```
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
```

```
154     phi = (*input.geometry.phi)[stuff.s]
155     ;; Rotate to proper starting position based on *output.time
156     if (input.options.motion) $
157       then locmoon, *output.time, phi, (*SystemConsts.a)[stuff.s], $
158         (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty , ang=ang $
159       else locmoon, fltarr(npack), phi, (*SystemConsts.a)[stuff.s], $
160         (*SystemConsts.orbrate)[stuff.s], x=satx, y=saty, ang=ang
161
162     ang = (ang + 2*!dpi) mod (2*!dpi)
163     *output.phi0 = ang  ;; Starting local time for each packet
164
165     ;; Rotate to proper starting orbital phase
166     *output.x = xx * cos(ang) - yy * sin(ang)
167     *output.y = xx * sin(ang) + yy * cos(ang)
168     *output.z = zz
169
170     *output.vx = vx * cos(ang) - vy * sin(ang)
171     *output.vy = vx * sin(ang) + vy * cos(ang)
172     *output.vz = vz
173   endelse
174
175   q = where(finite(*output.vx) EQ 0, nq) & if (nq NE 0) then stop
176   q = where(finite(*output.vy) EQ 0, nq) & if (nq NE 0) then stop
177   q = where(finite(*output.vz) EQ 0, nq) & if (nq NE 0) then stop
178
179   *output.frac = *output.f0
180   output.totalsource = total(*output.frac)
181
182   end
```