```
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
```