```
function get_gvalue, atom, a, path=path

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Looks up the gvalues given by Killen et al 2008. The function returns a
;; 2xn array with the velocities and the radiation pressure constant for the
;; species. Keywords can be used to get the g values for each line  which may
;; be used to calculate the emission.
;;
;; INPUTS:
;;    * a = distance from the sun (AU) -- must be a single value, not an array
;;    * atom = atom for which to look up g-values
;;
;; OUTPUTS
;;    * Function returns 2xn array with velocities and radiation pressure constant.
;;        units = km/s^2
;;    * lines = resonance transitions included
;;    * velocity = radial velocities (km/s)
;;    * gval = array containing g-value vs. velocity for each transition
;;
;;  Version History
;;    3.1: 10 May 2011
;;      * New way of saving g-values. Use set_up_gvals to save into structures
;;    2.1: 30 Jan 2009
;;      - added g-values for all species included in Killen et al. [C I, Ca I, Ca II,
;;         H I, He I, K I, Mg I, Mg II, Na I, OH, O I, S I
;;    2.0: original -- only looks up Na g values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (n_elements(a) GT 1) then stop
if (a EQ !null) then a = 1.
if (n_elements(path) EQ 0) then $
  path = !model.basepath + 'Work/AtomicData/g-values/' + atom
if ~(file_test(path)) then stop

files = file_search(path, '*.sav') & nf = n_elements(files)

gval = {species:atom, $
        a:a, $                        ;; AU
        wavelength:ptr_new(0), $  ;; Angstrom
        v:ptr_new(0), $             ;; km/s
        g:ptr_new(0), $             ;; photons cm^-2 s^-1
        radaccel:ptr_new(0)}      ;; km^-2 s^-1
case (nf) of
  0: begin
     *gval.v = [0., 1.]
     *gval.g = [0., 0.]
     print, 'g-values not found. Radiation acceleration = 0'
     end
  1: begin
```

```
     restore, files[0]
     *gval.wavelength = gvalue.wavelength
     *gval.v = *gvalue.v
     *gval.g = *gvalue.g * gvalue.a^2/a^2 ;; normalize to specified distance
     end
  else: begin
     lambda = fltarr(nf)
     vv = ptrarr(nf, /allocate)
     gg = ptrarr(nf, /allocate)
     for i=0,nf-1 do begin
       restore, files[i]
       lambda[i] = gvalue.wavelength
       *vv[i] = *gvalue.v
       *gg[i] = *gvalue.g * gvalue.a^2/a^2
     endfor

     ;; Test if all wavelengths are unique
     u = uniq(lambda, sort(lambda))
     if (n_elements(u) NE nf) then begin
       ;; Need to decide which to use
       stop
     endif
     *gval.wavelength = lambda

     ;; Get a common velocity axis
     allv = !null
     for i=0,nf-1 do allv = [allv, *vv[i]]
     allv = allv[sort(allv)]
     *gval.v = allv[uniq(allv)]
     *gval.g = fltarr(n_elements(*gval.v),nf)
     for i=0,nf-1 do (*gval.g)[*,i] = interpol(*gg[i], *vv[i], *gval.v)
     end
endcase

;; radpres_const = h/(m*lambda) * g
rr = !const.h / atomicmass(atom) / (*gval.wavelength*1e-8)
qq = 0.
for i=0,nf-1 do qq += rr[i]*(*gval.g)[*,i]*1e-5  ;; km s^-2
*gval.radaccel = qq

return, gval

end
```

```
;;; combine the two plasma files into a single file with the info I want
;;; These are not really valid near the moons, but are good for large scale
;;; clouds

restore, '$HOME/NeutralModel/modelpro/data/CAPSplasma/Enceladus.plasma.sav'
ltemp = mtorus

restore, '$HOME/NeutralModel/modelpro/data/CAPSplasma/SOI.plasma.sav'

Mtorus = lgrid
LatTorus = latgrid

t_etorus = interpol(t_e, ltemp, mtorus)
t_wtorus = interpol(t_i[*,1], ltemp, mtorus)
t_htorus = interpol(t_i[*,0], ltemp, mtorus)

n_ehotgrid = n_egrid * .2/70.
t_ehottorus = t_etorus * 12.5/1.5

save, Mtorus, LatTorus, N_egrid, n_hgrid, n_wgrid, t_etorus, t_wtorus, t_htorus, $
  n_ehotgrid, t_ehottorus, $
  file='$HOME/NeutralModel/modelpro/data/CAPSplasma/SaturnPlasma.sav'

end
```

```
pro SystemConstants, planet, SystemConsts, DipoleConsts

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version 2.0: 15 June 2010
;;   Creates the systemconsts and dipoleconsts structures from data stored
;;   in the !Planet system variables
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SystemConsts = {Planet:'', rPlan:0d, aPlan:0d, epsPlan:0d, $
  Objects:ptr_new(0), GM:ptr_new(0), radius:ptr_new(0), a:ptr_new(0), eps:ptr_new(0), $
  orbvel:ptr_new(0), period:ptr_new(0), orbrate:ptr_new(0)}

case strlowcase(planet) of
  'sun': plan = !sun
  'mercury': plan = !Mercury
  'venus': plan = !Venus
  'earth': plan = !Earth
  'mars': plan = !Mars
  'jupiter': plan = !Jupiter
  'saturn': plan = !Saturn
  'uranus': plan = !Uranus
  'neptune': plan = !Neptune
  'pluto': plan = !Pluto
endcase

SystemConsts.planet = plan.name
SystemConsts.rplan = plan.radius
SystemConsts.aplan = plan.a
SystemConsts.epsplan = plan.e

tt = tag_names(plan)
if (total(strcmp(tt, 'satellites', /fold))) then begin
  *SystemConsts.objects = [plan.name, plan.satellites]

  mm = [plan.mass, plan.msat]
  rr = [plan.radius, plan.rsat]
  tt = [0d, plan.orbsat*24.*3600.]
  vv = [0d, 2*!dpi*plan.asat*plan.radius/tt[1:*]]

  *SystemConsts.GM = -!const.G*mm*1e3/(plan.radius*1e5)^3
  *SystemConsts.radius = rr/plan.radius
  *SystemConsts.a = [0d, plan.asat]
  *SystemConsts.eps = [0d, plan.esat]
  *SystemConsts.period = tt
  *SystemConsts.orbvel = vv
  *SystemConsts.orbrate = [0d, 2*!dpi/tt[1:*]]
endif else begin
  *SystemConsts.objects = plan.name
  *SystemConsts.GM = -!const.G*plan.mass*1d3/(plan.radius*1d5)^3
  *SystemConsts.radius = 1d
  *SystemConsts.a = 0d
  *SystemConsts.eps = 0d
  *SystemConsts.period = 0d
  *SystemConsts.orbvel = 0d
  *SystemConsts.orbrate = 0d
endelse

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Read in the dipole constants
file = !model.basepath + 'Work/Data/PhysicalData/DipoleConstants.dat'
if ~file_test(file) then stop ;; file = (file_search('$HOME', 'DipoleConstants.dat'))[0]
readcol, /silent, file, delim=':', ob, mm, t, tdir, o, olon, olat, per, $
  format='A,D,D,D,D,D,D,D'
ob = strtrim(ob, 2)
q = (where(strcmp(planet, ob, /fold), nq))[0]
if (nq) $
  then DipoleConsts = {$
    strength:mm[q], $
    tilt:t[q]*!dtor, $
    lam3:tdir[q]*!dtor, $
    offset:o[q], $
    offlong:olon[q]*!dtor, $
    offlat:olat[q]*!dtor, $
    period:per[q], $
    magrat:2*!dpi/per[q]} $
   else DipoleConsts = -1

end
```

```
function determine_image_rotation, input, format

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; There are lots of ways to specify what the proper viewing geometry should be.
;; Need to break that down into three angles: rotations about the x, y, and z axis
;; in model coordinates.
;;
;; Tags that can be specified:
;;   a) SubObsLongitude, SubObsLatitude, PolarAngle
;;      * SubObsLong and Lat give the intersection point on the surface in
;;        longitude and latitude relative to the sub-solar point
;;      * On a planet, define the SubObsLongitude positive in the ccw direction when
;;        looking down from above.
;;        * long=0 -> looking down over sub-solar meridian
;;        * long=!pi/2 -> looking down over dusk meridian
;;        * long=!pi -> looking down over midnight meridian
;;        * long=3*!pi/2 -> looking down over dawn meridian
;;      * Latitude defined positive is north
;;        * lon = -!pi/2 -> looking down over south pole
;;        * lon = 0 -> looking down over equator
;;        * lon = !pi/2 -> looking down over north pole
;;   b) Observer (e.g. Earth, MESSENGER, ...), time
;;
;; Version History
;;   4.0: 25 Jan 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

tags = strlowcase(tag_names(format.geometry))

q0 = total(strmatch(tags, 'SubObsLongitude', /fold))
w0 = total(strmatch(tags, 'SubObsLatitude', /fold))

q1 = total(strmatch(tags, 'observer', /fold))
w1 = total(strmatch(tags, 'time', /fold))

case (1) of
  (q0+w0 EQ 2): begin
    pSun = [0.,-1.,0.]
    pObs = [sin(format.geometry.SubObsLongitude)*cos(format.geometry.SubObsLatitude), $
      -cos(format.geometry.SubObsLongitude)*cos(format.geometry.SubObsLatitude), $
      sin(format.geometry.SubObsLatitude)]
    end
  (q1+w1 EQ 2): begin
    relative_position, 0., 0., 0., format.geometry.observer, input.geometry.planet, $
      frame='J2000', pos=pObs, havetime=utc2et(time)
    relative_position, 0., 0., 0., 'Sun', input.geometry.planet, frame='J2000', $
      pos=pSun, havetime=utc2et(time)
    end
endcase
```

```

M = rotationmatrix(pSun, pObs) ;; This is the rotation in space relative to the
       ;; model coordinate system

;; Determine rotation of FOV -- rotation about new y-axis
q = rotation([0,0,0], [0,1.,0], format.geometry.PolarAngle, R=R)
M = R # M

return, M

end
```

```
function display_hull, pts

;; pts = an array of points to look at computed by results_voronoi
;;   (pts = *regions[i])

sz = size(pts)

hullfile = ('hull' + strint(round(random_nr(1)*1000000)) + '.dat')[0]
openw, lun, hullfile, /get_lun
printf, lun, sz[2]
printf, lun, sz[1]
printf, lun, transpose(pts)
free_lun, lun

spawn, 'qconvex s Fv TI hullpts.dat TO ' + hullfile
spawn, 'rm ' + hullfile

nfac = long(out[0])
facets = out[1:*]
if (n_elements(facets) NE nfac) then stop
connect = !null
for i=0,nfac-1 do begin
  w = long(strsplit(line, /extract)
  connect = [connect, w]
endfor

s0 = plot3d(pts[*,0], pts[*,1], pts[*,2], dimensions=[1000,1000], symbol='*', $
  linestyle=' ', /aspect_ratio, /aspect_z, /sym_filled)
s2 = polygon(pts[*,0], pts[*,1], pts[*,2], connectivity=connect, fill_color='blue', $
  fill_transparency=50, /data)

return, [s0, s1]

end
```

```
function display_model_image, result, savefile, brange=brange, log=log, _extra=e

if (n_elements(brange) NE 2) then $
  brange = minmax((*result.image)[where(*result.image NE 0)])
if (log EQ !null) then log = 0

xcyc, xc, yc

etags = (e NE !null) ? tag_names(e) : ''
rgb = (total(strcmp(etags, 'rgb_table', /fold))) ? e.rgb_table : 3
title = (total(strcmp(etags, 'title', /fold))) ? e.title : 'Image'
xtitle = (total(strcmp(etags, 'xtitle', /fold))) ? e.xtitle : 'Distance'
ytitle = (total(strcmp(etags, 'ytitle', /fold))) ? e.ytitle : 'Distance'
ztitle = (total(strcmp(etags, 'ztitle', /fold))) ? e.ztitle : 'Intensity'

if (log) $
  then im = bytscl(alog10(*result.image), alog10(brange[0]), alog10(brange[1])) $
  else im = bytscl(*result.image, brange[0], brange[1])

pp = image2(im, *result.xaxis, *result.zaxis, rgb_table=rgb, $
  dimensions=[800,600], location=[0,0], $
  position=[120,100,520,500], /dev, $
  font_size=20, title=title, xtitle=xtitle, ytitle=ytitle)
pp[0].refresh, /disable
p1 = plot(/overplot, xc, yc, thick=3, color='blue')
p2 = plotsquare2(minmax(*result.xaxis), minmax(*result.zaxis), thick=3)

pos = [550,140,600,460]
cb = colorbar2(pos, brange, log=log, rgb_table=rgb, thick=2, font_size=20, $
  title=ztitle)
pp[0].refresh

if (savefile NE !null) then pp[0].save, savefile, width=800
pp = [pp, p1, p2, cb]

return, pp

end
```

```
function emission_measure, atom, line, vy=vy, aplanet=aplanet, ee=ee

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes the emission measure for each packet. This is used in
;; line_of_sight, model_images, and density_track.
;;
;; Required parameters:
;;    * atom
;;    * line = vector of lines to compute emission for in Å
;; Optional depending on the emission type and line
;;    * vy = radial velocity relative to the sun
;;    * aplanet = heliocentric planet distance. If not specified, then resonant scattering
;;        is not computed
;;
;; Outputs:
;;   Function returns the emission measure per atom for the requested lines
;;   ee = resonant scattering emission measure for each line
;;
;; Version 2.0: 19 April 2010
;;    * written based on already existing method in line_of_sight and model_images.
;;    * need a new version to make sure things are done consistently.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

nl = n_elements(line)
doresscat = (n_elements(aplanet) EQ 1)
doeimp = 0

;; Correct for Na wavelength issues
if (atom EQ 'Na') then begin
  q = where(line EQ 5890, nq)
  if (nq EQ 1) then line[q] = 5891.
  q = where(line EQ 5896, nq)
  if (nq EQ 1) then line[q] = 5897.
endif

;; Resonant Scattering
if (doresscat) then begin
  q = get_gvalue(aplanet, atom, lines=ll, velocity=radvel, gval=gval)
  w = where(vy LT min(radvel), nw) & if (nw NE 0) then vy[w] = min(radvel)
  w = where(vy GT max(radvel), nw) & if (nw NE 0) then vy[w] = max(radvel)

  ee = fltarr(n_elements(vy),nl)
  for i=0,nl-1 do begin
    q = where(ll EQ line[i], nq)
    if (nq NE 1) $
      then print, 'Error: g-value not found for emission line ' + string(line[i]) $
      else ee[*,i] = interpol(gval[*,q], radvel, vy)/1e6
  endfor
  resscat = (nl EQ 1) ? ee : total(ee,2)
```

```
endif else resscat = 0.

;; Electron Impact
if (doeimp) then begin
  stop
endif else eimp = 0.

result = resscat + eimp
return, result

end
```

```
pro model_view, image, x0, b0, b1

sz = (size(image))[1:2]

xc = cos(findgen(361)*!dtor) & yc = sin(findgen(361)*!dtor)
plot, findgen(10), /nodata, xr=minmax(x0), yr=minmax(x0), /xst, /yst, $
  xtit='Distance (R!dObj!n)', ytit='Distance (R!dObj!n)', $
  pos=[100,100,100+sz[0],100+sz[1]], /dev
tv, bytscl(image, b0, b1, top=220)+35, 100, 100, /dev
polyfill, xc, yc, color=4
plots, [100,100,100+sz[0],100+sz[0],100], [100,100+sz[1],100+sz[1],100,100], /dev

end
```

```
function produce_density, files, data, savefile=savefile

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; determine density at points data.x, data.y, data.z
;;
;; if format.dr = 0, then determines density from the voronoi region
;; if format.dr > 0, then determines density from packets within sphere
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

;; Determine how dr is set
formtags = strlowcase(tag_names(geometry))
q = fix(total(strmatch(tag_names(format), 'dr', /fold)))
if (q) $
  then dr = format.dr $
  else dr = geometry.dr

if (size(data, /type) NE 8) then stop ;; data must be given as a structure

nf = n_elements(files)
nspec = n_elements(*data.x)

loadem = (savefile EQ !null) ? 1 : ~file_test(savefile)

if (loadem) then begin
  xx = !null & yy = !null & zz = !null & frac = !null & radvel_sun = !null
  for ff=0,nf-1 do begin
    results_loadfile, files[ff], pts, vels_sun, frac2 ;; note - not keeing frac=0
    xx = [xx, pts[*,0]] & yy = [yy, pts[*,1]] & zz = [zz, pts[*,2]]
    frac = [frac, frac2]
    radvel_sun = [radvel_sun, vels_sun[*,1]+stuff.vrplanet]  ;; for g-value
    print, 'Loaded inputs ' + strint(ff+1) + ' of ' + strint(nf)
  endfor

  out = {x:ptr_new(temporary(xx)), y:ptr_new(temporary(yy)), $
    z:ptr_new(temporary(zz)), frac:ptr_new(temporary(frac)), $
    radvel_sun:ptr_new(temporary(radvel_sun))}
  rhosqr_sun = *out.x^2 + *out.z^2
  if (savefile NE !null) then save, out, rhosqr_sun, file=savefile
endif else restore, savefile

;; remove packets outside region of interest
q = where((*out.x GE min(*data.x)-.1) and (*out.x LE max(*data.x)+.1) and $
  (*out.y GE min(*data.y)-.1) and (*out.y LE max(*data.y)+.1) and $
  (*out.z GE min(*data.z)-.1) and (*out.z LE max(*data.z)+.1) and $
  (*out.frac GT 0), nq)
if (nq GT 0) then begin
```

```
  *out.x = (*out.x)[q]
  *out.y = (*out.y)[q]
  *out.z = (*out.z)[q]
  *out.frac = (*out.frac)[q]
  *out.radvel_sun = (*out.radvel_sun)[q]
endif

;; determine packet weighting
*out.frac = results_packet_weighting(out)

if (format.quantity NE 'density') then stop ;; only can do points at the moment

if (dr EQ 0) then begin
  print, 'Using voronoi regions to determine density'

  regions = results_voronoi(out)
  tree = results_kd_tree(out)

  density = results_density(*data.x, *data.y, *data.z, out, regions, tree)
endif else begin
  vpix = 4./3.*!pi*(dr*SystemConsts.rplan*1e5)^3
  density = fltarr(n_elements(*data.x))
  for i=0,n_elements(*data.x)-1 do begin
    xpr = *out.x-(*data.x)[i]
    ypr = *out.y-(*data.y)[i]
    zpr = *out.z-(*data.z)[i]
    rpr = sqrt(xpr^2 + ypr^2 + zpr^2)
    q = where(rpr LT dr, nq)
    if (nq GT 0) then density[i] = total((*out.frac)[q])/vpix
  endfor
endelse

result = {density:ptr_new(density), format:format}
return, result

end
```

```
function produce_image, files, savefile=savefile

common constants
common results

;;;;;;;;;;;;;;;;
;; Determine the image origin
s = (where(strcmp(*SystemConsts.objects, format.geometry.origin, /fold), ns))[0]
if (ns NE 1) then stop

;;;;;;;;;;;;;;;
;; Determine image field of view and rotation
geometry = format.geometry

image = dblarr((geometry.dims)[0],(geometry.dims)[1])
immin = geometry.center - geometry.width/2.
immax = geometry.center + geometry.width/2.

scale = geometry.width/(geometry.dims-1)   ;; [xscale,zscale] in Rplan/pix
Apix = (scale[0]*scale[1])*((*SystemConsts.radius)[s]*SystemConsts.rplan*1e5)^2
   ;; cm^2/pix

;; xaxis and zaxis in Robj measured from center of object
xaxis = findgen((geometry.dims)[0])*scale[0] + immin[0]
zaxis = findgen((geometry.dims)[1])*scale[1] + immin[1]

;; Determine frame rotation
M = determine_image_rotation(input, format)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
for ff=0,n_elements(files)-1 do begin
  ;; restore output file and extract useful packets
  ;; pts_sun is in solar reference frame with origin=Object center, units R_obj
  ;; vels_sun in km/s
  results_loadfile, files[ff], pts_sun, vels_sun, frac, /keepall
  radvel_sun = vels_sun[*,1] + stuff.vrplanet  ;; for g-value
     ;; note -- want to keep the ones with frac = 0 to make sure those regions
     ;; are counted as not contributing

  ;; Rotate the packets to observer frame
  pts_obs = M ## pts_sun    ;; observer along -y axis
  vels_obs = M ## vels_sun

  ;; Determine which packets are not blocked by the planet
  rhosqr_obs = pts_obs[*,0]^2 + pts_obs[*,2]^2  ;; rho in observer's frame
  inview = ((rhosqr_obs GT 1) or (pts_obs[*,1] LT 0))
  frac *= inview

  rhosqr_sun = pts_sun[*,0]^2 + pts_sun[*,2]^2
  out_of_shadow = ((rhosqr_sun GT 1) or (pts_sun[*,1] LT 0))

  ;; Determine which packets are in the FOV
  h = where((pts_obs[*,0] GE immin[0]) and (pts_obs[*,0] LE immax[0]) and $
    (pts_obs[*,2] GE immin[1]) and (pts_obs[*,2] LE immax[1]), nh)
  if (nh GT 0) then begin
    out = {x:ptr_new(pts_obs[h,0]), y:ptr_new(pts_obs[h,1]), z:ptr_new(pts_obs[h,2]), $
      frac:ptr_new(frac[h]), radvel_sun:ptr_new(radvel_sun[h])}

    ;; Packet weighting
    weight = results_packet_weighting(out, out_of_shadow)

    ;; Additional factors:
    case (format.quantity) of
      'column': weight /= Apix
      'intensity': weight /= Apix
      'density': weight /= Vpix
      else: stop
    endcase

    ;; Now make the image
    newh = where(weight GT 0, nh)
    if (nh GT 0) then begin
      qx = round(interpol(findgen((geometry.dims)[0]), xaxis, (*out.x)[newh]))
      qz = round(interpol(findgen((geometry.dims)[1]), zaxis, (*out.z)[newh]))
      for j=0,nh-1 do image[qx[j],qz[j]] += weight[newh[j]]
    endif
    ;tv, bytscl(image)
  endif ;; (nh GT 0)
  print, 'Completed image ' + strint(ff) + ' of ' + strint(n_elements(files))
endfor

result = {image:ptr_new(image), xaxis:ptr_new(xaxis), zaxis:ptr_new(zaxis), $
  format:format}
return, result

end
```

```
function produce_los, files, dataall

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; If given, data needs to have x, y, z, dx, dy, dz or the corners
;;
;; common block contains
;;   * input
;;   * format
;;   * SystemConsts
;;   * stuff = {aplanet, vrplanet, atoms_per_packet, mod_rate, totalsource}
;;   * gvalue = {lines, velocity, g}
;;   * plasma = TBD
;;
;; Version History:
;;   4.11: 12/8/2011
;;     * Need to make sure it doesn't use too many packets at once
;;   4.8: 7/20/2011
;;     * adding ability to use cylinder instead of instrument FOV
;;     * adding more comments
;;     * possible bug fixes
;;   4.6: 4/21/2011
;;     * Makes use of parallelized kd_tree code
;;   4.5: 4/20/2011
;;     * same as 4.6 with debugging info still included.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

;; Determine which mechamisms to do
doresscat = (max(strcmp(format.emission.mechanism, 'resscat', /fold)))
doeimp = (max(strcmp(format.emission.mechanism, 'eimp', /fold)))

;; Determine points and lines of sights
geometry = format.geometry
geotags = strlowcase(tag_names(geometry))

;; Determine how dr is set
formtags = strlowcase(tag_names(geometry))
q = fix(total(strmatch(tag_names(format), 'dr', /fold)))
if (q) $
  then dr = format.dr $
  else dr = geometry.dr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Load the data if not given
if (size(dataall, /type) NE 8) then begin
  ;; figure out what information is given
  tag_spacecraft = total(stregex(geotags, 'spacecraft', /bool))
  tag_orbit = total(stregex(geotags, 'orbit', /bool))
  tag_phase = total(stregex(geotags, 'phase', /bool))
  tag_tstart = total(stregex(geotags, 'tstart', /bool))
  tag_tend = total(stregex(geotags, 'tend', /bool))
  tag_dt = total(stregex(geotags, 'dt', /bool))

  if (tag_spacecraft EQ 0) then begin
    print, 'A spacecraft must be specified for LOS measurements.'
    stop
  endif
  sc = strlowcase(geometry.spacecraft)
  case (sc) of
    'messenger': begin
      ;; can specify either (tstart, tend) or (orbit, phase)
      ;; Orbit only currently makes sense for the flybys
      case (1) of
        (tag_orbit): begin
          phase =  (tag_phase) ? geometry.phase : 'all'
          dataall = load_MASCS_data(input.options.atom, geometry.orbit, phase, /Level3, $
            /model)
          end
        (tag_tstart) and (tag_tend): $
          dataall = load_MASCS_data(input.options.atom, geometry.tstart, geometry.tend, $
            /Level3, /model)
        else: begin
          print, 'Not set up yet.'
          stop
          endelse
      endcase
      end
    else: stop
  endcase
  if (strcmp(dataall.species, 'none', /fold)) then stop
endif
;; Now have data = {x, y, z, xbore, ybore, zbore, xcorner, ycorner, zcorner},

sss = (where(strlowcase(*SystemConsts.Objects) EQ $
  strlowcase(input.geometry.StartPoint)))[0]
robj = (sss EQ 0) ? SystemConsts.rplan*1e5 : $    ;; radius of object in cm
  SystemConsts.rplan*(*SystemConsts.radius)[s]*1e5

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine which observations are too far from the planet and which
;; look at the planet
;; Distance of s/c from planet
dist_from_plan = sqrt(*dataall.x^2 + *dataall.y^2 + *dataall.z^2)

;; Angle between look dir and planet -- negative since want from look pt to planet
ang = acos((-*dataall.x**dataall.xbore - *dataall.y**dataall.ybore - $
  *dataall.z**dataall.zbore)/dist_from_plan)
```

```
;; Remove observations not looking close enough to the object
if ~(input.options.fullsystem) then begin
  mindist = dist_from_plan * sin(ang)
  todo = (mindist LE input.options.OuterEdge)
endif else todo = replicate(1, n_elements(*dataall.x))

data = data_extract(dataall, todo)

;; check to see if look direction intersects the planet anywhere
;; angular size of planet from look pt.
asize_plan = asin(1./dist_from_plan)

;; Don't worry about lines of sight that don't hit the planet
missp = where(ang GT asize_plan, nmissp, comp=hitp)
if (nmissp GT 0) then dist_from_plan[missp] = 1e30


t0 = systime(1)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Now look at the model outputs
nf = n_elements(files)
nspec = n_elements(*data.x)
nall = n_elements(*dataall.x)

if (dr EQ 0) then begin
  ;; Use Voronoi method
  stop

  rhosqr_sun = *out.x^2 + *out.z^2
  out_of_shadow = ((rhosqr_sun GT 1) or (*out.y LT 0))
  ;; construct the voronoi regions and a kdtree to determine the los density
  print, 'Using instrument FOV'

  ;; make the voronoi region for these points
  regions = results_voronoi(out2)

  ;; make the kd_tree for these points
  tree = results_kd_tree(out2)

  ;; Determine FOV
  phic = atan(*data.ycorner, *data.xcorner)   ;; Corners
  thc = asin(*data.zcorner) & sinthc = *data.zcorner
  phib = atan(*data.ybore, *data.xbore) & thb = asin(*data.zbore) ;; Boresight

  ;; Determine where LOS intersects modeled region
  limits = results_find_intersection_points(data, input)
  lim0 = reform(limits[0,*]) & lim1 = reform(limits[1,*])
  m0 = min(phic, dim=1) & m1 = max(phic, dim=1)
  l0 = min(sinthc, dim=1) & l1 = max(sinthc, dim=1)

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; Loop over each individual LOS
  nr = 1000L & nphi = 15L & nth = 15L

  radiance = dblarr(nspec)
  density = dblarr(nr,nspec)
  denr = dblarr(nr,nspec)
  rrr = dindgen(nr)/(nr-1) & ppp = dindgen(nphi)/(nphi-1) & ttt = dindgen(nth)/(nth-1)
  iii = one(ppp) & jjj = one(ttt)

  tstart = systime(1)
  for i=0,nspec-1 do begin
    if (todo[i]) then begin
      t0 = systime(1)
      rtemp = rrr*(lim1[i]-lim0[i]) + lim0[i]
      ddr = rtemp[1]-rtemp[0]
      phitemp = ((ppp*(m1[i]-m0[i]) + m0[i]) # jjj)[*]
      sinthtemp = (iii # (ttt*(l1[i]-l0[i]) + l0[i]))[*]
      thtemp = asin(sinthtemp)

      xtemp0 = cos(phitemp)*cos(thtemp)
      ytemp0 = sin(phitemp)*cos(thtemp)
      ztemp0 = sinthtemp

      roi = obj_new('IDLanROI', phic[*,i], sinthc[*,i])
      q = where(roi.ContainsPoints(phitemp, sinthtemp), nq)
      obj_destroy, roi

      xden = (xtemp0[q]#rtemp)[*] + (*data.x)[i]
      yden = (ytemp0[q]#rtemp)[*] + (*data.y)[i]
      zden = (ztemp0[q]#rtemp)[*] + (*data.z)[i]

      den1 = results_density(xden, yden, zden, out2, regions, tree)
      den1 = reform(den1, nq, nr)
      denr[*,i] = rtemp
      density[*,i] = total(den1, 1)/nq
      radiance[i] = total(density[*,i])*ddr*robj
      t1 = systime(1)
      print, 'LOS Spec Number: ' + strint(i+1) + ' of ' + strint(nspec), t1-t0
    endif
  endfor
  tend = systime(1)
  print, 'LOS time: ', tend-tstart

;; Determine slit solid angle
;;omega = slit_solidangle(data)

endif else begin
  ;; Use a uniform thickness cylinder to determine los column density
  ;; Area of a column
  Apix = !pi * (dr*robj)^2

```

```
;; Load the packets
xx = !null & yy = !null & zz = !null & frac = !null & radvel_sun = !null
for ff=0,nf-1 do begin
  results_loadfile, files[ff], pts, vels_sun, frac2 ;; note - not keeing frac=0
  xx = [xx, pts[*,0]] & yy = [yy, pts[*,1]] & zz = [zz, pts[*,2]]
  frac = [frac, frac2]
  radvel_sun = [radvel_sun, vels_sun[*,1]+stuff.vrplanet]  ;; for g-value
  print, 'Loaded inputs ' + strint(ff+1) + ' of ' + strint(nf)
endfor
out = {x:ptr_new(temporary(xx)), y:ptr_new(temporary(yy)), $
  z:ptr_new(temporary(zz)), frac:ptr_new(temporary(frac)), $
  radvel_sun:ptr_new(temporary(radvel_sun))}

;; Determine emission measure for each packet
;; base shadow on whether los goes through shadow
out_of_shadow = replicate(1, n_elements(*out.x))
weight = results_packet_weighting(out, out_of_shadow)

radiance = fltarr(nspec)
for i=0,nspec-1 do begin
  ;; Determine which packets are close to the line of sight
  xpr = *out.x - (*data.x)[i]
  ypr = *out.y - (*data.y)[i]
  zpr = *out.z - (*data.z)[i]
  rpr = sqrt(xpr^2 + ypr^2 + zpr^2)
  costheta = (xpr*(*data.xbore)[i] + ypr*(*data.ybore)[i] + $
    zpr*(*data.zbore)[i])/rpr

  ;; delta = perpendicular distance to the line of sight
  delta = rpr * sin(acos(costheta))
  q = where(finite(delta) EQ 0, nq) & if (nq GT 0) then stop

  inview = where((delta LT dr) and (costheta GT 0) and (*out.frac GT 0), nin)

  if (nin GT 0) then begin
    ftemp = (*out.frac)[inview]/apix
    if (doresscat) then begin
      ;; Determine whether the point along the LOS the packet represents is in
      ;; shadow
      losr = rpr[inview] * costheta[inview] ;; projection of packet onto LOS
      xhit = (*data.x)[i] + (*data.xbore)[i]*losr ;; point packet represents
      yhit = (*data.y)[i] + (*data.ybore)[i]*losr
      zhit = (*data.z)[i] + (*data.zbore)[i]*losr
      rhohit = xhit^2 + zhit^2
      out_of_shadow = (rhohit GT 1) or (yhit LT 0)
      ftemp *= out_of_shadow
    endif

    radiance[i] = total(ftemp)
  endif
  if ((i mod 100) EQ 0) then print, 'Finished spec #' + strint(i)
  endfor
  result = {radiance:ptr_new(radiance), format:format}
endelse

return, result

end
```

```
function produce_results, inputtemp, formattemp, data=data, npackets=npackets, $
  savefile=savefile, local=local

common constants
common results
time0 = systime(1)

if (local EQ !null) then local = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; For instructions, see: modelpro_2.0/Docs/produce_results.tex
;;
;; Given and inputfile and an output format file, produce the
;; desired output.
;;
;; All positions and angles need to be given in a reference frame with
;; the +y axis pointed away from the sun -- i.e. in the model reference frame
;;
;; Inputs:
;;  inputtemp - can be
;;    (a) inputfile - restore input and search for outputfiles
;;    (b) input structure - search for outputfiles
;;    (c) outputfile - restore
;;  formattemp = either a format structure or a file with the format
;;
;; Keyword Inputs:
;;   * npackets = minimum number of packets that are needed to continue.
;;
;; Version History:
;;   4.0: 25 Jan 2011
;;     * Original based on previous routines
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (npackets EQ !null) then npackets = 0 ;; If not specified, only need 1 packet
if (data EQ !null) then data = -1

fname = 'produce_results: '
stuff = {aplanet:0d, vrplanet:0d, atoms_per_packet:0d, mod_rate:0d, totalsource:0d, $
  local:local}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; restore the inputs and determine outputfiles to use
ss = size(inputtemp, /type)
case (1) of
  (ss EQ 7) and (stregex(inputtemp[0], '.output', /fold, /bool)): begin
    ;; A list of output files has been given
    ofile = obj_new('IDL_savefile', inputtemp[0])
    ofile.restore, 'input'
    obj_destroy, ofile
    files = inputtemp
    ;SystemConstants, input.geometry.planet, SystemConsts
    end
  (ss EQ 7) and (stregex(inputtemp, '.input', /fold, /bool)): begin
    ;; the name of an input file is given
    input = inputs_restore(inputtemp)
    SystemConstants, input.geometry.planet, SystemConsts
    files = modeloutput_search(input, nfiles=n0)
    end
  (ss EQ 8): begin
    ;; an input structure is given
    input = inputtemp
    SystemConstants, input.geometry.planet, SystemConsts
    files = modeloutput_search(input, nfiles=n0)
    end
  else: stop
endcase
if (size(input, /type) NE 8) then stop
nfiles = (files[0] EQ '') ? 0 : n_elements(files)
print, fname + strint(nfiles) + ' output files found.'

;; Restore the system constants
planet_dist, input.geometry.taa, SystemConsts, distance=aplanet, velocity=vrplanet
stuff.aplanet = aplanet
stuff.vrplanet = vrplanet

;; Determine the number of packets available
if (nfiles GT 0) then begin
  pack = extract_parameter('savedpackets', files)
  totalpackets = long(total((pack.values()).ToArray(type='long')))
  pack = 0  ; get around an IDL bug
endif else totalpackets = 0L
print, fname + strint(totalpackets) + ' packets found.'

;; If there are enough packets, process the result
if (totalpackets GT npackets) then begin
  ;; Restore the results format file
  case (size(formattemp, /type)) of
    7: format = read_resultformat(formattemp)
    8: format = formattemp
    else: stop
  endcase
  if (size(format, /type) NE 8) then stop

  ;; Determine the packet conversion
  tt = extract_parameter('totalsource', files)
  stuff.totalsource = total((tt.values()).ToArray(type='double'))
  tt = 0  ; get around an IDL bug

  stuff.mod_rate = stuff.totalsource / input.options.endtime ;; packets ejected per sec
  stuff.atoms_per_packet = (format.strength *1e26) / stuff.mod_rate
```

```
    print, fname + strint(stuff.mod_rate) + ' packets ejected per second'
    print, fname + strint(stuff.atoms_per_packet) + ' atoms per packet'

    ;;;;;;;;;;;;;;;;
    ;; Set up intensity if needed
    if (format.quantity EQ 'intensity') then results_intensity_setup

    ;; take different path for each result type
    case strlowcase(format.type) of
      'image': result = produce_image(files, savefile=savefile)
      'voronoi image': result = produce_voronoi_image(files, savefile=savefile)
      'los': result = produce_los(files, data)
      'points': result = produce_density(files, data, savefile=savefile)
      else: stop
    endcase
  endif else begin ;; (totalpackest < npackets)
    print, fname + 'Too few packets found.'
    result = -1
  endelse

  time1 = systime(1)
  print, 'Total runtime = ' + strint(round(time1-time0)) + ' seconds'

  return, result

  end

```

```
function produce_voronoi_image, files

common constants
common results

;;;;;;;;;;;;;;;;
;; Determine the image origin
s = (where(strcmp(*SystemConsts.objects, format.geometry.origin, /fold), ns))[0]
if (ns NE 1) then stop

;;;;;;;;;;;;;;;
;; Determine image field of view and rotation
geometry = format.geometry

image = dblarr((geometry.dims)[0],(geometry.dims)[1])
immin = geometry.center - geometry.width/2.
immax = geometry.center + geometry.width/2.

scale = geometry.width/(geometry.dims-1)         ;; [xscale,zscale] in Rplan/pix
Apix = (scale[0]*scale[1])*(SystemConsts.rplan*1e5)^2 ;; cm^2/pix

;; xaxis and zaxis in Robj measured from center of object
xaxis = findgen((geometry.dims)[0])*scale[0] + immin[0]
zaxis = findgen((geometry.dims)[1])*scale[1] + immin[1]

;; Determine frame rotation
M = determine_image_rotation(input, format)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
xx = !null & yy = !null & zz = !null & frac = !null
vx = !null & vy = !null & vz = !null & radvel_sun = !null
nf = n_elements(files)
for ff=0,nf-1 do begin
  ;; restore output file and extract useful packets
  ;; pts_sun is in solar reference frame with origin=Object center, units R_obj
  ;; vels_sun in km/s
  results_loadfile, files[ff], pts_sun, vels_sun, frac2 ;; note - not keeing frac=0

  ;; Rotate the packets to observer frame
  pts_obs = M ## pts_sun    ;; observer along -y axis
  vels_obs = M ## vels_sun

  ;; Determine which packets are not blocked by the planet
  rhosqr_obs = pts_obs[*,0]^2 + pts_obs[*,2]^2  ;; rho in observer's frame
  inview = ((rhosqr_obs GT 1) or (pts_obs[*,1] LT 0))
  frac2 *= inview

  h = where((pts_obs[*,0] GE immin[0]) and (pts_obs[*,0] LE immax[0]) and $
    (pts_obs[*,2] GE immin[1]) and (pts_obs[*,2] LE immax[1]), nh)
  if (nh GT 0) then begin
    xx = [xx, pts_obs[h,0]] & yy = [yy, pts_obs[h,1]] & zz = [zz, pts_obs[h,2]]
    vx = [vx, vels_obs[h,0]] & vy = [vy, vels_obs[h,1]] & vz = [vz, vels_obs[h,2]]
    frac = [frac, frac2[h]]
    radvel_sun = [radvel_sun, vels_sun[h,1]+stuff.vrplanet]  ;; for g-value
  endif
  print, 'Loaded inputs ' + strint(ff+1) + ' of ' + strint(nf)
endfor
out = {x:ptr_new(temporary(xx)), y:ptr_new(temporary(yy)), $
  z:ptr_new(temporary(zz)), frac:ptr_new(temporary(frac)), $
  vx:ptr_new(temporary(vx)), vy:ptr_new(temporary(vy)), $
  vz:ptr_new(temporary(vz)), radvel_sun:ptr_new(temporary(radvel_sun))}

weight = results_packet_weighting(out, format)
;; Additional factors:
case (format.quantity) of
  'column': weight /= Apix
  'intensity': weight /= Apix
  'density': weight /= Vpix
  else: stop
endcase

;; Determine voronoi regions
regions = results_voronoi(out)

;; make the kd_tree for these points
tree = results_kd_tree(out)

dy = min(scale)/2.
ny = round((max(*out.y)-min(*out.y))/dy)+1
yaxis = findgen(ny)*dy + min(*out.y)

density = dblarr((geometry.dims)[0],(geometry.dims)[1],ny)
yy = (one(zaxis)#yaxis)[*]
zz = (zaxis#one(yaxis))[*]
nn = n_elements(yy)
for i=0,(geometry.dims)[0]-1 do begin
  t0 = systime(1)
  xx = replicate(xaxis[i],nn)
  temp = results_density(xx, yy, zz, out, regions, tree)
  density[i,*,*] = reform(temp, (geometry.dims)[1], ny)
  t1 = systime(1)
  print, i, t1-t0
endfor

;;for i=0,(geometry.dims)[0]-1 do begin
;;  for j=0,(geometry.dims)[1]-1 do begin
;;    density[i,j,*] = results_density(replicate(xaxis[i],ny), yaxis, $
;;      replicate(zaxis[i],ny), out, regions, tree)
;;  endfor
;;  print, i
;;endfor

```

```
result = {image:ptr_new(image), xaxis:ptr_new(xaxis), zaxis:ptr_new(zaxis), $
  format:format, yaxis:ptr_new(yaxis), density:ptr_new(density)}
return, result

end
```

```
pro quick_look, outfile, geomfile, image=image, x0=x0, imtype=imtype

if (n_elements(imtype) NE 1) then imtype = 'column'

;; If geofile isn't given then get it
if (n_elements(geomfile) NE 1) then stop

;; restore the outputs
restore, outfile
SystemConstants, run_info.planet, c

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Print out some basic information
print, outfile
print, 'Planet: ' + run_info.planet
print, 'Starting Point: ' + run_info.startpoint
q = where(*run_info.gravity EQ 1, nq)
if (nq EQ 0) $
  then print, 'Gravity was not turned on' $
  else for i=0,nq-1 do print, (*c.objects)[q[i]] + '''s gravity is on'
print, 'Total run time = ' + strtrim(string(run_info.endtime/3600.), 2) + ' hours'
print, 'Neutral Species = ' + run_info.atom
print, 'Radiation Pressure is ' + ((run_info.radpres) ? 'on' : 'off')

if (run_info.fullsystem) $
  then print, 'Tracking full system' $
  else print, 'Only tracking packets within ' + $
    strtrim(string(run_info.outeredge),2) + ' object radii.'

print, '******************'
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Show the initial velocity distribution
;;window, 0
wset, 0
show_veldist, proc_info, run_info, vrange=vrange, theo=theo, /disp

vv = sqrt(*startloc.vx^2 + *startloc.vy^2 + *startloc.vz^2)
mm = minmax(vrange) & dv = vrange[1]-vrange[0]
actual = histw(vv, *loc.frac, min=mm[0], max=mm[1], bin=dv)/dv

;plot, vrange, theo, xr=[0,15], /ylog, yr=[10,1e6], /xst
oplot, vrange, actual, color=2
xyouts, .55, .85, /norm, 'Initial Velocity Distribution!c  (all packets)'
xyouts, .55, .75, /norm, 'Initial Velocity Distribution!c  (remaining packets)', color=2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Print out Loss Processes
if (run_info.lifetime EQ 0) then begin
  print, 'Loss Processes Included'
  for i=0,n_elements(*loss_info.reactions)-1 do $
    print, '  (' + strtrim(string(i),2) + ') ' + (*loss_info.reactions)[i]
  print, '**********************'
endif else print, strtrim(string(round(run_info.lifetime/3600)), 2) + ' hour lifetime'

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Make a Column density image
case (imtype) of
  'intensity': image = model_images('intensity', outfile, geomfile, 1., line='5890')
  'density': image = model_images('density', outfile, geomfile, 1., dz=0.1, zplane=0)
  else: image = model_images('column', outfile, geomfile, 1.)
endcase

restore, geomfile
;window, 1, xs=geoms.xs+150, ys=geoms.xs+150
wset, 1

xc = cos(findgen(361)*!dtor) & yc = sin(findgen(361)*!dtor)
x0 =(findgen(geoms.xs)/(geoms.xs-1)-.5)*((geoms.xr)[1]-(geoms.xr)[0])/ $
  (*c.radius)[geoms.center]
plot, findgen(10), /nodata, xr=minmax(x0), yr=minmax(x0), /xst, /yst, $
  xtit='Distance from ' + (*c.objects)[geoms.center] + ' (R!dObj!n)', $
  ytit='Distance from ' + (*c.objects)[geoms.center] + ' (R!dObj!n)', $
  pos=[100,100,100+geoms.xs,100+geoms.ys], /dev, tit=file_basename(outfile)
disparr, image, 3, /log, result=image2, /nodisp, low=l, high=h
tv, bytscl(image2, l, h, top=220)+35, 100, 100, /dev
polyfill, xc, yc, color=4
plots, [100,100,100+geoms.xs,100+geoms.xs,100], [100,100+geoms.ys,100+geoms.ys,100,100],$
  /dev

@destroy_all
destroy_constants, c

end
```

```
function read_resultformat, formatfile

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Read in the result format file.
;;
;; Version History:
;;   4.2 1 Dec 2011
;;     * A few updates
;;   4.1: 24 Oct 2011
;;     * Reworking this
;;   4.0: 25 Jan 2011
;;     * Original
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

readcol, formatfile, param, value, delim='=', format='A,A', /silent
param = strlowcase(strtrim(param, 2))

;; strip off any comments in the values
q = stregex(value, ';')
w = where(q NE -1, nq)
if (nq GT 0) then for i=0,nq-1 do $
  value[w[i]] = strmid(value[w[i]], 0, q[w[i]]-1)
value = strtrim(value, 2)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Make the format structure
form = where(strmatch(param, 'format*'))
fparam = strmid(param[form], strlen('format.'))
fval = value[form]

q = (where(fparam EQ 'type', nq))[0]
if (nq EQ 1) then type = fval[q] else stop

q = (where(fparam EQ 'quantity', nq))[0]
if (nq EQ 1) then quantity = fval[q] else stop

q = (where(fparam EQ 'strength', nq))[0]
strength = (nq EQ 1) ? double(fval[q]) : 1.

;; Test these:
if ((type NE 'image') and (type NE 'voronoi image') and (type NE 'los') and $
  (type NE 'points')) then begin
    print, 'Not a valid result type.'
    print, 'Valid options are: image, voronoi, los, points'
    stop
endif

if ((quantity NE 'column') and (quantity NE 'intensity') and (quantity NE 'density')) $
  then begin
  print, 'Not a valid result quantity.'
  print, 'Valid options are: column, intensity, density.'
  stop
 endif

if (strength LE 0) then begin
 print, 'Strength must be >0.'
 stop
endif

;;;;;;;;;;;;;;;;;
;; Make the geometry structure
geo = where(strmatch(param, 'geometry*'))
gparam = strmid(param[geo], strlen('geometry.'))
gval = value[geo]

q = (where(gparam EQ 'origin', nq))[0]
if (nq EQ 1) then origin = gval[q] else stop

case (1) of
 (type EQ 'image') or (type EQ 'voronoi image'): begin
  q = (where(gparam EQ 'dims', nq))[0]
  if (nq EQ 1) then begin
   dims = strcompress(gval[q], /remove_all)
   dims = fix(strsplit(dims, ',', /extract))
  endif else stop

  q = (where(gparam EQ 'center', nq))[0]
  if (nq EQ 1) then begin
   center = strcompress(gval[q], /remove_all)
   center = float(strsplit(center, ',', /extract))
  endif else stop

  q = (where(gparam EQ 'width', nq))[0]
  if (nq EQ 1) then begin
   width = strcompress(gval[q], /remove_all)
   width = float(strsplit(width, ',', /extract))
  endif else stop

  q = (where(gparam EQ 'subobslongitude', nq))[0]
  if (nq EQ 1) then subobslong = float(gval[q]) else stop
  if ((subobslong LT 0) or (subobslong GT 2*!dpi)) then begin
   print, 'Sub-Observer Longitude must be between 0 and 2¿'
   stop
  endif

  q = (where(gparam EQ 'subobslatitude', nq))[0]
  if (nq EQ 1) then subobslat = float(gval[q]) else stop
  if ((subobslat LT -!dpi/2) or (subobslat GT !dpi/2)) then begin
   print, 'Sub-Observer Latitude must be between -¿/2 and ¿/2'
   stop
```

```
      endif

      q = (where(gparam EQ 'polarangle', nq))[0]
      if (nq EQ 1) then polarangle = float(gval[q]) else stop
      if ((polarangle LT 0) or (polarangle GT 2*!dpi)) then begin
        print, 'Polar angle must be between 0 and 2¿'
        stop
      endif

      geometry = {origin:origin, dims:dims, center:center, width:width, $
        subobslongitude:subobslong, subobslatitude:subobslat, $
        polarangle:polarangle}
      end
    (type EQ 'los') or (type EQ 'density'): begin
      ;; Note: dr can be either in format or geometry part
      q = (where(fparam EQ 'dr', nq))[0]
      if (nq EQ 1) $
        then dr = double(fval[q]) $
        else begin
          q = (where(gparam EQ 'dr', nq))[0]
          if (nq EQ 1) then dr = double(gval[q]) else stop
        endelse

      q = (where(gparam EQ 'usedata', nq))[0]
      usedata = (nq EQ 1) ? fix(gval[q]) : 1

      if (usedata) then begin
        q = (where(gparam EQ 'spacecraft', nq))[0]
        spacecraft = gval[q]

        if (type EQ 'density') then begin
          q = (where(gparam EQ 'dt', nq))[0]
          dt = (nq EQ 1) ? double(gval[q]) : 0.
        endif else dt = 0.

        q = (where(gparam EQ 'orbit', nq))[0]
        case (nq) of
          0: begin  ;; tstart, tend specified
             q = (where(gparam EQ 'tstart', nq))[0]
             if (nq EQ 1) then tstart = gval[q] else stop

             q = (where(gparam EQ 'tend', nq))[0]
             if (nq EQ 1) then tend= gval[q] else stop
             geometry = {origin:origin, dr:dr, spacecraft:spacecraft, usedata:usedata, $
               tstart:tstart, tend:tend, usedata:usedata, dt:dt}
            end
          1: begin  ;; orbit # specified
             orbit = fix(gval[q])
             geometry = {origin:origin, dr:dr, spacecraft:spacecraft, orbit:orbit, $
               usedata:usedata, dt:dt}
             end
        else: stop
      endcase
    endif
    end
  else: stop ;; problem
endcase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Make the emission structure if necessary
if (quantity EQ 'intensity') then begin
  emi = where(strmatch(param, 'emission*'))
  eparam = strmid(param[emi], strlen('emission.'))
  eval = value[emi]

  q = (where(eparam EQ 'mechanism', nq))[0]
  if (nq EQ 1) then mech = eval[q] else stop
  mech = strsplit(mech, ',', /extract)
  if (n_elements(mech) EQ 1) then mech = mech[0]

  q = (where(eparam EQ 'line', nq))[0]
  if (nq EQ 1) then line = eval[q] else stop
  line = float(strsplit(line, ',', /extract))
  if (n_elements(line) EQ 1) then line = line[0]

  emission = {mechanism:mech, line:line}
endif else emission = !null

format = {type:type, quantity:quantity, strength:strength, geometry:geometry, $
  emission:emission}

return, format

end
```

```
pro results_common

;; Set up a common block

common results, input, format, gvalue, plasma
common density, x0, x1, output0, regions

end
```

```
function results_density, x, y, z, output, regions, tree, volume=volume, points=points

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version History
;;   4.5: 4/21/2011
;;     * First version that works with parallelized kd_tree nearest neighbor
;;       search
;;   4.4: 4/20/2011
;;     * Same as 4.5 but still has the debug code in it -- use this when
;;       writing up the comparisons
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

if (n_elements(x) NE n_elements(y)) then stop
if (n_elements(x) NE n_elements(z)) then stop
npts = n_elements(x)

volume = dblarr(npts)
density = dblarr(npts)
points = replicate(-1L, npts)

;; Enforce density outside modeled region or inside planet = 0
r = sqrt(x^2 + y^2 + z^2)
if (input.options.fullsystem) $
  then nonzero = where(r GT 1, num, comp=zero) $
  else nonzero = where((r GT 1) and (r LT input.options.outeredge), num, comp=zero)

if (num GT 0) then begin
  x2 = float(x[nonzero])
  y2 = float(y[nonzero])
  z2 = float(z[nonzero])

  ;; Determine closest packet to each point
  outpts = ptr_new([[*output.x], [*output.y], [*output.z]])
  results_find_closest, outpts, tree, [[x2], [y2], [z2]], pmin=pt
  outpts = 0

  ;; Determine the volume for each of the needed regions
  results_voronoi_volume, regions, pt
  volume2 = (*regions.volume)[pt]
  q = where(volume2 EQ 0, nq) & if (nq NE 0) then stop

  density2 = (*output.frac)[pt]/volume2/(SystemConsts.rplan*1e5)^3 ;; volume = cm^3
  q = where(volume2 GT 1e10, nq)
  if (nq NE 0) then density2[q] = 0.

  volume[nonzero] = volume2
```

```
    density[nonzero] = density2
    points[nonzero] = pt
  endif

  q = where(density LT 0, nq) & if (nq NE 0) then stop
  return, density

  end
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Some functions to help out computing the results
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_loadfile, file, pts_sun, vels_sun, frac, keepall=keepall

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Load results file and convert to proper reference frame
;;
;; Input:
;;   file = output file to restore
;;
;; Outputs:
;;   pts_sun = x,y,z in the solar frame with (0,0,0)=object center and units=R_obj
;;   vels_sun = vx,vy,vz in the solar frame, units=km/s
;;   frac = packet fraction remaining
;;
;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

if (keepall EQ !null) then keepall=0

;; Determine the image origin
s = (where(strcmp(*SystemConsts.objects, format.geometry.origin, /fold), ns))[0]
if (ns NE 1) then stop

if (s NE 0) then begin
  ;; Will need to translate packets to satellite frame
  origin = (*SystemConsts.a)[s]*[-sin((*input.geometry.phi)[s]), $
    cos((*input.geometry.phi)[s]), 0.]   ;; location of satellite
  sc = 1./(*SystemConsts.radius)[s]      ;; scale factor
endif else begin
  origin = [0., 0., 0.]
  sc = 1.
endelse

;; Reuseable script to load the output file and get the packets to use
ofile = obj_new('IDL_savefile', file)
ofile.restore, 'output'
obj_destroy, ofile

;; Extract packets to use
touse = (keepall) ? lindgen(n_elements(*output.frac)) : where(*output.frac NE 0, npack)

;; Determine position relative to origin -- not rotated
pts_sun = [[(*output.x)[touse]-origin[0]], $
  [(*output.y)[touse]-origin[1]], $
  [(*output.z)[touse]-origin[2]]]
pts_sun *= sc  ;; Units = R_obj

;; Velocities not adjusted -- still includes orbital motion
vels_sun = [[(*output.vx)[touse]], [(*output.vy)[touse]], [(*output.vz)[touse]]]
vels_sun *= SystemConsts.rplan

frac = (*output.frac)[touse]
destroy_structure, output

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_intensity_setup

common constants
common results

if (max(strcmp(format.emission.mechanism, 'resscat', /fold))) then begin
  ;; get g-values
  gvalue = get_gvalue(input.options.atom, stuff.aplanet)
endif

if (max(strcmp(format.emission.mechanism, 'eimp', /fold))) then begin
  stop
  ;; load plasma info
endif

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function slit_solidangle, data

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Determine the solid angle subtended by the slit
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

temp = [[[*data.xcorner]], [[*data.ycorner]], [[*data.zcorner]]]
c0 = reform(temp[0,*,*]) & c1 = reform(temp[1,*,*])
c2 = reform(temp[2,*,*]) & c3 = reform(temp[3,*,*]) & temp = 0

xxx = c0[*,1]*c2[*,2] - c0[*,2]*c2[*,1]
yyy = -c0[*,0]*c2[*,2] + c0[*,2]*c2[*,0]
zzz = c0[*,0]*c2[*,1] - c0[*,1]*c2[*,0]
ccc = total(c0*c2,2)
```

```

q0 = abs(c1[*,0]*xxx + c1[*,1]*yyy + c1[*,2]*zzz)
q1 = 1 + ccc + total(c1*c0,2) + total(c1*c2,2)
omega0 = atan(q0,q1)
q = where(omega0 LT 0, nq) & if (nq NE 0) then omega0[q] += !pi

q0 = abs(c3[*,0]*xxx + c3[*,1]*yyy + c3[*,2]*zzz)
q1 = 1 + ccc + total(c3*c0,2) + total(c3*c2,2)
omega1 = atan(q0,q1)
q = where(omega1 LT 0, nq) & if (nq NE 0) then omega1[q] += !pi

omega = 2*(omega0+omega1) ;; slit solid angle for each spectrum

return, omega

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function results_find_intersection_points, data, input

nn = n_elements(*data.x)
tt = dblarr(2,nn)

oedge = (input.options.outeredge*1.25)^2 ;; give 25% leeway
dist_from_plan = sqrt(*data.x^2 + *data.y^2 + *data.z^2)
for i=0,nn-1 do begin
  r0 = dist_from_plan[i]
  t = findgen(1001)/1000. * (dist_from_plan[i]+input.options.outeredge*1.5)

  p0x = (*data.x)[i] + t*(*data.xbore)[i]
  p0y = (*data.y)[i] + t*(*data.ybore)[i]
  p0z = (*data.z)[i] + t*(*data.zbore)[i]
  r2 = p0x^2 + p0y^2 + p0z^2
  if (dist_from_plan[i] LT input.options.outeredge) then begin
    tt[0,i] = 0.
    tt[1,i] = interpol(t, r2, oedge)
  endif else begin
    q = (where(r2 EQ min(r2)))[0]
    tt[0,i] = interpol(t[0:q], r2[0:q], oedge)
    tt[1,i] = interpol(t[q:*], r2[q:*], oedge)
  endelse
endfor

return, tt

end
```

```
function results_trace_tree, pts, tree, points, stpt=stpt

treesize = size(*pts) & dim = treesize[2]
npts = (size(points))[1]

if (stpt EQ !null) then stpt = (where(*tree.level EQ 0))[0]
if ((n_elements(stpt) NE 1) and (n_elements(stpt) NE npts)) then stop

branches = replicate(-1L,npts,max(*tree.level)+2)
branches[*,0] = stpt

ct = 0
q = where(branches[*,0] NE -1L, nq)
while (nq NE 0) do begin
  ;; Current node
  a = branches[q,ct]

  ;; Level of current node and dimension to look at
  lev = (*tree.level)[a]
  dd = lev mod dim

  pp = points[q,dd]  ;; Points still to do
  tpp = (*pts)[a,dd] ;; Node values to compare with
  www = (pp LT tpp)

  ;; Determine whether to take hi or low branch
  newa = www*(*tree.lowchild)[a] + (1-www)*(*tree.hichild)[a]

  ;; Increment count
  ct++

  ;; add in the next node
  branches[q,ct] = newa
  q = where(branches[*,ct] NE -1L, nq)
endwhile

return, branches

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_find_closest, pts, tree, points, stpt=stpt, rmin=rmin, pmin=pmin, $
  lll=lll

if (lll EQ !null) then lll = 0

treesize = size(*pts) & dim = treesize[2]
npts = (size(points))[1]
;;print, 'Tree Level = ', lll, npts

```

```
if (stpt EQ !null) then begin
  stpt = (where(*tree.level EQ 0))[0]
;;  rmin = replicate(1d30, npts)
;;  pmin = lonarr(npts)
endif

if ((n_elements(stpt) NE 1) and (n_elements(stpt) NE npts)) then stop
if (n_elements(rmin) NE npts) then begin
  rmin = replicate(1d30, npts)
  pmin = lonarr(npts)
endif

;; First follow the tree to see where each point belongs
branch = results_trace_tree(pts, tree, points, stpt=stpt)

bb = max(branch, dim=1)
w = (where(bb EQ -1, nw))[0]
if (nw NE 0) then branch = branch[*,0:w-1]

temp = (size(branch))
brmax = (temp[0] EQ 1) ? 0 : temp[2]-1

;; if (dd LT rmin) then it is possible that there is a closer point in that branch
for i=brmax,0,-1 do begin
  nodes = branch[*,i]
;;  printf, 1, lll, i, nodes
  q = where(nodes NE -1, nq)
  if (nq NE 0) then begin
    pp = points[q,*]
    nn = nodes[q]
    ll = (*tree.level)[nn] mod dim
    dd = ((*pts)[nn,ll]-points[q,ll])^2

    w = q[where(dd LT rmin[q], nw)]
    if (nw GT 0) then begin
      pp2 = points[w,*]
      nn2 = nodes[w]
      ll2 = (*tree.level)[nn2] mod dim
      node_pts = (*pts)[nn2,*]
      if (n_elements(pp2) NE n_elements(node_pts)) then stop

      r0 = total((node_pts-pp2)^2,2)
      if (n_elements(r0) NE nw) then stop
      qr = where(r0 LT rmin[w], nr)
      if (nr NE 0) then begin
        rmin[w[qr]] = r0[qr]
        pmin[w[qr]] = nn2[qr]
      endif

      ;; follow the branch not previously used
      dir = (pp2[lindgen(nw),ll2] LT node_pts[lindgen(nw),ll2])
      newst = dir*(*tree.hichild)[nn2] + (1-dir)*(*tree.lowchild)[nn2]
      e = where(newst NE -1L, nee)
      if (nee GT 0) then begin
        pp3 = pp2[e,*]
        newst = newst[e]
        rtemp = rmin[w[e]]
        ptemp = pmin[w[e]]
        results_find_closest, pts, tree, pp3, stpt=newst, rmin=rtemp, pmin=ptemp, $
          lll=lll+1
        rmin[w[e]] = rtemp
        pmin[w[e]] = ptemp
      endif
    endif
  endif
endfor

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_kd_node, tree, pts, index, parent, ndim, level

n = n_elements(index)
case (n) of
  1: begin
     (*tree.level)[index] = level
     (*tree.parent)[index] = parent
     (*tree.lowchild)[index] = -1
     (*tree.hichild)[index] = -1
     indnode = index
     end
  2: begin
     (*tree.level)[index[0]] = level
     (*tree.parent)[index[0]] = parent
     (*tree.lowchild)[index[0]] = index[1]
     (*tree.hichild)[index[0]] = index[1]

     (*tree.level)[index[1]] = level+1
     (*tree.parent)[index[1]] = index[0]
     (*tree.lowchild)[index[1]] = -1
     (*tree.hichild)[index[1]] = -1
     end
  else: begin
     dim = level mod ndim

     p = (*pts)[dim,index]
     s = sort(p)

     ;; this node is at index[n2] = where(pts[*,dim] EQ median(pts[*,dim]))
     n2 = n/2 & if ((n2 EQ 0) or (n2 EQ n-1)) then stop
     indnode = index[s[n2]]
     indlow = index[s[0:n2-1]] & indhi = index[s[n2+1:*]]

     (*tree.level)[indnode] = level
     (*tree.parent)[indnode] = parent
     results_kd_node, tree, pts, indlow, indnode, ndim, level+1
     results_kd_node, tree, pts, indhi, indnode, ndim, level+1

     ii = indlow[(where((*tree.level)[indlow] EQ level+1, nq))[0]]
     if (nq NE 1) then stop
     if ((*tree.parent)[ii] NE indnode) then stop
     (*tree.lowchild)[indnode] = ii

     ii = indhi[(where((*tree.level)[indhi] EQ level+1, nq))[0]]
     if (nq NE 1) then stop
     if ((*tree.parent)[ii] NE indnode) then stop
     (*tree.hichild)[indnode] = ii
   endelse
endcase

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function results_kd_tree, output

tstart = systime(1)

;;pts = n x 3 array
pts = ptr_new(transpose([[*output.x], [*output.y], [*output.z]]))

sz = size(*pts)
ndim = sz[1]

tree = {level:ptr_new(lonarr(sz[2])), parent:ptr_new(lonarr(sz[2])), $
  lowchild:ptr_new(lonarr(sz[2])), hichild:ptr_new(lonarr(sz[2]))}

index = lindgen(sz[2])
results_kd_node, tree, pts, index, -1, ndim, 0
pts = 0.

tend = systime(1)
print, 'results_kd_tree: ', tend-tstart

return, tree

end
```

```
function results_packet_weighting, output, out_of_shadow

common constants
common results

case (format.quantity) of
  'column': weight = *output.frac * stuff.atoms_per_packet ;; atoms per packet
  'density': weight = *output.frac * stuff.atoms_per_packet
  'intensity': begin
    if (max(strcmp(format.emission.mechanism, 'resscat', /fold))) then begin
      ;; trim min and max vy_sun values
      w = where(*output.radvel_sun LT min(*gvalue.v), nw)
      if (nw NE 0) then (*output.radvel_sun)[w] = min(*gvalue.v)
      w = where(*output.radvel_sun GT max(*gvalue.v), nw)
      if (nw NE 0) then (*output.radvel_sun)[w] = max(*gvalue.v)

      ;; sum g-value over observed lines
      gg = 0.
      for j=0,n_elements(format.emission.line)-1 do begin
        w = (where(abs(*gvalue.wavelength-(format.emission.line)[j]) LE $
          1e-2, nw))[0]
        if (nw NE 1) then stop
        gg += interpol((*gvalue.g)[*,w], *gvalue.v, $
          *output.radvel_sun)
      endfor

      ;; Compute emission measure for each packet
      weight_resscat = (*output.frac*stuff.atoms_per_packet) * out_of_shadow * $
        (gg/1e6) ;; Ra
      ;; gg/1e6 = 10^6 photons/atom/sec
      ;; *output.frac * atoms_per_packet = atoms
      ;; f_resscat = 10^6 photons/sec
    endif

    ;; Compute electron impact emission
    if (total(strcmp(format.emission.mechanism, 'eimp', /fold))) then stop

    ;; Sum emission measures for each process
    weight = weight_resscat ;; + weigth_eimp
    end
  'spectrum': stop
  else: stop
endcase

q = where(finite(weight) EQ 0, nq) & if (nq NE 0) then stop

return, weight

end
```

```
pro results_voronoi_volume, regions, q, volume=volume

if (q EQ !null) then q = lindgen(n_elements(*regions.volume))

for i=0,n_elements(q)-1 do $
  if ((*regions.volume)[q[i]] EQ 0) then begin
    vv = *regions.vertices[q[i]]

    hullfile = ('hull' + strint(round(random_nr(1)*1000000)) + '.dat')[0]
    openw, lun, hullfile, /get_lun
    printf, lun, '3'
    printf, lun, n_elements(vv)/3
    printf, lun, transpose(vv)
    free_lun, lun

    spawn, ['qconvex', 's', 'FS', 'TI', hullfile], out, ss, /noshell
    out = out[1]
    (*regions.volume)[q[i]] = double((strsplit(out, /extract))[2])
    if ((*regions.volume)[q[i]] LE 0) then stop

    spawn, 'rm ' + hullfile
  endif

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function results_voronoi, output

tstart = systime(1)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes the Voronoi connectivity for a set of points
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pts = [[*output.x], [*output.y], [*output.z]]

;; Save the points to a temporary file
sz = size(pts)

ptsfile = ('pts' + strint(round(random_nr(1)*1000000)) + '.dat')[0]
openw, lun, ptsfile, /get_lun
printf, lun, sz[2]
printf, lun, sz[1]
printf, lun, transpose(pts)
free_lun, lun

;; Compute the voronoi regions
```

```
spawn, ['qvoronoi', 's', 'p', 'FN', 'TI', ptsfile], out, ss, /noshell
spawn, 'rm ' + ptsfile

dim = long(out[0]) & nvert = long(out[1])
vertstring = out[2:2+nvert-1]
vertices = fltarr(nvert, dim)
for i=0L,nvert-1 do vertices[i,*] = float(strsplit(vertstring[i], /extract))

ct = 2+nvert
nreg = long(out[ct])
reg = out[ct+1:*]
if (n_elements(reg) NE nreg) then stop
if (nreg NE sz[1]) then stop

;; Find the voronoi region for each packet
regions = {vertices:ptrarr(nreg, /allocate), volume:ptr_new(dblarr(nreg))}
for i=0,nreg-1 do begin
  w = long(strsplit(reg[i], /extract))
  if (n_elements(w) GT 1) then begin
    w = w[1:*]
    q = where(w LT 0, onedge)
  endif else onedge = 1

  ;; if there are negative indices, at edge of region, set Volume=infinite
  if (onedge GT 0) then begin
    *regions.vertices[i] = -1
    (*regions.volume)[i] = 1e30
  endif else *regions.vertices[i] = vertices[w,*]
endfor
tend = systime(1)
print, 'Results_voronoi: ', tend-tstart

return, regions

end
```

```
function accel, loc, input, magcoord, which

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes acceleration on a packet due to the specified forces
;; Adapted from accel in rkas.pro
;;
;; Forces given in units of Rplan/s^2
;;
;; Possible forces:
;;  * Gravity
;;  * radiation pressure
;;  * Lorentz [not yet]
;;
;; Equations of motion:
;;   dvxdt = sum_objects (GM * (x-x_obj))/(r_obj)^3
;;   dvydt = sum_objects (GM * (y-y_obj))/(r_obj)^3 + C* gamma(vy)
;;   dvzdt = sum_objects (GM * (z-z_obj))/(r_obj)^3
;;     -- r_obj = sqrt( (x-x_obj)^2 + (y-y_obj)^2 + (z-z_obj)^2 )
;;     -- radiation pressure only valid for Na
;;   dndt = instantaneous change in density
;;
;;  Version history:
;;    3.0: 7/21/2010
;;      * rewritten with new structure architecture
;;    2.1: 4/29/10
;;      * adding optional out_of_shadow input
;;    2.0: Rewritten so that individual forces are farmed out
;;    1.0: Original version
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

agrav =  (input.Forces.gravity) $
  ? gravity(loc, input.geometry, input.options, which) $
  : 0.
aradpres = (input.Forces.radpres) $
  ? radiation_pressure(loc, input.geometry, input.options.atom, *magcoord.out_of_shadow) $
  : 0.
alor = (input.Forces.Lorentz) $
  ? Lorentz(loc, options) $
  : 0.

accel = {dvdt:ptr_new(0)}
*accel.dvdt = agrav + aradpres + alor

q = where(finite(*accel.dvdt) EQ 0) & if (q[0] NE -1) then stop
return, accel

end

```

```
function gravity, loc, geometry, options, which

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes gravitational acceleration
;;
;; Equations of motion:
;;   dvxdt = sum_objects (GM * (x-x_obj))/(r_obj)^3
;;   dvydt = sum_objects (GM * (y-y_obj))/(r_obj)^3
;;   dvzdt = sum_objects (GM * (z-z_obj))/(r_obj)^3
;;     -- r_obj = sqrt( (x-x_obj)^2 + (y-y_obj)^2 + (z-z_obj)^2 )
;;     -- radiation pressure only valid for Na
;;   dndt = instantaneous change in density
;;
;;  Version History
;;  3.0 - 7/20/2010
;;    * Small upgrade
;;  2.0 - 10/22/08
;;    * Created file from accel.pro
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

n = (size(*loc.x))[1]
ct = n_elements(which) ;; number of objects

;; Determine positions of satellites for each packet
if (options.motion) $
  then locmoon, *loc.t, (*geometry.phi)[which], (*SystemConsts.a)[which], $
    (*SystemConsts.orbrate)[which], x=xsat, y=ysat, z=zsat $
  else locmoon, fltarr(n_elements(*loc.t)), (*geometry.phi)[which], $
    (*SystemConsts.a)[which], (*SystemConsts.orbrate)[which], x=xsat, y=ysat, z=zsat

;; Compute distances between packets and satellites
ii = replicate(1., ct)
jj = replicate(1., n)

xdiff = (*loc.x)[*,0]#ii - xsat
ydiff = (*loc.x)[*,1]#ii - ysat
zdiff = (*loc.x)[*,2]#ii - zsat
r3 = (xdiff^2 + ydiff^2 + zdiff^2)^1.5

;; Compute gravitational acceleration
GM = jj # (*SystemConsts.GM)[which]
ax = GM * xdiff/r3
ay = GM * ydiff/r3
az = GM * zdiff/r3

if (ct NE 1) then begin
  ax = total(ax, 2)
  ay = total(ay, 2)
  az = total(az, 2)
endif

; Final resutls
accel = dblarr(n,3)
accel[*,0] = ax
accel[*,1] = ay
accel[*,2] = az

return, accel

end

```

```
function Lorentz, loc, options

common constants

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Compute the Lorentz force on an ion
;;
;; Assumes the dipole is aligned north-south
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

qm = atomiccharge(options.atom)/atomicmass(options.atom)

r = sqrt(total(*loc.x^2, 2))
x = (*loc.x)[*,0]
y = (*loc.x)[*,1]
z = (*loc.x)[*,2]
;; Field strength in Gauss
Bx = 3*x*z*DipoleConsts.strength*r^(-5)
By = 3*y*z*DipoleConsts.strength*r^(-5)
Bz = (3*z^2-r^2)*DipoleConsts.strength*r^(-5)

;; Determine speed of ion relative to magnetic field
Bvx = -DipoleConsts.magrat * y
Bvy = DipoleConsts.magrat * x
vx = ((*loc.v)[*,0]-Bvx)/!const.c
vy = ((*loc.v)[*,1]-Bvy )/!const.c
vz = (*loc.v)[*,2]/!const.c

;; Electric field
Ex = 0. & Ey = 0. & Ez = 0.

ax = qm * (Ex + vy*Bz - vz*By)
ay = qm * (Ey + vz*Bx - vx*Bz)
az = qm * (Ez + vx*By - vy*Bx)

accel = dblarr(n_elements(x),3)
accel[*,0] = ax
accel[*,1] = ay
accel[*,2] = az

return, accel

end
```

```
function radiation_pressure, loc, geometry, atom, out_of_shadow

common constants

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Get radiation acceleration as function of radial velocity
;; Radiation pressure depends on the species
;;
;; Radiation_const = h*g/(m*lambda)/R_planet as fn of v_rad [units = R_plan/s^2]
;;
;; Version history
;;   3.0: 7/20/2010
;;     * Revised for new structures
;;   2.1: Added support for multiple species based on gvalues from Killen et al 2008
;;   2.0: original
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (n_elements(out_of_shadow) EQ 0) then stop ;out_of_shadow = 1.

gg = interpol(*stuff.radpres_const, *stuff.radpres_v, (*loc.v)[*,1]+stuff.vrplanet)
arad = out_of_shadow * gg

n = (size(*loc.x))[1]
accel = dblarr(n,3)
accel[*,1] = arad

return, accel

end

```

```
pro driver, input, output, seed=seed

;;************************************************************************
;;
;;  Driver routine to run the 5th order RK integrator from Numerical
;;  Recipies, 3rd Ed.
;;
;;  Version History:
;;    3.1: 4/27/2011
;;      -- Need to speed up when modeling satellites
;;    3.0: 7/20/2010
;;      -- Revising for new structure architecture
;;    2.7: 7/6/2010:
;;      -- Adding impact_check_2.9 to this program so it does not use the include
;;    2.6: 4/26/2010:
;;      -- Added moon's temperature map
;;      -- removed thermalized option from emitfn case (no longer used)
;;    2.5: 1/14/2010:
;;      -- Keep track of fate of each packet
;;      -- Keep track of deposition on the surface
;;      -- Replace ptr_free with destory_structure
;;    2.4: 12/7/2009
;;      -- Allowing variable surface temperature for Maxwellian reemission
;;    2.3: 11/6/2009
;;      -- changing the way it does the thermalization. New velocity is determined from
;;         partial accomodation to thermal speed at surface
;;    2.2: Added variable surface temperature for particle sticking (for Mercury,
;;      Based on surface temperature in Leblanc & Johnson 2003). -- I don't think I
;;      did this [12/7/09]
;;    2.1: Added support for elastic bouncing of particles from the surface
;;    2.0: Revised for current structure setup
;;    1.2: Previous working version
;;    1.1: Older version to work with rk4
;;    1.0: Similar to the original version to work with rk7
;;
;; Impact_check version history (before inclusion into this program):
;;    Version 2.9 4/28/10
;;       -- Changing definition of accommodation coefficient
;;          * Need energy accommodation rather than velocity accommodation
;;          * Before: v_1 = a v_th + (1-a) v_0
;;          * After: v_1^2 = a v_th^2 + (1-a) v_0^2
;;    Version 2.8 3/9/10
;;       -- Fixing issue with thermal accomodation - Now choose speed based on
;;      thermal distribution.
;;    Version 2.7 1/19/10
;;       -- fixing some problems with v2.6
;;    Version 2.6 1/14/2010
;;       -- keep track of what happens to each packet
;;       -- keep track of surface deposition
;;            * doesn't get the map right for satellites.
;;    Version 2.4 11/6/2009
;;      -- added thermal accomodation to the surface
;;   Version 2.1 -- added option for elastic bouncing
;;
;;***************************************************************

common constants

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Remake the loc structure to speed up the math and make it easier to read
if (input.options.trackloss) $
  then loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0), $
    lossfrac:ptr_new(0), hitfrac:ptr_new(0), ringfrac:ptr_new(0)} $
  else loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0)}
*loc.x = [[*output.x], [*output.y], [*output.z]]
*loc.v = [[*output.vx], [*output.vy], [*output.vz]]
*loc.frac = *output.frac
*loc.t = *output.time ;; This is how much time before present and works up to zero
npack = n_elements(*output.x)

if (input.options.trackloss) then begin
  lossfrac = dblarr(npack)
  hitfrac = dblarr(npack,n_elements(*SystemConsts.objects))
  ringfrac = dblarr(npack)
  leftfrac = dblarr(npack)
  deposition = {longitude:ptr_new(findgen(360)*!dtor), $
    latitude:ptr_new(findgen(180)*!dtor-!pi/2), $
    map:ptr_new(dblarr(360,180,n_elements(*SystemConsts.objects)))}
  ilon = findgen(360)
  ilat = findgen(180)
endif

which = where(*input.geometry.include, nw)
pp = replicate(1., n_elements(*SystemConsts.objects))

;; Set up the stepsizes
h = replicate(1000d, npack)     ;initial guess at best stepsize
hold = h  ;; last step used by each packet

;Set variables in preparation for iteration
count = 0L  ;; number of steps taken

;These control how quickly the stepsize is increased or decreased between iterations
safety = .95
shrink = -.25
grow = -.2

;; yscale = scaling parameter for each variable
;;   x,y,z ~ R_plan
;;   vx,vy,vz ~ 1 km/s (1/Rplan Rplan/s)
;;   frac ~ exp(-t/lifetime) ~ mean(frac)

```

```
resolution = input.options.resolution

;; Set up the bounce conditions
if (input.sticking_info.stickcoef NE 1.) then begin
  case strlowcase(input.sticking_info.emitfn) of
    'maxwellian': begin
      if (input.sticking_info.Tsurf EQ 0) then begin   ;; Use surf temp model
        case (input.geometry.startpoint) of
          'Mercury': begin
            temp0 = 100.
            temp1 = 600 + 125*(cos(input.geometry.taa)-1)/2. ;; sub-solar temp fn of taa
            nn = .25
            end
          'Moon': begin
            temp0 = 151
            temp1 = 162.
            nn = .25
            end
          else: stop
        endcase

        nt = 21 & nv = 101
        temperature = dindgen(nt)/(nt-1)*temp1 + temp0
        v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        prob = dindgen(101)/100.
        vgrid = dblarr(nt,101)
        for i=0,nt-1 do begin
          vrange = dindgen(nv)/(nv-1)*v_temp[i]*3.
          f_v = MaxwellianDist(vrange, temperature[i], input.options.atom)
          sumdist = f_v
          for j=1,nv-1 do sumdist[j] += sumdist[j-1]
          sumdist /= max(sumdist)
          vgrid[i,*] = interpol(vrange, sumdist, prob)
        endfor
      endif else begin   ;; use constant surf temp
        v_th = sqrt(2*input.Sticking_info.Tsurf*!const.kb/atomicmass(input.options.atom)) /1e5
        vrange = findgen(1001)/1000 * v_th*5 & vrange = vrange[1:*]
        f_v = MaxwellianDist(vrange, input.Sticking_info.Tsurf, options.atom)

        sumdist = f_v
        for i=1,n_elements(vrange)-1 do sumdist[i] += sumdist[i-1]
        sumdist /= max(sumdist)
      endelse
      end
    'elastic scattering':
    else: stop
  endcase
end

;******************************************************************************
;Keep takeing R.K. steps until every packet has reached the time of "image taken"
```

```
;*******************************************************************************

moretogo = where((*loc.t GT resolution) and (*loc.frac GT 0), ntogo)
done = (ntogo EQ 0)

while ~(done) do begin
  ;Now generate sub-arrays containing only the particles that are still being tracked

  loc0 = {t: ptr_new((*loc.t)[moretogo]), x:ptr_new((*loc.x)[moretogo,*]), $
    v:ptr_new((*loc.v)[moretogo,*]), frac:ptr_new((*loc.frac)[moretogo])}

  w = where(*loc0.frac EQ 0, nw) & if (nw NE 0) then stop
  w = where(finite(*loc0.x) EQ 0, nw) & if (nw NE 0) then stop
  w = where(finite(*loc0.v) EQ 0, nw) & if (nw NE 0) then stop

  oldx = *loc0.x  ;; This is used for determining if anything hit the rings
  oldf = *loc0.frac
  h = hold[moretogo]

  ;Adjust stepsize to be no more than time remaining
  h = (h LE (*loc0.t))*h + (h GT (*loc0.t))*(*loc0.t)

  ;; Run the rk5 step
  rk_5, loc0, h, input, which, delta

  ;; Do the error check
  ;; scale = a_tol + |y| * r_tol
  ;;   for x: a_tol = r_tol = resolution
  ;;   for v: a_tol = r_tol = resoltuon/10. -- require v to be more precise
  ;;   for f: a_tol = 0.01 ; r_tol = 0 -- set fractional tolerance to 1%
  scalespace = resolution + abs(*loc0.x) * resolution
  scalevel = 0.1*(resolution + abs(*loc0.v) * resolution)
  scaleabund = 0.01 ;; resolution + abs(*loc0.frac) * resolution

  ;; difference relative to acceptable difference
  *delta.x /= scalespace
  *delta.v /= scalevel
  *delta.frac /= scaleabund
  xerrmax = max(*delta.x, dim=2)
  verrmax = max(*delta.v, dim=2)

  ;; Maximum error for each packet
  errmax = (xerrmax GE verrmax)*xerrmax + (xerrmax LT verrmax)*verrmax
;  errmax = (errmax GE *delta.frac)*errmax + (errmax LT *delta.frac)**delta.frac
  if ((where(finite(errmax) EQ 0))[0] NE -1) then stop

  ;; Check where difference is very small - adjust step size
  noerr = where(errmax LE 1e-7)
  if (noerr[0] NE -1) then begin
    errmax[noerr] = 1.
    h[noerr] = h[noerr]*10.
    endif

    ;; Put the post-step values
    g = where(errmax LE 1.0, ng, comp=b)
    if (ng GT 0) then begin
      (*loc.t)[moretogo[g]] = (*loc0.t)[g]
      (*loc.x)[moretogo[g],*] = (*loc0.x)[g,*]
      (*loc.v)[moretogo[g],*] = (*loc0.v)[g,*]
      (*loc.frac)[moretogo[g]] = (*loc0.frac)[g]
      if (input.options.trackloss) then $
        lossfrac[moretogo[g]] += (oldf[g]-(*loc0.frac)[g])  ;; add in change in frac
      h[g] = safety*h[g]*errmax[g]^grow
    endif

    if (ng NE ntogo) then begin
      ;; don't adjust the bad values, but do fix the stepsize
      htemp = safety * h[b] * errmax[b]^shrink
      q = where(htemp LT 0.0, nq) & if (nq NE 0) then stop

      ;; don't let step size drop below 1/10th previous step size
      h[b] = max([[htemp], [0.1*h[b]]], dim=2)
    endif
    qqq = where(h LT 1e-7, nqq) & if (nqq NE 0) then stop  ;; error test

    destroy_structure, loc0
    destroy_structure, delta

    ;save new values of h
    hold[moretogo] = h

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Impact check
    ;; Only look at packets which moved during this step
    if (ng GT 0) then begin
      ;; Make a new structure with just the packets that moved this step
      loc1 = {t:ptr_new((*loc.t)[moretogo[g]]), x:ptr_new((*loc.x)[moretogo[g],*]), $
        v:ptr_new((*loc.v)[moretogo[g],*]), frac:ptr_new((*loc.frac)[moretogo[g],*])}
      if (input.options.trackloss) then begin
        oldfrac = *loc1.frac
        hitfrac1 = hitfrac[moretogo[g],*]
        ringfrac1 = ringfrac[moretogo[g]]
        leftfrac1 = leftfrac[moretogo[g]]
      endif

      jj = replicate(1., ng)

      ;; 1) Did the packets hit anything?
      ;Get object positions
      if (input.options.motion) $
        then locmoon, *loc1.t, *input.geometry.phi, *SystemConsts.a, $
          *SystemConsts.orbrate, x=xSat, y=ySat, z=zSat, ang=ang $
      else locmoon, fltarr(ng), *input.geometry.phi, *SystemConsts.a, $
        *SystemConsts.orbrate, x=xSat, y=ySat, z=zsat, ang=ang

    ;; Distance of packets from each object
    tempR = sqrt(((*loc1.x)[*,0]#pp - xSat)^2 + ((*loc1.x)[*,1]#pp - ySat)^2 + $
      ((*loc1.x)[*,2]#pp - zSat)^2)

    ;Is r < satellite radius?
    eps = 0.
    satrad = jj # (*SystemConsts.radius)*(1-eps)
    hhh = where((tempR-satrad) LT 0, nhits)
    if (nhits NE 0) then begin
      ;;w = where((*loc1.t)[hhh mod ng] EQ 0, nw) & if (nw NE 0) then stop
      hx = hhh mod ng & hy = hhh/ng

      ;; adject the frac values
      if (input.sticking_info.stickcoef EQ 1) $
        then (*loc1.frac)[hx] = 0 $
        else (*loc1.frac)[hx] = (*loc1.frac)[hx] * (1.-input.sticking_info.stickcoef)

      ;; If need to know where things hit the surface, do this
      if ((input.options.trackloss) or (input.sticking_info.stickcoef LT 1)) then begin
        ;; Figure out where things hit the surface
        srad = satrad[hhh]
        r0 = tempR[hhh]  ;; R_plan
        x0 = (*loc1.x)[hx,0]  ;; R_plan
        y0 = (*loc1.x)[hx,1]  ;; R_plan
        z0 = (*loc1.x)[hx,2]  ;; R_plan
        r0 = sqrt(x0^2 + y0^2 + z0^2)

        ;; Position of the satellites
        xcent = xSat[hx,hy]
        ycent = ySat[hx,hy]
        zcent = zSat[hx,hy]

        ;; Vector from center of satellite to packet
        ;;   -- packet positions relative to satellite
        x1 = (x0-xcent)/srad  ;; rsat
        y1 = (y0-ycent)/srad  ;; rsat
        z1 = (z0-zcent)/srad  ;; rsat

        ;; Velocity - orbital vel = vel relative to satellite
        vxsat = -(*SystemConsts.orbvel)[hy]*cos(ang[hx,hy])*input.options.motion/$
          SystemConsts.rplan
        vysat = -(*SystemConsts.orbvel)[hy]*sin(ang[hx,hy])*input.options.motion/$
          SystemConsts.rplan

        vx0 = (*loc1.v)[hx,0] - vxsat  ;; rplan/s
        vy0 = (*loc1.v)[hx,1] - vysat  ;; rplan/s
        vz0 = (*loc1.v)[hx,2]  ;; rplan/s
        ;; Find where the packet hit the surface
        ;;     |x + vt| = 1 -- see ResearchNotes from 4/28/08
        a = vx0^2 + vy0^2 + vz0^2
        b = 2*(x1*vx0 + y1*vy0 + z1*vz0)
        c = x1^2 + y1^2 + z1^2 - 1

        dd = b^2 - 4*a*c
        q = where(dd LT 0, nq) & if (nq NE 0) then stop
        t0 = (-b - sqrt(b^2-4*a*c))/(2*a)
        t1 = (-b + sqrt(b^2-4*a*c))/(2*a)
        t = (t0 LE 0)*t0 + (t1 LT 0)*t1

        ;; Point where packet hit the surface
        x2 = x1 + vx0*t
        y2 = y1 + vy0*t
        z2 = z1 + vz0*t
        ;; r2 = sqrt(x2^2 + y2^2 + z2^2)  ;; -- this should be = 1.

        lonhit = (atan(x2, -y2) + 2*!pi) mod (2*!pi)
        lathit = asin(z2)

        ;; Put new coordinates into the array
        x_final = xcent + x2*srad
        y_final = ycent + y2*srad
        z_final = zcent + z2*srad

        q = where(finite(x_final) EQ 0, nq) & if (nq NE 0) then stop
        q = where(finite(y_final) EQ 0, nq) & if (nq NE 0) then stop
        q = where(finite(z_final) EQ 0, nq) & if (nq NE 0) then stop
        (*loc1.x)[hx,0] = x_final
        (*loc1.x)[hx,1] = y_final
        (*loc1.x)[hx,2] = z_final
      endif

      if (input.options.trackloss) then begin
        lonind = (fix(interpol(ilon, *deposition.longitude, lonhit))) mod $
          n_elements(*deposition.longitude)
        latind = fix(interpol(ilat, *deposition.latitude, lathit)) mod $
          n_elements(*deposition.latitude)
        for i=0L,nhits-1 do (*deposition.map)[lonind[i],latind[i],hy[i]] += $
          oldfrac[hx[i]]*input.sticking_info.stickcoef
        hitfrac1[hhh] += oldfrac[hx]
      endif

      if (input.sticking_info.stickcoef LT 1) then begin
        ;; Figure out rebound velocity
        vv02 = vx0^2 + vy0^2 + vz0^2  ;; rplan/s
        PE = 2*(*SystemConsts.GM)[hy]*(1./r0-1./srad)
        vv02 += PE
        q = where(vv02 LT 0, nq) & if (nq NE 0) then vv02[q] = 0.
        q = where(finite(vv02) EQ 0, nq) & if (nq NE 0) then stop
      case strlowcase(input.sticking_info.emitfn) of
        'maxwellian': begin ;; Re-emit the packets with a thermal distribution
          if (input.sticking_info.Tsurf EQ 0) then begin
            surftemp = temp0 + (temp1*(abs(cos(lonhit)*cos(lathit)))^nn)*$
              (abs(lonhit) LT !pi/2)
            rr = random_nr(nhits, seed=seed, routine=0)
            vv_new = interpolate_xy(vgrid, temperature, prob, surftemp, rr)/$
              SystemConsts.rplan
          endif else vv_new = interpol(vrange, sumdist, $
            random_nr(seed=seed, nhits))/SystemConsts.rplan ;; rplan/s

          vv2 = sqrt(input.Sticking_info.accom_factor*vv_new^2 + $
            (1-input.Sticking_info.accom_factor)*vv02)
          end
        'elastic scattering': vv2 = sqrt(vv02)
      endcase

      ;; Determine new direction with F(v) ~ cos(theta)
      alt = acos(random_nr(seed=seed, nhits))
      az = 2*!pi * random_nr(seed=seed, nhits)

      v_rad = sin(alt)          ;; Radial component of velocity
      v_east = -cos(alt) * sin(az) ;; Component along latitude line (points east)
      v_north = cos(alt) * cos(az) ;; Component along longitude line (points to NP)

      vx2 = v_rad*x2
      vy2 = v_rad*y2
      vz2 = v_rad*z2

      lat = asin(z2)
      lon = atan(x2, y2)

      vx2 = dblarr(nhits) & vy2 = dblarr(nhits) & vz2 = dblarr(nhits)
      for i=0L,nhits-1 do begin
        M = transpose([ $
          [cos(lat[i])*sin(lon[i]), cos(lat[i])*cos(lon[i]), sin(lat[i])], $
          [-cos(lon[i]), sin(lon[i]), 0], $
          [-sin(lat[i])*sin(lon[i]), -sin(lat[i])*cos(lon[i]), cos(lat[i])] ])
        v_ren = [v_rad[i], v_east[i], v_north[i]]
        v_xyz = invert(M) # v_ren
        vx2[i] = v_xyz[0] * vv2[i]
        vy2[i] = v_xyz[1] * vv2[i]
        vz2[i] = v_xyz[2] * vv2[i]
      endfor

      ;; The new position in planet-centered coords
      vx_final = vx2 + vxsat
      vy_final = vy2 + vysat
      vz_final = vz2
```

```
    q = where(finite(vx_final) EQ 0, nq) & if (nq NE 0) then stop
    q = where(finite(vy_final) EQ 0, nq) & if (nq NE 0) then stop
    q = where(finite(vz_final) EQ 0, nq) & if (nq NE 0) then stop

    (*loc1.v)[hx,0] = vx_final
    (*loc1.v)[hx,1] = vy_final
    (*loc1.v)[hx,2] = vz_final
  endif
endif

;; 2) Have the packets left the corona? (only check if not tracking the full system)
if ~(input.options.fullSystem) then begin
  leftCor = where(tempR[*,stuff.s] GT input.options.OuterEdge * $
    (*SystemConsts.radius)[stuff.s], hh)
  if (hh NE 0) then begin
    if (input.options.trackloss) then leftfrac1[leftcor] += (*loc1.frac)[leftcor]
    (*loc1.frac)[leftCor] = 0
  endif
endif

;; 3) If Saturn, check to see if anything hit the rings
if (input.geometry.planet EQ 'Saturn') then begin
  ;; Ring region within 2.3 Rs
  ox = oldx[g,*]
  cross = ox[*,2] * (*loc1.x)[*,2]  ;; if cross is negative, then crossed eq. plane
  MayHit = where(cross LE 0 , nmay)
  if (nmay NE 0) then begin
    orho = sqrt(total(ox[MayHit,0:1]^2, 2))
    nrho = sqrt(total((*loc1.x)[MayHit,0:1]^2, 2))
    w = where((orho LT 2.3) or (nrho LT 2.3), nw)
    for j=0,nw-1 do begin $
      crosspt = interpol([orho[w[j]],nrho[w[j]]], [ox[MayHit[w[j]],2], $
        (*loc1.x)[MayHit[w[j]],2]], 0.)
      if (crosspt LT 2.3) then begin
        if (input.options.trackloss) then ringfrac1[MayHit[w[j]]] += $
          (*loc1.frac)[MayHit[w[j]]]
        (*loc1.frac)[MayHit[w[j]]] = 0.
      endif
    endfor
  endif
endif

;; Check to see if any packets have shrunk out of existence
q = where((*loc1.frac GT 0) and (*loc1.frac LT 1e-10), nq)
if (nq NE 0) then (*loc1.frac)[q] = 0.

;; If any new hits, set the time remaining to 0.
w = where(*loc1.frac EQ 0, nw)
if (nw NE 0) then (*loc1.t)[w] = 0.

;Put new values back into original array (again)
```

```
      (*loc.t)[moretogo[g]] = *loc1.t
      (*loc.x)[moretogo[g],*] = *loc1.x
      (*loc.v)[moretogo[g],*] = *loc1.v
      (*loc.frac)[moretogo[g]] = *loc1.frac
      if (input.options.trackloss) then begin
        hitfrac[moretogo[g],*] = hitfrac1
        ringfrac[moretogo[g]] = ringfrac1
        leftfrac[moretogo[g]] = leftfrac1
      endif

      destroy_structure, loc1
    endif
    ;;end impact check
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    moretogo = where(*loc.t GT resolution, ntogo)  ;; check to see which ones aren't done
    if (count mod 100 EQ 0) then print, stuff.strstart + 'Step Number: ' + string(count) + $
      ', Packets Remaining: ' + string(ntogo)
    count += 1 ;; step counter

    ;If it goes 100000 steps then it will never stop!
    done = ((ntogo EQ 0) or (count GT 100000.))
  endwhile

  *output.x = reform((*loc.x)[*,0])
  *output.y = reform((*loc.x)[*,1])
  *output.z = reform((*loc.x)[*,2])
  *output.vx = reform((*loc.v)[*,0])
  *output.vy = reform((*loc.v)[*,1])
  *output.vz = reform((*loc.v)[*,2])
  *output.frac = *loc.frac
  if (input.options.trackloss) then begin
    *output.hitfrac = reform(hitfrac)
    *output.lossfrac = lossfrac
    *output.ringfrac = ringfrac
    *output.leftfrac = leftfrac
    output.deposition = deposition
  endif

  destroy_structure, loc

  end
```

```
pro result_rk5, t, h, xx0, vv0, NN, delta, output, regions

compile_opt idl2, hidden

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; This does a 5th order RK step and computes the error estimate
;; See Numerical Recipes, ch 17.2
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

hh = h # [1., 1., 1.]
tt = t # [1., 1., 1.]

;; RK coefficients
c = [0., 0.2d, 0.3d, 0.8d, 8./9.d,  1d]
b = [35d/384d,  0d, 500d/1113d, 125d/192d, -2187d/6784d, 11d/84d]
bs = [5179d/57600d, 0d, 7571d/16695d, 393d/640d, -92097d/339200d, 187d/2100d]
bdiff = b-bs

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Figure out which points to look at

x0 = xx0 + (tt+c[0]*hh)*vv0
x1 = xx0 + (tt+c[1]*hh)*vv0
x2 = xx0 + (tt+c[2]*hh)*vv0
x3 = xx0 + (tt+c[3]*hh)*vv0
x4 = xx0 + (tt+c[4]*hh)*vv0
x5 = xx0 + (tt+c[5]*hh)*vv0

;; Compute density at each point
k0 = hh * results_density(x0, output, regions)
k1 = hh * results_density(x1, output, regions)
k2 = hh * results_density(x2, output, regions)
k3 = hh * results_density(x3, output, regions)
k4 = hh * results_density(x4, output, regions)
k5 = hh * results_density(x5, output, regions)

;; Compute N at t+h
t += h

temp = k0*b[0] + k1*b[1] + k2*b[2] + k3*b[3] + k4*b[4] + k5*b[5]
if (min(temp) LT 0) then stop
NN += temp

;; Estimate the error
delta = abs(k0*bdiff[0] + k1*bdiff[1] + k2*bdiff[2] + k3*bdiff[3] + k4*bdiff[4] + $
  k5*bdiff[5])

q = where(nn LT 0, nq) & if (nq NE 0) then stop
```

```

end
```

```
function result_rkintegrate, x0temp, x1temp, output, regions, resolution=resolution

compile_opt idl2, hidden

;;**************************************************************************
;;
;;  Driver routine to run the 5th order RK integrator from Numerical
;;  Recipies, 3rd Ed.
;;
;;  x0,x1 should be nx3 arrays
;;
;;  Solve function:
;;    dN/dt = n(r(t)), r(t) = x0 + t*(x1-x0), N(t=0) = 0,
;;    integrate from t=0->t=1 or x0->x1
;;
;; Function returns integral over path.
;;
;; Written by Matthew Burger
;; Version 4.0: 3/24/2011
;;
;;***************************************************************

if (n_elements(resolution) EQ 0) then resolution = 1d-6

x0 = x0temp & x1 = x1temp
sz0 = size(x0) & sz1 = size(x1)
if ~(array_equal(sz0, sz1)) then stop
if (sz0[0] EQ 1) then begin
  x0 = transpose(x0)
  x1 = transpose(x1)
  sz0 = size(x0)
endif
npts = sz0[1]

h = replicate(0.1d, npts)       ;initial guess at best stepsize

;Set variables in preparation for iteration
count = 0L  ;; number of steps taken

;These control how quickly the stepsize is increased or decreased between iterations
safety = .95
shrink = -.25
grow = -.2

; -- don't use this right now - may want to change
;; yscale = scaling parameter for each variable

timeinc = resolution

;*****************************************************************************
;Keep takeing R.K. steps until every packet has reached the time of "image taken"
```

```
;******************************************************************************
;
t = dblarr(npts)
N = dblarr(npts)
v0 = x1-x0
tend = 1. ;; integrate from 0 to 1

t_step = dblarr(npts,10000)
N_step = dblarr(npts,10000)

if (regions EQ !null) then regions = results_voronoi(output)

tremain = tend - t
moretogo = where(tremain GT timeinc, ntogo)
done = (ntogo EQ 0)
while ~(done) do begin
  ;Now generate sub-arrays containing only the particles that are still being tracked

  trem = tremain[moretogo]
  t_sub = t[moretogo]
  x0_sub = x0[moretogo,*]
  v0_sub = v0[moretogo,*]
  N_sub = N[moretogo]
  h_sub = h[moretogo]

  ;Adjust stepsize to be no more than time remaining
  h_sub = (h_sub LE trem)*h_sub + (h_sub GT trem)*trem

  ;; Run the rk5 step
  result_rk5, t_sub, h_sub, x0_sub, v0_sub, N_sub, delta, output, regions

  ;; Do the error check
  ;; scale = a_tol + |y| * r_tol
  scale = resolution + abs(N_sub)*resolution

  ;; difference relative to acceptable difference
  delta /= scale

  ;; Check where difference is very small - adjust step size
  noerr = where(delta LE 1e-7)
  if (noerr[0] NE -1) then begin
    ;print, n_elements(noerr)
    delta[noerr] = 1.
    h_sub[noerr] = h_sub[noerr]*10.
  endif

  ;; Put the post-step values
  g = where(delta LE 1.0, ng, comp=b)
  if (ng GT 0) then begin
    t[moretogo[g]] = t_sub[g]
    N[moretogo[g]] = N_sub[g]
    h[moretogo[g]] = safety*h_sub[g]*delta[g]^grow
  endif

  if (ng NE ntogo) then begin
    ;; don't adjust the bad values, but do fix the stepsize
    htemp = safety * h_sub[b] * delta[b]^shrink
    q = where(htemp LT 0.0, nq) & if (nq NE 0) then stop

    ;; don't let step size drop below 1/10th previous step size
    h[moretogo[b]] = max([[htemp], [0.1*h_sub[b]]], dim=2)
  endif
  qqq = where(h LT 1e-7) & if (qqq[0] NE -1) then stop  ;; error test
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  tremain = tend - t
  moretogo = where(tremain GT timeinc, ntogo)  ;; check to see which ones aren't done
  count++ ;; step counter
  if (count mod 10 EQ 0) then print, 'Step Number: ' + strint(count) + $
    ', Points Remaining: ' + strint(ntogo)

  ;If it goes 100000 steps then it will never stop!
  done = ((ntogo EQ 0) or (count GT 100000.))
endwhile
q = where(N LT 0, nq) & if (nq GT 0) then stop

return, N

end
```

```
pro rk_5, loc, h, input, which, delta

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; This does a 5th order RK step and computes the error estimate
;; See Numerical Recipes, ch 17.2
;;
;; For each step:
;;   f_x = v
;;   f_v = a
;;   f_f = -f * ioniz
;;
;; Version History
;; * 3.1: 4/27/2011
;;       * cleaning up a bit and checking the error estimate
;; * 3.0: 7/21/10
;;       * Updating for new structure architecture
;; * 2.2: 4/26/10
;;       * The radiation pressure function only looks to see if the packet is in the
;;         planet shadow. Adding fix to check for moon shadow also
;; * 2.1: 4/26/10
;;       * xyz_to_magcoords and ionization_rate now determine whether the packet is
;;         shadowed by planets and moons. rk_5 is consistent with changes in those
;;         routines
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

hh = h # [1., 1., 1.]
hh2 = h^2 # [1., 1., 1.]

;; RK coefficients
c2 = 0.2d & c3 = 0.3d & c4 = 0.8d & c5 = 8./9.d & c6 = 1d & c7 = 1d

b1 = 35d/384d & b2 = 0d & b3 = 500d/1113d & b4 = 125d/192d & b5 = -2187d/6784d
b6 = 11d/84d & b7 = 0d
b1s = 5179d/57600d & b2s = 0d & b3s = 7571d/16695d & b4s = 393d/640d
  b5s = -92097d/339200d & b6s = 187d/2100d & b7s = 1d/40d
b1d = b1-b1s & b2d = b2-b2s & b3d = b3-b3s & b4d = b4-b4s & b5d = b5-b5s & b6d = b6-b6s
  b7d = b7-b7s

a21 = 0.2d
a31 = 3d/40d & a32 = 9d/40d
a41 = 44d/45d & a42 = -56d/15d & a43 = 32d/9d
a51 = 19372d/6561d & a52 = -25360d/2187d & a53 = 64448d/6561d & a54 = -212d/729d
a61 = 9017d/3168d & a62 = -355d/33d & a63 = 46732d/5247d & a64 = 49d/176d
  a65 = -5103d/18656d
a71 = b1 & a72 = b2 & a73 = b3 & a74 = b4 & a75 = b5 & a76 = b6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;; Step 1
;; k1 = h*f(x_n,y_n)
magcoord = xyz_to_magcoord(loc, input)
ioniz1 = ionization_rate(loc, input, magcoord)
a1 = accel(loc, input, magcoord, which)
magcoord = 0

k1x = hh**loc.v
k1v = hh**a1.dvdt
k1f = -h**loc.frac*ioniz1
a1 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 2
loc2 = {t: ptr_new(*loc.t-c2*h), $
        x: ptr_new(*loc.x+a21*k1x), $
        v: ptr_new(*loc.v+a21*k1v), $
        frac: ptr_new(*loc.frac+a21*k1f)}

magcoord = xyz_to_magcoord(loc2, input)
ioniz2 = ionization_rate(loc2, input, magcoord)
a2 = accel(loc2, input, magcoord, which)
magcoord = 0

k2x = hh**loc2.v
k2v = hh**a2.dvdt
k2f = -h**loc2.frac*ioniz2
loc2 = 0 & a2 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 3
loc3 = {t: ptr_new(*loc.t - c3*h), $
        x: ptr_new(*loc.x + a31*k1x + a32*k2x), $
        v: ptr_new(*loc.v + a31*k1v + a32*k2v), $
        frac: ptr_new(*loc.frac + a31*k1f + a32*k2f)}

magcoord = xyz_to_magcoord(loc3, input)
ioniz3 = ionization_rate(loc3, input, magcoord)
a3 = accel(loc3, input, magcoord, which)
magcoord = 0

k3x = hh**loc3.v
k3v = hh**a3.dvdt
k3f = -h**loc3.frac*ioniz3
loc3 = 0 & a3 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 4
loc4 = {t: ptr_new(*loc.t - c4*h), $
        x: ptr_new(*loc.x + a41*k1x + a42*k2x + a43*k3x), $
        v: ptr_new(*loc.v + a41*k1v + a42*k2v + a43*k3v), $
        frac: ptr_new(*loc.frac + a41*k1f + a42*k2f + a43*k3f)}

magcoord = xyz_to_magcoord(loc4, input)
ioniz4 = ionization_rate(loc4, input, magcoord)
a4 = accel(loc4, input, magcoord, which)
magcoord = 0

k4x = hh**loc4.v
k4v = hh**a4.dvdt
k4f = -h**loc4.frac*ioniz4
loc4 = 0 & a4 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 5
loc5 = {t: ptr_new(*loc.t - c5*h), $
        x: ptr_new(*loc.x + a51*k1x + a52*k2x + a53*k3x + a54*k4x), $
        v: ptr_new(*loc.v + a51*k1v + a52*k2v + a53*k3v + a54*k4v), $
        frac: ptr_new(*loc.frac + a51*k1f + a52*k2f + a53*k3f + a54*k4f)}

magcoord = xyz_to_magcoord(loc5, input)
ioniz5 = ionization_rate(loc5, input, magcoord)
a5 = accel(loc5, input, magcoord, which)
magcoord = 0

k5x = hh**loc5.v
k5v = hh**a5.dvdt
k5f = -h**loc5.frac*ioniz5
loc5 = 0 & a5 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 6
loc6 = {t: ptr_new(*loc.t - c6*h), $
        x: ptr_new(*loc.x + a61*k1x + a62*k2x + a63*k3x + a64*k4x + a65*k5x), $
        v: ptr_new(*loc.v + a61*k1v + a62*k2v + a63*k3v + a64*k4v + a65*k5v), $
        frac: ptr_new(*loc.frac + a61*k1f + a62*k2f + a63*k3f + a64*k4f + a65*k5f)}

magcoord = xyz_to_magcoord(loc6, input)
ioniz6 = ionization_rate(loc6, input, magcoord)
a6 = accel(loc6, input, magcoord, which)
magcoord = 0

k6x = hh**loc6.v
k6v = hh**a6.dvdt
k6f = -h**loc6.frac*ioniz6
loc6 = 0 & a6 = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 7
loc7 = {t: ptr_new(*loc.t - c7*h), $
        x: ptr_new(*loc.x + a71*k1x + a72*k2x + a73*k3x + a74*k4x + a75*k5x + a76*k6x), $
        v: ptr_new(*loc.v + a71*k1v + a72*k2v + a73*k3v + a74*k4v + a75*k5v + a76*k6v), $
        frac: ptr_new(*loc.frac + a71*k1f + a72*k2f + a73*k3f + a74*k4f + a75*k5f + $
          a76*k6f)}

magcoord = xyz_to_magcoord(loc7, input)
ioniz7 = ionization_rate(loc7, input, magcoord)
a7 = accel(loc7, input, magcoord, which)
destroy_structure, magcoord

k7x = hh**loc7.v
k7v = hh**a7.dvdt
k7f = -h**loc7.frac*ioniz7
ptr_free, loc7.t, loc7.x, loc7.v, loc7.frac, a7.dvdt

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 7 -- Compute the result

*loc.t = *loc.t - h
*loc.x = *loc.x + b1*k1x + b2*k2x + b3*k3x + b4*k4x + b5*k5x + b6*k6x
*loc.v = *loc.v + b1*k1v + b2*k2v + b3*k3v + b4*k4v + b5*k5v + b6*k6v
*loc.frac = *loc.frac + b1*k1f + b2*k2f + b3*k3f + b4*k4f + b5*k5f + b6*k6f

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 8 -- Estimate the error
deltax = abs(b1d*k1x + b2d*k2x + b3d*k3x + b4d*k4x + b5d*k5x + b6d*k6x + b7d*k7x)
deltav = abs(b1d*k1v + b2d*k2v + b3d*k3v + b4d*k4v + b5d*k5v + b6d*k6v + b7d*k7v)
deltaf = abs(b1d*k1f + b2d*k2f + b3d*k3f + b4d*k4f + b5d*k5f + b6d*k6f + b7d*k7f)
delta = {x:ptr_new(deltax), v:ptr_new(deltav), frac:ptr_new(deltaf)}

end
```

```
pro JupiterPlasma, loc, M, zeta, plasma_info, lam, phi, $
  ElecTherm=ElecTherm, ElecEner=ElecEner, IonTherm=IonTherm, IonEner=IonEner

common constants
common ratecoefs
common plasma

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Determines the plasma density and temperature as a function of location in
;;  the IPT
;;
;;  OUTPUTS:
;;    ElecTherm: state of the thermal electrons
;;    IonTherm: state of thermal ions
;;    ElecEner: state of the energetic electrons
;;    IonEner: state of energetic ions
;;
;;  Version History
;;    3.1: 1/31/2011
;;    3.0: 12/6/2010
;;      * updating
;;    2.0: 5/27/2009
;;      Starting over from scratch. Removing all the varibility and using a simple
;;      offset, tilted dipole. Can add other effects in later.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

num = n_elements(*loc.t)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; State of the thermal electrons
if ((kappa.eimp) and (plasma_info.thermal)) then begin
  n_e = interpol(*plasma.n_e, *plasma.L, M)
  t_e = interpol(*plasma.t_e, *plasma.L, M)
  H = interpol(*plasma.h_e, *plasma.L, M)

  hq = where(n_e LE 0, hct) & if (hct NE 0) then n_e[hq] = 0.
  hq = where(t_e LE 0.01, hct) & if (hct NE 0) then t_e[hq] = 0.01
  hq = where(H LE .1, hct)  & if (hct NE 0) then H[hq] = .1

  q = where(M GT max(*plasma.L), nq)
  if (nq NE 0) then begin
    t_e[q] = 0.01
    H[q] = .1
  endif

  n_e = n_e * exp( -(zeta/H)^2 )
  ElecTherm = {n_e:ptr_new(n_e), t_e:ptr_new(t_e)}
endif
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; State of the energetic electrons
if ((kappa.eimp) and (plasma_info.energetic)) then begin
  n_ehot = interpol(*plasmahot.n_e, *plasmahot.L, M)
  t_ehot = interpol(*plasmahot.t_e, *plasmahot.L, M)
  Hhot = interpol(*plasmahot.h_e, *plasmahot.L, M)

  hq = where(n_ehot LE 0, hct) & if (hct NE 0) then n_ehot[hq] = 0.
  hq = where(t_ehot LE 0.01, hct) & if (hct NE 0) then t_ehot[hq] = 0.01
  hq = where(HHot LE .1, hct) & if (hct NE 0) then HHot[hq] = .1

  q = where(M GT max(*plasmahot.L), nq)
  if (nq NE 0) then begin
    t_ehot[q] = 0.01
    Hhot[q] = .1
  endif

  n_ehot = n_ehot * exp( -(zeta/Hhot)^2 )
  ElecEner = {n_e:ptr_new(n_ehot), t_e:ptr_new(t_ehot)}
endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; State of the Thermal ions
if ((kappa.chx) and (plasma_info.thermal)) then begin
  nion = n_elements(kappa.ions)
  ThermDen = fltarr(num, nion)
  ThermH = fltarr(num, nion)
  for i=0,nion-1 do begin
    ;; For each ion in the rate coefficient, determine the ion density and scale height
    q = (where(*plasma.ions EQ (kappa.ions)[i]))[0]
    if (q NE -1) then begin
      ThermDen[*,i] = interpol((*plasma.n_i)[*,q], *plasma.L, M)
      ThermH[*,i] = interpol((*plasma.h_i)[*,q], *plasma.L, M)
    endif
  endfor
  hq = where(ThermDen LE 0, hct) & if (hct NE 0) then ThermDen[hq] = 0
  hq = where(ThermH LE 0.1, hct) & if (hct NE 0) then ThermH[hq] = 0.1

  ii = replicate(1., nion)
  ThermDen = ThermDen * exp(-((zeta#ii)/ThermH)^2)
  ThermT = ThermH*0.  ;; Currently do not include effects of ion thermal motion

  IonTherm = {n_i:ptr_new(ThermDen), t_i:ptr_new(ThermT)}
endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; State of the Energetic ions
;; Don't include charge exchange with hot ions
;;if ((chx) and (plasma_info.energetic)) then begin
;;  IonEner = {n_i:ptr_new(0), t_i:ptr_new(0)}
;;endif
```

```

end

```

```
pro SaturnPlasma, Lshell, maglat, loss_info, ElecTherm=ElecTherm, IonTherm=IonTherm, $
  ElecEner=ElecEner

common constants
common ratecoefs
common plasma

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  OUTPUTS:
;;    ElecTherm: state of the thermal electrons
;;    IonTherm: state of thermal ions
;;    ElecEner: state of the energetic electrons
;;    IonEner: state of energetic ions
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

num = n_elements(Lshell)

eimp = strcmp(coef_eimp.type, 'Electron Impact', /fold_case)
chx = strcmp(coef_chx.type, 'Ion-Neutral', /fold_case)

;; NOTE: Only cool ions and electrons are included for Saturn
;; indices for grid interpolation -- needed to find densities
xind = interpol(findgen(n_elements(*plasma.L)), *plasma.L, Lshell)
yind = interpol(findgen(n_elements(*plasma.latitude)), *plasma.latitude, maglat)
badL = where((Lshell LT min(*plasma.L)) or (Lshell GT max(*plasma.L)), nl)

;;;;;;;;;;;;;;;;;
;; State of electrons
if (eimp) then begin
  elecden = interpolate(*plasma.elecden, xind, yind)
  q = where(elecden LT 0, nq) & if (nq NE 0) then elecden[q] = 0.

  electemp = interpol(*plasma.electemp, *plasma.L, Lshell)
  q = where(electemp LE 0.01, nq) & if (nq NE 0) then electemp[q] = 0.01

  if (nl NE 0) then begin
    elecden[badl] = 0.
    electemp[badl] = 0.01
  endif
  ElecTherm = {n_e:ptr_new(elecden), t_e:ptr_new(electemp)}
  ElecEner = {n_e:ptr_new(-1), t_e:ptr_new(-1)}
endif

;;;;;;;;;;;;;;;;;;
;; State of the ions
;; Currently have info for H+ and W+
if (chx) then begin
  nion = n_elements(*coef_chx.ion)
  ThermDen = fltarr(num, nion)
```

```
 ThermTemp = fltarr(num, nion)
 for i=0,nion-1 do begin
  case ((*coef_chx.ion)[i]) of
   'H+': begin
    w = (where(*plasma.ions EQ 'H+'))[0]
    ThermDen[*,i] = interpolate((*plasma.ionden)[*,*,w], xind, yind) ;; Protons
    ThermTemp[*,i] = interpol((*plasma.iontemp)[*,w], *plasma.L, Lshell)
    end
   'H_2O+': begin
    w = (where(*plasma.ions EQ 'W+'))[0]
    ratio = interpol((*plasma.ratio)[*,0], *plasma.L, Lshell)
    ThermDen[*,i] = interpolate((*plasma.ionden)[*,*,w], xind, yind)*ratio ;; W+
    ThermTemp[*,i] = interpol((*plasma.iontemp)[*,w], *plasma.L, Lshell)
    end
   'O+': begin
    w = (where(*plasma.ions EQ 'W+'))[0]
    ratio = interpol((*plasma.ratio)[*,1], *plasma.L, Lshell)
    ThermDen[*,i] = interpolate((*plasma.ionden)[*,*,w], xind, yind)*ratio ;; W+
    ThermTemp[*,i] = interpol((*plasma.iontemp)[*,w], *plasma.L, Lshell)
    end
   'OH+': begin
    w = (where(*plasma.ions EQ 'W+'))[0]
    ratio = interpol((*plasma.ratio)[*,2], *plasma.L, Lshell)
    ThermDen[*,i] = interpolate((*plasma.ionden)[*,*,w], xind, yind)*ratio ;; W+
    ThermTemp[*,i] = interpol((*plasma.iontemp)[*,w], *plasma.L, Lshell)
    end
   'H_3O+': begin
    w = (where(*plasma.ions EQ 'W+'))[0]
    ratio = interpol((*plasma.ratio)[*,3], *plasma.L, Lshell)
    ThermDen[*,i] = interpolate((*plasma.ionden)[*,*,w], xind, yind)*ratio ;; W+
    ThermTemp[*,i] = interpol((*plasma.iontemp)[*,w], *plasma.L, Lshell)
    end
   else:stop
  endcase
 endfor
 hq = where(ThermDen LE 0, hct) & if (hct NE 0) then ThermDen[hq] = 0
 hq = where(ThermTemp LE 0.01, hct) & if (hct NE 0) then ThermTemp[hq] = 0.01

 if (nl NE 0) then begin
  ThermDen[badl,*] = 0.
  ThermTemp[badl,*] = 0.01
 endif
 IonTherm = {ions:ptr_new(*coef_chx.ion), n_i:ptr_new(ThermDen), $
  t_i:ptr_new(thermtemp)}
endif

end

```

```
function ionization_rate, loc, input, magcoord

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Compute the ionozation rate of the species due to each possible process
;;
;; Version History
;; 3.3: 12/13/2010
;;    * rewriting with new kappa structure
;; 3.2: 7/21/2010
;;    * rewritten with new structure architecture
;; 3.1: 4/26/10
;;    * Added support for Earth (photoionization only)
;;    * Added check for moon's shadow -- before only checked to see if the packet
;;      was in the planet's shadow
;; 3.0: original based on neutlt
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common ratecoefs

if (input.options.lifetime NE 0) $    ;;; Explicitly set
  then rate = 1./replicate(input.options.lifetime, n_elements(*loc.t)) $
  else begin
    num = n_elements(*loc.t)

    ;; Get plasma parameters
    dotherm = 0
    doener = 0
    case (input.geometry.planet) of
      'Mercury':
      'Earth':
      'Jupiter': begin
        JupiterPlasma, loc, *magcoord.M, *magcoord.zeta, input.plasma_info, $
          *magcoord.lam, ElecTherm=ElecTherm, ElecEner=ElecEner, IonTherm=IonTherm
        dotherm = 1 & doener = 1
        end
      'Saturn': begin
        SaturnPlasma, *magcoord.M, *magcoord.zeta, ElecTherm=ElecTherm, $
          IonTherm=IonTherm, ElecEner=ElecEner
        dotherm = 1
        doener = 0
        end
      else: stop
    endcase

    chxrate = dblarr(num)

    ;; Compute photo-loss rate
    if (kappa.photo) then begin
      if (n_elements(*magcoord.out_of_shadow) NE num) then stop
      photorate = double(*magcoord.out_of_shadow * *kappa.kappa_photo)
    endif else photorate = dblarr(num)
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    ;; Compute electron impact rate
    eimprate = dblarr(num)
    if ((kappa.eimp) and (dotherm)) then begin
      K = loginterpol(*kappa.kappa_ei, *kappa.t_e, *ElecTherm.t_e)
      w = where((*ElecTherm.n_e GE 0) and (K GT 0), ctw)
      if (ctw NE 0) then eimprate[w] = (*ElecTherm.n_e)[w] * K[w]
      destroy_structure, ElecTherm
    endif

    if ((kappa.eimp) and (doener)) then begin
      K = loginterpol(*kappa.kappa_ei, *kappa.t_e, *ElecEner.t_e)
      w = where((*ElecEner.n_e GE 0) and (K GT 0), ctw)
      if (ctw NE 0) then eimprate[w] += (*ElecEner.n_e)[w] * K[w]
      destroy_structure, ElecEner
    endif
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    ;; Compute charge exchange rate
    if (kappa.chx) then begin
      ;; Calculate relative velocity
      Bvx = -DipoleConsts.magrat * (*loc.x)[*,1]
      Bvy = DipoleConsts.magrat * (*loc.x)[*,0]
      vrel = (sqrt(((*loc.v)[*,0]-Bvx)^2 + ((*loc.v)[*,1]-Bvy)^2 + (*loc.v)[*,2]^2)) $
        * SystemConsts.rplan
      q = where(vrel GT max(*kappa.v_rel), nq)
      if (nq NE 0) then vrel[q] = max(*kappa.v_rel)
      q = where(vrel LT min(*kappa.v_rel), nq)
      if (nq NE 0) then vrel[q] = min(*kappa.v_rel)

      ;; Compute the rate
      for kk=0,n_elements(kappa.ions)-1 do chxrate += (*IonTherm.n_i)[*,kk] * $
        interpolate_xy((*kappa.kappa_chx)[*,*,kk], *kappa.t_i, *kappa.v_rel, $
        (*IonTherm.t_i)[*,kk], vrel)
      destroy_structure, IonTherm
    endif

    rate = photorate + eimprate + chxrate
    q = where(rate EQ 0, nq) & if (nq NE 0) then rate[q] = 1d-30
  endelse

;Return the lifetimes
q = where(finite(rate) EQ 0) & if (q[0] NE -1) then stop
return, rate

end

```

```
function lifetime_setup, input

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Set up the default reactions and plasma for use later
;; Function returns loss_info which is only used to keep track of what
;; reactions were used in running the model
;;
;; Version History
;;   3.1: 12/9/2010
;;     * new rate coeficient structure
;;   3.0: 7/19/2010
;;     * Rewriting with input structure
;;   2.4: 4/26/2010
;;     * Added option for Earth - photoionization only
;;   2.3: 9/14/2009
;;     * moved load_plasma section to separate program
;;   2.2: 5/27/2009
;;     * replaced GetReactionList with create_lossinfo
;;     * added Jupiter plasma
;;   2.1: 4/23/2009
;;     * For Mercury, changing the routine so that I can bock photoionization
;;       in the shadow. Will keep the coef_photo structure and ionization rk5 will
;;       call ionization_rate
;;   2.0: 10/22/2008 (MHB)
;;     * Routine created.
;;     * Need to add Jupiter plasma
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common ratecoefs
common plasma

;; Find the default reactions
restore, !model.basepath + 'Work/AtomicData/Defaults.Loss.sav'

;; Figure out which reactions to use
q = where(defaults.species EQ input.options.atom, nq)
if (nq EQ 0) then stop
loss_info = defaults[q]

pp = where(strcmp(loss_info.mechanism, 'photo', /fold), np)
ie = where(strcmp(loss_info.mechanism, 'Electron Impact', /fold), nie)
in = where(strcmp(loss_info.mechanism, 'Ion-Neutral', /fold), nin)
case (input.geometry.planet) of
  'Mercury': loss_info = reform(loss_info[pp]) ;; only use photoreactions for now
  'Earth': loss_info = reform(loss_info[pp]) ;; only use photoreactions for now
  else: begin ;; Load the plasma
    load_plasma, input.geometry.planet, input.plasma_info, plasma=plasma, $
      hotplasma=plasmahot
    ionlist = [*plasma.ions, *plasmahot.ions]
    ionlist = ionlist[uniq(ionlist, sort(ionlist))]
  endelse
endcase
pp = where(strcmp(loss_info.mechanism, 'photo', /fold), np)
ie = where(strcmp(loss_info.mechanism, 'Electron Impact', /fold), nie)
in = where(strcmp(loss_info.mechanism, 'Ion-Neutral', /fold), nin)

;; Load the photo-reaction rate coefficients
photrate = 0d
for i=0,np-1 do begin
  restore, !model.basepath + strmid(loss_info[pp[i]].file, strlen('/Users/mburger/'))
  photrate += ratecoef.kappa / stuff.aplanet^2
  destroy_structure, ratecoef
endfor

;; Load the electron impact rate coefficients
if (nie GT 0) then begin
  electemp = 10.^(dindgen(41)/20.)  ;; electrons valid for 1 eV < t_e < 100 eV
  eimpcoef = 0d
  for i=0,nie-1 do begin
    restore, !model.basepath + strmid(loss_info[ie[i]].file, strlen('/Users/mburger/'))
    eimpcoef += loginterpol(*ratecoef.kappa, *ratecoef.t_e, electemp)
    destroy_structure, ratecoef
  endfor
endif else begin
  electemp = 0. & eimpcoef = 0.
endelse

;; Load the ion-neutral rate coefficeints
if (nin GT 0) then begin
  vrel = dindgen(101)*2                 ;; 0 km/s < v_rel < 200 km/s
  iontemp = dindgen(101)
  chxcoef = dblarr(101,101,n_elements(ionlist))
  for i=0,nin-1 do begin
    restore, !model.basepath + strmid(loss_info[in[i]].file, strlen('/Users/mburger/'))
    q = (where(ionlist EQ ratecoef.ion, nq))[0]
    if (nq GT 1) then stop ;; problem
    if (nq EQ 1) then chxcoef[*,*,q] += interpolate_xy(*ratecoef.kappa, *ratecoef.t_i, $
      *ratecoef.v_rel, iontemp, vrel, /grid)
    print, ratecoef.reaction, '  ', q
    destroy_structure, ratecoef
  endfor
endif else begin
  vrel = 0. & iontemp = 0. & chxcoef = 0. & ionlist = ''
endelse

kappa = {photo:(photrate NE 0), eimp:(nie GT 0), chx:(nin GT 0), $
  kappa_photo:ptr_new(photrate), t_e:ptr_new(electemp), $
  kappa_ei:ptr_new(eimpcoef), ions:ionlist, t_i:ptr_new(iontemp), $
  v_rel:ptr_new(vrel), kappa_chx:ptr_new(chxcoef)}
```

```

return, loss_info

end
```

```
pro load_plasma, planet, plasma_info, plasma=plasma, hotplasma=plasmahot

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Load the plasma and make necessary adjustments
;;
;; Version History:
;;   2.1: 12/6/2010
;;      * some small adjustments
;;   2.0: created 9/14/2009
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

case (planet) of
  'Jupiter': begin
    restore, !model.basepath + 'Work/Jupiter/PlasmaTorus/VoyagerTorus/VoyagerTorus.sav'
    elecscale = *plasma.h_e
    q = where(finite(elecscale) EQ 0, comp=c)
    elecscale[q] = elecscale[c[0]]
    plasma = create_struct('elecscale', ptr_new(elecscale), plasma)

    hotscale = *plasmahot.h_e
    q = where(finite(hotscale) EQ 0, comp=c)
    hotscale[q] = hotscale[c[0]]
    plasmahot = create_struct('elecscale', ptr_new(hotscale), plasmahot)
    end
  'Saturn': begin
    restore, !model.basepath + $
      'Work/NeutralModel/modelpro_2.0/data/CAPSplasma/SaturnPlasma.2008-04-16.sav'
    *plasma.elecden *= plasma_info.ElecDenMod   ;; constant elec den changes
    *plasma.electemp *= plasma_info.ElecTempMod ;; constant elec temp changes
    end
  else:
endcase

end
```

```
function xyz_to_magcoord, loc, input

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes the position of each packet in the torus coordinates M and zeta
;;
;; Inputs:
;;   * *loc.x, *loc.y, *loc.z = cartesian coordinates of packets (R_J)
;;   * phi = orbital longitude of packets (radians)
;;   * lam = magnetic longitude of packets (radians)
;;   * consts = list of magnetic dipole constants
;;   * plamsa_info = contains plasma torus information
;; Outputs:
;;   M = M shell (modified L shell) (R_J)
;;   zeta = distance along field line from centrifugal equator to packet (R_J)
;;   L = true L shell (R_J)
;;
;; Version History
;;   3.1: 4/27/2011
;;     * changing out_of_shadow -- does the planet but not moons
;;   3.0: 7/21/2010
;;     * Updating for new structure architecture
;;   -- 4/26/10 -- Added empty case statement 'Earth'
;;   2.0: 5/27/2009
;;     * Fixed issues with position in IPT
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

case (input.geometry.planet) of
  'Mercury': magcoord = {out_of_shadow:ptr_new(0)}
  'Earth': magcoord = {out_of_shadow:ptr_new(0)}
  'Jupiter': begin
    magcoord = {L:ptr_new(0), M:ptr_new(0), zeta:ptr_new(0), lam:ptr_new(0), $
      out_of_shadow:ptr_new(0)}

    ;; See notes from 2008-05-13 for full description of this calculation.
    locx = (*loc.x)[*,0]
    locy = (*loc.x)[*,1]
    locz = (*loc.x)[*,2]
    phi = atan(-locx, locy)
    CML = input.geometry.cml - DipoleConsts.magrat*(*loc.t)  ;; current CML
    *magcoord.lam = CML - phi + !pi

    alpha = -DipoleConsts.tilt * cos(*magcoord.lam-DipoleConsts.lam3) ;angle of B equator

    ;;; Location of the dipole center in xyz
    lam_d = CML - DipoleConsts.offlong
    delx = DipoleConsts.offset * sin(lam_d)
    dely = -DipoleConsts.offset * cos(lam_d)
    delz = 0.

    ;; Positions relative to center of dipole
    xx = locx - delx
    yy = locy - dely
    zz = locz - delz

    ;; Account for E/W electric field
    r0 = sqrt(xx^2 + yy^2 + zz^2)
    xx -= input.plasma_info.eps*R0  ;; E/W electric field effectively moves packets east
    r1 = sqrt(xx^2 + yy^2 + zz^2) ;; Recompute distance from center

    ;;; Determine L
    orblat = asin(zz/r0)
    maglat = orblat - alpha
    centlat = orblat - 2./3.*alpha ;; centrifugal latitude
    *magcoord.L = r1 / (cos(maglat))^2
      ;; M = L * (cos(alpha/3.))^2 ;; M = dist from Jup that field line hits cent. eq.
      ;;; Don't actually want L since need the centrifugal equator
      ;; The Mag latitude of the centrifugal equator is alpha/3.
    *magcoord.M = *magcoord.L * cos(alpha/3.)^2

    ;; Determine zeta -- perp distance from packet to cent. equator
    *magcoord.zeta = r1 * sin(centlat)

    ;;;; Determine zeta -- old way
;;    cos2lat = sqrt(5.-3*cos(2*latD))
;;    cos2th = sqrt(5.-3*cos(2*theta))
;;
 ;;   x1 = sqrt(6.)*sinlat/cos2lat
 ;;   atanhx = .5 * alog((1.+x1)/(1.-x1))
 ;;   analy1 = ( sqrt((coslat)^4 + 4.*(coslat)^2*(sinlat)^2 )) * $
 ;;     ( atanhx / (sqrt(6.) * coslat * cos2lat) + sinlat/coslat/2. )
;;
;;    x2 = sqrt(6.)*sintheta/cos2th
 ;;   atanhx = .5 * alog((1.+x2)/(1.-x2))
 ;;   analy2 = ( sqrt((costheta)^4 + 4.*(costheta)^2*(sintheta)^2 )) * $
 ;;     ( atanhx / (sqrt(6.) * costheta * cos2th) + sintheta/costheta/2. )
;;
;;    zeta = L * (analy1-analy2)
    end
  'Saturn': begin
    magcoord = {L:ptr_new(0), M:ptr_new(0), zeta:ptr_new(0), out_of_shadow:ptr_new(0)}
    r0 = sqrt( ((*loc.x)[*,0])^2 + ((*loc.x)[*,1])^2 + ((*loc.x)[*,2])^2)
    *magcoord.zeta = asin((*loc.x)[*,2]/r0) ;; magnetic latitude
    *magcoord.L = r0 / (cos(zeta))^2
    *magcoord.M = *magcoord.L
    end
  'Pluto': magcoord = {out_of_shadow:ptr_new(0)}
endcase

```

```
;; Check to see in packets are shadowed by planet or a moon
rho = sqrt((*loc.x)[*,0]^2 + (*loc.x)[*,2]^2)
*magcoord.out_of_shadow = ((rho GT 1) or ((*loc.x)[*,1] LT 0))

return, magcoord

end

```

```
function BennaPrecipitationFilename, orbit, mnum, proton=proton, $
  electron=electron, params=params

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; mnum =
;;   0: inbound IMF conditions, Best fit
;;   1: outbound IMF conditions, Best fit
;;   2: inbound IMF conditions, low density
;;   3: inbound IMF conditions, medium density
;;   4: inbound IMF conditions, high density
;;   5: outbound IMF conditions, low density
;;   6: outbound IMF conditions, medium density
;;   7: outbound IMF conditions, high density
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (proton EQ !null) then proton = 0
if (electron EQ !null) then electron = 0
if (proton+electron EQ 0) then proton = 1
if (proton+electron NE 1) then stop

if (mnum EQ !null) then begin
  print, '  0 = Inbound IMF, Best fit'
  print, '  1 = Outbound IMF, Best fit'
  print, '  2 = Inbound IMF, low density SW'
  print, '  3 = Inbound IMF, medium density SW'
  print, '  4 = Inbound IMF, high density SW'
  print, '  5 = Outbound IMF, low density SW'
  print, '  6 = Outbound IMF, medium density SW'
  print, '  7 = Outbound IMF, high density SW'
  read, mnum, prompt='Enter the IMF conditions: '
endif
if ((mnum LT 0) or (mnum GT 7)) then stop

case (mnum) of
  0: begin
     imfstr = 'Inbound IMF, best fit'
     o = 0
     end
  1: begin
     imfstr = 'Outbound IMF, best fit'
     o = 1
     end
  2: begin
     imfstr = 'Inbound IMF, low density'
     o = 0
     end
  3: begin
     imfstr = 'Inbound IMF, medium density'
     o = 0
```

```
      end
  4: begin
      imfstr = 'Inbound IMF, high density'
      o = 0
      end
  5: begin
      imfstr = 'Outbound IMF, low density'
      o = 1
      end
  6: begin
      imfstr = 'Outbound IMF, medium density'
      o = 1
      end
  7: begin
      imfstr = 'Outbound IMF, high density'
      o = 1
      end
endcase

;; Determine which model to use
restore, !model.basepath + 'Work/Data/surfacemaps/Mercury/PrecipModelCrossRef.sav'

q = (where(*precip_orbit.orbit EQ orbit, nq))[0]
if (nq NE 1) then stop

modelnumber = (*precip_orbit.models)[mnum,q]

if (modelnumber NE -1) then begin
  modelden = (*precip_orbit.mod_den)[mnum,q]
  modelBx = (*precip_orbit.mod_Bx)[mnum,q]
  modelBy = (*precip_orbit.mod_By)[mnum,q]
  modelBz = (*precip_orbit.mod_Bz)[mnum,q]

  Bx = (*precip_orbit.Bx)[o,q]
  By = (*precip_orbit.By)[o,q]
  Bz = (*precip_orbit.Bz)[o,q]

  ;; Determine name of precipitation file
  case (1) of
    (modelnumber LT 10): mstr = '000' + strint(modelnumber)
    (modelnumber LT 100): mstr = '00' + strint(modelnumber)
    (modelnumber LT 1000): mstr = '0' + strint(modelnumber)
    else: mstr = strint(modelnumber)
  endcase

  part = (proton) ? 'Proton' : 'Electron'
  filename = !model.basepath + 'Work/Data/surfacemaps/Mercury/' + $
    part + 'Precipitation/' + mstr + '.' + part + '.sav'

  params = {orbit:orbit, IMF:imfstr, model:modelnumber, filename:filename, $
    modelden:modelden, modelbx:modelbx, modelby:modelby, modelbz:modelbz, $
    bx:bx, by:by, bz:bz}

  print, 'Orbit # = ' + strint(orbit)
  print, 'Model # = ' + strint(modelnumber)
  print, 'IMF conditions: ' + imfstr
  print, 'Model Density = ' + strint(modelden)
  print, 'Bx: Observed = ' + strint(Bx) + ' Modeled = ' + strint(modelbx)
  print, 'By: Observed = ' + strint(By) + ' Modeled = ' + strint(modelby)
  print, 'Bz: Observed = ' + strint(Bz) + ' Modeled = ' + strint(modelbz)
endif else begin
  filename = ''
  print, 'Orbit # = ' + strint(orbit)
  print, 'IMF conditions: ' + imfstr
  print, 'No model satisfies these conditions'
endelse

return, filename

end
```

```
function MercuryModelEndTime, atoms, taa

na = n_elements(atoms) & nt = n_elements(taa)
SystemConstants, 'Mercury', c
planet_dist, taa, c, d=rr, v=vv

data = search_atomicdata()
result = dblarr(nt,na)
for i=0,na-1 do begin
  q = (where((data.mechanism EQ 'photo') and (data.species EQ atoms[i]), nq))[0]
  if (nq NE 1) then stop

  print, data[q].file
  restore, data[q].file
  if (n_elements(ratecoef.kappa) NE 1) then stop
  kappa = ratecoef.kappa / rr^2
  life = 1./kappa
  result[*,i] = life * 4.
endfor

return, reform(result)

end
```

```
pro combine_iterations, filename, filelist, delete=delete

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Combine the individual interations of a model run into a single file
;; Set keyword start if this is a startloc structure
;; Set keyword after if combining after all model runs are completed
;;
;; * filename is the name of the new savefile
;; * If filelist is given, it is an array with the names of files to combine
;; * If filelist is not given, then it looks for all the filename.#
;;
;; Version history:
;;   3.2: 4/27/2011
;;       * trackloss is now optional
;;   3.1: 1/6/2011
;;       * Changing the way it finds files
;;   3.0: 7/21/2010
;;       * rewriting for new structure architecture
;;   2.3: 19 Jan 2010
;;       * Combines the deposition and hitfrac, etc. fields.
;;   2.2: 20 November 2009
;;       * Saves the final structures with single precision floating point rather than
;;         double precision.
;;   2.0: Rewritten to conform with new structure definitions.
;;   1.0: originial
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (delete EQ !null) then delete = 0

if (filelist EQ !null) then begin
  filelist = file_search(filename+'.*', count=numit)
  savenames = 0
endif else begin
  numit = n_elements(filelist)
  savenames = 1
endelse

case (numit) of
  0: stop
  1: spawn, 'cp ' + filelist + ' ' + filename
  else: begin
    packets = 0
    for it=0,numit-1 do begin
      restore, filelist[it]
      if (packets EQ 0) $
        then begin
          outnew = temporary(output)
          innew = temporary(input)
          vernew = temporary(version)
        endif else begin
          *outnew.x0 = float([*outnew.x0, *output.x0])
          *outnew.y0 = float([*outnew.y0, *output.y0])
          *outnew.z0 = float([*outnew.z0, *output.z0])
          *outnew.f0 = float([*outnew.f0, *output.f0])

          *outnew.vx0 = float([*outnew.vx0, *output.vx0])
          *outnew.vy0 = float([*outnew.vy0, *output.vy0])
          *outnew.vz0 = float([*outnew.vz0, *output.vz0])

          *outnew.phi0 = float([*outnew.phi0, *output.phi0])
          *outnew.time = float([*outnew.time, *output.time])

          *outnew.x = float([*outnew.x, *output.x])
          *outnew.y = float([*outnew.y, *output.y])
          *outnew.z = float([*outnew.z, *output.z])
          *outnew.frac = float([*outnew.frac, *output.frac])

          *outnew.vx = float([*outnew.vx, *output.vx])
          *outnew.vy = float([*outnew.vy, *output.vy])
          *outnew.vz = float([*outnew.vz, *output.vz])

          if (input.options.trackloss) then begin
            *outnew.lossfrac = float([*outnew.lossfrac, *output.lossfrac])
            *outnew.ringfrac = float([*outnew.ringfrac, *output.ringfrac])
            *outnew.leftfrac = float([*outnew.leftfrac, *output.leftfrac])

            s = size(*outnew.hitfrac)
            if (s[0] EQ 1) $
              then *outnew.hitfrac = float([*outnew.hitfrac, *output.hitfrac]) $
              else begin
                temp = fltarr(n_elements(*outnew.x),s[2])
                temp[0:s[1]-1,*] = float(*outnew.hitfrac)
                temp[s[1]:*,*] = float(*output.hitfrac)
                *outnew.hitfrac = temp
              endelse
            *outnew.deposition.map += *output.deposition.map
          endif

          outnew.totalsource = total(double(*outnew.f0))

          if (savenames) then *outnew.sourcefile = [*outnew.sourcefile, $
            *output.sourcefile]

          destroy_structure, output
          destroy_structure, input
        endelse
      packets = n_elements(*outnew.x)
    endfor

    output = temporary(outnew)
    input = temporary(innew)
    version = temporary(vernew)

    save, output, input, version, file=filename
    destroy_structure, output
    destroy_structure, input
  end
endcase

make_model_header, filename

;; Delete intermediate filelist
if (delete) then for i=0,numit-1 do spawn, 'rm ' + filelist[i]

end

```

```
function compare_inputs_value, value0, value1, tagname

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Compares two values when given a tagname
;; For floating point values, an acceptable tolerance is specified.
;; Otherwise, the values have to be exact
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

same = array_equal(value0, value1) ;; first do quick check
if ~(same) then begin  ;; Now check to see if things are close
  s = (n_elements(value0) EQ n_elements(value1))
  if (s) then begin
    case strlowcase(tagname) of
      'phi': tol = 3.0*!dtor  ;; 5 degree tolerance
      'taa': tol = 3.0*!dtor  ;; 5 degree tolerance
      'lifetime': tol = min([value0,value1]) * 0.05  ;; 5% tolerance
      'endtime': tol = min([value0,value1]) * 0.05  ;; 5% tolerance
      'temperature': tol = 1.*(value0 NE 0)*(value1 NE 0) ; 1 deg unless one = 0
      'outeredge': tol = min([value0,value1]) * 0.05 ;; 5% tolerance
      'longitude': tol = 1*!dtor
      'latitude': tol = 1*!dtor
      'stickcoef': tol = 0.01
      'accom_factor': tol = 0.01
      'kappa': tol = 0.1
      'diffusionlimit': tol = min([value0,value1]) * 0.01
      else: stop
    endcase
    same = (max(abs(value0-value1)) LT tol)
  endif else same = 0  ;; different number of elements
endif

return, same

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function compare_inputs, input0temp, input1temp, verbose=verbose

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Make a comparison between two input files
;;
;; Version 3.0: 21 Dec 2010
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (verbose EQ !null) then verbose = 0
```

```

input0 = (isa(input0temp, 'string')) ? inputs_restore(input0temp) : input0temp
input1 = (isa(input1temp, 'string')) ? inputs_restore(input1temp) : input1temp

;; Check top level structure
t0 = tag_names(input0) & t1 = tag_names(input1)
q0 = sort(t0) & q1 = sort(t1)

same = 1
if (array_equal(t0[q0], t1[q1])) then begin
  ;; Compare next level down
  i = 0
  while ((i LT n_elements(t0)) and (same)) do begin
    if (~strcmp(t0[q0[i]], t1[q1[i]])) then stop
    struct0 = input0.(q0[i]) & struct1 = input1.(q1[i])
    s0 = tag_names(struct0) & s1 = tag_names(struct1)
    w0 = sort(s0) & w1 = sort(s1)

    if (array_equal(s0[w0], s1[w1])) then begin
      ;; Compare values of each tag
      j = 0
      while ((j LT n_elements(s0)) and (same)) do begin
        if (~strcmp(s0[w0[j]], s1[w1[j]])) then stop

        ;; make sure types are the same, lengths are the same, values are the same
        value0 = struct0.(w0[j]) & value1 = struct1.(w1[j])
        type0 = size(value0, /type) & type1 = size(value1, /type)
        case (1) of
          (type0 NE type1): same = 0
          (type0 EQ 10): $ ;; value0 and value1 are pointers
            same = compare_inputs_value(*value0, *value1, s0[w0[j]])
          (type0 EQ 4) or (type0 EQ 5): $ ;; value0 and value1 are floats
            same = compare_inputs_value(value0, value1, s0[w0[j]])
          else: same = array_equal(value0, value1) ;; byte or integer
        endcase
        if (~same and verbose) then print, 'failed at = ' + s0[w0[j]], xxx
        j++
      endwhile
      i++
    endif else begin
      same = 0
      if (verbose) then print, 'failed at ' + t0[q0[i]], xxx
    endelse
  endwhile
endif else begin
  same = 0
  if (verbose) then begin
    q = strcmp(t0[q0], t1[q1])
    w = where(q EQ 0, nw)
    print, 'Failed at Top Level', xxx
    for i=0,nw-1 do $
```

```
      print, '  input0.' + t0[q0[w[i]]] + '; input1.' + t1[q1[w[i]]], xxx
    endif
  endelse

  return, same

  end
```

```
function extract_distribution, genericfiles, tempinput, firstfile=firstfile

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Takes a model output file with an isotropic surface distribution and flat
;;  speed distribution and creates a new outputfile with a specified speed and
;;  surface distribution
;;
;;  Required Inputs:
;;    genericfiles: list of generic output files
;;    input: new inputs structure
;;    firstfile: name of the first file to save (will increment automatically)
;;       If not given, then chooses file based on inputs
;;
;;  Version History:
;;    3.3: 7/8/2011
;;      * added PSD spatial distribution
;;    3.2: 4/27/2011
;;      * trackloss is now optional
;;    3.1: 1/6/2011
;;      * now extracts the distribution from a list of files and combines them
;;        to make a series of files with ~10^6 packets in each
;;    3.0: 7/16/2010
;;      * rewriting to make use of inputs structure
;;    2.1: 3/10/2010
;;      * allowing option of saving the results (which is slow) or just outputing them
;;          with keywords
;;    2.0: Created. 12/10/2009
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

if ~(file_test('tempoutput/')) then file_mkdir, 'tempoutput'

;; Remember these values for later
temptime = tempinput.options.endtime
temptaa = tempinput.geometry.taa

ngen = n_elements(genericfiles)
tempfiles = strarr(ngen)
packets = lonarr(ngen)
oldtaa = dblarr(ngen)
for i=0,ngen-1 do begin
  tempfiles[i] = 'tempoutput/temp' + strint(i) + '.output'

  ;; restore a generic file
  restore, genericfiles[i]
  oldinput = temporary(input)
  oldtaa[i] = oldinput.geometry.taa

  ;; This is the new input
```

```
input = temporary(tempinput)

;; use the endtime and taa from the generic file
input.options.endtime = oldinput.options.endtime
input.geometry.taa = oldinput.geometry.taa

ss = (where(*SystemConsts.objects EQ input.geometry.startpoint))[0]

;;;;;;;;;;;;;;;;
;; Determine the new surface distribution
SpatialDist = input.SpatialDist
if (ss EQ 0) then begin
  longitude = atan(*output.x0, -*output.y0)
  latitude = asin(*output.z0/SpatialDist.exobase)
endif else begin
  longitude = atan(-*output.x0, -*output.y0)
  latitude = asin(*output.z0/SpatialDist.exobase)
endelse
longitude = (longitude + 2*!pi) mod (2*!pi)
q = where(finite(longitude) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(latitude) EQ 0, nq) & if (nq NE 0) then stop

case strlowcase(SpatialDist.type) of
  'surface': begin
    case (1) of
      (SpatialDist.use_map): begin
        restore, SpatialDist.mapfile
        q = where(finite(*sourcemap.map) EQ 0, nq) & if (nq NE 0) then stop
        newf_spat = interpolate_xy(*sourcemap.map, *sourcemap.longitude, $
          *sourcemap.latitude, longitude, latitude)
        destroy_structure, sourcemap
        end
      (SpatialDist.longitude)[0] LE (SpatialDist.longitude)[1]: $
        newf_spat = ((longitude GE (SpatialDist.longitude)[0]) and $
          (longitude LE (SpatialDist.longitude)[1]) and $
          (latitude GT (SpatialDist.latitude)[0]) and $
          (latitude LE (SpatialDist.latitude)[1]))
      (SpatialDist.longitude)[0] GT (SpatialDist.longitude)[1]: $
        newf_spat = ((longitude GE (SpatialDist.longitude)[0]) or $
          (longitude LE (SpatialDist.longitude)[1])) and $
          (latitude GT (SpatialDist.latitude)[0]) and $
          (latitude LE (SpatialDist.latitude)[1])
      else: stop
    endcase
    end
  'psd': begin
    sourcemap = PSDfluxmap(input)
    *sourcemap.map /= max(*sourcemap.map)
    newf_spat = interpolate_xy(*sourcemap.map, *sourcemap.longitude, $
      *sourcemap.latitude, longitude, latitude)
    destroy_structure, sourcemap
```

```
        end
      else: stop
    endcase
    if (max(newf_spat) EQ 0) then stop
    newf_spat /= max(newf_spat) ;; normalize to 1
    q = where(newf_spat LT 0, nq) & if (nq GT 0) then newf_spat[q] = 0
    q = where(finite(newf_spat) EQ 0, nq) & if (nq GT 0) then stop

    ;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Determine the new speed distribution
    vold = sqrt(*output.vx0^2 + *output.vy0^2 + *output.vz0^2)*SystemConsts.rplan
    SpeedDist = input.SpeedDist
    newf_vel = SpeedDistribution(input, vold)
    if (max(newf_vel) EQ 0) then stop
    q = where(newf_vel LT 0, nq) & if (nq GT 0) then newf_vel[q] = 0
    q = where(finite(newf_vel) EQ 0, nq) & if (nq GT 0) then stop

    ;;;;;;;;;;;;;;;;;;
    ;; Determine the new angular distribution
    AngularDist = input.AngularDist

    zenang = (*output.x0**output.vx0 + *output.y0**output.vy0 + $
      *output.z0**output.vz0)/sqrt(*output.x0^2+*output.y0^2+*output.z0^2)/$
      sqrt(*output.vx0^2+*output.vy0^2+*output.vz0^2)
    q = where(zenang LT 0L, nq) & if (nq GT 0) then zenang[q] = 0
    q = where(zenang GT 1L, nq) & if (nq GT 0) then zenang[q] = 1
    altitude = !pi/2-acos(zenang)
    q = where(abs(altitude) GT !pi/2., nq) & if (nq GT 0) then stop

    azimuth = fltarr(n_elements(*output.x0))

    case (AngularDist.type) of
      'radial': stop
      'isotropic': newf_ang = ((altitude GE (AngularDist.altitude)[0]) and $
        (altitude LE (AngularDist.altitude)[1]) and $
        (azimuth GE (AngularDist.azimuth)[0]) and $
        (azimuth LE (AngularDist.azimuth)[1]))
      'costheta': begin
        ;; tested this -- can reproduce costheta distribution in angular_distribution.pro
        inrange = ((altitude GE (AngularDist.altitude)[0]) and $
          (altitude LE (AngularDist.altitude)[1]) and $
          (azimuth GE (AngularDist.azimuth)[0]) and $
          (azimuth LE (AngularDist.azimuth)[1]))
        newf_ang = sin(altitude)^AngularDist.n * inrange
        end
    endcase
    if (max(newf_ang) EQ 0) then stop
    newf_ang /= max(newf_ang)
    q = where(newf_ang LT 0, nq) & if (nq GT 0) then newf_ang[q] = 0
    q = where(finite(newf_ang) EQ 0, nq) & if (nq GT 0) then stop

```

```
weight = float(newf_vel*newf_spat*newf_ang)
if (max(weight) EQ 0) then stop
weight/= max(weight)
q = where(weight LT 0, nq) & if (nq GT 0) then stop
q = where(finite(weight) EQ 0, nq) & if (nq GT 0) then stop

*output.f0 *= weight
*output.frac *= weight
q = where(*output.f0 GE 1e-6, nq)
packets[i] = nq

*output.x0 = (*output.x0)[q]
*output.y0 = (*output.y0)[q]
*output.z0 = (*output.z0)[q]
*output.f0 = (*output.f0)[q]
*output.vx0 = (*output.vx0)[q]
*output.vy0 = (*output.vy0)[q]
*output.vz0 = (*output.vz0)[q]
*output.phi0= (*output.phi0)[q]
*output.time = (*output.time)[q]
*output.x = (*output.x)[q]
*output.y = (*output.y)[q]
*output.z = (*output.z)[q]
*output.frac = (*output.frac)[q]
*output.vx = (*output.vx)[q]
*output.vy = (*output.vy)[q]
*output.vz = (*output.vz)[q]
*output.sourcefile = genericfiles[i]
output.totalsource = total(double(*output.f0))

if (input.options.trackloss) then begin
  *output.lossfrac = (*output.lossfrac)[q]
  *output.ringfrac = (*output.ringfrac)[q]
  *output.leftfrac = (*output.leftfrac)[q]
  s = (size(*output.hitfrac))[0]
  if (s EQ 1) $
    then *output.hitfrac = (*output.hitfrac)[q] $
    else *output.hitfrac = (*output.hitfrac)[q,*]
  *output.deposition.longitude = 0
  *output.deposition.latitude = 0
  *output.deposition.map = 0
endif

;; Note -- not saving deposition
save, input, output, version, file=tempfiles[i]

destroy_structure, output
tempinput = temporary(input)
tempinput.geometry.taa = temptaa
tempinput.options.endtime = temptime
print, 'extracted file ' + strint(i+1) + ' of ' + strint(ngen)
```

```
endfor

;; Now go through and combine files to get ~10^6 packets per file
totpack = packets
i = 0 & cur = 0
curpack = 0L & maxpack = 1000000L
fnumber = lonarr(ngen)
while (i LT ngen) do begin
  curpack += packets[i]
  if (curpack LE maxpack) then begin
    fnumber[i] = cur
    i++
  endif else begin
    cur++
    curpack = 0L
  endelse
endwhile

n = max(fnumber)
newpack = lonarr(n+1)
outputfiles = strarr(n+1)
for i=0,n do begin
  tempinput.geometry.taa = oldtaa[i]
  q = where(fnumber EQ i, nq)
  if (firstfile EQ !null) $
    then outputfiles[i] = output_filename(tempinput) $
    else stop
  combine_iterations, outputfiles[i], tempfiles[q]
  print, 'Created file: ' + file_basename(outputfiles[i]) + ' out of ' + $
    strint(nq) + ' smaller files.'
  print, 'Number of packets = ' + strint(total(packets[q]))
endfor
tempinput.geometry.taa = temptaa

;; Remove the temporary files
spawn, 'rm -r tempoutput'

return, outputfiles

end
```

```
function extract_parameter, parameter, filelist

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Searches through the given filelist for the specified parameter
;; Returns the list as a hash with key=filename, value=scalar or array
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

temp = filelist
w = where(stregex(filelist, '.output', /bool), nw)
if (nw GT 0) then temp[w] = headername(filelist[w])

if (n_elements(temp) EQ 1) then filetemp = temp else begin
  filetemp = temp[0]
  for i=1,n_elements(temp)-1 do filetemp += ' ' + temp[i]
endelse

spawn, 'grep -ir ' + parameter + ' ' + filetemp, list, error
if (error NE '') then stop
if (list[0] EQ '') then result = !null else begin
  res0 = strarr(n_elements(list))
  res1 = strarr(n_elements(list))
  for i=0,n_elements(list)-1 do begin
    x = strsplit(list[i], ':|=', /regex, /extract)
    if (n_elements(x) EQ 2) then begin
      res0 = filelist
      res1 = strtrim(x[1], 2)
    endif else begin
      res0[i] = strtrim(x[0], 2)
      res1[i] = strtrim(x[2], 2)
    endelse
  endfor

  u = uniq(res0, sort(res0)) & nu = n_elements(u)
  if (nu NE n_elements(list)) then begin
    ;; Some parameters were repeated
    ulist = (res0[sort(res0)])[u]
    result = hash()
    for i=0,nu-1 do begin
      w = where(res0 EQ ulist[i], nw)
      if (nw EQ 1) $
        then result = result + hash(ulist[i], res1[w[0]]) $
        else result = result + hash(ulist[i], res1[w])
    endfor
  endif else result = hash(res0, res1)
endelse

return, result

end
```

```
pro inputtree_event, ev

widget_control, ev.ID, get_uvalue=uname

case strlowcase(uname) of
  'base': widget_control, ev.top, xsize=ev.x, ysize=ev.y
  'done': widget_control, ev.top, /destroy
  else:
endcase

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro input_view, inputtemp

if (isa(inputtemp, 'string')) then begin
  input = inputs_restore(inputtemp)
  inputfile = inputtemp
endif else begin
  input = inputtemp
  inputfile = 'input'
endelse


base = widget_base(/column, title='Input', xoffset=1200, xsize=600, ysize=800, $
  /tlb_size_events, uvalue='base')
donebut = widget_button(base, value='Done', uvalue='Done')

tree = widget_tree(base, xsize=600, ysize=800)
root = widget_tree(tree, value=inputfile, /folder, uvalue='root', /expanded)

geotree = widget_tree(root, value='geometry', uvalue='geometry', /folder)
tname = tag_names(input.geometry)
geobranches = bytarr(n_elements(tname))
tname[0] = widget_tree(geotree, value='planet: '+input.geometry.planet, uvalue='planet')
tname[1] = widget_tree(geotree, value='start point: '+input.geometry.startpoint, $
  uvalue='startpoint')
tname[2] = widget_tree(geotree, value='phi: ', uvalue='phi', /folder)
tname[3] = widget_tree(geotree, value='include: ', uvalue='include', /folder)
tname[4] = widget_tree(geotree, value='TAA: '+string(input.geometry.taa), $
  uvalue='taa')


sticktree = widget_tree(root, value='sticking info', uvalue='StickingInfo', /folder)

forcetree = widget_tree(root, value='forces', uvalue='forces', /folder)

spatialtree = widget_tree(root, value='spatial distribution', uvalue='SpatialDist', $
  /folder)

```

```
speedtree = widget_tree(root, value='speed distribution', uvalue='SpeedDist', /folder)

angtree = widget_tree(root, value='angular distribution', uvalue='AngularDist', /folder)

;;perturbtree = widget_tree(root, value='velocity perturbation', uvalue='PerturbVel', $
;;  /folder)
;;plasmatree = widget_tree(root, value='plasma info', uvalue='PlasmaInfo', /folder)

opttree = widget_tree(root, value='options', uvalue='options', /folder)

widget_control, base, /realize
xmanager, 'inputtree', base, /no_block

end
```

```
function inputs_restore, file, geometry=geometry, sticking_info=sticking_info, $
  forces=forces, spatialdist=spatialdist, angulardist=angulardist, speeddist=speeddist, $
  perturbvel=perturbvel, options=options, plasma_info=plasma_info

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Revision History
;;    3.3: 7/7/11
;;     * Added PSD spatial distribution option
;;    3.2: 11/23/10
;;     * added size option to SO2 exosphere spatial dist
;;    3.0: 7/13/10
;;     * Put all the inputs into a single structure that is the output of the function
;;
;;    2.10 4/26/10
;;     * Added Earth/Moon options
;;    2.9  3/9/10
;;     * removed partial thermal accommodation option and added accom_factor to
;;         Maxwellian sticking function
;;    2.8: 1/13/10
;;      * added local temperature option to maxwellian speed ditsribution
;;         -- distribution fn depends on local surface temperature
;;      * added options.datapath and options.modelpath
;;    2.7: 11/20/09
;;      * added powerlaw/exponential options to exosphere spatial distribution
;;    2.6: 11/6/09
;;      * added Sticking_info.accom_factor option
;;    2.3: 2/12/09
;;      * added parameters for SpatialDist.type = 'exosphere'
;;          * SpatialDist.b -- powerlaw slope of density in exospehre
;;    2.2:
;;    2.1: 12/19/08
;;      * added fields geometry.subSolarLong and subSolarLat
;;      * These are currently used in determining the observation geometry
;;    2.0:
;;      * 12 November 2008: remade with new structure information
;;      * need to add in input validation to make sure there are no
;;          contradictions - for example - if using circular orbits, need a
;;          perturbation distribution but no angular distribution
;;      * need to check capitalization, etc.
;;    1.0: original
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

readcol, file, param, value, delim='=', format='A,A', /silent
param = strlowcase(strtrim(param, 2))

;; strip off any comments in the values
q = stregex(value, ';')
w = where(q NE -1, nq)
if (nq GT 0) then for i=0,nq-1 do $
```

```
  value[w[i]] = strmid(value[w[i]], 0, q[w[i]]-1)
value = strtrim(value, 2)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Make the geometry structure
;; Fields:
;;   planet -- required
;;   StartPoint -- required
;;   phi -- required for Jupiter and Saturn with one for each object
;;       -- phi[0] = 0 always
;;       -- for Mercury, the input is ignored and phi set to 0.
;;   include -- optional with one for each object
;;           -- if not provided set to 1 for each object
;;           -- if included, must have correct number (Jup=5, Sat=10, Merc=1)
;;   CML -- required for Jupiter only
;;   taa -- required for Mercury only
;;   SubSolarLong -- default = 0 deg.
;;   SubSolLat -- default = 0 deg.
;;   aplanet ---| these are added to the structure but default value is deteremined
;;   vrplanet --| in modeldriver.
geom = where(strmatch(param, 'geometry*'))
gparam = strmid(param[geom], strlen('geometry.'))
gval = value[geom]

q = (where(gparam EQ 'planet'))[0]
planet = gval[q]

q = (where(gparam EQ 'subsolarlong', nq))[0]
subslong = (nq EQ 1) ? gval[q] : 0.

q = (where(gparam EQ 'subsolarlat', nq))[0]
subslat = (nq EQ 1) ? gval[q] : 0.

case (planet) of
  'Mercury': begin
    q = (where(gparam EQ 'taa', nq))[0]
    if (nq NE 1) then stop else taa = double(gval[q])
    q = (where(gparam EQ 'include', nq))[0]
    case (nq) of
      0: inc = 1
      1: inc = fix(gval[q]) NE 0
      else: stop
    endcase

    geometry = {planet:'Mercury', StartPoint:'Mercury', phi:ptr_new(0d), $
      include:ptr_new(inc), taa:taa, subsolarlong:subslong, subsolarlat:subslat}
    end
  'Earth': begin
    q = (where(gparam EQ 'startpoint', nq))[0]
    stpt = (nq EQ 1) ? gval[q] : stop
    q = where(gparam EQ 'phi', nq)
  case (nq) of
    1: phi = double([0., gval[q]])
    2: phi = double(gval[q])
    else: stop
  endcase
  q = where(gparam EQ 'include', nq)
  case (nq) of
    0: inc = [1,1]
    2: inc = fix(gval[q]) NE 0
    else: stop
  endcase
  geometry = {planet:'Earth', StartPoint:stpt, phi:ptr_new(phi), $
    include:ptr_new(inc), subsolarlong:subslong, subsolarlat:subslat, taa:0.}
  end
 'Jupiter': begin
  q = (where(gparam EQ 'startpoint'))[0]
  stpt = gval[q]
  q = where(gparam EQ 'phi', nq)
  if (nq NE 5) then stop else phi = double(gval[q]) & phi[0] = 0d
  q = (where(gparam EQ 'cml', nq))[0]
  if (nq NE 1) then stop else cml = double(gval[q])

  q = where(gparam EQ 'include', nq)
  case (nq) of
    0: inc = replicate(1, 5)
    5: inc = fix(gval[q]) NE 0
    else: stop
  endcase

  geometry = {planet:'Jupiter', StartPoint:stpt, phi:ptr_new(phi), $
    include:ptr_new(inc), CML:CML, subsolarlong:subslong, $
    subsolarlat:subslat, taa:0.}
  end
 'Saturn': begin
  q = (where(gparam EQ 'startpoint'))[0]
  stpt = gval[q]
  q = where(gparam EQ 'phi', nq)
  if (nq NE 10) then stop else phi = double(gval[q]) & phi[0] = 0d
  q = where(gparam EQ 'include', nq)
  case (nq) of
    0: inc = replicate(1, 10)
    10: inc = fix(gval[q]) NE 0
    else: stop
  endcase

  geometry = {planet:'Saturn', StartPoint:stpt, phi:ptr_new(phi), $
    include:ptr_new(inc), subsolarlong:subslong, subsolarlat:subslat, taa:0.}
  end
  else: stop
endcase

```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Sticking_info
;; This is optional - if not there, assumed that everything sticks to the surface
;; fields:
;;   * stickcoef  -- if stickcoef = 1. then there are no other options
;;   * emitfn -- options = 'Maxwellian', 'elastic scattering'
;;   * surftemp -- if emitfn = 'Maxwellian'
;; -- will need to expand on this

st = where(strmatch(param, 'sticking_info*'), ns)
if (ns EQ 0) $
  then stop $
  else begin
    sparam = strmid(param[st], strlen('sticking_info.'))
    sval = value[st]

    q = (where(sparam EQ 'stickcoef', nq))[0]
    if (nq NE 1) then stop else stick = float(sval[q])
    if (stick GE 1) $
      then sticking_info = {stickcoef:1.} $
      else begin
        q = (where(sparam EQ 'emitfn'))[0]
        fn = sval[q]
        case strlowcase(fn) of
          'maxwellian': begin
            q = (where(sparam EQ 'tsurf', nq))[0]
            if (nq EQ 1) then Tsurf = max([0d, double(sval[q])]) else Tsurf = 0

            q = (where(sparam EQ 'accom_factor', nq))[0]
            if (nq EQ 1) then accom_factor = double(sval[q]) else stop
            if (accom_factor LT 0) then accom_factor = 0d
            if (accom_factor GT 1) then accom_factor = 1d

            sticking_info = {stickcoef:stick, emitfn:fn, Tsurf:tsurf, $
              accom_factor:accom_factor}
            end
          'elastic scattering': sticking_info = {stickcoef:stick, emitfn:fn}
          else: stop ;; not set up yet
        endcase
      endelse
  endelse

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  Forces
;;  Any force not explicitly set is turned off
;;  options:
;;    * gravity
;;    * radpres
;;    * lorentz
;;
forces = {gravity:0, radpres:0, lorentz:0}
```

```
ff = where(strmatch(param, 'forces*'), ns)
if (ns NE 0) then begin
  fparam = strmid(param[ff], strlen('forces.'))
  fval = value[ff]
  q = (where(fparam EQ 'gravity', nq))[0]
  if (nq EQ 1) then forces.gravity = fix(fval(q))
  q = (where(fparam EQ 'radpres', nq))[0]
  if (nq EQ 1) then forces.radpres= fix(fval(q))
  q = (where(fparam EQ 'lorentz', nq))[0]
  if (nq EQ 1) then forces.lorentz= fix(fval(q))
endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SpatialDist
;;   * surface -- default is evenly spread out over sphere with radius=1
;;   * torus -- no default - r0,r1,r2 must be specified
spat = where(strmatch(param, 'spatialdist*'), ns)
sparam = strmid(param[spat], strlen('spatialdist.'))
sval = value[spat]
q = (where(sparam EQ 'type'))[0]
spatdist = sval[q]

case strlowcase(spatdist) of
  'surface': begin
    q = (where(sparam EQ 'use_map', nq))[0]
    usemap = (nq EQ 1) ? fix(sval[q]) : 0

    q = (where(sparam EQ 'exobase', nq))[0]
    exobase = (nq EQ 1) ? double(sval[q]) : 1d

    if (usemap) then begin
      q = (where(sparam EQ 'mapfile'))[0]
      mapfile = sval[q]
      SpatialDist = {type:'surface', exobase:exobase, use_map:1, $
        mapfile:mapfile}
    endif else begin
      q = where(sparam EQ 'longitude0', nq)
      lon0 = (nq EQ 1) ? double(sval[q]) : 0d
      q = where(sparam EQ 'longitude1', nq)
      lon1 = (nq EQ 1) ? double(sval[q]) : 2*!dpi

      q = where(sparam EQ 'latitude0', nq)
      lat0 = (nq EQ 1) ? double(sval[q]) : -!dpi/2.
      q = where(sparam EQ 'latitude1', nq)
      lat1 = (nq EQ 1) ? double(sval[q]) : !dpi/2.

      SpatialDist = {type:'surface', exobase:exobase, use_map:0, $
        longitude:[lon0,lon1], latitude:[lat0,lat1]}
    endelse
    end
  'torus': begin
  q = (where(sparam EQ 'torus_radius0', nq))[0]
  if (nq EQ 1) then r0 = double(sval[q]) else stop
  q = (where(sparam EQ 'torus_radius1', nq))[0]
  if (nq EQ 1) then r1 = double(sval[q]) else stop
  q = (where(sparam EQ 'torus_radius2', nq))[0]
  if (nq EQ 1) then r2 = double(sval[q]) else stop

  SpatialDist = {type:'torus', torus_radii:[r0, r1, r2]}
  end
 'exosphere': begin
  q= (where(sparam EQ 'exotype', nq))[0]
  if (nq EQ 1) then exotype = sval[q] else stop
  q = (where(sparam EQ 'b', nq))[0]
  if (nq EQ 1) then b = float(sval[q]) else stop
  q = (where(sparam EQ 'rmax', nq))[0]
  rmax = (nq EQ 1) ? sval[q] : 10.
  q = (where(sparam EQ 'block_shadow', nq))[0]
  block_shadow = (nq EQ 1) ? fix(sval[q]) : 0

  SpatialDist = {type:'exosphere', exotype:exotype, b:b, rmax:rmax, $
    block_shadow:block_shadow}
  end
 'so2 exosphere': begin
  q = (where(sparam EQ 'size', nq))[0]
  case (1) of
    stregex(sval[q], 'large', /fold, /bool): size = 'large'
    stregex(sval[q], 'small', /fold, /bool): size = 'small'
    else: stop
  endcase
  SpatialDist = {type:'SO2 exosphere', size:size}
  end
 'psd': begin
  q = (where(sparam EQ 'exobase', nq))[0]
  exobase = (nq EQ 1) ? double(sval[q]) : 1d

  q = (where(sparam EQ 'diffusionlimit', nq))[0]
  dlimit = (nq EQ 1) ? double(sval[q]) : 1d30 ; default = unlimited

  q = (where(sparam EQ 'kappa', nq))[0]
  kappa = (nq EQ 1) ? double(sval[q]) : 0d

  if (kappa GT 0) then begin
    q = (where(sparam EQ 'protonprecipfile', nq))[0]
    if (nq EQ 1) then ff = sval[q] else stop
  endif else ff = ''

  SpatialDist = {type:'PSD', diffusionlimit:dlimit, kappa:kappa, $
    ProtonPrecipFile:ff, exobase:exobase}
  end
 else: stop
endcase
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  VelocityDist: SpeedDist and AnglularDist
;;
vdist = where(strmatch(param, 'speeddist*'))
vparam = strmid(param[vdist], strlen('speeddist.'))
vval = value[vdist]

q = (where(vparam EQ 'type'))[0]
spd = strlowcase(vval[q])

case (spd) of
  'gaussian': begin   ;; vprob, sigma
    q = (where(vparam EQ 'vprob'))[0]
    vprob = double(vval[q])
    q = (where(vparam EQ 'sigma'))[0]
    sigma = double(vval[q])
    speeddist = {type:'gaussian', vprob:vprob, sigma:sigma}
    end
  'trigaussian': begin ;; vxprob, vxsigma, vyprob, vysigma, vzprob, vzsigma
    q = (where(vparam EQ 'vxprob'))[0]
    vxprob = double(vval[q])
    q = (where(vparam EQ 'vxsigma'))[0]
    vxsigma = double(vval[q])

    q = (where(vparam EQ 'vyprob'))[0]
    vyprob = double(vval[q])
    q = (where(vparam EQ 'vysigma'))[0]
    vysigma = double(vval[q])

    q = (where(vparam EQ 'vzprob'))[0]
    vzprob = double(vval[q])
    q = (where(vparam EQ 'vzsigma'))[0]
    vzsigma = double(vval[q])

    speeddist = {type:'trigaussian', vxprob:vxprob, vxsigma:vxsigma, vyprob:vyprob, $
      vysigma:vysigma, vzprob:vzprob, vzsigma:vzsigma}
    end
  'dolsfunction': begin  ;; dols0, dols1
    q = (where(vparam EQ 'dols0'))[0]
    dols0 = vval[q]
    q =  (where(vparam EQ 'dols1'))[0]
    dols1 = vval[q]
    speeddist = {type:'dolsfunction', dols0:dols0, dols1:dols1}
    end
  'sputtering': begin  ;; U, alpha, beta
    q = (where(vparam EQ 'u'))[0]
    U = double(vval[q])
    q = (where(vparam EQ 'alpha'))[0]
    alpha = double(vval[q])
    q = (where(vparam EQ 'beta'))[0]
    beta = double(vval[q])
    speeddist = {type:'sputtering', U:U, alpha:alpha, beta:beta}
    end
  'maxwellian': begin ;; temperature
    q = (where(vparam EQ 'temperature'))[0]
    temp = double(vval[q])
    speeddist = {type:'maxwellian', temperature:temp}
    end
  'flat': begin  ;; vprob, delv
    q = (where(vparam EQ 'vprob'))[0]
    vprob = double(vval[q])
    q = (where(vparam EQ 'delv'))[0]
    delv = double(vval[q])
    speeddist = {type:'flat', vprob:vprob, delv:delv}
    end
  'circular orbits': speeddist = {type:'circular orbits'}  ;; no options
  'user defined': begin
    q = (where(vparam EQ 'distfile', nq))[0]
    if (nq NE 0) then distfile = vval[q] else stop
    speeddist = {type:'user defined', distfile:distfile}
    end
  else: stop
endcase

vdist = where(strmatch(param, 'angulardist*'))
vparam = strmid(param[vdist], strlen('angulardist.'))
vval = value[vdist]

q = (where(vparam EQ 'type'))[0]
ang = strlowcase(vval[q])
if (SpeedDist.type EQ 'circular orbits') then ang = 'none'

case (ang) of  ;; none, radial, isotropic, costheta
  'none': angulardist = {type:'none'}
  'radial': angulardist = {type:'radial'}
  'isotropic': begin
    ;; For distributions starting at the surface, make sure the packets are pointed
    ;; outward
    case (SpatialDist.type) of
      'surface': altmin = 0.
      'torus': altimin = -!dpi/2.
      'exosphere': altmin = -!dpi/2.
      'SO2 exosphere': altmin = -!dpi/2.
      else: stop
    endcase

    q = where(vparam EQ 'azimuth0', nq)
    az0 = (nq EQ 1) ? double(vval[q]) : 0d
    q = where(vparam EQ 'azimuth1', nq)
    az1 = (nq EQ 1) ? double(vval[q]) : 2*!dpi

    q = where(vparam EQ 'altitude0', nq)
    alt0 = (nq EQ 1) ? double(vval[q]) : altmin
    q = where(vparam EQ 'altitude1', nq)
    alt1 = (nq EQ 1) ? double(vval[q]) : !dpi/2.

    angulardist = {type:'isotropic', azimuth:[az0, az1], altitude:[alt0, alt1]}
    end
  'costheta': begin
    q = where(vparam EQ 'azimuth0', nq)
    az0 = (nq EQ 1) ? double(vval[q]) : 0d
    q = where(vparam EQ 'azimuth1', nq)
    az1 = (nq EQ 1) ? double(vval[q]) : 2*!dpi

    q = where(vparam EQ 'altitude0', nq)
    alt0 = (nq EQ 1) ? double(vval[q]) : 0.
    q = where(vparam EQ 'altitude1', nq)
    alt1 = (nq EQ 1) ? double(vval[q]) : !dpi/2.

    q = (where(vparam EQ 'n', nq))[0]
    n = (nq EQ 1) ? double(vval[q]): 1.

    angulardist = {type:'costheta', azimuth:[az0, az1], altitude:[alt0, alt1], n:n}
    end
endcase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PerturbVel -- not set up
pdist = where(strmatch(param, 'perturbvel*'), npert)
if (npert NE 0) then begin
  pparam = strmid(param[pdist], strlen('perturbvel.'))
  pval = value[pdist]

  q = (where(pparam EQ 'type'))[0]
  type = pval[q]

  case (type) of
    'none': PerturbVel = {type:'none'}
    'gaussian': begin
      q = (where(pparam EQ 'vprob'))[0]
      vprob = pval[q]
      q = (where(pparam EQ 'sigma'))[0]
      sigma = pval[q]

      PerturbVel = {type:'gaussian', vprob:vprob, sigma:sigma}
      end
    'trigaussian': begin ;; vxprob, vxsigma, vyprob, vysigma, vzprob, vzsigma
      q = (where(pparam EQ 'vxprob'))[0]
      vxprob = double(pval[q])
      q = (where(pparam EQ 'vxsigma'))[0]
      vxsigma = double(pval[q])

      q = (where(pparam EQ 'vyprob'))[0]
      vyprob = double(pval[q])
      q = (where(pparam EQ 'vysigma'))[0]
      vysigma = double(pval[q])

      q = (where(pparam EQ 'vzprob'))[0]
      vzprob = double(pval[q])
      q = (where(pparam EQ 'vzsigma'))[0]
      vzsigma = double(pval[q])

      PerturbVel = {type:'trigaussian', vxprob:vxprob, vxsigma:vxsigma, vyprob:vyprob, $
        vysigma:vysigma, vzprob:vzprob, vzsigma:vzsigma}
      end
    'charge exchange': begin ;; flowvel
      q = (where(pparam EQ 'flowvel'))[0]
      flowvel = double(pval[q])
      PerturbVel = {type:'charge exchange', flowvel:flowvel}
      end
    else: stop
  endcase
endif else PerturbVel = {type:'none'}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Plasma_info
case (planet) of
  'Mercury': plasma_info = {type:'none'}
  'Earth': plasma_info = {type:'none'}
  'Jupiter': begin
    ;; These are the current defaults, but need to review this.
    ;; plamsa parameters:
    ;;   eps - default = 0.14/5.7
    ;;   thermal: default = 1
    ;;   energetic: default = 1
    pdist = where(strmatch(param, 'plasma*'))
    pparam = strmid(param[pdist], strlen('plasma.'))
    pval = value[pdist]

    q = (where(pparam EQ 'eps', nq))[0]
    eps = (nq EQ 1) ? double(pval[q]) : 0.14/5.7

    q = (where(pparam EQ 'thermal', nq))[0]
    th = (nq EQ 1) ? fix(pval[q]) : 1
    q = (where(pparam EQ 'energetic', nq))[0]
    en = (nq EQ 1) ? fix(pval[q]) : 1

    plasma_info = {eps:eps, thermal:th, energetic:en}
    end
  'Saturn': begin
    ;;   ElecDenMod: default = 1
    ;;   ElecTempMod: default = 1
    pdist = where(strmatch(param, 'plasma*'),np)
```

```
      if (np NE 0) then begin
        pparam = strmid(param[pdist], strlen('plasma.'))
        pval = value[pdist]
      endif else begin
        pparam = ''
        pval = 0.
      endelse

      q = (where(pparam EQ 'elecdenmod', nq))[0]
      denmod = (nq EQ 1) ? double(pval[q]) : 1d

      q = (where(pparam EQ 'electempmod', nq))[0]
      tmod = (nq EQ 1) ? double(pval[q]) : 1d

      plasma_info = {ElecDenMod:denmod, ElecTempMod:tmod}
      end
  endcase

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; Options
  ;;    * endtime
  ;;    * resolution - default = 1e-6
  ;;    * motion - default = 1 for Jupiter and Saturn, no effect for Mercury
  ;;    * lifetime - default = 0
  ;;    * atom
  ;;    * at_once - default = 0 but reset when doing streamlines
  ;;    * fullsystem - default = 1 for Jupiter and Saturn, default=0 for Mercury
  ;;    * outeredge - if fullsystem set to 0, default = 20
  ;;

  odist = where(strmatch(param, 'options*'))
  oparam = strmid(param[odist], strlen('options.'))
  oval = value[odist]

  q = (where(oparam EQ 'endtime', nq))[0]
  if (nq NE 1) then stop else endtime = double(oval[q])

  q = (where(oparam EQ 'resolution', nq))[0]
  res = (nq EQ 1) ? double(oval[q]) : 1d-6

  q = (where(oparam EQ 'at_once', nq))[0]
  at = (nq EQ 1) ? fix(oval[q]) : 0

  q = (where(oparam EQ 'atom', nq))[0]
  if (nq NE 1) then stop else atom = oval[q]

  q = (where(oparam EQ 'lifetime', nq))[0]
  life = (nq EQ 1) ? double(oval[q]) : 0d

  f = (where(oparam EQ 'fullsystem', nf))[0]
  if (planet EQ 'Mercury') then begin
  full = (nf EQ 1) ? fix(oval[f]) : 0
  m = 0
endif else begin
  full = (nf EQ 1) ? fix(oval[f]) : 1

  q = (where(oparam EQ 'motion', nq))[0]
  m = (nq EQ 1) ? fix(oval[q]) : 1
endelse

if ~(full) then begin
  q = (where((oparam EQ 'outeredge'), nq))[0]
  outer = (nq EQ 1) ? double(oval[q]) : 20d
endif else outer = 0.

q = (where(oparam EQ 'trackloss', nq))[0]
trackloss = (nq EQ 1) ? fix(oval[q]) : 0

options = {endtime:endtime, resolution:res, motion:m, lifetime:life, $
  atom:atom, at_once:at, fullsystem:full, outeredge:outer, trackloss:trackloss}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Put all the inputs into a single structure
input = {geometry:geometry, sticking_info:sticking_info, forces:forces, $
  spatialdist:spatialdist, angulardist:angulardist, speeddist:speeddist, $
  options:options, perturbvel:perturbvel, plasma_info:plasma_info}

return, input

end
```

```
function make_generic_input, inputtemp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Standard function for making the generic version of an input file
;;
;; Version History
;;   3.1: 7/8/2011
;;     * Added PSD spatial distribution option
;;   3.0: 1/7/2011
;;     * created based on code removed from modeldriver_3.4
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

input = (isa(inputtemp, 'string')) ? inputs_restore(inputtemp) : inputtemp

case strlowcase(input.SpatialDist.type) of
  'surface': begin
    SpatialDist = {type:'surface', use_map:0, longitude:[0, 2*!dpi], $
      latitude:[-!dpi/2,!dpi/2], exobase:input.SpatialDist.exobase}
    AngularDist = {type:'isotropic', azimuth:[0,2*!dpi], altitude:[0,!dpi/2.]}
    end
  'exosphere': begin
    AngularDist = {type:'isotropic', azimuth:[0,2*!dpi], $
      altitude:[-!dpi/2.,!dpi/2.]}
    stop ;; don't know what generic exosphere dist should be
    end
  'psd': begin
    SpatialDist = {type:'surface', use_map:0, longitude:[0, 2*!dpi], $
      latitude:[-!dpi/2,!dpi/2], exobase:input.SpatialDist.exobase}
    AngularDist = {type:'isotropic', azimuth:[0,2*!dpi], altitude:[0,!dpi/2.]}
    end
endcase

case strlowcase(input.geometry.planet) of
  'mercury': SpeedDist = {type:'flat', vprob:4d, delv:3.99d}
  'jupiter': SpeedDist = {type:'flat', vprob:4d, delv:3.9d}
  'saturn': SpeedDist = {type:'flat', vprob:4d, delv:3.9d}
  else: stop
endcase

genericinput = {geometry:input.geometry, sticking_info:input.sticking_info, $
  forces:input.forces, options:input.options, $
  perturbvel:input.perturbvel, plasma_info:input.plasma_info, $
  SpatialDist:SpatialDist, SpeedDist:SpeedDist, AngularDist:AngularDist}

return, genericinput

end
```

```
pro make_model_header, outputfile

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; make_model_header: make a text format header file for the model output
;;
;; Inputs:
;;    * outputfile = model output file in IDLsave format
;;
;; Writen by Matthew Burger
;; Version History:
;;   3.2: 7/19/10
;;     * converted to new architecture
;;   3.1: 5/13/10
;;     * Added num keyword
;;     * Added code versions to the header
;;   3.0: 5/10/10
;;     * Created.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

result = obj_new('IDL_Savefile', outputfile)

;; Extract identifying info
contents = result.contents()
id = {file:outputfile, time:contents.date, user:contents.user, computer:contents.host}
t = strtrim(tag_names(id), 2)
idparam = strarr(n_elements(t))
idvalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  idparam[i] = t[i]
  idvalue[i] = string(id.(i))
endfor
idvalue = strtrim(idvalue, 2)

result.restore, 'output'
npackets = n_elements(*output.x)
result.restore, 'input'

;; Extract geometry info
geometry = input.geometry
t = strtrim(tag_names(geometry), 2)
geoparam = strarr(1000)
geovalue = strarr(1000)
ct = 0
for i=0,n_elements(t)-1 do begin
  if (ptr_valid(geometry.(i))) then for j=0,n_elements(*geometry.(i))-1 do begin
    geoparam[ct] = t[i]
    geovalue[ct] = string((*geometry.(i))[j])
    ct++
  endfor else begin
```

```
    geoparam[ct] = t[i]
    geovalue[ct] = string(geometry.(i))
    ct++
  endelse
endfor
geoparam = geoparam[0:ct-1]
geovalue = strtrim(geovalue[0:ct-1], 2)

;; Extract Sticking_info
sticking_info = input.sticking_info
t = strtrim(tag_names(sticking_info), 2)
stickparam = strarr(n_elements(t))
stickvalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  stickparam[i] = t[i]
  stickvalue[i] = string(sticking_info.(i))
endfor
stickvalue = strtrim(stickvalue, 2)

;; Extract Forces
forces = input.forces
t = strtrim(tag_names(forces), 2)
forceparam = strarr(n_elements(t))
forcevalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  forceparam[i] = t[i]
  forcevalue[i] = string(forces.(i))
endfor
forcevalue = strtrim(forcevalue, 2)

;; Extract spatialdist
SpatialDist = input.SpatialDist
t = strtrim(tag_names(SpatialDist), 2)
spatparam = strarr(100)
spatvalue = strarr(100)
ct = 0
for i=0,n_elements(t)-1 do begin
  n = n_elements(SpatialDist.(i))
  if (n EQ 1) then begin
    spatparam[ct] = t[i]
    spatvalue[ct] = string(SpatialDist.(i))
    ct++
  endif else for j=0,n-1 do begin
    spatparam[ct] = t[i] + strtrim(string(j),2)
    spatvalue[ct] = string((SpatialDist.(i))[j])
    ct++
  endfor
endfor
spatparam = spatparam[0:ct-1]
spatvalue = strtrim(spatvalue[0:ct-1], 2)

```

```
;; Extract speeddist
SpeedDist = input.SpeedDist
t = strtrim(tag_names(SpeedDist), 2)
speedparam = strarr(n_elements(t))
speedvalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  speedparam[i] = t[i]
  speedvalue[i] = string(SpeedDist.(i))
endfor
speedvalue = strtrim(speedvalue, 2)

;; Extract angular_dist
AngularDist = input.AngularDist
t = strtrim(tag_names(AngularDist), 2)
angparam = strarr(100)
angvalue = strarr(100)
ct = 0
for i=0,n_elements(t)-1 do begin
  n = n_elements(AngularDist.(i))
  if (n EQ 1) then begin
    angparam[ct] = t[i]
    angvalue[ct] = string(AngularDist.(i))
    ct++
  endif else for j=0,n-1 do begin
    angparam[ct] = t[i] + strtrim(string(j),2)
    angvalue[ct] = string((AngularDist.(i))[j])
    ct++
  endfor
endfor
angparam = angparam[0:ct-1]
angvalue = strtrim(angvalue[0:ct-1], 2)

;; Extract PerturbVel
PerturbVel = input.PerturbVel
t = strtrim(tag_names(PerturbVel), 2)
pertparam = strarr(n_elements(t))
pertvalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  pertparam[i] = t[i]
  pertvalue[i] = string(PerturbVel.(i))
endfor
pertvalue = strtrim(pertvalue, 2)

;; Extract plasma_info (if present)
PlasmaInfo = input.plasma_info
t = strtrim(tag_names(plasmainfo), 2)
plasmaparam = strarr(n_elements(t))
plasmavalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  plasmaparam[i] = t[i]
  plasmavalue[i] = string(plasmainfo.(i))
```

```
endfor
plasmavalue = strtrim(plasmavalue, 2)

;; extract options
options = input.options
t = strtrim(tag_names(options), 2)
optparam = strarr(n_elements(t))
optvalue = strarr(n_elements(t))
for i=0,n_elements(t)-1 do begin
  optparam[i] = t[i]
  optvalue[i] = string(options.(i))
endfor
optvalue = strtrim(optvalue, 2)

;; Extract version info
result.restore, 'version'
if (n_elements(version) LE 1) then stop
version = file_basename(version[1:*])
verparam = version
vervalue = version
q = stregex(verparam, '[0-9]\.[0-9]+$')
for i=0,n_elements(verparam)-1 do $
  if (q[i] NE -1) then begin
    verparam[i] = strmid(version[i], 0, q[i]-1)
    vervalue[i] = strmid(version[i], q[i])
  endif else vervalue[i] = 'XX'
obj_destroy, result

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Save header file
hdrfile = strmid(outputfile, 0, strlen(outputfile)-strlen('output')) + 'header'
print, hdrfile
openw, lun, hdrfile, width=100, /get_lun

form = '(A-30,A3,A-)'
for i=0,n_elements(idparam)-1 do printf, lun, 'id.' + idparam[i], ' = ', $
  idvalue[i], format=form
printf, lun

printf, lun, 'savedpackets', ' = ', strint(npackets),  format=form
printf, lun, 'output.totalsource', ' = ', strint(output.totalsource), format=form
for i=0,n_elements(*output.sourcefile)-1 do $
  printf, lun, 'output.sourcefile', ' = ', (*output.sourcefile)[i], format=form
printf, lun

for i=0,n_elements(geoparam)-1 do printf, lun, 'geometry.' + geoparam[i], ' = ', $
  geovalue[i], format=form
printf, lun

for i=0,n_elements(stickparam)-1 do printf, lun, 'sticking_info.' + stickparam[i], $
  ' = ', stickvalue[i], format=form
```

```
printf, lun

for i=0,n_elements(forceparam)-1 do printf, lun, 'forces.' + forceparam[i], ' = ', $
  forcevalue[i], format=form
printf, lun

for i=0,n_elements(spatparam)-1 do printf, lun, 'SpatialDist.' + spatparam[i], ' = ', $
  spatvalue[i], format=form
printf, lun

for i=0,n_elements(speedparam)-1 do printf, lun, 'SpeedDist.' + speedparam[i], ' = ', $
  speedvalue[i], format=form
printf, lun

for i=0,n_elements(angparam)-1 do printf, lun, 'AngularDist.' + angparam[i], ' = ', $
  angvalue[i], format=form
printf, lun

for i=0,n_elements(pertparam)-1 do printf, lun, 'PerturbVel.' + pertparam[i], ' = ', $
  pertvalue[i], format=form
printf, lun

for i=0,n_elements(plasparam)-1 do printf, lun, 'PlasmaInfo.' + plasmaparam[i], ' = ', $
  plasmavalue[i], format=form
printf, lun

for i=0,n_elements(optparam)-1 do printf, lun, 'options.' + optparam[i], ' = ', $
  optvalue[i], format=form
printf, lun

printf, lun, 'Program Versions'
for i=0,n_elements(verparam)-1 do printf, lun, verparam[i], ' = ', vervalue[i], $
  format=form
printf, lun

free_lun, lun
destroy_structure, output
destroy_structure, input

end
```

```
function modeloutput_search, inputtemp, verbose=verbose, nfiles=nfiles

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Program to search through the available model output files to compare
;; with an input set.
;;
;; Help file: Docs/modeloutput_search.pdf
;;
;;   3.5: 8 Dec 2011
;;     * minor updates
;;   3.4: 3 Jan 2011
;;     * Rewriting using compare_inputs
;;   3.3: 26 August 2010
;;     * improving the efficiency a bit by making use of the directory tree
;;   3.1: 7/15/10
;;     * seraches for exact matches only. Use genericmodel_search to find the
;;       generic models
;;   3.0: 7/14/10
;;     * original
;;     * looks for files - if none found, gives the generics
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


input0 = (isa(inputtemp, 'string')) ? inputs_restore(inputtemp) : inputtemp
if (verbose EQ !null) then verbose = 0

nfiles = 0
ct = 0
taa0 = input0.geometry.taa

;;;;;;;;
;; Find what output files are available
while ((nfiles EQ 0) and (ct LE 5)) do begin
  if (ct EQ 0) then begin
    if (verbose) then print, 'Trying within +/-0.5 deg'
    q = output_filename(input0, path=path)
    filelist = file_search(path, '*.output', count=nfiles)
  endif else begin
    if (verbose) then print, 'Trying within +/-' + strint(ct) + ' deg'
    input0.geometry.taa = taa0-ct*!dtor
    q = output_filename(input0, path=path)
    filelist0 = file_search(path, '*.output', count=nfiles0)

    input0.geometry.taa = taa0+ct*!dtor
    q = output_filename(input0, path=path)
    filelist1 = file_search(path, '*.output', count=nfiles1)

    nfiles = nfiles0+nfiles1
    case (1) of
```

```
        nfiles0 EQ 0 and nfiles1 EQ 0: filelist = ''
        nfiles1 EQ 0: filelist = filelist0
        nfiles0 EQ 0: filelist = filelist1
        else: filelist = [filelist0, filelist1]
      endcase

      input0.geometry.taa = taa0
    endelse

    if (nfiles GT 0) then begin
      same = intarr(nfiles)
      for i=0,nfiles-1 do begin
        out = obj_new('IDL_savefile', filelist[i])
        out.restore, 'input'

        ;; do a quick check
        t = tag_names(input.options)
        if ~(total(strcmp(t, 'trackloss', /fold))) then stop
        same[i] = compare_inputs(input0, input, verbose=verbose)
        obj_destroy, out
      endfor
      q = where(same, nq)
      filelist = (nq NE 0) ? filelist[q] : ''
    endif
    nfiles = (filelist[0] EQ '') ? 0 : n_elements(filelist)
    ct++
  endwhile

  if (verbose) then print, strint(nfiles) + ' files found with these inputs.'

  return, filelist

  end
```

```
function headername, outputfile

return, reform((strmid(outputfile, 0, strlen(outputfile)-6) + 'header')[0,*])

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function outputname, headerfile

return, reform((strmid(headerfile, 0, strlen(headerfile)-6) + 'output')[0,*])

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function output_filename, inputtemp, path=path, file=file

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Create a unique output filename
;;
;; These are not user readable fileneames to keep them short. The header files
;; are user readable.
;;
;; path = '/Volumes/DroboData/burger/modeloutputs/PLANET/' +
;;              TAA_DEG/ATOM/SPEEDDIST/SPATIALDIST/'  or
;;              MOON/PHI_DEG/ATOM/SPEEDDIST/SPATIALDIST/'
;; file = 'USER.HOSTNAME.####.output'
;;
;; Version History
;; 3.4: 9/28/2011
;;    * adding option to search locally
;; 3.3: 3 Jan 2011
;;    * changing path to DroboData
;; 3.1: 26 August 2010
;;    * Making a bit more of a directory tree
;;    * Making filenames a bit more unique
;;    * adding computer identifier to avoid an ambiguity
;; 3.0: Original
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

input = (isa(inputtemp, 'string')) ? inputs_restore(inputtemp) : inputtemp
if (SystemConsts EQ !null) then SystemConstants, input.geometry.planet, SystemConsts

;; Determine if startpoint = planet
s = (where(strcmp(*SystemConsts.objects, input.geometry.startpoint, /fold)))[0]

```

```
;; Create the path
path = input.geometry.planet + '/'

if (s EQ 0) then begin
  taastr = strint(round(input.geometry.taa/!dtor))
  path += taastr + '/'
endif else begin
  path += input.geometry.startpoint + '/'

  phistr = strint(round((*input.geometry.phi)[s]/!dtor))
  path += phistr + '/'
endelse

path += input.options.atom + '/'
path += strlowcase(input.speeddist.type) + '/'
path += strlowcase(input.spatialdist.type) + '/'
path = strcompress(path, /remove_all)

sh = file_test(!model.SharedOutputPath)
loc = file_test(!model.LocalOutputPath)

case (1) of
  (stuff EQ !null) and (sh): path = !model.SharedOutputPath + path
  (stuff.local) and (loc): path = !model.LocalOutputPath + path
  ~(stuff.local) and (sh): path = !model.SharedOutputPath + path
  else: stop
endcase

if ~(file_test(path)) then file_mkdir, path

;; Create the filename
filest = !model.user + '.' + !model.hostname + '.'

q = file_search(path, filest+'*.output', count=nq)

if (nq EQ 0) $
  then filename = filest + '0000.output' $
  else begin
    file = file_basename(q, '.output')
    num = max(fix(stregex(file, '[0-9]+$', /extract))) + 1
    case (1) of
      (num LT 10): nn = '000' + strint(num)
      (num LT 100): nn = '00' + strint(num)
      (num LT 1000): nn = '0' + strint(num)
      else: nn = strint(num)
    endcase
    filename = filest + nn + '.output'
  endelse
if file_test(path+filename) then stop

return, path+filename
```

```

end
```

```
pro print_inputs, inputs, geometry=geometry, sticking_info=sticking_info, $
  forces=forces, spatialdist=spatialdist, angulardist=angulardist, $
  speeddist=speeddist, options=options, perturbvel=perturbvel, plasma_info=plasma_info

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Prints the content of an input structure to the screen
;;
;; Version History
;;   3.0: 7/19/10
;;     * created
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

geometry = 1
sticking_info = 1
forces = 0
spatialdist = 1
angulardist = 0
speeddist = 1
options = 1
pertrubvel = 0
plasma_info = 0

itags = strlowcase(tag_names(inputs))

for t=0,n_elements(itags)-1 do begin
  t0 = inputs.(t)
  tags = itags[t] + '.' + strlowcase(tag_names(t0))
  for i=0,n_elements(tags)-1 do begin
    val = t0.(i)
    case size(val, /type) of
      7:
      10: val = strint(*val)
      else: val = strint(val)
    endcase

    print, tags[i] + ' = ' + val
  endfor
  print
endfor

end
```

```
function PhotonLimit_PSD, d

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Compute the expected photon-limited PSD flux at the subsolar point.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

photflux = (2.8e15/d^2) ;; flux 3between 115 and 310 nm at 1 AU 2.8e15 phot cm^-2 s^-1
sigma = 3e-21 ;; cm^-2, PSD cross section
n = 7.5e14 ;; cm^-2, surface density
c = 0.005  ;; Na fraction

photon_limit = photflux * sigma * c * n

return, photon_limit

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function PSDfluxmap, input, photmap=photmap, difmap=difmap

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Create a surface map to use for PSD given a TAA, assummed diffusion limited
;; flux, ion-enhanced diffusion factor, and proton precipitation file
;;
;; Current version assumes:
;;   (a) diffision rate is constant over surface -- independent of temperature
;;   (b) desorption cross section is constant over surface -- independent of temperature
;;   (c) Photon limited desorption flux only depends on dist. from sun and SZA
;;   (d) diffusion limited flux depends on kappa and model of proton precipitation
;;
;; Version 1.1: 3 November 2011
;; Version 1.0: 7 July 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

if (SystemConsts EQ !null) then SystemConstants, input.geometry.planet, SystemConsts
planet_dist, input.geometry.taa, SystemConsts, distance=dd, velocity=vv

if (input.geometry.planet NE 'Mercury') then stop ;; uses mercury specific constants

longitude = findgen(361)*!dtor
latitude = findgen(181)*!dtor - !pi/2.
dcos = one(longitude) # cos(latitude)
dlon = longitude[1]-longitude[0] & dlat = latitude[1]-latitude[0]
```

```

;; Photon limited flux (normalized)
photmap = cos(longitude) # cos(latitude)
photmap[where(longitude GT !pi/2 and longitude LT 3*!pi/2),*] = 0.
q = where(photmap LT 0, nq) & if (nq NE 0) then photmap[q] = 0
;;;;;;;;;;;;;;;;;;

;; Diffusion limited flux (normalize)
if (input.SpatialDist.kappa GT 0) then begin
  restore, input.SpatialDist.ProtonPrecipFile
  if (n_elements(photmap) NE n_elements(*sourcemap.map)) then stop
  difmap = (1 + input.SpatialDist.kappa/1e8 * *sourcemap.map)
endif else difmap = replicate(1., n_elements(longitude), n_elements(latitude))
;;;;;;;;;;;;;;;;;

difmap = difmap*input.SpatialDist.DiffusionLimit
q = (photmap LT difmap)
map = q*photmap + (1-q)*difmap
q = where(finite(map) EQ 0, nq) & if (nq NE 0) then stop

;; compute total PSD source rate for normalization purposes later
rate = total(map*dcos) * (!Mercury.radius*1e5)^2 * dlon * dlat

sourcemap = {longitude:ptr_new(longitude), latitude:ptr_new(latitude), $
  map:ptr_new(map), rate:rate}

return, sourcemap

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro PSD_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Distribute packets according to PSD spatial distribution parameters
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

sourcemap = PSDfluxmap(input)

*sourcemap.map /= max(*sourcemap.map)
RandomDeviates_2d, *sourcemap.map, *sourcemap.longitude, sin(*sourcemap.latitude), $
  npack, lon, lat
lat = asin(lat)
destroy_structure, sourcemap

if strcmp(input.geometry.planet, input.geometry.StartPoint, /fold) then begin
  ;; Starting at a planet.
```

```
    ;; 0 deg longitude = subsolar pt = (0, -1, 0)
    ;; 90 deg longitude = dusk pt = (1, 0, 0)
    ;; 270 deg longitude = dawn pt = (-1, 0, 0)
    *output.x0 = double(SpatialDist.exobase * sin(lon)*cos(lat))
    *output.y0 = -double(SpatialDist.exobase * cos(lon)*cos(lat))
    *output.z0 = double(SpatialDist.exobase * sin(lat))
  endif else begin
    ;; Starting at a satellite
    ;; Treats the satellite as if it were at phi = 0.
    ;; 0 deg longitude = subsolar pt = (0, -1, 0)
    ;; 90 deg longitude = leading pt = (-1, 0, 0)
    ;; 270 deg longitude = trailing pt = (1, 0, 0)
    ;; lon=0 -> sub-planet point; lon=90 -> leading point
    *output.x0 = -double(SpatialDist.exobase * sin(lon)*cos(lat))
    *output.y0 = -double(SpatialDist.exobase * cos(lon)*cos(lat))
    *output.z0 = double(SpatialDist.exobase * sin(lat))
  endelse

  end



```

```
pro SO2exosphere_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; O source rate based on SO2 exosphere modeled by Vincent Dols. See notes.
;;
;; Written by Matthew Burger
;;
;; Version History
;;   3.1  11/23/201
;;     * 2nd try
;;   3.0: 11/23/2010
;;     * initial version - doesn't work
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

num = n_elements(x)
case (1) of
  stregex(input.spatialdist.size, 'large', /fold, /bool): begin
    name_atmos = 'ATMOS_BOTH_LARGE'
    delta1 = 0.1  ;ATMOS_LARGE
    delta2 = 0.22  ;ATMOS_LARGE
    HZ = 1.10            ;ATMOS_LARGE
    shift_coef =   0.9   ;ATMOS_LARGE
    r1 = 1.42   ;ATMOS_LARGE
    r2 = r1 + 0.1    ;ATMOS_LARGE
    rmin = 1.04  ;distance where the rate drops to zero from the shifted center
    rmax = 2.16
    phi_drop = 2.  ;power index of the cos drop with longitude (2 + 2=4 for Z=1)
    end
  stregex(input.spatialdist.size, 'small', /fold, /bool): begin
    name_atmos = 'ATMOS_BOTH_SMALL'
    delta1 = 0.17  ;ATMOS_SMALL
    delta2 = 0.15  ;ATMOS_SMALL
    HZ = 0.95 ;ATMOS_SMALL
    shift_coef =   0.8   ;ATMOS_SMALL
    r1 = 1.02   ;ATMOS_SMALL
    r2 = r1 + 0.04   ;ATMOS_SMALL
    rmin = 1.04  ;distance where the rate drops to zero from the shifted center
    rmax = 2.16
    phi_drop = 7.  ;power index of the cos drop with longitude (2 + 7=9 for Z=1)
    end
  else: stop
endcase

;PLASMA DATA
;************
nel0 = 3778.0  ;upstream plasma densitycm-3
Bio = 1781.e-9 ; magn field at Io
vfl =  57.e3 ; upstream flow velocity m/s
mu0 =  4.* !pi *1.e-7 ;mgn permitivity
```

```
Valf =  Bio/sqrt(mu0 * (nel0 *1e6) * 22. * 1.67e-27) ;Alf velocity in m/s
Malf =   Vfl/Valf; Mach number
ANg_ALF =  atan(Malf) * 180./!pi; angle of alfven tube

;; Determine r' = modified radial component
rr_pr = dindgen(1001)/1000 * rmax
fr_pr1 = exp(-(rr_pr-r1)^2/delta1^2) * (rr_pr GT 1)
fr_pr2 = 0.25*exp(-(rr_pr-r2)^2/delta2^2) * (rr_pr GT r1)
fr_pr = fr_pr1 + fr_pr2
r_pr = RandomDeviates_1d(rr_pr, fr_pr, npack, seed=seed)

;; Determine latitudinal (z) and modified azimuthal (phi') components together
zz = (dindgen(201)/100-1)*2*Hz
pp_pr = dindgen(361)*!dtor

fz = exp(-(zz/Hz)^6)
f_zphi = dblarr(201,361)
for i=0,n_elements(zz)-1 do $
  for j=0,n_elements(pp_pr)-1 do $
    f_zphi[i,j] = fz[i] * (.5*(cos(!dpi-pp_pr[j])+1))^(2+phi_drop*abs(zz[i]))

RandomDeviates_2d, f_zphi, zz, pp_pr, npack, z, phi_pr, seed=seed

x_pr = r_pr * cos(phi_pr)
delX = shift_coef * Malf * abs(z)

*output.x0 = -(x_pr + delX)
*output.y0 = r_pr * sin(phi_pr)
*output.z0 = z

end
```

```
function SourceFlux, sourcemap, sourcerate, map=map

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Given a sourcemap structure and (optionally) the source rate, returns
;; the peak flux. If sourcerate is not given, assumes = 1e26
;;
;; Note - currently assumes that planet = Mercury
;;
;; Keyword output:
;;   map = re-normalized flux map with maximum=peakflux
;;
;; Version History
;;   1.0: written 8 Nov 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (sourcerate EQ !null) then sourcerate = 1e26

dlon = (*sourcemap.longitude)[1]-(*sourcemap.longitude)[0]
dlat = (*sourcemap.latitude)[1]-(*sourcemap.latitude)[0]
dcos = one(*sourcemap.longitude) # cos(*sourcemap.latitude)

maptotal = total(*sourcemap.map*dcos) * (!Mercury.radius*1e5)^2 * dlon * dlat
map = *sourcemap.map * sourcerate/maptotal
peakflux = max(map)

return, peakflux

end
```

```
function SourceRate, sourcemap, peakflux

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Given a sourcemap structure and (optionally) the peak flux, calculates
;; the total source rate
;;
;; Note - currently assumes that planet = Mercury
;;
;; Version History:
;;   1.0: written 8 Nov 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (peakflux EQ !null) $
  then map = *sourcemap.map $
  else map = *sourcemap.map/max(*sourcemap.map)*peakflux

dlon = (*sourcemap.longitude)[1]-(*sourcemap.longitude)[0]
dlat = (*sourcemap.latitude)[1]-(*sourcemap.latitude)[0]
dcos = one(*sourcemap.longitude) # cos((*sourcemap.latitude))

sourcerate = total(map*dcos) * (!Mercury.radius*1e5)^2 * dlon * dlat

return, sourcerate

end


```

```
pro add_perturbation, startloc, PerturbVel, options, seed

common constants

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Adds a perturbation to a pre-existing velocity distribution
;;
;; Version History
;;   2.0: created 10/24/08
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

npack = options.packets
case (PerturbVel.type) of
  'none':
  'gaussian': if (PerturbVel.sigma EQ 0) $
    then vperturb = replicate(PerturbVel.vprob, npack) $
    else begin
      maxv = PerturbVel.vprob + 4*PerturbVel.sigma
      velocity = findgen(1001)/1000.*maxv
      velocity = velocity[where(velocity GT PerturbVel.vprob - 4*PerturbVel.sigma)]
      f_v = GaussianDist(velocity, PerturbVel.vprob, PerturbVel.sigma)
      vperturb = MonteCarloDistribution(velocity, f_v, npack)

      ;; Choose the altitude -- f(alt) = cos(alt)
      altitude = dindgen(1001)/1000. * ((PerturbVel.altitude)[1]-$
        (PerturbVel.altitude)[0]) + (PerturbVel.altitude)[0]
      f_alt = cos(altitude)
      alt = MonteCarloDistribution(altitude, f_alt, npack)

      ;; Choose the aziumth
      if ((PerturbVel.azimuth)[0] GT (PerturbVel.azimuth)[1]) $
        then m = [(PerturbVel.azimuth)[0], (PerturbVel.azimuth)[1]+2*!pi] $
        else m = PerturbVel.azimuth
      az = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)

      *startloc.altitude = alt
      *startloc.azimuth = az

      v_north = sin(alt)
      v_corot = -cos(alt) * cos(az)
      v_rad = cos(alt) * sin(az)

      vxperturb = v_rad * vperturb
      vyperturb = v_corot * vperturb
      vzperturb = v_north * vperturb

      ;; Need to rotate the perturbation vectors to proper orientation
      ;; Want az=0 => corotational direction
      ;;      az=90 => radial direction
```

```

        ;; Starting velocity
        *startloc.vx += vxperturb
        *startloc.vy += vyperturb
        *startloc.vz += vzperturb
      endelse
    'trigaussian': begin
      if (PerturbVel.vxsigma EQ 0) $
        then vxperturb = replicate(PerturbVel.vxprob, npack) $
        else begin
          maxv = PerturbVel.vxprob + 4*PerturbVel.vxsigma
          velocity = (findgen(2001)/1000.-1)*maxv
          f_v = GaussianDist(velocity, PerturbVel.vxprob, PerturbVel.vxsigma)
          vxperturb = -MonteCarloDistribution(velocity, f_v, npack)
        endelse

      if (PerturbVel.vysigma EQ 0) $
        then vyperturb = replicate(PerturbVel.vyprob, npack) $
        else begin
          maxv = PerturbVel.vyprob + 4*PerturbVel.vysigma
          velocity = (findgen(2001)/1000.-1)*maxv
          f_v = GaussianDist(velocity, PerturbVel.vyprob, PerturbVel.vysigma)
          vyperturb = MonteCarloDistribution(velocity, f_v, npack)
        endelse

      if (PerturbVel.vzsigma EQ 0) $
        then vzperturb = replicate(PerturbVel.vzprob, npack) $
        else begin
          maxv = PerturbVel.vzprob + 4*PerturbVel.vzsigma
          velocity = (findgen(2001)/1000.-1)*maxv
          f_v = GaussianDist(velocity, PerturbVel.vzprob, PerturbVel.vzsigma)
          vzperturb = MonteCarloDistribution(velocity, f_v, npack)
        endelse

        ;; Need to rotate to the location of the packets
        ang = atan(-*startloc.x, *startloc.y)
        vxpert2 = vxperturb * cos(ang) - vyperturb * sin(ang)
        vypert2 = vxperturb * sin(ang) + vyperturb * cos(ang)
        vzpert2 = vzperturb

        ;; Starting velocity
        *startloc.vx += vxpert2/SystemConsts.rplan
        *startloc.vy += vypert2/SystemConsts.rplan
        *startloc.vz += vzpert2/SystemConsts.rplan
        end
    'sputtering': stop
    'charge exchange': charge_exchange_perturbation, startloc, PerturbVel, options
  endcase

  end
```

```
pro angular_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version History
;;  3.1: 1/5/2011
;;   * Changing the way the altitude is chosen. sin(alt) is evenly distributed between
;;     minimum and maximum angles.
;;  3.0: 7/19/2010
;;   * revise for new structure architecture
;;  2.2: 17 November 2009
;;    * changed the way the costheta distrubution works.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common Constants

AngularDist = input.AngularDist
vv = *output.vx0
case strlowcase(input.AngularDist.type) of
  'none':
  'radial': begin
    rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
    alt = replicate(!pi/2., npack) ;; set all packets going directly up
    az = fltarr(npack)
    end
  'isotropic': begin
    ;; Choose the altitude -- f(alt) = cos(alt)
    aa = sin(AngularDist.altitude)
    sinalt = random_nr(seed=seed, npack) * (aa[1]-aa[0]) + aa[0]
    alt = asin(sinalt)

    ;; Choose the longitude -- f(lon) = 1 / (lonmax-lonmin)
    if ((AngularDist.azimuth)[0] GT (AngularDist.azimuth)[1]) $
      then m = [(AngularDist.azimuth)[0], (AngularDist.azimuth)[1]+2*!pi] $
      else m = AngularDist.azimuth
    az = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)
    end
  'costheta': begin
    aa = sin(AngularDist.altitude)
    sinalt = dindgen(1001)/1000. * (aa[1]-aa[0]) + aa[0]
    f_sinalt = sinalt^AngularDist.n
    sinalt = RandomDeviates_1d(sinalt, f_sinalt, npack)
    alt = asin(sinalt)

    if ((AngularDist.azimuth)[0] GT (AngularDist.azimuth)[1]) $
      then m = [(AngularDist.azimuth)[0], (AngularDist.azimuth)[1]+2*!pi] $
      else m = AngularDist.azimuth
    az = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)
    end
endcase
```

```

;; Find the velocity components in coordinate system centered on the packet
v_rad = sin(alt)              ;; Radial component of velocity
v_tan0 = cos(alt) * cos(az)  ;; Component along latitude line (points east)
v_tan1 = cos(alt) * sin(az)   ;; Component along longitude line (points to NP)
;; Now rotate to the proper surface point
;; v_ren = M # v_xyz => v_xyz = invert(M) # v_ren

rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
x0 = *output.x0/rr & y0 = *output.y0/rr & z0 = *output.z0/rr

*output.vx0 = dblarr(npack)
*output.vy0 = dblarr(npack)
*output.vz0 = dblarr(npack)
for i=0L,npack-1 do begin
  rad = [x0[i], y0[i], z0[i]]
  east = [y0[i], -x0[i], 0]
  north = [-z0[i]*x0[i], -z0[i]*y0[i], x0[i]^2+y0[i]^2]

  east /= sqrt(total(east*east))
  north /= sqrt(total(north*north))

  v0 = v_tan0[i]*north + v_tan1[i]*east + v_rad[i]*rad
  if (abs(total(v0*v0))-1 GT 1e-3) then stop
  (*output.vx0)[i] = v0[0] * vv[i]
  (*output.vy0)[i] = v0[1] * vv[i]
  (*output.vz0)[i] = v0[2] * vv[i]
endfor

end
```

```
pro charge_exchange_perturbation, startloc, PerturbVel, options

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Add a perturbation velocity based on charge exchange
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

;; 1) find the appropriate charge exchange reactions
path = '$HOME/Data/AtomicData/Loss/'
defaults = path+'DefaultsList.dat'
readcol, defaults, species, reac, file, delim=':', /silent, skip=1, format='A,A,A'
species = strtrim(species, 2)
reac = strtrim(reac, 2)
file = strtrim(file, 2)

q = where(species EQ options.atom, nq)
if (nq EQ 0) then stop

rsub = reac[q]
fsub = file[q]

for i=0,nq-1 do begin
  restore, path+fsub[i]
  print, ratecoef.type
  if (strlowcase(ratecoef.type) EQ 'ion-neutral') then begin
    ;; determine if correct product is formed

    stop

  endif else destroy_ratecoef, ratecoef
endfor

stop
end
```

```
pro exosphere_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;   Distribute packets from a spherically symmetric exosphere
;;     f(r) = r^b
;;   or
;;     f(r) = exp(-r/h)
;;
;; Version History
;;   2.1: 20 November 2009
;;     * Added option to prevent packet creation in planet's geometric shadow
;;     * Added option to choose between specifying a scale height or a powerlaw exponent
;;   2.0: File created
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

stop
todo = lindgen(npack)
SpatialDist = input.SpatialDist

;; Set the angular distribution
ll = !dpi*dindgen(1001)/1000. - !dpi/2.
f_lat = cos(ll)

r = findgen(10001)/100.+1
r = r[where(r LE SpatialDist.rmax)]
case (SpatialDist.exotype) of
  'powerlaw': f_r = r^SpatialDist.b
  'exponential': f_r = exp(-(r-1)/SpatialDist.b)
endcase
f_r[0] = 0. ;; Don't allow packets to start right at the surface

*output.x0 = dblarr(npack)
*output.y0 = dblarr(npack)
*output.z0 = dblarr(npack)
while (npack GT 0) do begin
  lat = MonteCarloDistribution(ll, f_lat, npack)
  lon = 2*!dpi * random_nr(seed=seed, npack)

  rr = MonteCarloDistribution(r, f_r, npack)
  q = where(rr LT 1. or rr GT SpatialDist.rmax, nq)
  while (nq NE 0) do begin
    w = MonteCarloDistribution(r, f_r, nq)
    rr[q] = w
    q = where(rr LT 1. or rr GT SpatialDist.rmax, nq)
  endwhile

  if strcmp(input.geometry.planet, input.geometry.StartPoint, /fold) then begin
    ;; Starting at a planet
    (*output.x0)[todo] = double(rr * sin(lon)*cos(lat)) ;; longitude = 0 => -yaxis
    (*output.y0)[todo] = -double(rr * cos(lon)*cos(lat)) ;; longitude = 90 => axis
    (*output.z0)[todo] = double(rr * sin(lat))
  endif else begin
    ;; Starting at a satellite
    (*output.x0)[todo] = -double(rr * sin(lon)*cos(lat)) ;; longitude = 0 => -yaxis
    (*output.y0)[todo] = -double(rr * cos(lon)*cos(lat)) ;; longitude = 90 => axis
    (*output.z0)[todo] = double(rr * sin(lat))
  endelse

  rho = *output.x0^2 + *output.z0^2

  ;; not working right if block_shadow and starting at a satellite
  if (spatialdist.block_shadow) and ~(strcmp(input.geometry.planet, $
    input.geometry.StartPoint, /fold)) then stop

  if (spatialdist.block_shadow) $
    then todo = where((rho LE 1) and (*output.y0 GT 0), npack) $
    else npack = 0
endwhile

end
```

```
pro show_veldist, proc_info, run_info, vrange=vrange, theo=theo, actual=actual, $
  display=display

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; display = 0 --> does not plot
;; display = 1 --> only plots theoretical distribution function
;; display = 2 --> Also plots a random distribution function for correct # of packets
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SystemConstants, run_info.planet, systemconsts

if (n_elements(npack) NE 1) then npack = min([100000, run_info.packets])
if (n_elements(display) EQ 0) then display = 2

dv = 0.01
vrange = findgen(20000L)*dv
vrange = vrange[where(vrange LE 100.)] + dv

dodist = 0
case strlowcase(proc_info.speeddist) of
  'gaussian': begin
    a = [1., proc_info.vprob, proc_info.delv]
    theo = gauss(vrange, a)
    theo = theo * npack/total(theo)/dv

    bin = 0.01
    act = randomn(seed, npack)*proc_info.delV + proc_info.vprob
    actual = histw(act, min=0., max=max(vrange), bin=bin)/dv
    print, 'Gaussian speed distribution'
    print, 'P(v) ~ exp(-(v-v_prob)^2/(2*sigma^2))'
    print, '  v_prob = ' + strtrim(string(proc_info.vprob), 2) + ' km/s'
    print, '  sigma = ' + strtrim(string(proc_info.delv), 2) + ' km/s'
    print, '************'

    q = where(theo*dv GT .1, c)
    xr=[0,vrange[q[c-1]]]
    ylog = 1
    yr = [.1/dv, max(theo)]
    end
  'flat': begin
    q = where((vrange GE proc_info.vprob-proc_info.delV/2.) and $
      (vrange LE proc_info.vprob+proc_info.delV/2.))
    theo = fltarr(n_elements(vrange))
    theo[q] = 1.0
    theo = theo * npack/total(theo)/dv

    bin = 0.01
    act = random_nr(npack)*proc_info.delV + proc_info.vprob - proc_info.delV/2.
    actual = histw(act, min=0., max=max(vrange), bin=bin)/dv
    print, 'Flat speed distribution'
    print, 'v_prob-sigma/2. <= v <= v_prob+sigma/2.'
    print, '  v_prob = ' + strtrim(string(proc_info.vprob), 2) + ' km/s'
    print, '  sigma = ' + strtrim(string(proc_info.delv), 2) + ' km/s'
    print, '************'

    xr=[proc_info.vprob-proc_info.delv*1.5, proc_info.vprob+proc_info.delv*1.5]
    ylog = 0
    yr= [0,max(theo)]*1.1

    end
  'sputtering': begin
    theo = sputdist(vrange, proc_info.vprob, proc_info.alpha, proc_info.beta, $
      run_info.atom, v_b=v_b)
    theo = theo * npack/total(theo)/dv
    bin=.1
    dodist = 1
    print, 'Sputtering speed distribution'
    print, 'f(v) ~ v^(2*beta+1) / (v^2 + v_b^2)^alpha'
    print, '  alpha = ' + strtrim(string(proc_info.alpha),2)
    print, '  beta = ' + strtrim(string(proc_info.beta), 2)
    print, '  v_b = sqrt(2*U/m) = ' + strtrim(string(v_b), 2) + ' km/s'
    print, '  U = ' + strtrim(string(proc_info.vprob), 2) + ' eV'
    print, '************'
    end
  'exponential': begin
    ;theo = expflux(vrange, proc_info.temperature, run_info.atom, proc_info.beta, v_t=v_t)
    ;theo = theo * npack/total(theo)/dv
    theo = fltarr(n_elements(vrange))
    bin=.01
    dodist = 1
    if (n_elements(v_t) EQ 0) then v_t = 0.
    print, 'Exponential speed distribution'
    print, 'f(v) ~ v^beta * exp(-(v/v_t)^2)'
    print, '  beta = ' + strtrim(string(proc_info.beta), 2)
    print, '  v_t = sqrt(2kT/m) = ' + strtrim(string(v_t), 2) + ' km/s'
    print, '  T = ' + strtrim(string(proc_info.temperature), 2) + ' K'
    print, '************'
    end
  else: begin
    print, 'This option is not ready'
    return
    end
endcase

if (dodist) then begin
  if (run_info.mintrack GT 0) then begin
    nil = where(vrange LT run_info.mintrack, comp=notnil)
    proc_info.prodrate = proc_info.prodrate * total(theo[notnil])/total(theo)
    theo[nil] = 0.
  endif
```

```

    sumdist = theo
    n = n_elements(theo)
    for i=0,n-2 do sumdist[i+1] = sumdist[i+1] + sumdist[i]
    sumdist = sumdist/total(theo)
    t = random_nr(npack)
    act = interpol(vrange, sumdist, t)
    actual = histw(act, min=0, max=max(vrange), bin=bin)/bin

    q = where(theo*dv GT 1, c)
    xr=[0,vrange[q[c-1]]]
    ylog = 1
    yr = [.1/dv, max(theo)]
  endif

  q = where(strmatch(*systemconsts.objects, run_info.startpoint, /fold))
  v_esc = sqrt(-2*(*systemconsts.GM)[q]/(*systemconsts.radius)[q])*systemconsts.rplan

  if ((display EQ 1) or (display EQ 2)) then begin
    plot, vrange, theo, /xst, yr=yr, title='!17' + proc_info.speeddist + $
      ' Speed Distibution', xtit='Speed (km/s)', ytit='f(v) (km/s)!e-1!n', ylog=ylog, xr=xr
    if (display EQ 2) then oplot, findgen(n_elements(actual))*bin, actual, color=2
    oplot, [v_esc,v_esc], yr*[.01,100], color=5, linest=2
    xyouts, v_esc*1.1, yr[0]*2, 'v!desc!n', color=5
  endif

  destroy_constants, systemconsts

  end
```

```
pro source_distribution, input, npack, seed, output=output

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Determine the initial positions and velocities for each packet
;;  This puts everything into one program and removes it from modjup.
;;
;;  A description of each step in this program is given in MonteCarlo.tex
;;
;;  Options:
;;    (1) Spatial Distributions
;;      (a) Surface  -- satellite-centric
;;      (b) SO_2 Exosphere  -- satellite-centric
;;      (c) Torus -- planet-centric
;;      (d) cloud -- planet-centric
;;      (e) exosphere - satellite-centric
;;
;;    (2) Speed Distribution
;;      (a) Gaussian -- f(v) ~ v_prob + exp(.5*(v/vth)^2)
;;      (b) Sputtering
;;      (c) maxwellian
;;      (d) dolsfunction
;;      (e) curcular orbits
;;      (f) flat
;;
;;    (3) angular distributions
;;      (a) radial
;;      (b) cos(theta)
;;      (c) isotropic
;;
;;  Version History
;;    3.1: 1/3/2011
;;     * Makeing output.sourcefile a pointer
;;    3.0: 7/19/2010
;;      * Rewriting with new structure format
;;
;;    1.0 - 10/23/08
;;      * begin version control
;;      * originally written 19 June 2006
;;      * modified: 22 Oct 2007
;;         -- replaced randomu with random_nr
;;         -- still need to replace randomn
;;      * modified: 9 June 2008
;;    2.0 - 10/23/08
;;      * re-writing to include complete creation of loc structure and farm out more
;;        bits
;;    2.1 - 2/11/09
;;      * Add option for molecular dissociation of exospheric source
;;    2.2 - 1/14/10
;;      * Add fields to the loc structure to keep track of fate of packets
;;
```

```
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

;; Decide where the starting point is
s = stuff.s

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 1) Create the structures
if (input.options.trackloss) $
  then output = {x0:ptr_new(0), y0:ptr_new(0), z0:ptr_new(0), f0:ptr_new(0), $
    vx0:ptr_new(0), vy0:ptr_new(0), vz0:ptr_new(0), phi0:ptr_new(0), $
    totalsource:0., time:ptr_new(0), $
    x:ptr_new(0), y:ptr_new(0), z:ptr_new(0), frac:ptr_new(0), $
    vx:ptr_new(0), vy:ptr_new(0), vz:ptr_new(0), $
    lossfrac:ptr_new(0), hitfrac:ptr_new(0), ringfrac:ptr_new(0), $
    leftfrac:ptr_new(0), $
    deposition:{longitude:ptr_new(), latitude:ptr_new(), map:ptr_new()}, $
    loss_info:{reactions:ptr_new(), files:ptr_new(), type:ptr_new()}, $
    sourcefile:ptr_new('modeloutput')} $
  else output = {x0:ptr_new(0), y0:ptr_new(0), z0:ptr_new(0), f0:ptr_new(0), $
    vx0:ptr_new(0), vy0:ptr_new(0), vz0:ptr_new(0), phi0:ptr_new(0), $
    totalsource:0., time:ptr_new(0), $
    x:ptr_new(0), y:ptr_new(0), z:ptr_new(0), frac:ptr_new(0), $
    vx:ptr_new(0), vy:ptr_new(0), vz:ptr_new(0), $
    loss_info:{reactions:ptr_new(), files:ptr_new(), type:ptr_new()}, $
    sourcefile:ptr_new('modeloutput')}
  *output.f0 = replicate(1d, npack)

  ;; Determine the endtime of each packet
  *output.time = (input.options.at_once) ? $
    replicate(input.options.endtime, npack) : $
    random_nr(seed=seed, npack) * input.options.endtime

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; 2) Spatial distribution
  ;;  Choose a starting location for each packet.
  case strlowcase(input.SpatialDist.type) of
  ;; note -- torus and SO2 exosphere distributions not revised yet
  'surface': surface_distribution, input, output, npack, seed
  'torus': stop; torus_distribution, geometry, spatialdist, options, seed, startloc=startloc
  'exosphere': exosphere_distribution, input, output, npack, seed
  'so2 exosphere': SO2exosphere_distribution, input, output, npack, seed
  else: stop
endcase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Part 3: Velocity distribution
;; Choose a speed and direction for each packet
speed_distribution, input, output, npack, seed
```

```
q = where(finite(*output.vx0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz0) EQ 0, nq) & if (nq NE 0) then stop

if (strlowcase(input.angulardist.type) NE 'none') then $
  angular_distribution, input, output, npack, seed
q = where(finite(*output.vx0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz0) EQ 0, nq) & if (nq NE 0) then stop

if (input.PerturbVel.type NE 'none') then stop ;; not revised yet
;  add_perturbation, startloc, PerturbVel, options, seed

;; Now have initial positions
;;   x,y,z in either Rplan or Rsat
;;   vx,vy,vz in Rplan/s
;;   time
;; * Still need to move packets to the proper position relative to the planet
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Part 4: Rotate everything to proper position for running the model
;; * if using a planet-centered distribution (torus), then don't need to do
;;   anything special

if (stuff.s EQ 0) then begin ;; Everything is already setup correctly
  *output.x = *output.x0
  *output.y = *output.y0
  *output.z = *output.z0

  *output.vx = *output.vx0
  *output.vy = *output.vy0
  *output.vz = *output.vz0

  *output.phi0 = 0.  ;; this is meaningless for planet-centered distribution
endif else begin
  ;; Move packets out to their starting distance
  xx = *output.x0*(*SystemConsts.radius)[stuff.s]
  yy = *output.y0*(*SystemConsts.radius)[stuff.s] + (*SystemConsts.a)[stuff.s]
  zz = *output.z0*(*SystemConsts.radius)[stuff.s]

  ;; Add in orbital velocity if needed
  vx = *output.vx0 - input.options.motion*(*SystemConsts.orbvel)[stuff.s]/$
    SystemConsts.rplan
  vy = *output.vy0
  vz = *output.vz0

  phi = (*input.geometry.phi)[stuff.s]
  ;; Rotate to proper starting position based on *output.time
  if (input.options.motion) $
    then locmoon, *output.time, phi, (*SystemConsts.a)[stuff.s], $
      (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty , ang=ang $
    else locmoon, fltarr(npack), phi, (*SystemConsts.a)[stuff.s], $
      (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty, ang=ang

  ang = (ang + 2*!dpi) mod (2*!dpi)
  *output.phi0 = ang  ;; Starting local time for each packet

  ;; Rotate to proper starting orbital phase
  *output.x = xx * cos(ang) - yy * sin(ang)
  *output.y = xx * sin(ang) + yy * cos(ang)
  *output.z = zz

  *output.vx = vx * cos(ang) - vy * sin(ang)
  *output.vy = vx * sin(ang) + vy * cos(ang)
  *output.vz = vz
endelse

q = where(finite(*output.vx) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz) EQ 0, nq) & if (nq NE 0) then stop

*output.frac = *output.f0
output.totalsource = total(*output.frac)

end
```

```
pro source_distribution, input, npack, seed, output=output

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Determine the initial positions and velocities for each packet
;;  This puts everything into one program and removes it from modjup.
;;
;;  A description of each step in this program is given in MonteCarlo.tex
;;
;;  Options:
;;    (1) Spatial Distributions
;;      (a) Surface  -- satellite-centric
;;      (b) SO_2 Exosphere  -- satellite-centric
;;      (c) Torus -- planet-centric
;;      (d) cloud -- planet-centric
;;      (e) exosphere - satellite-centric
;;      (f) PSD - satellite-centric
;;
;;    (2) Speed Distribution
;;      (a) Gaussian -- f(v) ~ v_prob + exp(.5*(v/vth)^2)
;;      (b) Sputtering
;;      (c) maxwellian
;;      (d) dolsfunction
;;      (e) curcular orbits
;;      (f) flat
;;
;;    (3) angular distributions
;;      (a) radial
;;      (b) cos(theta)
;;      (c) isotropic
;;
;;  Version History
;;    3.1: 1/3/2011
;;     * Makeing output.sourcefile a pointer
;;    3.0: 7/19/2010
;;      * Rewriting with new structure format
;;
;;    1.0 - 10/23/08
;;      * begin version control
;;      * originally written 19 June 2006
;;      * modified: 22 Oct 2007
;;          -- replaced randomu with random_nr
;;          -- still need to replace randomn
;;      * modified: 9 June 2008
;;    2.0 - 10/23/08
;;      * re-writing to include complete creation of loc structure and farm out more
;;          bits
;;    2.1 - 2/11/09
;;      * Add option for molecular dissociation of exospheric source
;;    2.2 - 1/14/10
;;      * Add fields to the loc structure to keep track of fate of packets
```

```
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants

;; Decide where the starting point is
s = stuff.s

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 1) Create the structures
if (input.options.trackloss) $
  then output = {x0:ptr_new(0), y0:ptr_new(0), z0:ptr_new(0), f0:ptr_new(0), $
    vx0:ptr_new(0), vy0:ptr_new(0), vz0:ptr_new(0), phi0:ptr_new(0), $
    totalsource:0., time:ptr_new(0), $
    x:ptr_new(0), y:ptr_new(0), z:ptr_new(0), frac:ptr_new(0), $
    vx:ptr_new(0), vy:ptr_new(0), vz:ptr_new(0), $
    lossfrac:ptr_new(0), hitfrac:ptr_new(0), ringfrac:ptr_new(0), $
    leftfrac:ptr_new(0), $
    deposition:{longitude:ptr_new(), latitude:ptr_new(), map:ptr_new()}, $
    loss_info:{reactions:ptr_new(), files:ptr_new(), type:ptr_new()}, $
    sourcefile:ptr_new('modeloutput')} $
  else output = {x0:ptr_new(0), y0:ptr_new(0), z0:ptr_new(0), f0:ptr_new(0), $
    vx0:ptr_new(0), vy0:ptr_new(0), vz0:ptr_new(0), phi0:ptr_new(0), $
    totalsource:0., time:ptr_new(0), $
    x:ptr_new(0), y:ptr_new(0), z:ptr_new(0), frac:ptr_new(0), $
    vx:ptr_new(0), vy:ptr_new(0), vz:ptr_new(0), $
    loss_info:{reactions:ptr_new(), files:ptr_new(), type:ptr_new()}, $
    sourcefile:ptr_new('modeloutput')}
  *output.f0 = replicate(1d, npack)

  ;; Determine the endtime of each packet
  *output.time = (input.options.at_once) ? $
    replicate(input.options.endtime, npack) : $
    random_nr(seed=seed, npack) * input.options.endtime

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; 2) Spatial distribution
  ;;  Choose a starting location for each packet.
  case strlowcase(input.SpatialDist.type) of
  ;; note -- torus and SO2 exosphere distributions not revised yet
  'surface': surface_distribution, input, output, npack, seed
  'torus': stop; torus_distribution, geometry, spatialdist, options, seed, startloc=startloc
  'exosphere': exosphere_distribution, input, output, npack, seed
  'so2 exosphere': SO2exosphere_distribution, input, output, npack, seed
  'psd': PSD_distribution, input, output, npack, seed
  else: stop
endcase
q = where(finite(*output.x0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.y0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.z0) EQ 0, nq) & if (nq NE 0) then stop
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Part 3: Velocity distribution
;; Choose a speed and direction for each packet
speed_distribution, input, output, npack, seed
q = where(finite(*output.vx0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz0) EQ 0, nq) & if (nq NE 0) then stop

if (strlowcase(input.angulardist.type) NE 'none') then $
  angular_distribution, input, output, npack, seed
q = where(finite(*output.vx0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz0) EQ 0, nq) & if (nq NE 0) then stop

if (input.PerturbVel.type NE 'none') then stop ;; not revised yet
; add_perturbation, startloc, PerturbVel, options, seed

;; Now have initial positions
;;   x,y,z in either Rplan or Rsat
;;   vx,vy,vz in Rplan/s
;;   time
;; * Still need to move packets to the proper position relative to the planet
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Part 4: Rotate everything to proper position for running the model
;; * if using a planet-centered distribution (torus), then don't need to do
;;   anything special

if (stuff.s EQ 0) then begin ;; Everything is already setup correctly
  *output.x = *output.x0
  *output.y = *output.y0
  *output.z = *output.z0

  *output.vx = *output.vx0
  *output.vy = *output.vy0
  *output.vz = *output.vz0

  *output.phi0 = 0.  ;; this is meaningless for planet-centered distribution
endif else begin
  ;; Move packets out to their starting distance
  xx = *output.x0*(*SystemConsts.radius)[stuff.s]
  yy = *output.y0*(*SystemConsts.radius)[stuff.s] + (*SystemConsts.a)[stuff.s]
  zz = *output.z0*(*SystemConsts.radius)[stuff.s]

  ;; Add in orbital velocity if needed
  vx = *output.vx0 - input.options.motion*(*SystemConsts.orbvel)[stuff.s]/$
    SystemConsts.rplan
  vy = *output.vy0
  vz = *output.vz0

  phi = (*input.geometry.phi)[stuff.s]
  ;; Rotate to proper starting position based on *output.time
  if (input.options.motion) $
    then locmoon, *output.time, phi, (*SystemConsts.a)[stuff.s], $
      (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty , ang=ang $
    else locmoon, fltarr(npack), phi, (*SystemConsts.a)[stuff.s], $
      (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty, ang=ang

  ang = (ang + 2*!dpi) mod (2*!dpi)
  *output.phi0 = ang  ;; Starting local time for each packet

  ;; Rotate to proper starting orbital phase
  *output.x = xx * cos(ang) - yy * sin(ang)
  *output.y = xx * sin(ang) + yy * cos(ang)
  *output.z = zz

  *output.vx = vx * cos(ang) - vy * sin(ang)
  *output.vy = vx * sin(ang) + vy * cos(ang)
  *output.vz = vz
endelse

q = where(finite(*output.vx) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vy) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.vz) EQ 0, nq) & if (nq NE 0) then stop

*output.frac = *output.f0
output.totalsource = total(*output.frac)

end
```

```
pro speed_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version History:
;;   3.2: 12/13/2010
;;       * changes to random deviates methods
;;   3.1: 9/15/2010
;;       * added option for "user defined" speed distribution
;;   3.0: 7/19/2010
;;       * Revised with new structure architechture
;;   2.3: 1/12/10
;;       * Added thermal distribution where velocity distribution depends on local
;;         surface temperature
;;   2.2: 3/3/09
;;       * Added support for Weibull distribution [removed 1/20/10]
;;   2.1 --
;;       * broke velocity_distribution_2.0 into separate speed, angular, and perturbation
;;  components
;;
;; Returns either an array of speeds in *output.vx0 or the full velocity
;; in *output.vx0, *output.vy0, *output.vz0
;;   -- Right now only returns full velocity if "circular orbits" speed distribution is
;;      chosen
;;
;; All speeds are returned in units of Rplan
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common Constants

SpeedDist = input.SpeedDist
case strlowcase(SpeedDist.type) of
  'gaussian': begin
    if (SpeedDist.sigma EQ 0) $
      then *output.vx0 = replicate(SpeedDist.vprob, npack) $
      else *output.vx0 = RandomGaussian(npack, SpeedDist.vprob, SpeedDist.sigma)
    end
  'trigaussian': begin
    if (SpeedDist.vxsigma EQ 0) $
      then *output.vx0 = replicate(SpeedDist.vxprob, npack) $
      else *output.vx0 = RandomGaussian(npack, SpeedDist.vxprob, SpeedDist.vxsigma)

    if (SpeedDist.vysigma EQ 0) $
      then *output.vy0 = replicate(SpeedDist.vyprob, npack) $
      else *output.vy0 = RandomGaussian(npack, SpeedDist.vyprob, SpeedDist.vysigma)

    if (SpeedDist.vzsigma EQ 0) $
      then *output.vz0 = replicate(SpeedDist.vzprob, npack) $
      else *output.vz0 = RandomGaussian(npack, SpeedDist.vzprob, SpeedDist.vzsigma)
    end
    'dolsfunction': begin
      stop
      velocity = findgen(1001.)/100.
      f_v = dolsdist(velocity, SpeedDist.dols0, SpeedDist.dols1, input.options.atom)
      *output.vx0 = RandomDeviates_1d(velocity, f_v, npack)   ;; km/s
      end
    'sputtering': begin
      velocity = findgen(5000)/100.+.1
      f_v = sputdist(velocity, SpeedDist.U, SpeedDist.alpha, SpeedDist.beta,$
        input.options.atom)
      *output.vx0 = RandomDeviates_1d(velocity, f_v, npack)   ;; km/s
      end
    'maxwellian': begin
      if (SpeedDist.temperature NE 0) then begin
        ;; Use a constant surface temperature
        v_th = sqrt(2*SpeedDist.temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        velocity = findgen(1001)/1000 * v_th*5 & velocity = velocity[1:*]
        f_v = MaxwellianDist(velocity, SpeedDist.temperature, input.options.atom)
        *output.vx0 = RandomDeviates_1d(velocity, f_v, npack)   ;; km/s
      endif else begin
        ;; Use a surface temperature map
        rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
        SZA = acos(-*output.y0/rr)  ;; cos(SZA) = [0,-1,0]·[x,y,z]/r = -y/r
        q = where(finite(SZA) EQ 0, nq) & if (nq NE 0) then stio
        surftemp = surface_temperature(input.geometry, SZA)

        nt = 101 & np = 1001
        temperature = dindgen(nt)/(nt-1)*(max(surftemp)-min(surftemp)) + min(surftemp)
        v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        prob = dindgen(np)/(np-1)
        vgrid = dblarr(nt,np)
        for i=0,nt-1 do begin
          ;; Produces the velocity as fn of T and cumulative value.
          ;; Given T and random P, can get v
          vrange = dindgen(np)/(np-1)*v_temp[i]*3.
          f_v = MaxwellianDist(vrange, temperature[i], input.options.atom)
          sumdist = f_v
          for j=1,np-1 do sumdist[j] += sumdist[j-1]
          sumdist /= max(sumdist)
          vgrid[i,*] = interpol(vrange, sumdist, prob)
        endfor
        p = random_nr(seed=seed, npack)
        *output.vx0 = interpolate_xy(vgrid, temperature, prob, surftemp, p)
      endelse
      end
    'maxwellian2': begin
      if (SpeedDist.temperature NE 0) then begin
        ;; Use a constant surface temperature
        v_th = sqrt(2*SpeedDist.temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        velocity = findgen(1001)/1000 * v_th*5 & velocity = velocity[1:*]
        f_v = MaxwellianDist2(velocity, SpeedDist.temperature, input.options.atom)
        *output.vx0 = RandomDeviates_1d(velocity, f_v, npack)   ;; km/s
      endif else begin
        ;; Use a surface temperature map
        rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
        SZA = acos(-*output.y0/rr)  ;; cos(SZA) = [0,-1,0]·[x,y,z]/r = -y/r
        q = where(finite(SZA) EQ 0, nq) & if (nq NE 0) then stio
        surftemp = surface_temperature(input.geometry, SZA)

        nt = 101 & np = 1001
        temperature = dindgen(nt)/(nt-1)*(max(surftemp)-min(surftemp)) + min(surftemp)
        v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        prob = dindgen(np)/(np-1)
        vgrid = dblarr(nt,np)
        for i=0,nt-1 do begin
          ;; Produces the velocity as fn of T and cumulative value.
          ;; Given T and random P, can get v
          vrange = dindgen(np)/(np-1)*v_temp[i]*3.
          f_v = MaxwellianDist2(vrange, temperature[i], input.options.atom)
          sumdist = f_v
          for j=1,np-1 do sumdist[j] += sumdist[j-1]
          sumdist /= max(sumdist)
          vgrid[i,*] = interpol(vrange, sumdist, prob)
        endfor
        p = random_nr(seed=seed, npack)
        *output.vx0 = interpolate_xy(vgrid, temperature, prob, surftemp, p)
      endelse
      end
    'flat': *output.vx0 = random_nr(npack)*(2*SpeedDist.delv) + SpeedDist.vprob - $
      SpeedDist.delv
    'circular orbits': begin
      ;; Determine the Keplerian velocity
      rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
      velocity = sqrt(abs((*SystemConsts.GM)[0]/rr))*SystemConsts.rPlan ;; Kepler vel.

      ;; Determine the plane of the orbit
      ;; All orbits are in the z x r direction
      xhat = *output.x0/rr & yhat = *output.y0/rr & zhat = *output.z0/rr
      zaxis = [0., 0., 1]
      vhat = fltarr(npack, 3)
      for i=0L,npack-1 do begin
        vhat[i,*] = crossp(zaxis, [xhat[i], yhat[i], zhat[i]])
        vhat[i,*] = vhat[i,*]/sqrt(total(vhat[i,*]*vhat[i,*]))
      endfor

      ;; Starting velocity
      *output.vx0 = velocity * vhat[*,0]
      *output.vy0 = velocity * vhat[*,1]
      *output.vz0 = velocity * vhat[*,2]
      end
    'user defined': begin
      restore, speeddist.distfile
    *output.vx0 = RandomDeviates_1d(*speeddistribution.v, *speeddistribution.fv, npack)
    destroy_structure, speeddistribution
    end
  else: stop
endcase

*output.vx0 /= SystemConsts.rplan
*output.vy0 /= SystemConsts.rplan
*output.vz0 /= SystemConsts.rplan

end
```

```
function gaussiandist, velocity, vprob, sigma

;; Velocity, vprob, sigma must be in the same units.
f_v = exp(-(velocity-vprob)^2/2./sigma^2)
return, f_v

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function dolsdist, velocity, dols0, dols1, atom

;; Velocity must be in km/s
;; dols0 and dols1 are in eV, basically.

tt = .5*atomicmass(atom)*(velocity*1e5)^2/!const.erg_eV
f_v = (velocity*1e5) * exp(-(tt-dols0)^2/dols1^2)
f_v /= max(f_v)
return, f_v

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function sputdist, velocity, U, alpha, bet, atom, v_b=v_b

;; Generic sputtering distribution
;; See helpwiki for explanation

matom = atomicmass(atom)
vb = sqrt(2*U*!const.erg_eV/matom)/1e5
f_v = velocity^(2*bet+1) / (velocity^2 + vb^2)^alpha
return, f_v

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function MaxwellianDist, velocity, temperature, atom

;; Velocity must be in km/s
;; Temperature in K

v_th2 = 2*temperature*!const.kb/atomicmass(atom)/1e10
f_v = velocity^2 * exp(-velocity^2/v_th2)
return, f_v

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
function MaxwellianDist2, velocity, temperature, atom

;; Velocity must be in km/s
;; Temperature in K

v_th2 = 2*temperature*!const.kb/atomicmass(atom)/1e10
f_v = velocity^3 * exp(-velocity^2/v_th2)
return, f_v

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function WeibullDist, velocity, temperature, alpha, atom

v_th = sqrt(2*temperature*!const.kb/atomicmass(atom))/1e5
f_v = velocity^(alpha-1) * exp(-(velocity/v_th)^alpha)
return, f_v

end
```

```
pro surface_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;   Distribute pacets about a sphere with radius r=SpatialDist.exobase
;;
;; Version History
;;   3.2: 12/16/2010
;;     * rewrote the way random points on the surface are chosen
;;   3.1: 11/23/2010
;;     * minor revision in (SpatialDist.use_map EQ 1) section
;;   3.0: 7/19/2010
;;     * rewriting with new strucutre architecture
;;   2.1: Added better support for surface distributions
;;   2.0: File created.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; options: use_map, latitude, longitude
SpatialDist = input.SpatialDist

if (SpatialDist.use_map) then begin
  if ~(file_test(SpatialDist.mapfile)) then stop
  restore, SpatialDist.mapfile
  RandomDeviates_2d, *sourcemap.map, *sourcemap.longitude, sin(*sourcemap.latitude), $
    npack, lon, lat
  lat = asin(lat)
  destroy_structure, sourcemap
endif else begin
  ;; Choose the latitude -- f(lat) = cos(lat)
  if ((SpatialDist.latitude)[0] EQ (SpatialDist.latitude)[1]) $
    then lat = replicate((SpatialDist.latitude)[0], npack) $
    else begin
      sinlat = random_nr(seed=seed, npack)*2.-1
      lat = asin(sinlat)
    endelse

  ;; Choose the longitude -- f(lon) = 1 / (lonmax-lonmin)
  if ((SpatialDist.longitude)[0] GT (SpatialDist.longitude)[1]) $
    then m = [(SpatialDist.longitude)[0], (SpatialDist.longitude)[1]+2*!pi] $
    else m = SpatialDist.longitude
  lon = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)
endelse

if strcmp(input.geometry.planet, input.geometry.StartPoint, /fold) then begin
  ;; Starting at a planet.
  ;; 0 deg longitude = subsolar pt = (0, -1, 0)
  ;; 90 deg longitude = dusk pt = (1, 0, 0)
  ;; 270 deg longitude = dawn pt = (-1, 0, 0)
  *output.x0 = double(SpatialDist.exobase * sin(lon)*cos(lat))
  *output.y0 = -double(SpatialDist.exobase * cos(lon)*cos(lat))
  *output.z0 = double(SpatialDist.exobase * sin(lat))
endif else begin
  ;; Starting at a satellite
  ;; Treats the satellite as if it were at phi = 0.
  ;; 0 deg longitude = subsolar pt = (0, -1, 0)
  ;; 90 deg longitude = leading pt = (-1, 0, 0)
  ;; 270 deg longitude = trailing pt = (1, 0, 0)
  ;; lon=0 -> sub-planet point; lon=90 -> leading point
  *output.x0 = -double(SpatialDist.exobase * sin(lon)*cos(lat))
  *output.y0 = -double(SpatialDist.exobase * cos(lon)*cos(lat))
  *output.z0 = double(SpatialDist.exobase * sin(lat))
endelse

q = where(finite(*output.x0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.y0) EQ 0, nq) & if (nq NE 0) then stop
q = where(finite(*output.z0) EQ 0, nq) & if (nq NE 0) then stop

end

```

```
function surface_temperature, geometry, a, b, c, d, grid=grid, $
  longitude=longitude, latitude=latitude

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Compute the surface temperature for an object as a function of
;;  latitude and longitude on the surface
;;
;;  Input either SZA, longitude/latitude, or neither
;;  If neither, then produces a map in latutude/longitude
;;
;;  Version 3.0: 12/15/2010
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Figure out what inputs were given
nin = n_params()
case (nin) of
  1: begin
     lon0 = dindgen(361)*!dtor
     lat0 = dindgen(181)*!dtor-!pi/2.
     latlon = 1
     grid = 1
     end
  2: begin
     sza = a
     latlon = 0
     grid = 0
     end
  3: begin
     lon0 = a
     lat0 = b
     latlon = 1
     if (grid EQ !null) and (n_elements(a) NE n_elements(b)) then grid = 1
     if (grid EQ !null) then grid = 0
     end
  else: begin
    print, 'surface_temperature, geometry, grid=grid'
    print, 'surface_temperature, geometry, longitude, latitude, grid=1/0'
    print, 'surface_temperature, geometry, SZA'
    return, -1
    end
endcase

if (grid) then begin
  longitude = (lon0 # one(lat0))[*]
  latitude = (one(lon0) # lat0)[*]
endif

if (latlon) $
  then cosSZA = cos(longitude)*cos(latitude) $
  else cosSZA = cos(SZA)

case strlowcase(geometry.startpoint) of
  'mercury': begin
    temp0 = 100.
    temp1 = 600 + 125*(cos(geometry.taa)-1)/2. ;; sub-solar temp fn of taa
    n = .25
    day = where(cossza GT 0, nq)
    temperature = replicate(temp0, n_elements(cossza))
    if (nq GT 0) then temperature[day] += temp1*cosSZA[day]^n
    end
  else: stop
endcase

if (grid) then temperature = reform(temperature, n_elements(lon0), n_elements(lat0))

return, temperature

end
```

```
pro torus_distribution, geometry, spatialdist, options, seed, startloc=startloc

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;   Distribute packets in a torus centered on the central planet that is longitudinally
;;   symmetric.
;;
;;   Torus equation:
;;     x = (r0 + r1*cos(theta))*cos(phi)
;;     y = (r0 + r1*cos(theta))*sin(phi)
;;     z = r2*sin(theta)
;;
;;   If r2 = 0, then packets are confined to the equatorial plane.
;;
;; Version History
;;   2.0: File created.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

npack = options.packets
phi = random_nr(seed=seed, npack)*2*!pi
theta = random_nr(seed=seed, npack)*2*!pi
r0 = (SpatialDist.torus_radii)[0]
r1 = random_nr(seed=seed, npack)*(SpatialDist.torus_radii)[1]
r2 = random_nr(seed=seed, npack)*(SpatialDist.torus_radii)[2]

*startloc.x = (r0 + r1*cos(theta))*cos(phi)
*startloc.y = (r0 + r1*cos(theta))*sin(phi)
*startloc.z = r2*sin(theta)

*startloc.latitude = theta
*startloc.longitude = phi

end
```

```
pro out_cat, out0, out1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Combine two output structures
;; out0 is changed.
;;
;; Written 14 March 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

*out0.x0 = [*out0.x0, *out1.x0]
*out0.y0 = [*out0.y0, *out1.y0]
*out0.z0 = [*out0.z0, *out1.z0]
*out0.f0 = [*out0.f0, *out1.f0]
*out0.vx0 = [*out0.vx0, *out1.vx0]
*out0.vy0 = [*out0.vy0, *out1.vy0]
*out0.vz0 = [*out0.vz0, *out1.vz0]
*out0.phi0 = [*out0.phi0, *out1.phi0]
out0.totalsource += out1.totalsource
*out0.time = [*out0.time, *out1.time]
*out0.x = [*out0.x, *out1.x]
*out0.y = [*out0.y, *out1.y]
*out0.z = [*out0.z, *out1.z]
*out0.frac = [*out0.frac, *out1.frac]
*out0.vx = [*out0.vx, *out1.vx]
*out0.vy = [*out0.vy, *out1.vy]
*out0.vz = [*out0.vz, *out1.vz]
*out0.lossfrac = float([*out0.lossfrac, *out1.lossfrac])
*out0.ringfrac = float([*out0.ringfrac, *out1.ringfrac])
*out0.leftfrac = float([*out0.leftfrac, *out1.leftfrac])
*out0.sourcefile = [*out0.sourcefile, *out1.sourcefile]
s = size(*out0.hitfrac)
if (s[0] EQ 1) $
  then *out0.hitfrac = float([*out0.hitfrac, *out1.hitfrac]) $
  else begin
    temp = fltarr(n_elements(*out0.x),s[2])
    temp[0:s[1]-1,*] = float(*out0.hitfrac)
    temp[s[1]:*,*] = float(*out1.hitfrac)
    *out0.hitfrac = temp
  endelse
*out0.deposition.map += *out1.deposition.map

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro out_sub, output, q

*output.x0 = (*output.x0)[q]
```

```
*output.y0 = (*output.y0)[q]
*output.z0 = (*output.z0)[q]
*output.f0 = (*output.f0)[q]
*output.vx0 = (*output.vx0)[q]
*output.vy0 = (*output.vy0)[q]
*output.vz0 = (*output.vz0)[q]
*output.phi0 = (*output.phi0)[q]
*output.time = (*output.time)[q]
*output.x = (*output.x)[q]
*output.y = (*output.y)[q]
*output.z = (*output.z)[q]
*output.frac = (*output.frac)[q]
*output.vx = (*output.vx)[q]
*output.vy = (*output.vy)[q]
*output.vz = (*output.vz)[q]
*output.lossfrac = (*output.lossfrac)[q]
*output.ringfrac = (*output.ringfrac)[q]
s = size(*output.hitfrac)
if (s[0] EQ 1) $
  then *output.hitfrac = (*output.hitfrac)[q] $
  else *output.hitfrac = (*output.hitfrac)[*,q]

end
```

```
pro locmoon, time, theta0, radius, orbrate, x=x, y=y, z=z, ang=ang

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; - calculates the coordinates of each moon given a "final" orbital longitude and
;;     a time difference
;; - Where was moon (time) seconds ago?
;;      theta = (-time [s]) * (orbrate [rad/s]) + theta0 [rad]
;;
;; INPUTS
;; * time: Time before moon was at "theta0" (seconds)
;; * theta0: final orbital longitude of each moon (radians)
;; * radius: orbital radius of each moon (R_plan)
;; * orbrate: angular speed of each moon (rad/s)
;;
;; OUTPUTS
;; * x: moon's x-position relative to planet "time" seconds before
;;          it was at "theta0" (R_J)
;; * y:     "   y            "                "
;; * ang: theta  _time_ seconds ago
;;
;;                                +y
;;                                 |        +z is out of the screen (Jup north)
;;                                 |
;;                                O--------- West,+x
;;                                 |
;;                                 |
;;                                \ /
;;                                 v
;;                       To Earth/Sun
;;
;;   n = n_elements(time) = number of packets
;;   m = n_elements(radius) = number of moons
;;   x = float(n,m) = x position of each moon at each time step requested
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Calculate orbital longitude (radians)
i = replicate(1.,n_elements(time))
ang = double(-time)#orbrate + i#double(theta0)  ;;[n#m + n#m]

;Calculate x and y coordinates
x = -(i#radius) * sin(ang)
y = (i#radius) * cos(ang)
z = x * 0. ;; Assume inclination = 0

end
```

```
pro model_common_blocks

;; Defines the common blocks used in the model
common constants, SystemConsts, DipoleConsts, stuff
common ratecoefs, kappa
common plasma, plasma, plasmahot
;;common lifetimes, lossrate

end
```

```
function model_findpackets, input, strstart, overwrite=overwrite, generic=generic

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  * Given an inputfile, determines how many packets are available.
;;  * Will extract from generic files if requetsed.
;;  * Also returns the available output files and number of packets in each file
;;
;;  Revision history
;;    4.0 -- 12/8/2011
;;      * Moving this part of modeldriver_3.7 to a separate program.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Load in the common blocks
common constants

;; Determine run options
if (overwrite EQ !null) then overwrite = 0
if (generic EQ !null) then generic = 1

dogen = (generic EQ 1) or (generic EQ 2)

totalpackets = 0L    ;; reset the number of packets

if (dogen) then geninput = make_generic_input(input)

files = modeloutput_search(input, nfiles=nfiles)
print, stuff.strstart + strint(nfiles) + ' output files found.'

if (dogen) then begin
  genericfiles = modeloutput_search(geninput, nfiles=ngfiles)
  print, stuff.strstart + strint(ngfiles) + ' generic output files found.'
endif else ngfiles = 0

;; Delete old files if requested
if (overwrite) then begin
  print, stuff.strstart + 'Eraseing old, unwanted files.'

  for i=0,nfiles-1 do spawn, 'rm ' + files[i]
  if (generic EQ 2) then for i=0,ngfiles-1 do spawn, 'rm ' + genericfiles[i]
endif

if ((generic EQ 0) or (generic EQ 1)) then begin
  ;; Want specific output files

  ;; Extract from generic if any new generic files are available
  if (ngfiles GT 0) then begin
    extract = replicate(1, ngfiles)
    if (nfiles GT 0) then begin
      ;; check to make sure none of the generic files have been used already
```

```
   param0 = extract_parameter('sourcefile', files)
   usedfiles = (param0.values()).toArray(type='string')
   param0 = 0

   ;; go through each generic file to make sure it has been used no more than once
   for i=0,ngfiles-1 do begin
    w = (where(strcmp(usedfiles, genericfiles[i]), nw))[0]
    case (nw) of
     0: extract[i] = 1
     1: extract[i] = 0
     else: stop  ;; already two files extracted -- error
    endcase
   endfor
  endif

  q = where(extract, nq)
  print, stuff.strstart + strint(nq) + ' new generic files available.'
  if (nq GT 0) then begin
   newfiles = extract_distribution(genericfiles[q], input)

   ;; Reset the filelist
   files = modeloutput_search(input, nfiles=nfiles)
   if (nfiles EQ 0) then stop ;; error
  endif
 endif

 ;; Determine number of packets available
 if (nfiles GT 0) then begin
  pack = extract_parameter('savedpackets', files)
  totalpackets = long(total((pack.values()).ToArray(type='long')))
  pack = 0  ; get around an IDL bug
 endif else totalpackets = 0L
endif else begin ;; generic = 2
 if (ngfiles GT 0) then begin
  pack = extract_parameter('savedpackets', genericfiles)
  totalpackets = long(total((pack.values()).ToArray(type='long')))
  pack = 0  ; get around an IDL bug
 endif else totalpackets = 0L
endelse

return, totalpackets

end
```

```
pro modeldriver, inputfiles, npackets, seed, $
  quick=quick, overwrite=overwrite, generic=generic, local=local, $
  logfile=logfile, outputfile=outputfile, runmodel=runmodel

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Options
;;     * quick = 1/0 -- if running in quickmode, just does one run through of
;;         npackets to test a set of options. Otherwise, does the full check of
;;         what is already run. Quickmode implies overwrite=1 and generic=0
;;     * generic = 0/1/2
;;          = 0 => does not search through the existing generic files (runs
;;                 exact inputs specified)
;;          = 1 => Extracts specific inputs from generic files and runs more
;;                 generic files if needed.
;;          = 2 => Does not extract specific inputs and checks to see if there
;;                 are enough generic packets available. This is needed to prevent
;;                 extracting inputs when you want to run generic models only
;;
;;  Revision history
;;    3.8 -- 10/8/2011
;;      * moving the part where it searches for packets and extracts from generic files
;;        to a model_findpackets.pro
;;    3.7 -- 9/28/2011
;;      * Adding option to save models locally
;;    3.6 -- 9/16/2011
;;      * Changing the path method - using !model.basepath
;;    3.5 -- 5/10/2011
;;      * Changing the way the g-value is stored
;;    3.4 -- 12/17/2010
;;      * adding more options to make quick runs and streamlines easier
;;    3.3 -- 9/8/10
;;      * If generic=0, does not search through the existing generic files
;;    3.2 -- 8/31/10
;;      * added packets keyword to modeloutput_search
;;    3.0 -- 7/12/10
;;      * user no longer controls name of output file
;;      * want to look to see if the appropriate model already exists
;;      * need to move the streamline routines to separate programs
;;      * removing minimization option
;;    2.5 -- 4/19/10
;;      * added version to output
;;    2.4 -- replace destroy_XXX with destroy_structure
;;    2.3 -- added an option to minimize the structures to save memory
;;        -- Removed to_run option since only one integrator still works (5/27/09)
;;    2.2 -- added ability to use g-values from Killen et al. This will make it possible
;;      to compute radiation pressure for a number of species.
;;    2.1 -- revise modstream 1 to do everything all at once (if there aren't too
;;      many packets)
;;        -- need to add modstream 2
;;    2.0 -- 10/22/08
;;       Begin transition to version control
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Load in the common blocks
common constants
common ratecoefs
common plasma

;; Turn off the logging and open a new logfile
if (!journal) then journal
w = file_test('logs/') & if ~w then spawn, 'mkdir logs'
if (n_elements(logfile) NE 1) then begin
  spawn, 'date +"%Y-%m-%dT%H:%M"', date
  logfile = 'logs/modelrun.' + date + '.pro'
endif

print, 'Opening log file: ' + logfile
journal, logfile

tstart = systime(1)
tittot = 0.

;; Set up the stuff structure
if (local EQ !null) then local = 0
stuff = {s:0, aplanet:0., vrplanet:0., radpres_v:ptr_new(0), $
  radpres_const:ptr_new(0), local:local, strstart:''}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine run options
if (overwrite EQ !null) then overwrite = 0
if (generic EQ !null) then generic = 1
if (quick EQ !null) then quick = 0
if (runmodel EQ !null) then runmodel = 1

dogen = (generic EQ 1) or (generic EQ 2)
if (quick) then dogen = 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine program version
readfmt, 'version.dat', /silent, 'A100', version
version = strtrim(version, 2)
if (n_elements(version) EQ 1) then stop
ntot = 0L

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Loop over each inputfile
ninputs = n_elements(inputfiles)
for iii=0,ninputs-1 do begin
  trun0 = systime(1)
  stuff.strstart = 'Inputfile #' + strint(iii) + ': '
```

```

inputfile = inputfiles[iii]
print, '***********************'
print, stuff.strstart + 'Starting ' + inputfile
print, stuff.strstart + systime(0)

totalpackets = 0L    ;; reset the number of packets
runthrough = 0      ;; reset the run number
owrite = overwrite  ;; reset the overwrite state
;; Run model until have specified number of packets

while (totalpackets LT npackets) do begin
  print, stuff.strstart + 'Starting Simulation #' + strint(runthrough)

  ;; (a) Read in the inputs -- Reset to ensure no changes have been made
  input = inputs_restore(inputfile)
  SystemConstants, input.geometry.planet, SystemConsts, DipoleConsts
  geninput = inputs_restore(inputfile)

  ;; (b) Determine how many packets have already been run
  if ~(quick) then begin
    owrite = (runthrough EQ 0) ? overwrite : 0
    totalpackets = model_findpackets(input, generic=generic, overwrite=owrite)
    ntodo = npackets - totalpackets
  endif else ntodo = npackets ;; quick

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; If runmodel = 0, then just doing the packet extraction part
  if (runmodel) and (ntodo GT 0) then begin
    if (dogen) then input = temporary(geninput)

    ;; Read in the constants
    stuff.s = (where(strlowcase(*SystemConsts.Objects) EQ $
      strlowcase(input.geometry.StartPoint)))[0]

    ;; Determine distance and radial velocity of planet relative to the sun
    planet_dist, input.geometry.taa, SystemConsts, distance=dd, velocity=vv
    stuff.aplanet = dd
    stuff.vrplanet = vv/SystemConsts.rplan

    ;; find the default reactions and datasets
    if (input.options.lifetime EQ 0) $
      then loss_info = lifetime_setup(input) $
      else loss_info = !null

    ;; Set up the radiation pressure
    if (input.forces.radpres) then begin
      q = get_gvalue(input.options.atom, stuff.aplanet)
      ;;q /= SystemConsts.rplan ;; v in rplan/s, a in rplan/s^2
      *stuff.radpres_v = *q.v
      *stuff.radpres_const = *q.radaccel/SystemConsts.rplan
```

```
endif else begin
  *stuff.radpres_v = 0.
  *stuff.radpres_const = 0.
endelse

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine how to run the additional packets that are needed
;; run 10 iterations of 100,000 packets to produce a run of 1,000,000 packets
;; Each outputfile produced will have 1,000,000 packets
maxpack = 1000000L
nruns = ceil(ntodo/maxpack)
nits = 10
packs_per_it = 100000L
if (dogen) $
  then print, stuff.strstart + 'Running generic model.' $
  else print, stuff.strstart + 'Running Model'
print, stuff.strstart + 'Will complete ' + strint(nruns) + $
  ' runs of 1,000,000 packets.'

for i=0,nruns-1 do begin
  outputfile = output_filename(input)
  print, stuff.strstart + 'Outputfile = ' + outputfile
  for j=0,nits-1 do begin
    tit0 = systime(1)
    print, '** Starting run #' + strint(i+1) + ' of ' + strint(nruns)
    print, '** Starting iteration #' + strint(j+1) + ' of 10'

    ;; Determine the initial source distribution
    source_distribution, input, packs_per_it, seed, output=output
    if (input.options.lifetime EQ 0) $
      then output.loss_info = {reactions:ptr_new(loss_info.reaction), $
        files:ptr_new(loss_info.file), type:ptr_new(loss_info.type)}

    ;; Run the model
    print, '**** RUNNING *****'
    driver, input, output, seed=seed
    save, output, input, version, file=outputfile+'.' + strint(j)
    destroy_structure, output

    ;; Concluding stuff
    tit1 = systime(1)
    ntot++
    tittot += (tit1-tit0)
    print, stuff.strstart + 'Iteration time = ' + strint(tit1-tit0) + ' seconds'
    print, stuff.strstart + 'Mean Iteration time = ' + strint(tittot/ntot) + $
      ' seconds'
  endfor
  combine_iterations, outputfile, /delete
  print, 'Finished ' + outputfile
endfor  ;; runs

```

```
      ;; Cleanup the memory
      destroy_structure, input

      destroy_structure, plasma
      destroy_structure, plasmahot
      destroy_structure, plasma

      destroy_structure, coef_eimp
      destroy_structure, coef_chx
      destroy_structure, coef_photo

      print, stuff.strstart + 'Finishing Simulation #' + strint(runthrough)
      runthrough++
    endif else totalpackets = npackets ;; not running model, so end while loop

    ;; If not running quick version, always want to do another runthrough
    ;; to extract packets from newly available generic files
    ;; If running quick version then stop running
    if (quick) then totalpackets = npackets
    destroy_structure, SystemConsts
  endwhile

  trun1 = systime(1)
  print, stuff.strstart + 'Finishing ' + inputfile
  print, stuff.strstart + 'Time for inputfile: ' + strint((trun1-trun0)/3600.) + ' hours'
  print, stuff.strstart + 'Total elaspsed time: ' + strint((trun1-tstart)/3600.) + ' hours'
  print, '  ' + inputfile
  print
endfor ; iii

;; close the logfile
journal
print, 'Closed log file ' + logfile

end
```

```
pro modelstreamlines, inputfiles, dt, npackets, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Driver to determine particle streamlines.
;;
;; Method = 0 -> Satellite positions at end of model time are given
;; Method = 1 -> Satellite positions at begining of model time are given
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Load in the common blocks
common constants
common ratecoefs
common plasma

tstart = systime(1)
tittot = 0.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine program version
readfmt, 'version.dat', /silent, 'A100', version
version = strtrim(version, 2)
if (n_elements(version) EQ 1) then stop
ntot = 0L

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Loop over each inputfile
ninputs = n_elements(inputfiles)
for iii=0,ninputs-1 do begin
  trun0 = systime(1)
  strstart = 'Inputfile #' + strint(iii) + ': '

  inputfile = inputfiles[iii]
  print, '***********************'
  print, strstart + 'Starting ' + inputfile
  print, strstart + systime(0)

  input = inputs_restore(inputfile)
  outputfile = output_filename(input) + '.streamline'

  if (input.sticking_info.stickcoef NE 1) then stop ;; this won't work with bouncing

  ;; Set up the stuff structure
  stuff = {s:0, aplanet:0., vrplanet:0., radpres_v:ptr_new(0), $
    radpres_const:ptr_new(0), datapath:''}

  ;; Read in the constants
  SystemConstants, input.geometry.Planet, SystemConsts, DipoleConsts
  stuff.s = (where(strlowcase(*SystemConsts.Objects) EQ $
    strlowcase(input.geometry.StartPoint)))[0]
```

```

;; Determine distance and radial velocity of planet relative to the sun
planet_dist, input.geometry.taa, SystemConsts, distance=dd, velocity=vv
stuff.aplanet = dd
stuff.vrplanet = vv/SystemConsts.rplan

;; Set up the paths to necessary data
testdir = ['/Users/mburger/Data/AtomicData/', $
  '/Users/burger/Data/AtomicData/', $
  '$HOME/NeutralModel/AtomicData/']
w = (where(file_test(testdir)))[0]
if (w EQ -1) $
  then stop $
  else stuff.datapath = testdir[w]

;; find the default reactions and datasets
if (input.options.lifetime EQ 0) $
  then loss_info = lifetime_setup(input) $
  else loss_info = !null

;; Set up the radiation pressure
if (input.forces.radpres) then begin
  q = get_gvalue(stuff.aplanet, input.options.atom, path=stuff.datapath+'g-values/')
  q /= SystemConsts.rplan ;; v in rplan/s, a in rplan/s^2
  *stuff.radpres_v = q[*,0]
  *stuff.radpres_const = q[*,1]
endif else begin
  *stuff.radpres_v = 0.
  *stuff.radpres_const = 0.
endelse

;; For streamlines, set at_once = 1
input.options.at_once = 1

;; there are two ways to do the streamlines:
;;   a) Time given is the location at the end of the model period
;;   b) Time given is the location at the begining of the model period

;; Only doing method A
;; Determine the initial source distirbution
input.options.trackloss = 0 ;; for now
endtime = input.options.endtime
input.options.endtime = 0.
source_distribution, input, npackets, seed, output=output
if (input.options.lifetime EQ 0) then output.loss_info = $
  {reactions:ptr_new(loss_info.reaction), files:ptr_new(loss_info.file), $
  type:ptr_new(loss_info.type)}

;; Number of time steps
nt = long(endtime/dt)+1
runtime = dindgen(nt)*dt
```

```

  ;; Need starting position at each time
  t0 = dblarr(npackets,nt)
  x0 = dblarr(npackets,nt)
  y0 = dblarr(npackets,nt)
  z0 = dblarr(npackets,nt)
  f0 = dblarr(npackets,nt)
  vx0 = dblarr(npackets,nt)
  vy0 = dblarr(npackets,nt)
  vz0 = dblarr(npackets,nt)
  phi0 = dblarr(npackets,nt)

  ;; If starting at the planet or there is no motion, then starting point does not change
  if ((stuff.s EQ 0) or (input.options.motion EQ 0)) then begin
    for i=0,nt-1 do begin
      t0[*,i] = runtime[i]
      x0[*,i] = *output.x
      y0[*,i] = *output.y
      z0[*,i] = *output.z
      f0[*,i] = *output.frac
      vx0[*,i] = *output.vx
      vy0[*,i] = *output.vy
      vz0[*,i] = *output.vz
    endfor
  endif else begin
    ;; Determine where the object is at each time
    locmoon, runtime, (*input.geometry.phi)[stuff.s], (*SystemConsts.a)[stuff.s], $
      (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty, ang=ang
    rotang = ang-(*input.geometry.phi)[stuff.s]
    rr = transpose([[*output.x], [*output.y], [*output.z]])
    vv = transpose([[*output.vx], [*output.vy], [*output.vz]])
    ;; Rotate starting packets to proper starting point
    for i=0,nt-1 do begin
      rr2 = rotation(rr, [0,0,1.], rotang[i])
      vv2 = rotation(vv, [0,0,1.], rotang[i])
      t0[*,i] = runtime[i]
      x0[*,i] = reform(rr2[0,*])
      y0[*,i] = reform(rr2[1,*])
      z0[*,i] = reform(rr2[2,*])
      f0[*,i] = *output.frac
      vx0[*,i] = reform(vv2[0,*])
      vy0[*,i] = reform(vv2[1,*])
      vz0[*,i] = reform(vv2[2,*])
      phi0[*,i] = ang[i]
;;        plot, x0[*,i], y0[*,i], psym=8, /iso, xrange=satx[i]+[-.1,.1], $
;;        yrange=saty[i]+[-.1,.1], /xst, /yst
;;        plots, satx[i], saty[i], psym=8, color=2
;;        wait, 0.1
    endfor
  endelse

```

```
    ;; Set up the array to run
    nn = npackets * nt
    nmax = long(1e6) ;; never run more that 10^6 packets at once

    if (nn LT nmax) then begin
      *output.time = t0[*]
      *output.x = x0[*]
      *output.y = y0[*]
      *output.z = z0[*]
      *output.frac = f0[*]
      *output.vx = vx0[*]
      *output.vy = vy0[*]
      *output.vz = vz0[*]

      driver, input, output, seed=seed
      *output.x = reform(*output.x, npackets, nt)
      *output.y = reform(*output.y, npackets, nt)
      *output.z = reform(*output.z, npackets, nt)
      *output.frac = reform(*output.frac, npackets, nt)
      *output.vx = reform(*output.vx, npackets, nt)
      *output.vy = reform(*output.vy, npackets, nt)
      *output.vz = reform(*output.vz, npackets, nt)
    endif else begin
      stop
    endelse

    save, output, input, file='temp.sav'
  endfor

  end
```

```
pro modelstreamlines, inputfiles, dt, npackets, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Driver to determine particle streamlines.
;;
;; Method = 0 -> Satellite positions at end of model time are given
;; Method = 1 -> Satellite positions at begining of model time are given
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Load in the common blocks
common constants
common ratecoefs
common plasma

tstart = systime(1)
tittot = 0.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine program version
readfmt, 'version.dat', /silent, 'A100', version
version = strtrim(version, 2)
if (n_elements(version) EQ 1) then stop
ntot = 0L

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Loop over each inputfile
ninputs = n_elements(inputfiles)
for iii=0,ninputs-1 do begin
  trun0 = systime(1)
  strstart = 'Inputfile #' + strint(iii) + ': '

  inputfile = inputfiles[iii]
  print, '**********************'
  print, strstart + 'Starting ' + inputfile
  print, strstart + systime(0)

  input = inputs_restore(inputfile)
  outputfile = output_filename(input) + '.streamline'

  ;; Set up the stuff structure
  stuff = {s:0, aplanet:0., vrplanet:0., radpres_v:ptr_new(0), $
    radpres_const:ptr_new(0), datapath:''}

  ;; Read in the constants
  SystemConstants, input.geometry.Planet, SystemConsts, DipoleConsts
  stuff.s = (where(strlowcase(*SystemConsts.Objects) EQ $
    strlowcase(input.geometry.StartPoint)))[0]

  ;; Determine distance and radial velocity of planet relative to the sun
```

```
planet_dist, input.geometry.taa, SystemConsts, distance=dd, velocity=vv
stuff.aplanet = dd
stuff.vrplanet = vv/SystemConsts.rplan

;; Set up the paths to necessary data
testdir = ['/Users/mburger/Data/AtomicData/', $
  '/Users/burger/Data/AtomicData/', $
  '$HOME/NeutralModel/AtomicData/']
w = (where(file_test(testdir)))[0]
if (w EQ -1) $
  then stop $
  else stuff.datapath = testdir[w]

;; find the default reactions and datasets
if (input.options.lifetime EQ 0) $
  then loss_info = lifetime_setup(input) $
  else loss_info = !null

;; Set up the radiation pressure
if (input.forces.radpres) then begin
  q = get_gvalue(stuff.aplanet, input.options.atom, path=stuff.datapath+'g-values/')
  q /= SystemConsts.rplan ;; v in rplan/s, a in rplan/s^2
  *stuff.radpres_v = q[*,0]
  *stuff.radpres_const = q[*,1]
endif else begin
  *stuff.radpres_v = 0.
  *stuff.radpres_const = 0.
endelse

;; For streamlines, set at_once = 1
input.options.at_once = 1

;; there are two ways to do the streamlines:
;;   a) Time given is the location at the end of the model period
;;   b) Time given is the location at the begining of the model period

;; Only doing method A
;; Determine the initial source distirbution
input.options.trackloss = 0 ;; for now
source_distribution, input, npackets, seed, output=output
if (input.options.lifetime EQ 0) then output.loss_info = $
  {reactions:ptr_new(loss_info.reaction), files:ptr_new(loss_info.file), $
  type:ptr_new(loss_info.type)}

;; Number of time steps
nt = fix(input.options.endtime/dt)+1

xx = dblarr(npackets,nt) & xx[*,0] = *output.x
yy = dblarr(npackets,nt) & yy[*,0] = *output.y
zz = dblarr(npackets,nt) & zz[*,0] = *output.z
ff = dblarr(npackets,nt) & ff[*,0] = *output.frac
```

```
vx = dblarr(npackets,nt) & vx[*,0] = *output.vx
vy = dblarr(npackets,nt) & vy[*,0] = *output.vy
vz = dblarr(npackets,nt) & vz[*,0] = *output.vz

loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0)}
*loc.x = [[*output.x], [*output.y], [*output.z]]
*loc.v = [[*output.vx], [*output.vy], [*output.vz]]
*loc.frac = *output.frac
*loc.t = *output.time ;; This is how much time before present and works up to zero

h = replicate(dt, npackets)
which = where(*input.geometry.include, nw)
pp = replicate(1.,n_elements(*SystemConsts.objects))
for i=1,nt-1 do begin
  todo = where(*loc.frac GT 0, ntodo)
  if (ntodo GT 0) then begin
    ;; extract necessary packets
    loc1 = {t:ptr_new((*loc.t)[todo]), x:ptr_new((*loc.x)[todo,*]), $
      v:ptr_new((*loc.v)[todo,*]), frac:ptr_new((*loc.frac)[todo])}

    ;; advance the step
    rk_5, loc1, h, input, which, delta

    ;; Check to see if we hit anything
    jj = replicate(1., ntodo)

    ;; 1) Did the packets hit anything?
    ;Get object positions
    if (input.options.motion) $
      then locmoon, *loc1.t, *input.geometry.phi, *SystemConsts.a, $
        *SystemConsts.orbrate, x=xSat, y=ySat, z=zSat, ang=ang $
      else locmoon, fltarr(ntodo), *input.geometry.phi, *SystemConsts.a, $
        *SystemConsts.orbrate, x=xSat, y=ySat, z=zsat, ang=ang

    ;; Distance of packets from each object
    tempR = sqrt(((*loc1.x)[*,0]#pp - xSat)^2 + ((*loc1.x)[*,1]#pp - ySat)^2 + $
      ((*loc1.x)[*,2]#pp - zSat)^2)

    ;Is r < satellite radius?
    eps = 0.
    satrad = jj # (*SystemConsts.radius)[which]*(1-eps)
    hhh = where((tempR-satrad) LT 0, nhits)
    ;;print, nhits
    if (nhits NE 0) then begin
      ;;w = where((*loc1.t)[hhh mod nq] EQ 0, nw) & if (nw NE 0) then stop
      hx = hhh mod ntodo & hy = hhh/ntodo

      ;; adject the frac values
      if (input.sticking_info.stickcoef EQ 1) $
        then (*loc1.frac)[hx] = 0 $
        else (*loc1.frac)[hx] = (*loc1.frac)[hx] * (1.-input.sticking_info.stickcoef)

        ;; Figure out exactly where things hit the surface
        srad = satrad[hhh]
        r0 = tempR[hhh]  ;; R_plan
        x0 = (*loc1.x)[hx,0]  ;; R_plan
        y0 = (*loc1.x)[hx,1]  ;; R_plan
        z0 = (*loc1.x)[hx,2]  ;; R_plan
        r0 = sqrt(x0^2 + y0^2 + z0^2)

        ;; Position of the satellites
        xcent = xSat[hx,hy]
        ycent = ySat[hx,hy]
        zcent = zSat[hx,hy]

        ;; Vector from center of satellite to packet
        ;;   -- packet positions relative to satellite
        x1 = (x0-xcent)/srad  ;; rsat
        y1 = (y0-ycent)/srad  ;; rsat
        z1 = (z0-zcent)/srad  ;; rsat

        ;; Velocity - orbital vel = vel relative to satellite
        vxsat = -(*SystemConsts.orbvel)[hy]*cos(ang[hx,hy])*input.options.motion/$
          SystemConsts.rplan
        vysat = -(*SystemConsts.orbvel)[hy]*sin(ang[hx,hy])*input.options.motion/$
          SystemConsts.rplan

        vx0 = (*loc1.v)[hx,0] - vxsat  ;; rplan/s
        vy0 = (*loc1.v)[hx,1] - vysat  ;; rplan/s
        vz0 = (*loc1.v)[hx,2]  ;; rplan/s

        ;; Find where the packet hit the surface
        ;;    |x + vt| = 1 -- see ResearchNotes from 4/28/08
        a = vx0^2 + vy0^2 + vz0^2
        b = 2*(x1*vx0 + y1*vy0 + z1*vz0)
        c = x1^2 + y1^2 + z1^2 - 1

        dd = b^2 - 4*a*c
        q = where(dd LT 0, nq) & if (nq NE 0) then stop
        t0 = (-b - sqrt(b^2-4*a*c))/(2*a)
        t1 = (-b + sqrt(b^2-4*a*c))/(2*a)
        t = (t0 LE 0)*t0 + (t1 LT 0)*t1

        ;; Point where packet hit the surface
        x2 = x1 + vx0*t
        y2 = y1 + vy0*t
        z2 = z1 + vz0*t
        ;; r2 = sqrt(x2^2 + y2^2 + z2^2)  ;; -- this should be = 1.

        lonhit = (atan(x2, -y2) + 2*!pi) mod (2*!pi)
        lathit = asin(z2)

```

```
    ;; Put new coordinates into the array
    x_final = xcent + x2*srad
    y_final = ycent + y2*srad
    z_final = zcent + z2*srad

    q = where(finite(x_final) EQ 0, nq) & if (nq NE 0) then stop
    q = where(finite(y_final) EQ 0, nq) & if (nq NE 0) then stop
    q = where(finite(z_final) EQ 0, nq) & if (nq NE 0) then stop
    (*loc1.x)[hx,0] = x_final
    (*loc1.x)[hx,1] = y_final
    (*loc1.x)[hx,2] = z_final

    if (input.sticking_info.stickcoef LT 1) then begin
      ;; Figure out rebound velocity
      vv02 = vx0^2 + vy0^2 + vz0^2  ;; rplan/s
      PE = 2*(*SystemConsts.GM)[hy]*(1./r0-1./srad)
      vv02 += PE
      q = where(vv02 LT 0, nq) & if (nq NE 0) then vv02[q] = 0.
      q = where(finite(vv02) EQ 0, nq) & if (nq NE 0) then stop

      case strlowcase(input.sticking_info.emitfn) of
        'maxwellian': begin ;; Re-emit the packets with a thermal distribution
          if (input.sticking_info.Tsurf EQ 0) then begin
            surftemp = temp0 + (temp1*(abs(cos(lonhit)*cos(lathit)))^nn)*$
              (abs(lonhit) LT !pi/2)
            vv_new = interpolate_xy(vgrid, temperature, prob, surftemp, $
              random_nr(seed=seed, nhits))/SystemConsts.rplan
          endif else vv_new = interpol(vrange, sumdist, $
            random_nr(seed=seed, nhits))/SystemConsts.rplan ;; rplan/s

          vv2 = sqrt(input.Sticking_info.accom_factor*vv_new^2 + $
            (1-input.Sticking_info.accom_factor)*vv02)
          end
        'elastic scattering': vv2 = sqrt(vv02)
      endcase

      ;; Determine new direction with F(v) ~ cos(theta)
      alt = acos(random_nr(seed=seed, nhits))
      az = 2*!pi * random_nr(seed=seed, nhits)

      v_rad = sin(alt)          ;; Radial component of velocity
      v_east = -cos(alt) * sin(az) ;; Component along latitude line (points east)
      v_north = cos(alt) * cos(az) ;; Component along longitude line (points to NP)

      vx2 = v_rad*x2
      vy2 = v_rad*y2
      vz2 = v_rad*z2

      lat = asin(z2)
      lon = atan(x2, y2)

      vx2 = dblarr(nhits) & vy2 = dblarr(nhits) & vz2 = dblarr(nhits)
      for i=0L,nhits-1 do begin
        M = transpose([ $
          [cos(lat[i])*sin(lon[i]), cos(lat[i])*cos(lon[i]), sin(lat[i])], $
          [-cos(lon[i]), sin(lon[i]), 0], $
          [-sin(lat[i])*sin(lon[i]), -sin(lat[i])*cos(lon[i]), cos(lat[i])] ])
        v_ren = [v_rad[i], v_east[i], v_north[i]]
        v_xyz = invert(M) # v_ren
        vx2[i] = v_xyz[0] * vv2[i]
        vy2[i] = v_xyz[1] * vv2[i]
        vz2[i] = v_xyz[2] * vv2[i]
      endfor

      ;; The new position in planet-centered coords
      vx_final = vx2 + vxsat
      vy_final = vy2 + vysat
      vz_final = vz2

      q = where(finite(vx_final) EQ 0, nq) & if (nq NE 0) then stop
      q = where(finite(vy_final) EQ 0, nq) & if (nq NE 0) then stop
      q = where(finite(vz_final) EQ 0, nq) & if (nq NE 0) then stop

      (*loc1.v)[hx,0] = vx_final
      (*loc1.v)[hx,1] = vy_final
      (*loc1.v)[hx,2] = vz_final
     endif  ;; stickcoef LT 0
    endif    ;; nhits GT 0

    ;; 2) Have the packets left the corona?
    if ~(input.options.fullSystem) then begin
     leftCor = where(tempR[*,stuff.s] GT input.options.OuterEdge * $
      (*SystemConsts.radius)[stuff.s], hh)
     if (hh NE 0) then (*loc1.frac)[leftCor] = 0
    endif

    ;; 3) If Saturn, check to see if anything hit the rings
    if (input.geometry.planet EQ 'Saturn') then begin
     ;; Ring region within 2.3 Rs
     ox = oldx[g,*]
     cross = ox[*,2] * (*loc1.x)[*,2] ;;if cross is negative, then crossed eq. plane
     MayHit = where(cross LE 0 , nmay)
     if (nmay NE 0) then begin
      orho = sqrt(total(ox[MayHit,0:1]^2, 2))
      nrho = sqrt(total((*loc1.x)[MayHit,0:1]^2, 2))
      w = where((orho LT 2.3) or (nrho LT 2.3), nw)
      for j=0,nw-1 do begin $
        crosspt = interpol([orho[w[j]],nrho[w[j]]], [ox[MayHit[w[j]],2], $
          (*loc1.x)[MayHit[w[j]],2]], 0.)
        if (crosspt LT 2.3) then (*loc1.frac)[MayHit[w[j]]] = 0.
      endfor
     endif
```

```
        endif

        ;; Save these values and update the full loc structure
        xx[todo,i] = (*loc1.x)[*,0]
        yy[todo,i] = (*loc1.x)[*,1]
        zz[todo,i] = (*loc1.x)[*,2]
        ff[todo,i] = *loc1.frac
        vx[todo,i] = (*loc1.v)[*,0]
        vy[todo,i] = (*loc1.v)[*,1]
        vz[todo,i] = (*loc1.v)[*,2]

        (*loc.t)[todo] = *loc1.t
        (*loc.x)[todo,*] = *loc1.x
        (*loc.v)[todo,*] = *loc1.v
        (*loc.frac)[todo,*] = *loc1.frac
        loc1 = 0.
      endif  ;; nq > 0
    endfor   ;; loop over timesteps
    loc = 0

    ;; Save the output structure
    *output.x = xx
    *output.y = yy
    *output.z = zz
    *output.frac = ff
    *output.vx = vx
    *output.vy = vy
    *output.vz = vz
    save, output, input, version, file='temp.sav'
    output = 0 & input = 0 & SystemConsts = 0
  endfor  ;; inputfiles

  end

```

```
pro modstreamA, loc, startloc, geometry, spatialdist, speeddist, angulardist, $
  PerturbVel, options, dt, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Run streamlines - the result of this version is a set of packets in which
;; a packet with traveltime = t has been ejected from the object t seconds ago
;;
;; The satellites end up at their specified positions and
;; each packet has travelled for (runtime) seconds
;;
;; Version history:
;;   2.3: 1/14/2010
;;     - added ability to track packet loss and surface deposition
;;   2.1: created from a section in modeldriver_2.0 (2 Dec 2008)
;;     - Tries to do everything with a single call to the driver. If there are too
;;       many packets or time steps, then it will iterate
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (Sticking_info.stickcoef NE 1) then stop

maxsteps = 200000L ;; do everything at once if fewer points than maxsteps

endtime = options.endtime
runtime = dindgen(round(options.endtime/dt)+1)*dt
nt = n_elements(runtime)
options.at_once = 1
options.endtime = 0.

;; Find the initial conditions for t=tfinal
source_distribution, geometry, spatialdist, speeddist, angulardist, PerturbVel, $
 options, seed, loc=loc, startloc=startloc
*startloc.TravelTime = runtime
npack = options.packets
options.endtime = endtime

;; Save the starting values
x0 = *loc.x
y0 = *loc.y
z0 = *loc.z
vx0 = *loc.vx
vy0 = *loc.vy
vz0 = *loc.vz

;; Make arrays for the final values
x2 = dblarr(npack, nt)
y2 = dblarr(npack, nt)
z2 = dblarr(npack, nt)
vx2 = dblarr(npack, nt)
vy2 = dblarr(npack, nt)
```

```
vz2 = dblarr(npack, nt)
frac2 = dblarr(npack, nt)
time2 = dblarr(npack, nt)
*startloc.phi = dblarr(npack, nt)

nsteps = long(nt)*long(npack)
print, nsteps
if ((s NE 0) and (options.motion)) then locmoon, runtime, (*geometry.phi)[s], $
  (*SystemConsts.a)[s], (*SystemConsts.orbrate)[s], x=satx, y=saty, ang=ang

for i=0L,npack-1 do begin
  ;; Run one packet at a time for all the times

  x1 = replicate(x0[i], nt)
  y1 = replicate(y0[i], nt)
  z1 = replicate(z0[i], nt)
  vx1 = replicate(vx0[i], nt)
  vy1 = replicate(vy0[i], nt)
  vz1 = replicate(vz0[i], nt)

  ;; Determine the local starting point for each packet - only need to do this
  ;; if the starting position is a function of time
  if ((s NE 0) and (options.motion)) then begin
    (*startloc.phi)[i,*] = ang

    ;; Rotate packets to their proper start position
    aa = reform((*startloc.phi)[i,*] - (*geometry.phi)[s])

    xtemp = *loc.x & ytemp = *loc.y & ztemp = *loc.z
    x2[i,*] = x1 * cos(aa) - y1 * sin(aa)
    y2[i,*] = x1 * sin(aa) + y1 * cos(aa)
    z2[i,*] = z1

    vxtemp = *loc.vx &  vytemp = *loc.vy & vztemp = *loc.vz
    vx2[i,*] = vx1 * cos(aa) - vy1 * sin(aa)
    vy2[i,*] = vx1 * sin(aa) + vy1 * cos(aa)
    vz2[i,*] = vz1

    frac2[i,*] = 1d
    time2[i,*] = runtime
  endif else begin
    x2[i,*] = x1 & y2[i,*] = y1 & z2[i,*] = z1
    vx2[i,*] = vx1 & vy2[i,*] = vy1 & vz2[i,*] = vz1
    frac2[i,*] = 1d & time2[i,*] = runtime
  endelse
endfor

if (nsteps LT maxsteps) then begin
  options.packets = nsteps

  *loc.x = x2[*]
```

```
    *loc.y = y2[*]
    *loc.z = z2[*]
    *loc.vx = vx2[*]
    *loc.vy = vy2[*]
    *loc.vz = vz2[*]
    *loc.frac = frac2[*]
    *loc.finTime = time2[*]

    driver, loc, geometry, options, forces, plasma_info, loss_info, sticking_info, $
      deposition, seed=seed

    *loc.x = reform(*loc.x, npack, nt)
    *loc.y = reform(*loc.y, npack, nt)
    *loc.z = reform(*loc.z, npack, nt)
    *loc.vx = reform(*loc.vx, npack, nt)
    *loc.vy = reform(*loc.vy, npack, nt)
    *loc.vz = reform(*loc.vz, npack, nt)
    *loc.frac = reform(*loc.frac, npack, nt)
    *loc.fintime = reform(*loc.fintime, npack, nt)

    if (options.trackloss) then begin
      *loc.lossfrac = reform(*loc.lossfrac, npack, nt)
      *loc.hitfrac = reform(*loc.hitfrac, npack, nt, n_elements(*SystemConsts.objects))
      *loc.ringfrac = reform(*loc.ringfrac, npack, nt)
      *loc.leftfrac = reform(*loc.leftfrac, npack, nt)
    endif
  endif else begin
    ;; Make arrays for the final values
    x1 = dblarr(npack, nt)
    y1 = dblarr(npack, nt)
    z1 = dblarr(npack, nt)
    vx1 = dblarr(npack, nt)
    vy1 = dblarr(npack, nt)
    vz1 = dblarr(npack, nt)
    frac1 = dblarr(npack, nt)
    if (options.trackloss) then begin
      loss1 = dblarr(npack, nt)
      hit1 = dblarr(npack, n_elements(*SystemConsts.objects), nt)
      ring1 = dblarr(npack, nt)
      left1 = dblarr(npack, nt)
      map1 = dblarr(360, 180, n_elements(*SystemConsts.objects), nt)
    endif
    options.packets = npack

    for i=0L,nt-1 do begin
      *loc.x = x2[*,i]
      *loc.y = y2[*,i]
      *loc.z = z2[*,i]
      *loc.vx = vx2[*,i]
      *loc.vy = vy2[*,i]
      *loc.vz = vz2[*,i]
    *loc.frac = frac2[*,i]
    *loc.fintime = time2[*,i]

    if (i GT 0) then driver, loc, geometry, options, forces, plasma_info, $
      loss_info, sticking_info, deposition, seed=seed

    x1[*,i] = *loc.x
    y1[*,i] = *loc.y
    z1[*,i] = *loc.z
    vx1[*,i] = *loc.vx
    vy1[*,i] = *loc.vy
    vz1[*,i] = *loc.vz
    frac1[*,i] = *loc.frac
    if (options.trackloss) then begin
      loss1[*,i] = *loc.lossfrac
      hit1[*,*,i] = *loc.hitfrac
      ring1[*,i] = *loc.ringfrac
      left1[*,i] = *loc.leftfrac
      if (i GT 0) then begin
        map1[*,*,*,i] = *deposition.map
        lon1 = *deposition.longitude
        lat1 = *deposition.latitude
        destroy_structure, deposition
      endif
    endif
  endfor
  *loc.x = x1
  *loc.y = y1
  *loc.z = z1
  *loc.vx = vx1
  *loc.vy = vy1
  *loc.vz = vz1
  *loc.frac = frac1
  if (options.trackloss) then begin
    *loc.lossfrac = loss1
    *loc.hitfrac = hit1
    *loc.ringfrac = ring1
    *loc.leftfrac = left1
    deposition.longitude = ptr_new(lon1)
    deposition.latitude = ptr_new(lat1)
    deposition.map = ptr_new(map1)
  endif
endelse
options.packets = npack
```

```
;pro modstreamB

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Run packets by advancing packets dt at a time.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

endtime = options.endtime
runtime = dindgen(round(options.endtime)/dt+1)*dt
nt = n_elements(runtime)
options.at_once = 1
npack = options.packets

options.endtime = 0
source_distribution, geometry, spatialdist, speeddist, angulardist, PerturbVel, $
  options, seed, loc=loc, startloc=startloc
*startloc.TravelTime = runtime
options.endtime = endtime

;; Make arrays for the final values
x2 = dblarr(npack, nt) & x2[*,0] = *loc.x
y2 = dblarr(npack, nt) & y2[*,0] = *loc.y
z2 = dblarr(npack, nt) & z2[*,0] = *loc.z
vx2 = dblarr(npack, nt) & vx2[*,0] = *loc.vx
vy2 = dblarr(npack, nt) & vy2[*,0] = *loc.vy
vz2 = dblarr(npack, nt) & vz2[*,0] = *loc.vz
frac2 = dblarr(npack, nt) & frac2[*,0] = *loc.frac
time2 = dblarr(npack, nt)
if (options.trackloss) then begin
  loss2 = dblarr(npack,nt)
  hit2 = dblarr(npack,n_elements(*SystemConsts.objects), nt)
  ring2 = dblarr(npack,nt)
  left2 = dblarr(npack,nt)
  map2 = dblarr(360,180,nt)
endif

xcyc, xc, yc
plot, findgen(10), /nodata, xr=[-10,10], yr=[-10,10], /iso, /xst, /yst
polyfill, xc, yc, color=4

phi0 = *geometry.phi
locmoon, -runtime*options.motion, *geometry.phi, *SystemConsts.a, *SystemConsts.orbrate, $
  x=satx, y=saty, ang=ang

(*loc.fintime)[*] = dt
for i=1L,nt-1 do begin
  w = where(frac2[*,i-1] GT 0, nw, comp=c)
  if (nw NE 0) then begin
    options.packets = nw
    *loc.x = x2[w,i-1]
    *loc.y = y2[w,i-1]
    *loc.z = z2[w,i-1]
    *loc.frac = frac2[w,i-1]
    *loc.vx = vx2[w,i-1]
    *loc.vy = vy2[w,i-1]
    *loc.vz = vz2[w,i-1]
    *loc.fintime = replicate(dt, nw)
    *geometry.phi = reform(ang[i,*])

;    if (nw NE 10) then stop
    driver, loc, geometry, options, forces,plasma_info, loss_info, sticking_info, $
      deposition, seed=seed
    x2[w,i] = *loc.x
    y2[w,i] = *loc.y
    z2[w,i] = *loc.z
    frac2[w,i] = *loc.frac
    vx2[w,i] = *loc.vx
    vy2[w,i] = *loc.vy
    vz2[w,i] = *loc.vz
    time2[*,i] = runtime[i]
    if (options.trackloss) then begin
      loss2[w,i] = *loc.lossfrac
      hit2[w,*,i] = *loc.hitfrac
      ring2[w,i] = *loc.ringfrac
      left2[w,i] = *loc.leftfrac
   ;   if (max(*loc.leftfrac ) NE 0) then stop
      map2[*,*,i] = *deposition.map
      lon2 = *deposition.longitude
      lat2 = *deposition.latitude
      destroy_structure, deposition
    endif

    if (c[0] NE -1) then begin
      x2[c,i] = x2[c,i-1]
      y2[c,i] = y2[c,i-1]
      z2[c,i] = z2[c,i-1]
      frac2[c,i] = frac2[c,i-1]
      vx2[c,i] = vx2[c,i-1]
      vy2[c,i] = vy2[c,i-1]
      vz2[c,i] = vz2[c,i-1]
    endif

 ;   plots, *loc.x, *loc.y, psym=8, color=2
;    wait, .1
  endif
endfor

;; Rotate packets backward to fixed satellite
if (options.motion) then begin
  for i=0,nt-1 do begin
    aa = -(ang[i,s]-ang[0,s])
    x3 = x2[*,i] * cos(aa) - y2[*,i] * sin(aa)
    y3 = x2[*,i] * sin(aa) + y2[*,i] * cos(aa)

    vx3 = vx2[*,i] * cos(aa) - vy2[*,i] * sin(aa)
    vy3 = vx2[*,i] * sin(aa) + vy2[*,i] * cos(aa)

    x2[*,i] = x3 & y2[*,i] = y3
    vx2[*,i] = vx3 & vy2[*,i] = vy3
  endfor
endif

*geometry.phi = phi0
*loc.x = x2
*loc.y = y2
*loc.z = z2
*loc.frac = frac2
*loc.vx = vx2
*loc.vy = vy2
*loc.vz = vz2
*loc.fintime = time2
if (options.trackloss) then begin
  *loc.lossfrac = loss2
  *loc.hitfrac = hit2
  *loc.ringfrac = ring2
  *loc.leftfrac = left2
  deposition.longitude = ptr_new(lon2)
  deposition.latitude = ptr_new(lat2)
  deposition.map = ptr_new(map2)
endif

options.packets = npack
```

```
p o planet_dist, taa, SystemConsts, distance=distance, velocity=velocity

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Given a true anomaly angle, determine the distance and radial velocity of the planet
;; relative to the sun.
;;
;; Outputs: distance (AU) and radial velocity (km/s)
;;
;;  Version history
;;    2.0: 10/22/08 (MHB)
;;      * Computes distance and radial velocity for Mercury.
;;      * For other planets, just assumes circular orbits with dr/dt = 0.
;;      * Should change this, but would require a taa for all models
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (n_params() EQ 0) then begin
  print, 'planet_dat, taa, SystemConsts, distance=distance, velocity=velocity'
  return
endif

a = SystemConsts.aplan  ;AU             semi-major axis
eps = (SystemConsts.planet EQ 'Mercury') ? SystemConsts.epsplan : 0.

if (SystemConsts.planet EQ 'Mercury') then begin
  time = findgen(1001)/1000. ;; time = t/P

  ;; Mean anomaly
  M = (2*!pi*time) mod (2*!pi)

  ;; eccentric anomaly
  EEtemp = findgen(1001)/1000*2*!pi

  mm = EEtemp - eps*sin(EEtemp)
  EE = fltarr(n_elements(time))
  err = fltarr(n_elements(time))
  for i=0,n_elements(EE)-1 do ee[i] = interpol(eetemp, mm-m[i], 0)

  ;; true anomaly
  phi = (2*atan(sqrt((1+eps)/(1-eps)) * tan(EE/2)) + (2*!pi)) mod (2*!pi)
  r = a * (1-eps^2)/(1+eps*cos(phi))

  P = sqrt(a^3)
  drdt = deriv(time*!const.sec_year*0.241, r*!const.au/1e5)  ;; km/s

  distance = interpol(r, phi, taa)
  velocity = interpol(drdt, phi, taa)
endif else begin
  distance = a
  velocity = 0.
```

```
endelse

end
```

```
function get_gvalue, atom, a, path=path

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Looks up the gvalues given by Killen et al 2008. The function returns a
;; 2xn array with the velocities and the radiation pressure constant for the
;; species. Keywords can be used to get the g values for each line  which may
;; be used to calculate the emission.
;;
;; INPUTS:
;;    * a = distance from the sun (AU) -- must be a single value, not an array
;;    * atom = atom for which to look up g-values
;;
;; OUTPUTS
;;    * Function returns 2xn array with velocities and radiation pressure constant.
;;        units = km/s^2
;;    * lines = resonance transitions included
;;    * velocity = radial velocities (km/s)
;;    * gval = array containing g-value vs. velocity for each transition
;;
;;  Version History
;;    3.1: 10 May 2011
;;      * New way of saving g-values. Use set_up_gvals to save into structures
;;    2.1: 30 Jan 2009
;;      - added g-values for all species included in Killen et al. [C I, Ca I, Ca II,
;;         H I, He I, K I, Mg I, Mg II, Na I, OH, O I, S I
;;    2.0: original -- only looks up Na g values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

if (n_elements(a) GT 1) then stop
if (a EQ !null) then a = 1.
if (n_elements(path) EQ 0) then $
  path = !model.basepath + 'Work/AtomicData/g-values/' + atom
if ~(file_test(path)) then stop

files = file_search(path, '*.sav') & nf = n_elements(files)

gval = {species:atom, $
        a:a, $                      ;; AU
        wavelength:ptr_new(0), $  ;; Angstrom
        v:ptr_new(0), $             ;; km/s
        g:ptr_new(0), $             ;; photons cm^-2 s^-1
        radaccel:ptr_new(0)}      ;; km^-2 s^-1
case (nf) of
  0: begin
     *gval.v = [0., 1.]
     *gval.g = [0., 0.]
     print, 'g-values not found. Radiation acceleration = 0'
     end
  1: begin
```

```
     restore, files[0]
     *gval.wavelength = gvalue.wavelength
     *gval.v = *gvalue.v
     *gval.g = *gvalue.g * gvalue.a^2/a^2 ;; normalize to specified distance
     end
  else: begin
     lambda = fltarr(nf)
     vv = ptrarr(nf, /allocate)
     gg = ptrarr(nf, /allocate)
     for i=0,nf-1 do begin
       restore, files[i]
       lambda[i] = gvalue.wavelength
       *vv[i] = *gvalue.v
       *gg[i] = *gvalue.g * gvalue.a^2/a^2
     endfor

     ;; Test if all wavelengths are unique
     u = uniq(lambda, sort(lambda))
     if (n_elements(u) NE nf) then begin
       ;; Need to decide which to use
       stop
     endif
     *gval.wavelength = lambda

     ;; Get a common velocity axis
     allv = !null
     for i=0,nf-1 do allv = [allv, *vv[i]]
     allv = allv[sort(allv)]
     *gval.v = allv[uniq(allv)]
     *gval.g = fltarr(n_elements(*gval.v),nf)
     for i=0,nf-1 do (*gval.g)[*,i] = interpol(*gg[i], *vv[i], *gval.v)
     end
endcase

;; radpres_const = h/(m*lambda) * g
rr = !const.h / atomicmass(atom) / (*gval.wavelength*1e-8)
qq = 0.
for i=0,nf-1 do qq += rr[i]*(*gval.g)[*,i]*1e-5  ;; km s^-2
*gval.radaccel = qq

return, gval

end
```

```
;;; combine the two plasma files into a single file with the info I want
;;; These are not really valid near the moons, but are good for large scale
;;; clouds

restore, '$HOME/NeutralModel/modelpro/data/CAPSplasma/Enceladus.plasma.sav'
ltemp = mtorus

restore, '$HOME/NeutralModel/modelpro/data/CAPSplasma/SOI.plasma.sav'

Mtorus = lgrid
LatTorus = latgrid

t_etorus = interpol(t_e, ltemp, mtorus)
t_wtorus = interpol(t_i[*,1], ltemp, mtorus)
t_htorus = interpol(t_i[*,0], ltemp, mtorus)

n_ehotgrid = n_egrid * .2/70.
t_ehottorus = t_etorus * 12.5/1.5

save, Mtorus, LatTorus, N_egrid, n_hgrid, n_wgrid, t_etorus, t_wtorus, t_htorus, $
  n_ehotgrid, t_ehottorus, $
  file='$HOME/NeutralModel/modelpro/data/CAPSplasma/SaturnPlasma.sav'

end
```

```
pro SystemConstants, planet, SystemConsts, DipoleConsts

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version 2.0: 15 June 2010
;;   Creates the systemconsts and dipoleconsts structures from data stored
;;   in the !Planet system variables
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SystemConsts = {Planet:'', rPlan:0d, aPlan:0d, epsPlan:0d, $
  Objects:ptr_new(0), GM:ptr_new(0), radius:ptr_new(0), a:ptr_new(0), eps:ptr_new(0), $
  orbvel:ptr_new(0), period:ptr_new(0), orbrate:ptr_new(0)}

case strlowcase(planet) of
  'sun': plan = !sun
  'mercury': plan = !Mercury
  'venus': plan = !Venus
  'earth': plan = !Earth
  'mars': plan = !Mars
  'jupiter': plan = !Jupiter
  'saturn': plan = !Saturn
  'uranus': plan = !Uranus
  'neptune': plan = !Neptune
  'pluto': plan = !Pluto
endcase

SystemConsts.planet = plan.name
SystemConsts.rplan = plan.radius
SystemConsts.aplan = plan.a
SystemConsts.epsplan = plan.e

tt = tag_names(plan)
if (total(strcmp(tt, 'satellites', /fold))) then begin
  *SystemConsts.objects = [plan.name, plan.satellites]

  mm = [plan.mass, plan.msat]
  rr = [plan.radius, plan.rsat]
  tt = [0d, plan.orbsat*24.*3600.]
  vv = [0d, 2*!dpi*plan.asat*plan.radius/tt[1:*]]

  *SystemConsts.GM = -!const.G*mm*1e3/(plan.radius*1e5)^3
  *SystemConsts.radius = rr/plan.radius
  *SystemConsts.a = [0d, plan.asat]
  *SystemConsts.eps = [0d, plan.esat]
  *SystemConsts.period = tt
  *SystemConsts.orbvel = vv
  *SystemConsts.orbrate = [0d, 2*!dpi/tt[1:*]]
endif else begin
  *SystemConsts.objects = plan.name
  *SystemConsts.GM = -!const.G*plan.mass*1d3/(plan.radius*1d5)^3
  *SystemConsts.radius = 1d
  *SystemConsts.a = 0d
  *SystemConsts.eps = 0d
  *SystemConsts.period = 0d
  *SystemConsts.orbvel = 0d
  *SystemConsts.orbrate = 0d
endelse

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Read in the dipole constants
file = !model.basepath + 'Work/Data/PhysicalData/DipoleConstants.dat'
if ~file_test(file) then stop ;; file = (file_search('$HOME', 'DipoleConstants.dat'))[0]
readcol, /silent, file, delim=':', ob, mm, t, tdir, o, olon, olat, per, $
  format='A,D,D,D,D,D,D,D'
ob = strtrim(ob, 2)
q = (where(strcmp(planet, ob, /fold), nq))[0]
if (nq) $
  then DipoleConsts = {$
    strength:mm[q], $
    tilt:t[q]*!dtor, $
    lam3:tdir[q]*!dtor, $
    offset:o[q], $
    offlong:olon[q]*!dtor, $
    offlat:olat[q]*!dtor, $
    period:per[q], $
    magrat:2*!dpi/per[q]} $
   else DipoleConsts = -1

end
```

```
function determine_image_rotation, input, format

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; There are lots of ways to specify what the proper viewing geometry should be.
;; Need to break that down into three angles: rotations about the x, y, and z axis
;; in model coordinates.
;;
;; Tags that can be specified:
;;   a) SubObsLongitude, SubObsLatitude, PolarAngle
;;      * SubObsLong and Lat give the intersection point on the surface in
;;        longitude and latitude relative to the sub-solar point
;;      * On a planet, define the SubObsLongitude positive in the ccw direction when
;;        looking down from above.
;;        * long=0 -> looking down over sub-solar meridian
;;        * long=!pi/2 -> looking down over dusk meridian
;;        * long=!pi -> looking down over midnight meridian
;;        * long=3*!pi/2 -> looking down over dawn meridian
;;      * Latitude defined positive is north
;;        * lon = -!pi/2 -> looking down over south pole
;;        * lon = 0 -> looking down over equator
;;        * lon = !pi/2 -> looking down over north pole
;;   b) Observer (e.g. Earth, MESSENGER, ...), time
;;
;; Version History
;;   4.0: 25 Jan 2011
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

tags = strlowcase(tag_names(format.geometry))

q0 = total(strmatch(tags, 'SubObsLongitude', /fold))
w0 = total(strmatch(tags, 'SubObsLatitude', /fold))

q1 = total(strmatch(tags, 'observer', /fold))
w1 = total(strmatch(tags, 'time', /fold))

case (1) of
  (q0+w0 EQ 2): begin
    pSun = [0.,-1.,0.]
    pObs = [sin(format.geometry.SubObsLongitude)*cos(format.geometry.SubObsLatitude), $
      -cos(format.geometry.SubObsLongitude)*cos(format.geometry.SubObsLatitude), $
      sin(format.geometry.SubObsLatitude)]
    end
  (q1+w1 EQ 2): begin
    relative_position, 0., 0., 0., format.geometry.observer, input.geometry.planet, $
      frame='J2000', pos=pObs, havetime=utc2et(time)
    relative_position, 0., 0., 0., 'Sun', input.geometry.planet, frame='J2000', $
      pos=pSun, havetime=utc2et(time)
    end
endcase
```

```

M = rotationmatrix(pSun, pObs) ;; This is the rotation in space relative to the
       ;; model coordinate system

;; Determine rotation of FOV -- rotation about new y-axis
q = rotation([0,0,0], [0,1.,0], format.geometry.PolarAngle, R=R)
M = R # M

return, M

end
```

```
function display_hull, pts

;; pts = an array of points to look at computed by results_voronoi
;;   (pts = *regions[i])

sz = size(pts)

hullfile = ('hull' + strint(round(random_nr(1)*1000000)) + '.dat')[0]
openw, lun, hullfile, /get_lun
printf, lun, sz[2]
printf, lun, sz[1]
printf, lun, transpose(pts)
free_lun, lun

spawn, 'qconvex s Fv TI hullpts.dat TO ' + hullfile
spawn, 'rm ' + hullfile

nfac = long(out[0])
facets = out[1:*]
if (n_elements(facets) NE nfac) then stop
connect = !null
for i=0,nfac-1 do begin
  w = long(strsplit(line, /extract)
  connect = [connect, w]
endfor

s0 = plot3d(pts[*,0], pts[*,1], pts[*,2], dimensions=[1000,1000], symbol='*', $
  linestyle=' ', /aspect_ratio, /aspect_z, /sym_filled)
s2 = polygon(pts[*,0], pts[*,1], pts[*,2], connectivity=connect, fill_color='blue', $
  fill_transparency=50, /data)

return, [s0, s1]

end
```

```
function display_model_image, result, savefile, brange=brange, log=log, _extra=e

if (n_elements(brange) NE 2) then $
  brange = minmax((*result.image)[where(*result.image NE 0)])
if (log EQ !null) then log = 0

xcyc, xc, yc

etags = (e NE !null) ? tag_names(e) : ''
rgb = (total(strcmp(etags, 'rgb_table', /fold))) ? e.rgb_table : 3
title = (total(strcmp(etags, 'title', /fold))) ? e.title : 'Image'
xtitle = (total(strcmp(etags, 'xtitle', /fold))) ? e.xtitle : 'Distance'
ytitle = (total(strcmp(etags, 'ytitle', /fold))) ? e.ytitle : 'Distance'
ztitle = (total(strcmp(etags, 'ztitle', /fold))) ? e.ztitle : 'Intensity'

if (log) $
  then im = bytscl(alog10(*result.image), alog10(brange[0]), alog10(brange[1])) $
  else im = bytscl(*result.image, brange[0], brange[1])

pp = image2(im, *result.xaxis, *result.zaxis, rgb_table=rgb, $
  dimensions=[800,600], location=[0,0], $
  position=[120,100,520,500], /dev, $
  font_size=20, title=title, xtitle=xtitle, ytitle=ytitle)
pp[0].refresh, /disable
p1 = plot(/overplot, xc, yc, thick=3, color='blue')
p2 = plotsquare2(minmax(*result.xaxis), minmax(*result.zaxis), thick=3)

pos = [550,140,600,460]
cb = colorbar2(pos, brange, log=log, rgb_table=rgb, thick=2, font_size=20, $
  title=ztitle)
pp[0].refresh

if (savefile NE !null) then pp[0].save, savefile, width=800
pp = [pp, p1, p2, cb]

return, pp

end
```

```
function emission_measure, atom, line, vy=vy, aplanet=aplanet, ee=ee

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Computes the emission measure for each packet. This is used in
;; line_of_sight, model_images, and density_track.
;;
;; Required parameters:
;;    * atom
;;    * line = vector of lines to compute emission for in Å
;; Optional depending on the emission type and line
;;    * vy = radial velocity relative to the sun
;;    * aplanet = heliocentric planet distance. If not specified, then resonant scattering
;;        is not computed
;;
;; Outputs:
;;    Function returns the emission measure per atom for the requested lines
;;    ee = resonant scattering emission measure for each line
;;
;; Version 2.0: 19 April 2010
;;    * written based on already existing method in line_of_sight and model_images.
;;    * need a new version to make sure things are done consistently.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

nl = n_elements(line)
doresscat = (n_elements(aplanet) EQ 1)
doeimp = 0

;; Correct for Na wavelength issues
if (atom EQ 'Na') then begin
  q = where(line EQ 5890, nq)
  if (nq EQ 1) then line[q] = 5891.
  q = where(line EQ 5896, nq)
  if (nq EQ 1) then line[q] = 5897.
endif

;; Resonant Scattering
if (doresscat) then begin
  q = get_gvalue(aplanet, atom, lines=ll, velocity=radvel, gval=gval)
  w = where(vy LT min(radvel), nw) & if (nw NE 0) then vy[w] = min(radvel)
  w = where(vy GT max(radvel), nw) & if (nw NE 0) then vy[w] = max(radvel)

  ee = fltarr(n_elements(vy),nl)
  for i=0,nl-1 do begin
    q = where(ll EQ line[i], nq)
    if (nq NE 1) $
      then print, 'Error: g-value not found for emission line ' + string(line[i]) $
      else ee[*,i] = interpol(gval[*,q], radvel, vy)/1e6
  endfor
  resscat = (nl EQ 1) ? ee : total(ee,2)
```

```
endif else resscat = 0.

;; Electron Impact
if (doeimp) then begin
  stop
endif else eimp = 0.

result = resscat + eimp
return, result

end
```

```
pro model_view, image, x0, b0, b1

sz = (size(image))[1:2]

xc = cos(findgen(361)*!dtor) & yc = sin(findgen(361)*!dtor)
plot, findgen(10), /nodata, xr=minmax(x0), yr=minmax(x0), /xst, /yst, $
  xtit='Distance (R!dObj!n)', ytit='Distance (R!dObj!n)', $
  pos=[100,100,100+sz[0],100+sz[1]], /dev
tv, bytscl(image, b0, b1, top=220)+35, 100, 100, /dev
polyfill, xc, yc, color=4
plots, [100,100,100+sz[0],100+sz[0],100], [100,100+sz[1],100+sz[1],100,100], /dev

end
```