attributes
#       Returns cover type for each stand for each year
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# OK... Enough of that!  Lets get to the meat of this thing!!!!!
#

#
# Get all the necessary modules
#

# Stock Python stuff
import string, types, math, re, sys, operator, os
```

```
#
# Get and set system paths
#

if( sys.path[0] == '' ): sys.path.append( sys.path[0] + './lmslib' )
else: sys.path.append( sys.path[0] + '/lmslib' )

# import LMS modules
# LMS stuff that Jim McCarter built.  Thanks Jim!!
from trefile2 import *
from sdbfile2 import *
import inifile
import lms2
#from sngfile2 import *

#
# Set some globals
#

# Variables
(FALSE, TRUE) = (0, 1)
GP = 0                # Growth period from hsi.ini
PM = ''               # Percent conifer/deciduous from hsi.ini
RunType = ''          # Run Type from hsi.ini
Output = ''           # Output table type
proc = ''          # Type of program run - determined by command line options
verbose = 0

# Initialize all the lists
Conifer = []                  # conifer species, get from hsi.ini
Deciduous = []                        # deciduous species, get dynamically (not Conifer)
CoverTypes = []        # Cover types from hsi.ini
TimberedTypes = []     # Timbered cover types from hsi.ini
TTAttList = []         # Timbered cover type attribute name list from hsi.ini
HabAttList = []        # Habitat attribute name list from hsi.ini
ModelList = []         # List of habitat model names from hsi.ini
stands = []            # List of all stands created dynamically
years = []             # List of all years created dynamically

# Initialize all the dictionaries
HSIModel = {}          # References text model names to funcitons defined late on
CTVar = {}             # CTVar[CT][var] - stores all cover type thresholds from hsi.ini
HEPData = {}           # HEPData[CT][var] - stores all data from original HEP by cover
type from hsi.ini
SppCT = {}             # SppCT[species][CT list] - stores list of applicable cover types for
model runs from hsi.ini
```

```
LMSData = {}          # LMSData[year][stand][var] - stores all dynamic data calculated
from LMS data
CType = {}            # CType[year][stand][var] - stores cover types
HSI = {}              # HSI[year][stand][species][var] - stores variables and HSI numbers
HU = {}               # HU[year][species][var] - stores habitat units, habitat acreages
AAHU = {}
HabAc = {}
Acres = {}            # Acres[stand] - stores acreage for each stand
Type = {}             # Type[stand] - stores cover type (timbered or other) from
FOLIONAME.sdb

#
# Define all the functions
#
#
# argument processor
#
def proc_args( args ):
   import getopt, string
   options = 'vhc'                                  # define options: v = verbose, h = habitat, c
= cover type
   optlist, arglist = getopt.getopt( args, options )
   print 'optlist = %s, arglist = %s' % (optlist, arglist)
   index = 0
   for item in arglist:                                  # look for response files
      if( item[0] == '@' ):                                  # process it
         f1 = open( item[1:], 'r' )                    # open file
         while 1:                                              # read file, passing each
            line = f1.readline()               # line to getopt()
            if not line: break
            roptlist, rarglist = getopt.getopt( string.split(line), options )
            ##print 'roptlist = ', roptlist
            if( len( roptlist ) > 0 ):
               n = len( optlist )
               optlist[n:n] = roptlist
            if( len(rarglist) > 0 ):
               n = len( arglist )
               arglist[n:n] = rarglist
         f1.close()
         del arglist[index]                              # delete @file from args
      index = index + 1

   #print 'optlist = ', optlist, ' arglist = ', arglist
   for item in optlist:
      if( item[0] == '-v' ): verbose = 1
      if( item[0] == '-h' ): proc = 'hab'          # Sets up for HSI run
```

```
        if( item[0] == '-c' ): proc = 'cover'          # Sets up for cover typing only
        #print proc
    return arglist, proc

#
# process HSI.ini file for conifer and selected species lists
#
def ProcessHSIINI( hsiinifile ):
    #print hsiinifile
    INI = inifile.INIFile( hsiinifile )
    global Conifer, CoverTypes, TimberedTypes, TTAttList, CTVar, PM, Output,
RunType, GP, ModelList, SppCt, HepData

    # Get data used by all runs
    # Get conifer species list
    Conifer = string.split( string.strip( INI.GetKeyValue( 'Conifer Species', 'Conifer' ) ) )

    CoverTypes = string.split( string.strip( INI.GetKeyValue( 'Cover Types', 'CTypeList' )
) )

    TimberedTypes = string.split( string.strip( INI.GetKeyValue( 'Timbered Types',
'TTypeList' ) ) )

    TTAttList = string.split( string.strip( INI.GetKeyValue( 'Timbered Type Attributes',
'TTAttList' ) ) )

    # Loop through to make dicirtonary of cover type threshold attributes
    for ct in TimberedTypes:
        ##print ct
        if ( not CTVar.has_key ( ct ) ): CTVar[ct] = {}
        for a in TTAttList:
            if ( not CTVar[ct].has_key ( a ) ): CTVar[ct][a] = {}
            ##print ct, a
            var = string.atof( INI.GetKeyValue( ct, a ) )
            CTVar[ct][a] = var

    # Get percent conifer/deciduous method
    PM = string.strip( INI.GetKeyValue( 'Percent Method', 'PM' ) )
    ##print PM

    # Get cover type output type
    ##print 'proc = ', proc
    if proc == 'cover':
        Output = string.strip( INI.GetKeyValue( 'HEP Output', 'HEPOut' ) )
        ##print Output
```

```
    # Get data for habitat runs only
    if proc == 'hab':
        RunType = string.strip( INI.GetKeyValue( 'Run Type', 'RunType' ) )
        ##print RunType

        #GP = string.atof( INI.GetKeyValue( 'Growth Period', 'GP' ) )
        ##print GP

        ModelList = string.split( INI.GetKeyValue( 'HSI Models', 'ModelList' ) )
        ##print ModelList

        # loop through and make a dictionary of cover types applicable for each model
        if RunType == 'ModDoc':
            r = 'DOC'
        else: r = 'HEP'
        for m in ModelList:
            if ( not SppCT.has_key( m ) ): HEPData[m] = {}
            l = string.split( INI.GetKeyValue( m, r ) )
            ##print m, r, l
            SppCT[m] = l

        HabAttList = string.split( INI.GetKeyValue( 'Habitat Attributes', 'HabAtt' ) )

        # Loop through to make dictionary of habitat attribute data
        for ct in CoverTypes:
            if ( not HEPData.has_key( ct ) ): HEPData[ct] = {}
            for h in HabAttList:
                if ( not HEPData[ct].has_key( h ) ): HEPData[ct][h] = {}
                var = string.atof( INI.GetKeyValue( ct, h ) )
                ##print ct, h, var
                HEPData[ct][h] = var

        Output = string.strip( INI.GetKeyValue( 'HSI Output', 'HSIOut' ) )

    #return Conifer, CoverTypes, TimberedTypes, TTAttList, CTVar, PM, Output,
RunType, GP, ModelList, SppCt, HepData


#
# Manage species list, dynamically update deciduous species (Deciduous)
# to include all species not in Conifer
#
def CheckSpecies( sppcode ):
    for s in Conifer:                                       # check confier species list
        if( sppcode == s ): return              # found it, done
```

```
   for s in Deciduous:                                    # check deciduous list
      if( sppcode == s ): return             # found it, done
   Deciduous.append( sppcode )                            # else add it to deciduous list

#
# Check for sppcode in list of confer species
#
def IsConifer( sppcode ):
   for s in Conifer:                                                  # loop through conifer
list
      if( sppcode == s ): return( TRUE )           # found it, TRUE
   return( FALSE )                                                    # not found,
FALSE

#
#
#
def CalcLMSData( y, s ):
   #print y, s
   TPA = TRE.Count()
   CTPA = TRE.Count( Conifer )
   BT = TRE.Count( 'ALL', gt=20.9 )
   HT = TRE.AveHt( 'ALL', 40 )
   ADBH = TRE.AveDBH()
   OSDBH = TRE.AveDBH( 'ALL', 40 )
   layers = TRE.Layers()
   BA = TRE.SumBA( 'ALL' )
   CBA = 0
   for c in Conifer:
      CBA = CBA + TRE.SumBA( c )
   CC = 100 * ( 1.0 - math.exp( -0.01 * ( 100 * sumcc / 43560) ) )
   CCC = 100 * ( 1.0 - math.exp( -0.01 * (100 * sumccc / 43560) ) )

   if PM == 'T':
   #this does percentages by tpa
      if( TPA >0 ):
         PC = 100 * ( CTPA / TPA )
         PD = 100 * ( ( TPA - CTPA ) / TPA )
      else:
         PC = 0.0
         PD = 0.0

   if PM == 'B':
   #This does it by BA
      if( TPA > 0):
         PC = 100 * ( CBA / BA )
```

```
            PD = 100 * ( ( BA - CBA ) / BA )
        else:
            PC = 0.0
            PD = 0.0

    # Put everything in dictionary
    if ( not LMSData.has_key ( y ) ): LMSData[y] = {}
    if ( not LMSData[y].has_key ( s ) ): LMSData[y][s] = {}
    LMSData[y][s]['TPA'] = TPA
    LMSData[y][s]['CTPA'] = CTPA
    LMSData[y][s]['BT'] = BT
    LMSData[y][s]['HT'] = HT
    LMSData[y][s]['OSDBH'] = OSDBH
    LMSData[y][s]['ADBH'] = ADBH
    LMSData[y][s]['layers'] = layers
    LMSData[y][s]['BA'] = BA
    LMSData[y][s]['CC'] = CC
    LMSData[y][s]['CCC'] = CCC
    LMSData[y][s]['PC'] = PC
    LMSData[y][s]['PD'] = PD


def CHawk( y, s ):
    #print 'Running CHawk'
    # Get variables
    CT = CType[y][s]
    CTList = SppCT['CHawk']
    acres = Acres[s]

    #print CTList

    if operator.contains( CTList, CT ):
        #print CT
        if RunType != 'HepData':
            CC = LMSData[y][s]['CC']
            OSDBH = LMSData[y][s]['OSDBH']
            CCC = LMSData[y][s]['CCC']
            if CT == 'PF':
                CC = HEPData[CT]['CC']
                OSDBH = HEPData[CT]['OSDBH']
                CCC = HEPData[CT]['CCC']
        else:
            CC = HEPData[CT]['CC']
            OSDBH = HEPData[CT]['OSDBH']
            CCC = HEPData[CT]['CCC']
        #print CT, CC, OSDBH, CCC
```

```
        #print y, s, CC, OSDBH, CCC
        #Variable 1 % canopy closure
        if(CC <= 60): V1 = (CC / 60.0)
        else: V1 = 1

        #Variable 2 overstory size class.  Uses average DBH of top 40 trees
        if(OSDBH < 6): V2 = 0.2
        elif((OSDBH >=6) & (OSDBH < 10)): V2 = 0.6
        elif((OSDBH >=10) & (OSDBH <20)): V2 = 0.9
        else: V2 = 1

        #Variable 3 % conifer canopy closure
        if(CCC <= 10): V3 = 0.8 + ((0.2 * CCC) / 10.0)
        elif((CCC > 10) & (CCC < 30)): V3 = 1
        elif((CCC >= 30) & (CCC < 80)): V3 = 1 - ((0.8 / 50.0) * (CCC - 30.0))
        else: V3 = 0.2

    else:
        V1 = V2 = V3 = 0

    #Compute HSI
    hsi1 = (V1 * V2) ** (1.0 / 2.0)
    hsi2 = V3
    hsi = min (hsi1, hsi2)

    #Compute HU
    HU = hsi * acres

    # Put data into dictionaries
    if not HSI.has_key( y ): HSI[y] = {}
    if not HSI[y].has_key( s ): HSI[y][s] = {}
    if not HSI[y][s].has_key( 'CHawk' ): HSI[y][s]['CHawk'] = {}
    HSI[y][s]['CHawk']['V1'] = V1
    HSI[y][s]['CHawk']['V2'] = V2
    HSI[y][s]['CHawk']['V3'] = V3
    HSI[y][s]['CHawk']['hsi1'] = hsi1
    HSI[y][s]['CHawk']['hsi2'] = hsi2
    HSI[y][s]['CHawk']['hsi'] = hsi
    HSI[y][s]['CHawk']['HU'] = HU


def SRVole( y, s ):
    ##print 'Running SRVole'
    ##print y, s
    # Get variables
```

```
CT = CType[y][s]
CTList = SppCT['SRVole']
acres = Acres[s]
if operator.contains( CTList, CT ):
   DnF = HEPData[CT]['DnF']
   GR = HEPData[CT]['GR']
   if RunType != 'HepData':
      OSDBH = LMSData[y][s]['OSDBH']
      CCC = LMSData[y][s]['CCC']
   else:
      OSDBH = HEPData[CT]['OSDBH']
      CCC = HEPData[CT]['CCC']
   if(OSDBH <= 12): V1 = (1.0 / 12.0) * OSDBH
   else: V1 = 1.0
   #Variable 2: % downfall ground cover from HEP average
   if(DnF <= 20): V2 = (1.0 / 20.0) * DnF
   else: V2 = 1.0
   #Variable 3: % grass cover from HEP average
   if(GR < 10): V3 = 1
   elif((GR >=10) & (GR <= 80)): V3 = 1 - ((1 / 70) * (GR - 10))
   else: V3 = 0
   #Variable 4: % conifer canopy closure
   if(CCC <= 20): V4 = 0.05 + ((0.05 / 20) * CCC)
   elif((CCC > 20) & (CCC <= 50)): V4 = 0.1 + ((0.9 / 30) * (CCC - 20))
   else: V4 = 1.0

else:
   V1 = V2 = V3 = V4 = 0

#Compute HSI
hsi = ((V1 * V2 * V3) ** (1.0 / 3.0)) * (V4)

#Compute HU
HU = hsi * acres

if not HSI.has_key( y ): HSI[y] = {}
if not HSI[y].has_key( s ): HSI[y][s] = {}
if not HSI[y][s].has_key( 'SRVole' ): HSI[y][s]['SRVole'] = {}
HSI[y][s]['SRVole']['V1'] = V1
HSI[y][s]['SRVole']['V2'] = V2
HSI[y][s]['SRVole']['V3'] = V3
HSI[y][s]['SRVole']['V4'] = V4
HSI[y][s]['SRVole']['hsi'] = hsi
HSI[y][s]['SRVole']['HU'] = HU
```

```
def PWoodpecker( y, s ):
   ##print 'Running PWoodpecker'
   # Get varialbes
   CT = CType[y][s]
   CTList = SppCT['PWoodpecker']
   acres = Acres[s]

   if operator.contains( CTList, CT ):
      Stp = HEPData[CT]['Stp']
      L7 = HEPData[CT]['L7']
      BSn = HEPData[CT]['BSn']
      DBHBSn = HEPData[CT]['DBHBSn']
      if RunType != 'HepData':
         CC = LMSData[y][s]['CC']
         BT = LMSData[y][s]['BT']
         if CT == 'PF':
            CC = HEPData[CT]['CC']
            BT = HEPData[CT]['BT']
      else:
         CC = HEPData[CT]['CC']
         BT = HEPData[CT]['BT']
      #Variable 1 % canopy closure
      if(CC < 25): V1 = 0
      elif((CC >=25) & (CC <=75)): V1 = (1.0 / 50.0) * (CC - 25)
      else: V1 = 1
      #Variable 2: TPA > 20" dbh
      if(BT < 3): V2 = 0
      elif((BT >=3) & (BT <= 30)): V2 = (1.0 / 27.0) * (BT - 3)
      else: V2 = 1
      #Variable 3: # stumps > 1' tall / acre (data from HEP)
      dlgst = ( Stp + L7 )
      if(dlgst <= 10): V3 = 0.3 + (0.7 / 10.0) * dlgst
      elif(dlgst > 10): V3 = 1
      else: V3 = 'ERR'
      #Variable 6: # snags/ac >20" dbh (from HEP data)
      if(BSn <= 0.17): V6 = (1.0 / 0.17) * BSn
      elif(BSn > 0.17): V6 = 1
      else: V6 = 'ERR'
      #Variable 7: Ave dbh of snags >20" (from HEP data)
      if((DBHBSn >= 20) & (DBHBSn <= 30)): V7 = 0.25 + (0.75  / 10.0) * (DBHBSn -
20)
      elif(DBHBSn > 30): V7 = 1
      elif(DBHBSn < 20): V7 = 0
      else: V7 = 'ERR'

   else: V1 = V2 = V3 = V6 = V7 = 0
```

```
    #Compute HSI
    hsi1 = (V1 * V2 * V3) ** (1.0 / 2.0)
    hsi2 = (V6 * V7) ** (1.0 / 2.0)
    hsi = min (hsi1, hsi2)

    #Compute HU
    HU = hsi * acres

    # Put data into dictionaries
    if not HSI.has_key( y ): HSI[y] = {}
    if not HSI[y].has_key( s ): HSI[y][s] = {}
    if not HSI[y][s].has_key( 'PWoodpecker' ): HSI[y][s]['PWoodpecker'] = {}
    HSI[y][s]['PWoodpecker']['V1'] = V1
    HSI[y][s]['PWoodpecker']['V2'] = V2
    HSI[y][s]['PWoodpecker']['V3'] = V3
    HSI[y][s]['PWoodpecker']['V6'] = V6
    HSI[y][s]['PWoodpecker']['V7'] = V7
    HSI[y][s]['PWoodpecker']['hsi1'] = hsi1
    HSI[y][s]['PWoodpecker']['hsi2'] = hsi2
    HSI[y][s]['PWoodpecker']['hsi'] = hsi
    HSI[y][s]['PWoodpecker']['HU'] = HU

def STowhee( y, s ):
    ##print 'Running STowhee'
    # Get variables
    CT = CType[y][s]
    CTList = SppCT['STowhee']
    acres = Acres[s]

    if operator.contains( CTList, CT ):
        TGC = HEPData[CT]['TGC']
        SSI = HEPData[CT]['SSI']
        if RunType != 'HepData':
            CC = LMSData[y][s]['CC']
            if CT == 'PF' or CT == 'PE':
                CC = HEPData[CT]['CC']
        else:
            CC = HEPData[CT]['CC']
        #Variable 1: % ground cover (from HEP data)
        if( TGC < 50): V1 = (1.0 / 50.0) * TGC
        elif((TGC >= 50) & (TGC <= 90)): V1 = 1
        elif(TGC > 90): V1 = 1 - (0.5 / 10.0) * (TGC - 90)
        else: V1 = 'ERR'
        #Variable 2: Shrub SI (from HEP data)
        V2 = SSI
```

```
        #Variable 3: Tree canopy cover
        if(CC < 12): V3 = (0.042 / 12.0) * CC
        elif( ( CC >= 12 ) & ( CC < 22 ) ): V3 = 0.042 + ( 0.1 / 10.0 ) * ( CC - 12 )
        elif((CC >= 22) & (CC <= 60)): V3 = 0.12 + (0.88 / 40) * (CC - 22)
        elif((CC > 60) & (CC < 75)): V3 = 1
        elif((CC >= 75) & (CC <= 85)): V3 = 1- (0.20 / 10.0) * (CC - 75)
        elif((CC > 85) & (CC < 95)): V3 = 0.8 - (0.6 / 10.0) * (CC - 85)
        elif(CC > 95): V3 = 0.2 - ( (0.017 / 5.0) * (CC - 95) )
        else: V3 = 'ERR'
    else:
        V1 = V2 = V3 = 0

    #Compute HSI
    hsi1 = V1
    hsi2 = V2
    hsi3 = (V2 + V3) / 2.0
    hsi = min (hsi1, hsi2, hsi3)

    #Compute HU
    HU = hsi * acres

    if not HSI.has_key( y ): HSI[y] = {}
    if not HSI[y].has_key( s ): HSI[y][s] = {}
    if not HSI[y][s].has_key( 'STowhee' ): HSI[y][s]['STowhee'] = {}

    HSI[y][s]['V1'] = V1
    HSI[y][s]['V2'] = V2
    HSI[y][s]['V3'] = V3
    HSI[y][s]['hsi1'] = hsi1
    HSI[y][s]['hsi2'] = hsi2
    HSI[y][s]['hsi3'] = hsi3
    HSI[y][s]['hsi'] = hsi
    HSI[y][s]['HU'] = HU

    HSI[y][s]['STowhee']['V1'] = V1
    HSI[y][s]['STowhee']['V2'] = V2
    HSI[y][s]['STowhee']['V3'] = V3
    HSI[y][s]['STowhee']['hsi1'] = hsi1
    HSI[y][s]['STowhee']['hsi2'] = hsi2
    HSI[y][s]['STowhee']['hsi3'] = hsi3
    HSI[y][s]['STowhee']['hsi'] = hsi
    HSI[y][s]['STowhee']['HU'] = HU

def StoreHUData( y, ct, m, h, ha, hu ):
    HU[y][m] = h
    HabAc[y][m] = ha
```

```
    ##print y, s, cha
    sumh = CTSum[y][ct][m]
    if sumh == 0:
        sumh = hu
    else: sumh = sumh + hu
    CTSum[y][ct][m] = sumh

#
# figure out cover type (timbered or not) from sdbfile
#

# Run this guy in Cover Typing stuff
def TypeCode(typecode):
    ###print typecode
    if(typecode == 0):ctype = 'T'
    elif(typecode == 1):ctype = 'B'
    elif(typecode == 2):ctype = 'G'
    elif(typecode == 3):ctype = 'PF'
    elif(typecode == 4):ctype = 'PE'
    #elif(typecode == 5):ctype = 'PS'
    else: ctype = 'UNK'
    ###print typecode, ctype
    return(ctype)

#
# figure out HEPcode from variables
#

def CoverType( y, s ):
    # Get thresholds
    C4CC = CTVar['C4']['CCVar']
    C4PC = CTVar['C4']['PCVar']
    C4HT = CTVar['C4']['HTVar']
    C4BT = CTVar['C4']['BTVar']
    C4layers = CTVar['C4']['LVar']
    C4TCC = CTVar['C4T']['CCVar']
    C4TPC = CTVar['C4T']['PCVar']
    C4THT = CTVar['C4T']['HTVar']
    C4TMinDBH = CTVar['C4T']['MinDBHVar']
    C4TMaxDBH = CTVar['C4T']['MaxDBHVar']
    C3CC = CTVar['C3']['CCVar']
    C3PC = CTVar['C3']['PCVar']
    C3HT = CTVar['C3']['HTVar']
    C3MinDBH = CTVar['C3']['MinDBHVar']
    C3MaxDBH = CTVar['C3']['MaxDBHVar']
    C3TCC = CTVar['C3T']['CCVar']
```

```
C3TPC = CTVar['C3T']['PCVar']
C3TMinDBH = CTVar['C3T']['MinDBHVar']
C3TMaxDBH = CTVar['C3T']['MaxDBHVar']
C2CC = CTVar['C2']['CCVar']
C2PC = CTVar['C2']['PCVar']
C2MinDBH = CTVar['C2']['MinDBHVar']
C2MaxDBH = CTVar['C2']['MaxDBHVar']
C1CC = CTVar['C1']['CCVar']
C1PC = CTVar['C1']['PCVar']
C1MinDBH = CTVar['C1']['MinDBHVar']
C1MaxDBH = CTVar['C1']['MaxDBHVar']
C1TPA = CTVar['C1']['TPAVar']
M3CC = CTVar['M3']['CCVar']
M3PC = CTVar['M3']['PCVar']
M3PD = CTVar['M3']['PDVar']
M3HT = CTVar['M3']['HTVar']
M3MinDBH = CTVar['M3']['MinDBHVar']
M3MaxDBH = CTVar['M3']['MaxDBHVar']
M3TCC = CTVar['M3T']['CCVar']
M3TPC = CTVar['M3T']['PCVar']
M3TPD = CTVar['M3T']['PDVar']
M3THT = CTVar['M3T']['HTVar']
M3TMinDBH = CTVar['M3T']['MinDBHVar']
M3TMaxDBH = CTVar['M3T']['MaxDBHVar']
M2CC = CTVar['M2']['CCVar']
M2PC = CTVar['M2']['PCVar']
M2PD = CTVar['M2']['PDVar']
M2MinDBH = CTVar['M2']['MinDBHVar']
M2MaxDBH = CTVar['M2']['MaxDBHVar']
M1CC = CTVar['M1']['CCVar']
M1PC = CTVar['M1']['PCVar']
M1PD = CTVar['M1']['PDVar']
M1MinDBH = CTVar['M1']['MinDBHVar']
M1MaxDBH = CTVar['M1']['MaxDBHVar']
M1TPA = CTVar['M1']['TPAVar']
H3CC = CTVar['H3']['CCVar']
H3PD = CTVar['H3']['PDVar']
H3HT = CTVar['H3']['HTVar']
H3MinDBH = CTVar['H3']['MinDBHVar']
H3MaxDBH = CTVar['H3']['MaxDBHVar']
H2CC = CTVar['H2']['CCVar']
H2PD = CTVar['H2']['PDVar']
H2MinDBH = CTVar['H2']['MinDBHVar']
H2MaxDBH = CTVar['H2']['MaxDBHVar']
H1CC = CTVar['H1']['CCVar']
H1PD = CTVar['H1']['PDVar']
```

```
    H1MinDBH = CTVar['H1']['MinDBHVar']
    H1MaxDBH = CTVar['H1']['MaxDBHVar']
    BCC = CTVar['B']['CCVar']

    # Get data
    TPA = LMSData[y][s]['TPA']
    BT = LMSData[y][s]['BT']
    HT = LMSData[y][s]['HT']
    OSDBH = LMSData[y][s]['OSDBH']
    ADBH = LMSData[y][s]['ADBH']
    layers = LMSData[y][s]['layers']
    CC = LMSData[y][s]['CC']
    PC = LMSData[y][s]['PC']
    PD = LMSData[y][s]['PD']

    #print H3CC, H3PD, H3MinDBH, H3MaxDBH
    #print CC, PC, PD, ADBH, OSDBH, HT, BT, TPA, layers
    #HEP = 'UNK'
    if(   (CC >= C4CC) & (PC >= C4PC) & (HT > C4HT) & (BT >= C4BT) & ( layers >=
C4layers )    ): HEP = 'C4'
    #elif( (CC >= C4CC) & (PC >= C4PC) & (HT > C4HT) & (BT >= C4BT) & ( layers <
C4layers  )    ): HEP = 'C4T'
    elif( (PC >= C4TPC) & (HT > C4HT) & (OSDBH >= C4TMinDBH)                  ):
HEP = 'C4T'
    elif( (CC >= C3CC) & (PC >= C3PC) & (OSDBH >= C3MinDBH) & (OSDBH <
C3MaxDBH)                ): HEP = 'C3'
    elif( (CC < C3TCC) & (PC >= C3TPC) & (OSDBH >= C3TMinDBH) & (OSDBH <
C3TMaxDBH)             ): HEP = 'C3T'
    elif( (CC >= C2CC) & (PC >= C2PC) & (OSDBH >= C2MinDBH) & (OSDBH <
C2MaxDBH)                ): HEP = 'C2'
    #elif( (CC  < 50) & (PC >= 75) & (adbh >=  4) & (adbh < 12)                  ): HEP =
'C2T'
    elif(               (PC >= C1PC) & (OSDBH >= C1MinDBH) & (OSDBH < C1MaxDBH)
& (TPA >= 150) ): HEP = 'C1'
    elif( (CC >= M3CC) & (PC < M3PC) & (PD < M3PD) & (HT >= M3HT) & (OSDBH
>= M3MinDBH) & (OSDBH < M3MaxDBH) ): HEP = 'M3'
    #removed max diameter of 21 inches on M3 and M3T
    elif( (CC < M3TCC) & (PC < M3TPC) & (PD < M3TPD) & (HT >= M3THT) &
(OSDBH >= M3TMinDBH) & (OSDBH < M3TMaxDBH) ): HEP = 'M3T'
    elif( (CC >= M2CC) & (PC < M2PC) & (PD < M2PD) & (OSDBH >= M2MinDBH)
& (OSDBH < M2MaxDBH) ): HEP = 'M2'
    elif(             (PC < M1PC ) & (PD < M1PD) & (OSDBH >= M1MinDBH) &
(OSDBH < M1MaxDBH ) & (TPA >= 150) ): HEP = 'M1'
    elif( (CC >= H3CC) & (PD >= H3PD) & (OSDBH > H3MinDBH) & (OSDBH <=
H3MaxDBH)             ): HEP = 'H3'
```

```
   elif( (CC >= H2CC) & (PD >= H2PD) & (OSDBH >= H2MinDBH) & (OSDBH <=
H2MaxDBH) ): HEP = 'H2'
   elif( (CC >= H1CC) & (PD >= H1PD) & (OSDBH >= H1MinDBH) & (OSDBH <
H1MaxDBH)              ): HEP = 'H1'
   elif( (CC < BCC) ): HEP = 'B'
   else: HEP = 'UNK'
   #print y, s, TPA, BT, HT, OSDBH, ADBH, layers, CC, PC, PD, HEP
   return( HEP )

def seperator():
   fout.write('\n--------------------\n\n')


#
# begin main program
#

(result, proc) = proc_args( sys.argv[1:] )
#print result

hsiinifile = result[0]
sdbfile = result[1]
invfile = result[2]
outfile = result[3]

#Conifer = ( 'DF', 'WH', 'RC' )                  # conifer species
#sspp = [ 'DF' ]                                 # selected species list
#print 'Processing INIfile'
ProcessHSIINI( hsiinifile )             # process HSI.INI for conifer and selected spp lists

lastyear = 0
laststand = 0

#
# Set up model dictionary
#

HSIModel['CHawk'] = CHawk
HSIModel['SRVole'] = SRVole
HSIModel['PWoodpecker'] = PWoodpecker
HSIModel['STowhee'] = STowhee


#
# process sdb file
#
```

```
#print 'Processing SDBfile'
f1 = open( sdbfile, 'r' )

while 1:
   line = f1.readline()
   ##print line
   if not line: break
   if line[0] == ';':
      continue
   field = string.splitfields( line, ',' )   # comma separated
   s = string.strip( field[0] )
   typecode  = string.atof( field[9] )
   acres = string.atof( field[10] )

   # Put data in dictionary
   Acres[s] = acres
   Type[s] = typecode

f1.close()


#
# process inv file
#

TRE = TREFile( invfile )                  # open invfile
#print 'Processing INVfile'
sumcc = sumccc = 0.0
while 1:                                  # loop forever
   line = TRE.ReadLine()                  # read line file file
   if not line: break                     # if end of file, break loop
##print line
   if line[0] ==';': continue
#else: break
   year = string.atoi(TRE.year)
   if not operator.contains( years, year ):
      years.append( year )
   stand = TRE.stand

   # Create stand list
   #print stands
   if not operator.contains( stands, stand ):
      stands.append( stand )

   spp = TRE.spp
   tpa = TRE.tpa
```

```
   mcw = TRE.mcw
   ht = TRE.ht
   if( (year != lastyear) | (stand != laststand) ):
      if( (lastyear != 0) & (laststand != 0) ):
         CalcLMSData( lastyear, laststand )
#
      laststand = stand
      lastyear = year
      TRE.Clear()
      sumcc = sumccc = 0.0
      TRE.AddRecord()
      CheckSpecies( spp )
      if( mcw == 0.0 ): radius = ht * 0.33 / 2.0
      else: radius = mcw / 2.0
      sumcc = sumcc + (tpa * (3.141592654 * (radius*radius)))
      if( IsConifer( spp ) ): sumccc = sumccc + (tpa * (3.141592654 * (radius*radius)))

   else:                                   # accumulate information about stand
      TRE.AddRecord()
      CheckSpecies( spp )
      if( mcw == 0.0 ): radius = ht * 0.33 / 2.0
      else: radius = mcw / 2.0
      sumcc = sumcc + (tpa * (3.141592654 * (radius*radius)))
      if( IsConifer( spp ) ): sumccc = sumccc + (tpa * (3.141592654 * (radius*radius)))

CalcLMSData( year, stand )        # add statistics for last stand

# Figure out cover types
#print 'Cover typing'
CTSum = {}
for y in years:
   if not CTSum.has_key( y ): CTSum[y] = {}
   for c in CoverTypes:
      if not CTSum[y].has_key( c ): CTSum[y][c] = {}
      CTSum[y][c]['acres'] = 0
for y in years:
   #C4ac = C4Tac = C3ac = C3Tac = C2ac = C1ac = M3ac = M3Tac = M2ac = M1ac =
H3ac =H2ac = H1ac = Bac = Gac = PFac = PEac = 0
   if not CType.has_key( y ): CType[y] = {}
   for s in stands:
      if not CType[y].has_key( s ): CType[y][s] = {}
      acres = Acres[s]
      t = Type[s]
      #print s, t
      if t == 0:
         ct = CoverType( y, s )
```

```
        else:
            ct = TypeCode( t )
        CType[y][s] = ct
        ctac = CTSum[y][ct]['acres']
        if ctac == 0:
            ctac = ctac + acres
        else:
            ctac = ctac + acres
        CTSum[y][ct]['acres'] = ctac


if proc == 'hab':
    # Run HSI functions

    for y in years:
        for s in stands:
            for m in ModelList:
                ####print m
                HSIModel[ m ]( y, s )

    # Summarize and calculate HU's and AAHU's
    #print 'Summarizing HSI'
    for y in years:
        if not HU.has_key( y ): HU[y] = {}
        if not HabAc.has_key( y ): HabAc[y] = {}
        for m in ModelList:
            if not HU[y].has_key( m ): HU[y][m] = {}
            HU[y][m] = 0
            for c in CoverTypes:
                CTSum[y][c][m] = 0
        #print y
    for y in years:
        for m in ModelList:
            u = a = 0
            #ch = srv = st = pw = 0.0
            #cha = srva = sta = pwa = 0.0
            #c4h = c4th = c3h = c3th = c2h = c1h = m3h = m3th = m2h = m1h = h3h = h2h =
h1h = bh = gh = pfh = peh = 0.0
            #hu = a = 0
            for s in stands:
                ac = Acres[s]
                ct = CType[y][s]
                hu = HSI[y][s][m]['HU']
                #if m == 'CHawk':
                #print 'summing '+m
                #Sum Cooper's hawk HU's
```

```
            if(u == 0): u = hu
            else: u = u + hu
            if operator.contains( SppCT[m], ct ):
               if( a == 0 ): a = ac
               else: a = a + ac
            StoreHUData( y, ct, m, u, a, hu )



   #Calculate AAHU's
   FirstYear = min(years)
   LastYear = max(years)
   PlanLife = LastYear - FirstYear
   if len( years ) > 1:
      GrPer = ( years[1] - years[0] )
      cycles = (LastYear - FirstYear) / GrPer
   else: cycles = 1
   for m in ModelList:
      tothu = 0
      for y in years:
         hu = HU[y][m]
         if tothu == 0:
            tothu = hu
         else: tothu = tothu + hu
         aahu = tothu / cycles
      AAHU[m] = aahu


#
# write output file
#
fout = open( outfile, 'w' )                  # open output file

# Cover type output
if proc == 'cover':
   if Output == 'DEBUG':
      attlist = [ 'CC', 'PC', 'PD', 'HT', 'OSDBH', 'ADBH', 'BT', 'layers', 'TPA' ]
      fout.write( 'Year, Stand, Acres, CType' )
      for a in attlist:
         fout.write( ', %s' % ( a ) )
      fout.write(' \n' )
      for y in years:
         for s in stands:
            ac = Acres[s]
            c = CType[y][s]
            fout.write( '%s, """%s""", %s, %s' % ( y, s, ac, c ) )
```

```
                for a in attlist:
                    fout.write( ', %0.2f' % ( LMSData[y][s][a] ) )
                fout.write( '\n' )

        if Output == 'STD' or Output == 'ALL':
            fout.write( 'Year, Stand, Acres, CType\n' )
            for y in years:
                for s in stands:
                    ac = Acres[s]
                    c = CType[y][s]
                    ##print y, s, ac, c
                    fout.write( '%s, """%s""", %s, %s\n' % ( y, s, ac, c ) )

        if Output == 'ALL':
            seperator()

        if Output == 'SUM' or Output == 'ALL':
            fout.write( 'Cover_Type_Summary\n' )
            fout.write( 'CType' )
            for y in years:
                fout.write(', %s_ac' % ( y ) )
            fout.write( '\n' )
            for c in CoverTypes:
                fout.write( '%s' % ( c ) )
                for y in years:
                    ac = CTSum[y][c]['acres']
                    fout.write( ', %0.1f' % ( ac ) )
                fout.write( '\n' )

        if Output == 'AV':
            fout.write( 'Stand')
            for y in years:
                fout.write( ', %s_CT' % ( y ) )
            fout.write( '\n' )
            for s in stands:
                fout.write( """%s""" % ( s ) )
                for y in years:
                    c = CType[y][s]
                    fout.write( ', %s' % ( c ) )
                fout.write( '\n' )

    if proc == 'hab':
        if Output == 'DEBUG':
            for m in ModelList:
                if m == 'CHawk':
                    fout.write( m +'\n' )
```

```
            fout.write('Year, Stand, CT, CC, OSDBH, CCC, V1, V2, V3, HSI1, HSI2,
HSI\n')
            for y in years:
               for s in stands:
                  ct = CType[y][s]
                  if RunType == 'HepData':
                     cc = HEPData[ct]['CC']
                     dbh = HEPData[ct]['OSDBH']
                     ccc = HEPData[ct]['CCC']
                  else:
                     cc = LMSData[y][s]['CC']
                     dbh = LMSData[y][s]['OSDBH']
                     ccc = LMSData[y][s]['CCC']
                  v1 = HSI[y][s][m]['V1']
                  v2 = HSI[y][s][m]['V2']
                  v3 = HSI[y][s][m]['V3']
                  hsi1 = HSI[y][s][m]['hsi1']
                  hsi2 = HSI[y][s][m]['hsi2']
                  hsi = HSI[y][s][m]['hsi']
                  fout.write( '%s, """%s""", %s, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f,
%0.3f, %0.3f, %0.3f\n' % ( y, s, ct, cc, dbh, ccc, v1, v2, v3, hsi1, hsi2, hsi ) )
            fout.write( '\n' )

         elif m == 'SRVole':
            fout.write( m +'\n' )
            fout.write( 'Year, Stand, CT, OSDBH, DnF, GR, CCC, V1, V2, V3, V4, HSI\n'
)
            for y in years:
               for s in stands:
                  ct = CType[y][s]
                  if RunType == 'HepData':
                     dbh = HEPData[ct]['OSDBH']
                     ccc = HEPData[ct]['CCC']
                  else:
                     dbh = LMSData[y][s]['OSDBH']
                     ccc = LMSData[y][s]['CCC']
                  dnf = HEPData[ct]['DnF']
                  gr = HEPData[ct]['GR']
                  v1 = HSI[y][s][m]['V1']
                  v2 = HSI[y][s][m]['V2']
                  v3 = HSI[y][s][m]['V3']
                  v4 = HSI[y][s][m]['V4']
                  hsi = HSI[y][s][m]['hsi']
                  fout.write( '%s, """%s""", %s, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f,
%0.3f, %0.3f, %0.3f\n' % ( y, s, ct, dbh, dnf, gr, ccc, v1, v2, v3, v4, hsi ) )
            fout.write( '\n' )
```

```
        elif m == 'PWoodpecker':
            fout.write( m +'\n' )
            fout.write( 'Year, Stand, CT, CC, BT, Stp, L7, BSn, DBHDSn, V1, V2, V3, V6,
V7, HSI1, HSI2, HSI\n' )
            for y in years:
                for s in stands:
                    ct = CType[y][s]
                    if RunType == 'HepData':
                        cc = HEPData[ct]['CC']
                        bt = HEPData[ct]['BT']
                    else:
                        cc = LMSData[y][s]['CC']
                        bt = LMSData[y][s]['BT']
                    stp = HEPData[ct]['Stp']
                    l7 = HEPData[ct]['L7']
                    bsn = HEPData[ct]['BSn']
                    dbhsn = HEPData[ct]['DBHBSn']
                    v1 = HSI[y][s][m]['V1']
                    v2 = HSI[y][s][m]['V2']
                    v3 = HSI[y][s][m]['V3']
                    v6 = HSI[y][s][m]['V6']
                    v7 = HSI[y][s][m]['V7']
                    hsi1 = HSI[y][s][m]['hsi1']
                    hsi2 = HSI[y][s][m]['hsi2']
                    hsi = HSI[y][s][m]['hsi']
                    fout.write( '%s, """%s""", %s, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f,
%0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f\n'
                                % ( y, s, ct, cc, bt, stp, l7, bsn, dbhsn, v1, v2, v3, v6, v7, hsi1, hsi2,
hsi ) )
            fout.write( '\n' )

        elif m == 'STowhee':
            fout.write( m +'\n' )
            fout.write( 'Year, Stand, CT, TGC, SSI, CC, V1, V2, V3, HSI1, HSI1, HSI3,
HSI\n' )
            for y in years:
                for s in stands:
                    ct = CType[y][s]
                    if RunType == 'HepData':
                        cc = HEPData[ct]['CC']
                    else:
                        cc = LMSData[y][s]['CC']
                    tgc = HEPData[ct]['TGC']
                    ssi = HEPData[ct]['SSI']
                    v1 = HSI[y][s][m]['V1']
```

```
                    v2 = HSI[y][s][m]['V2']
                    v3 = HSI[y][s][m]['V3']
                    hsi1 = HSI[y][s][m]['hsi1']
                    hsi2 = HSI[y][s][m]['hsi2']
                    hsi3 = HSI[y][s][m]['hsi3']
                    hsi = HSI[y][s][m]['hsi']
                    fout.write( '%s, """%s""", %s, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f, %0.3f,
%0.3f, %0.3f, %0.3f, %0.3f\n'
                            % ( y, s, ct, tgc, ssi, cc, v1, v2, v3, hsi1, hsi2, hsi3, hsi ) )
            fout.write( '\n' )

    if Output == 'HSI':
        fout.write( 'Year, Stand, HEP, Acres')
        for m in ModelList:
            fout.write(', %s' % ( m ))
        fout.write('\n')
        # EXCEPT FROM HERE TO.....
        for y in years:
            for s in stands:
                (ct, a)  = (CType[y][s], Acres[s])
                fout.write('%s, """%s""", %s, %s' % ( y, s, ct, a))
                for m in ModelList:
                    if m == 'CHawk':
                        hsi = HSI[y][s][m]['hsi']
                        fout.write( ', %0.4f' % ( hsi ) )
                    if m == 'SRVole':
                        hsi = HSI[y][s][m]['hsi']
                        fout.write( ', %0.4f' % ( hsi ) )
                    if m == 'PWoodpecker':
                        hsi = HSI[y][s][m]['hsi']
                        fout.write( ', %0.4f' % ( hsi ) )
                    if m == 'STowhee':
                        hsi = HSI[y][s][m]['hsi']
                        fout.write( ', %0.4f' % ( hsi ) )
                fout.write( '\n' )

    if Output == 'BySpp':
        for m in ModelList:
            ctlist = SppCT[m]
            for y in years:
                ta = thu = 0
                fout.write( 'Year: %s, Species: %s\n' % ( y, m ) )
                fout.write( 'CType, Acres, HSI\n' )
                for c in ctlist:
                    a = CTSum[y][c]['acres']
                    if ta == 0:
```

```
                ta = a
            else: ta = ta + a
            hu = CTSum[y][c][m]
            if thu == 0:
                thu = hu
            else: thu = thu + hu
            if a != 0:
                hsi = hu / a
            else:
                hsi = 0
            fout.write( '%s, %0.1f, %0.3f\n' % ( c, a, hsi ) )
        if ta !=0:
            ohsi = thu / ta
        else: ohsi = 0
        fout.write( 'Overall: %s, %0.3f\n' % ( ta, ohsi ) )
        seperator()

if Output == 'ALL' or Output == 'HSISum':
    fout.write( 'Average_HSI\n' )
    for y in years:
        fout.write( 'Year: %s\n' % ( y ) )
        fout.write( 'Species, Acres, HSI\n' )
        for m in ModelList:
            ctlist = SppCT[m]
            ta = thu = 0
            for c in ctlist:
                a = CTSum[y][c]['acres']
                if ta == 0:
                    ta = a
                else: ta = ta + a
                hu = CTSum[y][c][m]
                if thu == 0:
                    thu = hu
                else: thu = thu + hu
                if a != 0:
                    hsi = hu / a
                else:
                    hsi = 0
            if ta != 0:
                ohsi = thu / ta
            else: ohsi = 0
            fout.write( '%s, %s, %0.3f\n' % ( m, ta, ohsi ) )
        seperator()

if Output == 'ALL' or Output == 'HU':
    fout.write( 'Habitat_Units\n' )
```

```
    fout.write('Year')
    for m in ModelList:
        fout.write(', %s' % ( m ))
    fout.write('\n')
    for y in years:
        fout.write('%s' % ( y ) )
        for m in ModelList:
            hu = HU[y][m]
            fout.write( ', %0.4f' % ( hu ) )
        fout.write( '\n' )

if Output == 'ALL':
    seperator()

if Output == 'ALL' or Output == 'AAHU':
    fout.write('Annual_Averge_Habitat_Units\n')
    c = 0
    for m in ModelList:
        if c == 0:
            fout.write( '%s' % ( m ) )
            c = c + 1
        else:
            fout.write( ', %s' % ( m ) )
    fout.write( '\n' )
    c = 0
    for m in ModelList:
        aahu = AAHU[m]
        if c == 0:
            fout.write( '%0.3f' % ( aahu ) )
            c = c + 1
        else: fout.write( ', %0.3f' % ( aahu ) )
    fout.write( '\n' )

if Output == 'AV':
    fout.write('Stand')
    for y in years:
        for m in ModelList:
            fout.write( ', %s_%s' % ( y, m ) )
    fout.write( '\n' )
    for s in stands:
        fout.write( """+s+""" )
        for y in years:
            for m in ModelList:
                hsi = HSI[y][s][m]['hsi']
                fout.write( ', %0.3f' % ( hsi ) )
        fout.write( '\n' )
```

```
TRE.CloseFile()
fout.close()
```