```
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
```