
Subject: % OPENW: Error opening file.

Posted by [BWood](#) on Thu, 26 Feb 2015 22:19:45 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi folks,

I'm attempting to run the LandTrendr model in IDL, and have hit a snag. I have virtually no IDL experience, but was told that since the code is already written and has been tested, I should have very few hiccups. I've made it through a good number of modules, but have hit a breaking point. I don't know if there is enough background provided here for any of you to help my get past this error, but any help you may be able to provide would be greatly appreciated.

Cheers.

This is the error message I am getting:

% OPENW: Error opening file. Unit: 100, File:

E:\034032\outputs\nbr\LT_v2.00_nbr_034032_eval01_2015-02-26T
20:31:39.00005310773531Z_vertyrs.bsq

The filename, directory name, or volume label syntax is incorrect.

And here is the code:

```
function process_tcbcd_chunks, run_params, progressbaryesno
```

```
  ;copy over just to make it work with historical code.
```

```
  image_info = run_params.image_info
```

```
  index = run_params.index ;string with name of index
```

```
  subset=run_params.subset
```

```
  mask_image = run_params.mask_image
```

```
  output_base = run_params.output_base
```

```
  kernelsize = run_params.kernelsize
```

```
  background_val = run_params.background_val
```

```
  skipfactor = run_params.skipfactor
```

```
  desawtooth_val = run_params.desawtooth_val
```

```
  pval = run_params.pval
```

```
  max_segments = run_params.max_segments
```

```
  fix_doy_effect = run_params.fix_doy_effect
```

```
  divisor =run_params.divisor
```

```
  minneeded = run_params.minneeded
```

```
  recovery_threshold=run_params.recovery_threshold
```

```
  distweightfactor = run_params.distweightfactor
```

```
  vertexcountovershoot = run_params.vertexcountovershoot
```

```
  bestmodelproportion = run_params.bestmodelproportion
```

```
  progressval = 0 ;set to record progress
```

```
  progressinc = 10 ;for the increment
```

```
  ;make a directory
```

```
  file_mkdir, file_dirname(output_base)
```

```

;tests
if vertexcountovershoot gt 3 then begin
    print, 'Vertexcountovershoot cannot exceed 3'
    return, -1
end
if bestmodelproportion gt 1 then begin
    print, 'bestmodelproportion cannot exceed 1. Should be in the 0.5 to 1.0 range.'
    return, -1
end

;first, check to see if the output image already exists. If so
; see if it has a save file to indicate that it was already in
; process, and needs to just be picked up again
diagfile = output_base+'_diag.sav'

if file_exists(diagfile) then begin
    print, 'This file has already had processing done on it'
    restore, diagfile

    image_info = diag_info.image_info
    index = diag_info.index
    mask_image = diag_info.mask_image
    output_image_group = diag_info.output_image_group

    n_chunks = diag_info.n_chunks
    chunks = diag_info.chunks
    current_chunk = diag_info.current_chunk
    pixels_per_chunk = diag_info.pixels_per_chunk
    kernelsize = diag_info.kernelsize

    background_val=diag_info.background_val
    skipfactor=diag_info.skipfactor
    desawtooth_val=diag_info.desawtooth_val
    pval=diag_info.pval
    max_segments=diag_info.max_segments
    ; normalize=diag_info.normalize
    fix_doy_effect=diag_info.fix_doy_effect
    divisor=diag_info.divisor
    recovery_threshold = diag_info.recovery_threshold
    distweightfactor = diag_info.distweightfactor
    vertexcountovershoot = diag_info.vertexcountovershoot
    bestmodelproportion = diag_info.bestmodelproportion
end else begin ;if this image has not been set up before, then
    ;get set up to do it.
    ;SET UP THE OUTPUT FILE AND CHUNK INFORMATION
    ;First, set up the processing chunks

    if n_elements(max_pixels_per_chunk) eq 0 then $

```

```

max_pixels_per_chunk = 500000l

;to get the pixel size of the image, assume that all are the same
if file_exists(image_info[0].image_file) eq 0 then begin
  print, "process_cm_biomass_image.pro: Image in
  print,"   image_list does not exist.  Failing."
  print,"   Image that was not found:
  print, image_info[0].image_file
end

mastersubset = subset
subset = mastersubset
zot_img, image_info[0].image_file, hdr, img, subset=subset, /hdronly
pixsize = hdr.pixsize

;now get the chunks
subset = mastersubset
ok = define_chunks3(subset, pixsize, max_pixels_per_chunk, kernelsize)
if ok.ok eq 0 then return, {ok:0}

;stop
chunks = ok.subsets
pixels_per_chunk = ok.pixels_per_chunk
n_chunks = n_elements(chunks)
current_chunk = 0      ;an index

;define the output images
;vertices:  the years of the vertices
;vertvals:  the values of the input band at those years
;mags:  the magnitude of the segment change between two vertices
;distrec:  three layer image
; 1:  largest single disturbance
; 2:  largest single recovery
; 3:  scaled ratio between disturbance and recovery
; -1000 is all recovery
; 0 is a balance
; 1000 is all disturbance

;fitted image
; Same number of years as all of the inputs,
; but with fitted values
;Stats image for entire fit
; P of f
; f_stat
; ms_regr
; ms_resid

output_image_base = {filename:"", n_layers:0, extension:"", layersize:0l, filesize:0ll, DATA:""}

```

```
output_image_group = replicate(output_image_base, 10)
output_image_group[0].extension = '_vertyrs.bsq'
output_image_group[0].n_layers = max_segments+1
output_image_group[0].DATA = "Vertex Year"
```

```
output_image_group[1].extension = '_vertvals.bsq'
output_image_group[1].n_layers = max_segments+1
output_image_group[1].DATA = "Vertex Spectral Value"
```

```
output_image_group[2].extension = '_mags.bsq'
output_image_group[2].n_layers = max_segments
output_image_group[2].DATA = "Segment Magnitude"
```

```
output_image_group[3].extension = '_durs.bsq'
output_image_group[3].n_layers = max_segments
output_image_group[3].DATA = "Segment Duration"
```

```
output_image_group[4].extension = '_distrec.bsq'
output_image_group[4].n_layers = 3
output_image_group[4].DATA = "Trajectory Disturbance/Recovery"
```

```
;6/27/08 for fitted image, the output will be the
; max of 1 image per year. If multiple images
; in a given year are provided with the idea of doing
; cloud-mosaicking, then we need to compensate for
; those doubled-images in stack.
```

```
years = image_info.year
un_years = fast_unique(years)
years = un_years[sort(un_years)]
```

```
output_image_group[5].extension = '_fitted.bsq'
output_image_group[5].n_layers = n_elements(years)
output_image_group[5].DATA = "Fitted Spectral Stack"
```

```
output_image_group[6].extension = '_stats.bsq'
output_image_group[6].n_layers = 10
output_image_group[6].DATA = "Segment Statistics"
```

```
output_image_group[7].extension = '_segmse.bsq'
output_image_group[7].n_layers = max_segments
output_image_group[7].DATA = "Segment MSE"
```

```
output_image_group[8].extension = '_source.bsq'
output_image_group[8].n_layers = n_elements(years)
output_image_group[8].DATA = "Source Spectral Stack"
```

```
output_image_group[9].extension = '_segmean.bsq'
output_image_group[9].n_layers = max_segments
output_image_group[9].DATA = "Segment Mean Spectral"

;this_tag = "_" + timestamp() + "_" + landtrendr_version()
for i = 0, n_elements(output_image_group)-1 do begin

    this_file = output_base + output_image_group[i].extension
    output_image_group[i].filename = this_file

    openw, un, output_image_group[i].filename, /get_lun
    n_output_layers = output_image_group[i].n_layers

    bytes_per_pixel = 2
    layersize = long(hdr.filesize[0]) * hdr.filesize[1] * bytes_per_pixel

    filesize = ulong64(layersize) * n_output_layers
```
