```
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]
```

```
 52
 53  ;; Photon limited flux (normalized)
 54  photmap = cos(longitude) # cos(latitude)
 55  photmap[where(longitude GT !pi/2 and longitude LT 3*!pi/2),*] = 0.
 56  q = where(photmap LT 0, nq) & if (nq NE 0) then photmap[q] = 0
 57  ;;;;;;;;;;;;;;;;;
 58
 59  ;; Diffusion limited flux (normalize)
 60  if (input.SpatialDist.kappa GT 0) then begin
 61    restore, input.SpatialDist.ProtonPrecipFile
 62    if (n_elements(photmap) NE n_elements(*sourcemap.map)) then stop
 63    difmap = (1 + input.SpatialDist.kappa/1e8 * *sourcemap.map)
 64  endif else difmap = replicate(1., n_elements(longitude), n_elements(latitude))
 65  ;;;;;;;;;;;;;;;;;
 66
 67  difmap = difmap*input.SpatialDist.DiffusionLimit
 68  q = (photmap LT difmap)
 69  map = q*photmap + (1-q)*difmap
 70  q = where(finite(map) EQ 0, nq) & if (nq NE 0) then stop
 71
 72  ;; compute total PSD source rate for normalization purposes later
 73  rate = total(map*dcos) * (!Mercury.radius*1e5)^2 * dlon * dlat
 74
 75  sourcemap = {longitude:ptr_new(longitude), latitude:ptr_new(latitude), $
 76    map:ptr_new(map), rate:rate}
 77
 78  return, sourcemap
 79
 80  end
 81
 82  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 83  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 84
 85  pro PSD_distribution, input, output, npack, seed
 86
 87  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 88  ;;
 89  ;; Distribute packets according to PSD spatial distribution parameters
 90  ;;
 91  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 92
 93  sourcemap = PSDfluxmap(input)
 94
 95  *sourcemap.map /= max(*sourcemap.map)
 96  RandomDeviates_2d, *sourcemap.map, *sourcemap.longitude, sin(*sourcemap.latitude), $
 97    npack, lon, lat
 98  lat = asin(lat)
 99  destroy_structure, sourcemap
100
101  if strcmp(input.geometry.planet, input.geometry.StartPoint, /fold) then begin
102    ;; 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
```