An Introduction to the Piecewise Parabolic Method (PPM) Library
II. I/O and Data Analysis

David H. Porter,
Sarah E. Anderson,
Paul R. Woodward,
and B Edgar,

Laboratory of Computational Science or Engineering,
Department of Astronomy, School of Physics and Astronomy
116 Church St. SE.
University of Minnesota
Minneapolis, Minnesota 55455

3. Restart Dumps

a. Introduction

Any interesting computational problem requires many hours (if not days) of resources, and most users have experienced first hand the frustration of seeing the computer crash just before the completion of a computation. Therefore the prudent periodically create data files from which the program can be restarted *exactly,* thus salvaging the time and resources expended up to the creation of each file. These files shall be referred to as *restart dumps.*

PPMLIB I/O routines will be described in the specific context of a two dimensional hydrodynamical calculation. The calculation is controlled by a program called a *driver*, which is a program, or collection of programs which perform the basic housekeeping chores inherent to most any calculation. In the specific example presented here, these chores consist of

a) allocating memory

b) problem initialization

c) performing the calculation

d) producing output, if desired

While the intent here is not to describe how to perform hydrodynamic calculations, † it is necessary to examine the driver in some detail. With this in mind, the basic driver `Driverdirect2dg0c_v00` will now be described.

```
      program Driverdirect2dg0c_v00
c
      parameter (NXZONES = 256)
      parameter (NYZONES = 256)
      parameter (EOSGAM = 1.4)
      parameter (DT = 0.001100)
      parameter (XMAX = 1.0)
      parameter (YMAX = 1.0)
      parameter (N2DUMP = 20)
      parameter (T2DUMP = 99999.)
c     Note that because of the fixed timestep, N, XMAX, and DT are not
c               independent.
c
c      NXZONES = number of real computational zones in the x-direction, set
c                by an m4 definition at beginning of code.
c      NYZONES = number of real computational zones in the y-direction, set
```

† A more complete description from which the drivers describe here have been derived may be found in volume I of this manual.

```
c                 by an m4 definition at beginning of code.
c      EOSGAM = the equation of state gamma
c      DT = the fixed timestep
c      XMAX = the righthand edge x-value of the grid.
c                                    The x-grid runs from (0,XMAX)
c      YMAX = the righthand edge y-value of the grid.
c                                    The y-grid runs from (0,YMAX)
c      N2DUMP is the frequency of image dumps in iterations
c      T2DUMP is the frequency of image dumps in program time.
c
c     Do not change this 1.:
      parameter (NBDY = 5)
c
c

      dimension  xl( 1-NBDY:NXZONES+NBDY+1)
      dimension  yl( 1-NBDY:NYZONES+NBDY+1)
      dimension ddd( 1:4, 1-NBDY:NXZONES+NBDY, 1-NBDY:NYZONES+NBDY )
c
c
c
c     ddd    = the 3-D array into which the 5 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c
c      xl    = the locations of the lefthand zone interfaces
c      yl    = the locations of the bottom zone interfaces
c     ddd    = the 3-D array into which the 4 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
c     ddd(4,i,j) = uuy(i,j)
c     rrho   = the zone-averaged densities
c     pp     = the zone-averaged pressures
c     uux    = the zone-averaged x-velocities
c     uuy    = the zone-averaged y-velocities
c
c
c     smlrho = a trivial value of density, used to protect divides.
c     smalle = a trivial value of energy, used to protect divides.
c     smallp = a trivial value of pressure, used to protect divides.
c     smallu = a trivial value of velocity, used to protect divides.
c
      smlrho = 1.0e-06
      smalle = 1.0e-06
      smallp = 1.0e-06
      smallu = 1.0e-06
c
c
      ndump = N2DUMP
      tdump = T2DUMP

c          nstop is the maximum number of iterations to take for the entire
c             calculation
c          tstop is the maximum program time for the entire calculation, here
c             assuming a constant timestep DT
c          ndump is the frequency of image dumps in iteratiopns
c          tdump is the frequency of image dumps in program time.

c   Request the number of timesteps to complete
       print *,' Please enter number of timesteps to perform.'
       read *,nstop
c
c   Adjust nstop so that it is a multiple of ndump.  This avoids waste.
c
      nstop = ndump * ( (nstop +ndump -1)/ndump )
      tstop = float (nstop) * DT
      print *,' Thank you.  The program'
      print *,' will stop after ',nstop,' timesteps.'

c   initialize counters
      ncycle = 0
      time = 0.0
c
c     initialize the grid:
      xlimit = XMAX
      ylimit = YMAX
c
      call setgrid(  xl,  yl, xlimit, ylimit, nxzones, nyzones, nbdy )
c
c
c     initialize the hydrodynamic quantities:
c
      call set2dshock( ddd, nxzones, nyzones, nbdy )
c
c
c   CONTROL performs the entire hydrodynamic calculation.
c
      dtime = DT
      gamma = EOSGAM

      call cntrol ( xl,  yl, ddd, gamma, dtime, smlrho, smalle, smallp,
     ^             smallu, ncycle, numdmp, time, ndump, tdump, nstop, tstop,
     ^             nxzones, nyzones, nbdy)
```

```
      stop
      end
```

The main program, `Driverdirect2dg0c_v00`, is deliberately simple. It performs a specified number of timesteps of a two-dimensional problem, with a fixed timestep and problem size. (The number of timesteps to complete, entered by the user at program execution, is adjusted to ensure that it is a multiple of the image dump frequency to avoid wasted effort.) There are three arrays which are allocated. Two define the computational grid by specifying the locations of the left- and right-hand zone interfaces, `xl` and `yl`, resp. The remaining array contains the zone averages of all the computed quantities which, for this problem, are densiy, pressure, x-, and y-velocity. These are stored in the array `ddd` in interleaved order, that is, the zone averages of density, pressure, and the two velocities for a zone *(i,j)* are contiguous in memory. making `ddd` a *three*-dimensional array, with the first dimension being the number of zone-averaged quantities in the array. For example, the density, pressure, x- and y-velocity for zone (i, j) are `ddd(1,i,j)`, `ddd(2,i,j)`, `ddd(3,i,j)`, `ddd(4,i,j)`, resp. This interleaving provides greater efficiency on cache-based computers.

```
      dimension  xl( 1-NBDY:NXZONES+NBDY+1)
      dimension  yl( 1-NBDY:NYZONES+NBDY+1)
      dimension ddd( 1:4, 1-NBDY:NXZONES+NBDY, 1-NBDY:NYZONES+NBDY )
```

The number of actual computational zones in the x- and y- directions are given by the parameters `NXZONES` and `NYZONES`, resp. An additional `NBDY` zones are added to the left and right of each dimension to allow for the imposition of boundary conditions. Consider that in order to produce the updated zone averages for any given zone, the hydrodynamic subroutine requires information from surrounding zones. Ordinarily this would mean that the zones along the edges of the computational domain could not be updated due to the lack of information corresponding to the zones not present. This is avoided here by appending a sufficient number of *fake zones*, as if a picture frame, around the computational domain. These fake zones will be filled with values appropriate to the user's chosen boundary conditions. This allows zones on the boundary of the *computation* to the treated as internal zones.

The arrays are initialized in two steps. First the computational grid is set by a call to the subroutine `setgrid`. This creates a uniform computational grid running from 0.0 to 1.0 in both dimensions. The reader should note that a computational grid of `N` zones would require `N+1` lefthand zone interface locations.

```
      subroutine setgrid(  xl,  yl, xmax, ymax, nx, ny, nbdy )
c
c
c
c    This routine initializes the locations of the lefthand zone interfaces in
c      both the x- and y-directions, assuming a uniform grid.  The grid consists
c      of NX zones (defined by NX+1 lefthand zone interfaces) in the x-direction
c      ranging from 0.0 to XMAX, with NBDY zones added to each end to allow for
c      boundary conditions.  there are NY zones (defined by NY+1 lefthand zone
c      interfaces) in the y-direction ranging from 0.0 to YMAX, with NBDY zones
c      added to each end to allow for boundary conditions.
c
c       xl    = the x-locations of the lefthand zone interfaces.
c       yl    = the y-locations of the lefthand zone interfaces.
c
c      xmax  = The real (non boundary) portion of the grid is
c                XXL(1)=0.0 to XXL(NX+1)=XMAX.
c      ymax  = The real (non boundary) portion of the grid is
c                YYL(1)=0.0 to YYL(NY+1)=YMAX.
c
c     nx = number of real zones in in the x-direction
c     ny = number of real zones in in the y-direction
c   nbdy = number of boundary zones added to each end of the arrays to
c          implement boundary conditions.

      dimension   xl(1-nbdy:nx+nbdy+1)
      dimension   yl(1-nbdy:ny+nbdy+1)

c
      deex = xmax / float(NX)
      DO I=1,NX+1
         xl(i) = deex*float(i-1)
      ENDDO
```

```
      deey = ymax / float(NY)
      DO I=1,NY+1
         yl(i) = deey*float(i-1)
      ENDDO

      return
      end
```

Next, the zone averages are initialized by `set2dshock`. The initial conditions are those of a “two-dimensional shocktube”, which is a rectangular computational domain divided into four smaller rectangles or subdomains, with all four meeting at the point (nxdisc, nydisc) within the computational deomain. Each subdomain consists of a constant hydrodynamic state. After problem initialization, the program flow proceeds to `cntrol` which will perform the rest of the calculation.

```
      subroutine set2dshock( ddd, nx, ny, nbdy )
c
c     This routine sets the initial values of all hydrodynamic
c     quantities for BKE’s 2d shock tube problem.  The initial
c     state consists of 4. different constant states:
c
c   rhoq2 = 2.0        rhoq1 = 1.0
c     pq2 = 2.0          pq1 = 1.0
c    uxq2 = 0.0         uxq1 = 0.0
c    uyq2 = 0.0         uyq1 = 0.0
c
c   rhoq3 = 0.5        rhoq4 = 0.1
c     pq3 = 0.5          pq4 = 0.1
c    uxq3 = 0.0         uxq4 = 0.0
c    uyq3 = 0.0         uyq4 = 0.0
c         .....................................................
c         .                       .                           .
c         .                       .                           .
c         .                       .                           .
c         .      Q2               .            Q1             .
c         .                       .                           .
c         .                       .                           .
c         .....................................................  < nydisc
c         .                       .                           .
c         .                       .                           .
c         .                       .                           .
c         .                       .                           .
c         .      Q3               .            Q4             .
c         .                       .                           .
c         .                       .                           .
c         .                       .                           .
c         .                       .                           .
c         .....................................................
c                                 ^
c                               nxdisc
c
c   assuming a gamma law equation of state: p = (gamma-1)*rho*ei
c
c
c     ddd    = the 3-D array into which the 4 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
c     ddd(4,i,j) = uuy(i,j)
c
      dimension ddd( 1:4, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c

      parameter( rhoq1 = 1.0)
      parameter(   pq1 = 1.0)
      parameter(  uxq1 = 0.0)
      parameter(  uyq1 = 0.0)

      parameter( rhoq2 = 2.0)
      parameter(   pq2 = 2.0)
      parameter(  uxq2 = 0.0)
      parameter(  uyq2 = 0.0)

      parameter( rhoq3 = 0.5)
      parameter(   pq3 = 0.5)
      parameter(  uxq3 = 0.0)
      parameter(  uyq3 = 0.0)

      parameter( rhoq4 = 0.1)
      parameter(   pq4 = 0.1)
      parameter(  uxq4 = 0.0)
      parameter(  uyq4 = 0.0)

c
c   Location of discontinuity in the x-direction:
      nxdisc = nx/4
```

```
c   Location of discontinuity in the y-direction:
      nydisc = (4*ny)/5

      do 100   k = nydisc,ny+nbdy
      do 100   i = nxdisc,nx+nbdy
      ddd(1,i,k) = rhoq1
      ddd(2,i,k) = pq1
      ddd(3,i,k) = uxq1
      ddd(4,i,k) = uyq1
100   continue
c
      do 200   k = nydisc,ny+nbdy
      do 200   i = -nbdy+1,nxdisc-1
      ddd(1,i,k) = rhoq2
      ddd(2,i,k) = pq2
      ddd(3,i,k) = uxq2
      ddd(4,i,k) = uyq2
200   continue
c
      do 300   k = 1-nbdy,nydisc-1
      do 300   i = -nbdy+1,nxdisc-1
      ddd(1,i,k) = rhoq3
      ddd(2,i,k) = pq3
      ddd(3,i,k) = uxq3
      ddd(4,i,k) = uyq3
300   continue
c
      do 400   k = 1-nbdy,nydisc-1
      do 400   i = nxdisc,nx+nbdy
      ddd(1,i,k) = rhoq4
      ddd(2,i,k) = pq4
      ddd(3,i,k) = uxq4
      ddd(4,i,k) = uyq4
400   continue
c
c
c
c
      return
      end
```

This subroutine, `cntrol` is essentially an iterative loop consisting of two steps. First, the hydrodynamical calculation is advanced two time steps by a call to `runhyd`. For the purposes of this I/O discussion, `runhyd` may be treated as a "black box" which replaces the initial zone averaged values stored in the non-fake zones of the array `ddd` with the new zone averages advanced two timesteps. Then, when appropriate, an image file is created by a call to `dump`.

```
      subroutine cntrol ( xl,  yl, ddd, gamma, dtime, smlrho, smalle,
     ^      smallp, smallu, ncycle, numdmp, time,  ndump, tdump, nstop,
     ^       tstop, nx, ny, nbdy)
c
      dimension  xl( 1-NBDY:NX+NBDY+1)
      dimension  yl( 1-NBDY:NY+NBDY+1)
      dimension ddd( 1:4, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c
c     This routine runs the entire hydro calculation.
c
c
c
c     ddd    = the 3-D array into which the 5 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     nx     = the number of real zones in the x-dimension for this
c              tile of the computational domain.
c     ny     = the number of real zones in the y-dimension for this
c              tile of the computational domain.
c     nbdy   = the number of fake zones needed to implement boundary
c              conditions so that in the hydrodynamical calculation
c              all zones can be treated as if they were interior zones.
c
c
c     dtime  = the time step.
c     courmx = the maximum value of the Courant number for this tile
c              of the computational domain.
c     gamma  = the value of the ratio of specific heats, as in the
c              gamma-law equation of state:  p = (gamma-1)*rho*ei
c
c     smlrho = a trivial value of density, used to protect divides.
c     smalle = a trivial value of energy, used to protect divides.
c     smallp = a trivial value of pressure, used to protect divides.
c     smallu = a trivial value of velocity, used to protect divides.
c     numdmp = a count of the number of image dumps completed so far
c     ncycle = a count of the number of time steps taken so far
c     time   = the elapsed problem time to this point
```

```
c
c      xl    = the locations of the lefthand zone interfaces
c      yl    = the locations of the bottom zone interfaces
c     ddd    = the 3-D array into which the 5 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
c     ddd(4,i,j) = uuy(i,j)
c     rrho   = the zone-averaged densities for this tile.
c     pp     = the zone-averaged pressures for this tile.
c     uux    = the zone-averaged x-velocities for this tile.
c     uuy    = the zone-averaged y-velocities for this tile.
c
c
c    Create an image of the initial state. The initial state will be
c     sequence number 0.

       if ( numdmp .eq. 0 )  then
         call dump (ddd, numdmp, nx, ny, nbdy)
c           numdmp = the sequential number of the image 0->whatever, numdmp
c           will be automatically incremented after write the image.
       endif
c
      ntodmp = ndump
      ttodump = tdump
c
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c      The 1000 loop advances the calculation by 2. timesteps per    c
c                            iteration                               c
c                                                                    c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
1000  continue
c
c   clear the old maximum courant number in preparation of the next pair of
c       timesteps:
      courmx = 0.
c
c   RUNHYD performs a symmetrized pair of passes x-y-y-x which
c          equals 2. complete timesteps.
c
      call runhyd_lr ( xl,  yl, ddd, dtime, courmx, gamma,
     1             smlrho, smalle, smallp, smallu,
     2             nx, ny, nbdy)
c
c
c
c   update counters
c
      ncycle = ncycle + 2
      ntodmp = ntodmp - 2
      time = time + dtime + dtime
c
      write (6,1710)   ncycle, time, courmx, dtime
c
c   Check to see if time to write am image file.  An image is
c   written every NDUMP steps or at intervals of DTDUMP in
c   problem time.
c
      if (ntodmp)   1600,1600,1500
1500  if (time - tdump)   1700,1600,1600
1600  ntodmp = ndump
      ttodump = ttodump + tdump

      call dump (ddd, numdmp, nx, ny, nbdy)
c     numdmp = the sequential number of the image 0->whatever, numdmp
c           will be automatically incremented after write the image.

      call flush (6)
c
1700  continue
1710  format (1h ,3hn =,i5,4x,3ht =,
     1        1pe12.5,4x,13hcourant no. =,e12.5,4x,4hdt =,e12.5)
c
c   Check to see if time to stop.  Progam execution ends when either
c    the maximum number of timesteps, NSTOP,  has been computed or
c    when the maximum problem time, TSTOP, has been reached.
c
      if (ncycle .ge. nstop) return
      if (time .ge. tstop) return
      goto 1000
      end
```

In dump a file name is constructed based upon two supplied one character strings and the sequential number of the image. The four hydrodynamic quantities of density, pressure, and the two velocity components are extracted from ddd, without the fake zones, individually scaled, and composited in a one byte array `igray.`. This will produce one image within which are images of the four quantities arranged as a two by two matrix. At this point the bytes comprising the composite image are now contiguous in memory in this array of bytes. The disk file is opened using **ppm98_binopen_trunc**, and the composite image values written to it by **ppm98_binwrite**, after which the file is closed by **ppm98_binclose**. The image sequence number is then incremented in preparation for the next call. Results from this example at four different times are shown in figure 1.

```
      subroutine dump (ddd, numdmp, nx, ny, nbdy )
c
c   we are counting on the F77 compiler to automatically allocate/deallocate
      character*1 igray(2*nx,2*ny)
      character*1 acharc, acharc2
      character*8 namdmp
c
c
c     ddd    = the 3-D array into which the 4 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
c     ddd(4,i,j) = uuy(i,j)
c     rrho   = the zone-averaged densities.
c     pp     = the zone-averaged pressures.
c     uux    = the zone-averaged x-velocities.
c     uuy    = the zone-averaged y-velocities.
c
c     numdmp = the sequential number of the image 0->whatever, numdmp
c            will be automatically incremented after write the image.
c      nx = number of real computational zones in the x-direction, set
c                by an m4 definition at beginning of code.
c      ny = number of real computational zones in the y-direction, set
c                by an m4 definition at beginning of code.
c
c      nbdy = the number fake zones necessary for boundary conditions.
c
      dimension ddd( 1:4, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c
      acharc = 'D'
      acharc2 = 'e'
c
c     determine individual min/max for scaling
c
      rhomin = ddd(1,1,1)
      rhomax = ddd(1,1,1)
      pmin = ddd(2,1,1)
      pmax = ddd(2,1,1)
      uxmin = ddd(3,1,1)
      uxmax = ddd(3,1,1)
      uymin = ddd(4,1,1)
      uymax = ddd(4,1,1)
      do 1000   j = 1,ny
      do 1000   i = 1,nx
      ii = ii + 1
       rhomin = min( rhomin,ddd(1,i,j))
       rhomax = max( rhomax,ddd(1,i,j))
       pmin = min( pmin,ddd(2,i,j))
       pmax = max( pmax,ddd(2,i,j))
       uxmin = min( uxmin,ddd(3,i,j))
       uxmax = max( uxmax,ddd(3,i,j))
       uymin = min( uymin,ddd(4,i,j))
       uymax = max( uymax,ddd(4,i,j))
1000  continue
c
c   create scaling factors
c
      denom = (rhomax - rhomin)
      if( denom .ne. 0.0 ) denom = 1.0/denom
      rhofac = 255.0 * denom
      denom = (pmax - pmin)
      if( denom .ne. 0.0 ) denom = 1.0/denom
      pfac = 255.0 * denom
      denom = (uxmax - uxmin)
      if( denom .ne. 0.0 ) denom = 1.0/denom
      uxfac = 255.0 * denom
      denom = (uymax - uymin)
      if( denom .ne. 0.0 ) denom = 1.0/denom
      uyfac = 255.0 * denom
c
      irhooff = 0
      ipoff = nx
      iuxoff = 2*nx*ny
      iuyoff = 2*nx*ny + nx
      do j = 1,2*ny
         do i = 1,2*nx
            igray(i,j) = char(100)
         enddo
         enddo
      ii = 0
```

```
      do j = 1,ny
         do i = 1,nx
         ii = ii + 1
            rthyng = rhofac * (ddd(1,i,j) - rhomin)
            pthyng = pfac * (ddd(2,i,j) - pmin)
            uxthyng = uxfac * (ddd(3,i,j) - uxmin)
            uythyng = uyfac * (ddd(3,i,j) - uymin)
            igray(i,j) = char(int(rthyng))
            igray(i+nx,j) = char(int(pthyng))
            igray(i,j+ny) = char(int(uxthyng))
            igray(i+nx,j+ny) = char(int(uythyng))
         enddo
      enddo
c
c
c
c   Construct name of image file and store in NAMDMP:
c
      write (namdmp,1004)   acharc,  acharc2, numdmp
1004  format (a1,a1,'im',i4.4)
       print *,' namdmp=',namdmp
c

c
c   now write it to file namdmp
      iunit = 34
      nbytes = 4*nx*ny
      call ppm98_binopen_trunc (namdmp, iunit)
      call ppm98_binwrite (iunit, igray, nbytes)
      call ppm98_binclose (iunit)
c
      numdmp = numdmp + 1
c
      return
      end
```

The minimum information required to be able to restart this program is small. It consists of the values of the array of zone averages, `ddd` plus the values of a small number of parameters: `time` the program time, `ncycle` the number of cycles computed, and `numdmp` the number of image dumps created so far. One may, of course, save more than the minimum amount.

The revised program, `Driverdirect2dg0c_v01` incorporates the use of restart dumps. In this version, the user is first interrogated to determine whether or not a restart is desired. If not, the array of zone averages `ddd` is initialized as before. If a restart is desired, this initialization step is skipped and the values appropriate values of the necessary scalars and the zone averages are read from file `RESTART` by the subroutine `rs_read`. The program then continues much as before. Before program termination, a new restart dump is written by a call to `rs_write`.

```
      program Driverdirect2dg0c_v01
c
      parameter (NXZONES = 256)
      parameter (NYZONES = 256)
      parameter (EOSGAM = 1.4)
      parameter (DT = 0.001100)
      parameter (XMAX = 1.0)
      parameter (YMAX = 1.0)
      parameter (N2DUMP = 20)
      parameter (T2DUMP = 99999.)
c     Note that because of the fixed timestep, N, XMAX, and DT are not
c               independent.
c
c      NXZONES = number of real computational zones in the x-direction, set
c                by an m4 definition at beginning of code.
c      NYZONES = number of real computational zones in the y-direction, set
c                by an m4 definition at beginning of code.
c        NXZONES and NYZONES should be changed by changing the values of
c         256, 256, and 256 at beginning.
c      EOSGAM = the equation of state gamma
c      DT = the fixed timestep
c      XMAX = the righthand edge x-value of the grid.
c                                    The x-grid runs from (0,XMAX)
c      YMAX = the righthand edge y-value of the grid.
c                                    The y-grid runs from (0,YMAX)
c      N2DUMP is the frequency of image dumps in iterations
c      T2DUMP is the frequency of image dumps in program time.
c
c     Do not change this 1.:
      parameter (NBDY = 5)
c
c========================================================================
c
c     The constant  256  gives the maximum length of a row in the
c     x-direction which will be encountered.
c     The constant  256  gives the maximum length of a column in the
c     y-direction which will be encountered.
c
```

```
c=======================================================================
c
      dimension  xl( 1-NBDY:NXZONES+NBDY+1)
      dimension  yl( 1-NBDY:NYZONES+NBDY+1)
      dimension ddd( 1:4, 1-NBDY:NXZONES+NBDY, 1-NBDY:NYZONES+NBDY )
      character*1 achar
c
c     ddd    = the 3-D array into which the 5 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c
c      xl    = the locations of the lefthand zone interfaces
c      yl    = the locations of the bottom zone interfaces
c     ddd    = the 3-D array into which the 4 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
c     ddd(4,i,j) = uuy(i,j)
c     rrho   = the zone-averaged densities
c     pp     = the zone-averaged pressures
c     uux    = the zone-averaged x-velocities
c     uuy    = the zone-averaged y-velocities
c
c
c     smlrho = a trivial value of density, used to protect divides.
c     smalle = a trivial value of energy, used to protect divides.
c     smallp = a trivial value of pressure, used to protect divides.
c     smallu = a trivial value of velocity, used to protect divides.
c
      smlrho = 1.0e-06
      smalle = 1.0e-06
      smallp = 1.0e-06
      smallu = 1.0e-06
c
c
      ndump = N2DUMP
      tdump = T2DUMP

c          nstop is the maximum number of iterations to take for the entire
c             calculation
c          tstop is the maximum program time for the entire calculation, here
c             assuming a constant timestep DT
c          ndump is the frequency of image dumps in iteratiopns
c          tdump is the frequency of image dumps in program time.
c
c     initialize the grid:
      xlimit = XMAX
      ylimit = YMAX
c
      call setgrid(  xl,  yl, xlimit, ylimit, nxzones, nyzones, nbdy )
c

      print *,' Enter 1 if you intend to restart from file ''RESTART'''
      print *,' Enter anything else to start the problem with the initial'
      print *,' conditions.'
      read(5,100)  achar
100   format(a1)
      if (achar .eq. "1" ) then
         print *,' restarting'
          call rs_read (ddd, time, ncycle, numdmp, 4, nxzones, nyzones, nbdy )
          print *,' ncycle=',ncycle
          print *,' numdmp=',numdmp
      else

c         initialize counters
          ncycle = 0
          time = 0.0
        numdmp = 0
c
c         initialize the hydrodynamic quantities:
c
          call set2dshock( ddd, nxzones, nyzones, nbdy )
c
      endif
c
c   Request the number of timesteps to complete
        print *,' Please enter number of timesteps to perform.'
        read *,nstop
c
c   Adjust nstop so that it is a multiple of ndump.  This avoids waste.
c
      nstop = ndump * ( (nstop +ndump -1)/ndump )
      print *,' Thank you.  The program'
      print *,' will stop after ',nstop,' timesteps.'
c
c   add the number of completed timesteps ncycle
      nstop = nstop + ncycle
      tstop = float (nstop) * DT

c
c   CONTROL performs the entire hydrodynamic calculation.
c
      dtime = DT
      gamma = EOSGAM

      call cntrol ( xl,  yl, ddd, gamma, dtime, smlrho, smalle, smallp,
```

```
     ^              smallu, ncycle, numdmp, time, ndump, tdump, nstop, tstop,
     ^             nxzones, nyzones, nbdy)

c     write a restart dump before exiting.
      call rs_write (ddd, time, ncycle, numdmp, 4, nxzones, nyzones, nbdy )

      stop
      end
```

The subroutine `rs_write` opens a file `RESTART` and assigns it to FORTRAN unit 34. (This specific unit number is not significant, only that it not conflict with any other open unit numbers.) The three scalers are then written followed by the array of zone averages. This order is unimportant, but the user must take care to ensure that when reading from the restart dump, the quantities are read in the same order in which they were written, as illustrated by the subroutine `rs_read`. In `rs_read`, the file `RESTART` is opened. The values of the scalers are read followed by the values of the zone averages. The file is closed and program direction returns to the calling routine.

```
      subroutine rs_write (ddd, time, ncycle, numdmp, nv, nx, ny, nbdy )
c
c    This subroutine writes a minimal restart dump.  A restart dump contains
c       all the information necessary to restart the program from the point
c       at which the restart dump was written.
c
      dimension ddd( 1:NV, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c     ddd    = the 3-D array into which the NV principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c              These are, at a minimum:
c     ddd(1,i,j) = rrho(i,j) = the zone-averaged densities.
c     ddd(2,i,j) = pp(i,j)   = the zone-averaged pressures.
c     ddd(3,i,j) = uux(i,j)  = the zone-averaged x-velocities.
c     ddd(4,i,j) = uuy(i,j)  = the zone-averaged y-velocities.
c
c
c     numdmp = the sequential number of the nextimage 0->whatever, numdmp
c      NV = number of different zone averaged quantities in DDD
c      nx = number of real computational zones in the x-direction, set
c               by an m4 definition at beginning of code.
c      ny = number of real computational zones in the y-direction, set
c               by an m4 definition at beginning of code.
c
c      nbdy = the number fake zones necessary for boundary conditions.
c      ncycle = the number of completed timesteps
c      time = the program time.
c
c      Open the file "RESTART"
      iunit = 34
      call ppm98_binopen_trunc ("RESTART", iunit)
c
c      write out the scalars first
c
      nbytes = 4
      call ppm98_binwrite (iunit, ncycle, nbytes)
      call ppm98_binwrite (iunit, numdmp, nbytes)
      call ppm98_binwrite (iunit, time, nbytes)
c
c     now write out ddd
c
      nbytes = 4 * nv * (nx+2*nbdy)* (ny+2*nbdy)
      call ppm98_binwrite (iunit, ddd, nbytes)
      call ppm98_binclose (iunit)
      return
      end
c
      subroutine rs_read (ddd, time, ncycle, numdmp, nv, nx, ny, nbdy )
c
c    This subroutine reads a minimal restart dump.  A restart dump contains
c       all the information necessary to restart the program from the point
c       at which the restart dump was written.
c
      dimension ddd( 1:NV, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c     ddd    = the 3-D array into which the NV principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c              These are, at a minimum:
c     ddd(1,i,j) = rrho(i,j) = the zone-averaged densities.
c     ddd(2,i,j) = pp(i,j)   = the zone-averaged pressures.
c     ddd(3,i,j) = uux(i,j)  = the zone-averaged x-velocities.
c     ddd(4,i,j) = uuy(i,j)  = the zone-averaged y-velocities.
c
c
c      NV = number of different zone averaged quantities in DDD
c      nx = number of real computational zones in the x-direction, set
c               by an m4 definition at beginning of code.
```

```
c      ny = number of real computational zones in the y-direction, set
c               by an m4 definition at beginning of code.
c
c      nbdy = the number fake zones necessary for boundary conditions.
c     numdmp = the sequential number of the nextimage 0->whatever, numdmp
c      ncycle = the number of completed timesteps
c      time = the program time.
c
c      Open the file "RESTART"
      iunit = 34
      call ppm98_binopen ("RESTART", iunit)
c
c      read the scalars first
c
      nbytes = 4
      call ppm98_binread (iunit, ncycle, nbytes)
      call ppm98_binread (iunit, numdmp, nbytes)
      call ppm98_binread (iunit, time, nbytes)
c
c     now read ddd
c
      nbytes = 4 * nv * (nx+2*nbdy)* (ny+2*nbdy)
      call ppm98_binread (iunit, ddd, nbytes)
      call ppm98_binclose (iunit)
      return
      end
```

An obvious weakness is that if the data are read in a different order from that in which they were writted, the calculation cannot resume properly. At this point, there is no means which allow the inclusion of commentary. In the following example, this weakness is ameliorated to some degree by additional PPM-LIB routines for writing and reading restart dumps.

```
      program Driverdirect2dg0c_v02
c
      parameter (NXZONES = 256)
      parameter (NYZONES = 256)
      parameter (EOSGAM = 1.4)
      parameter (DT = 0.001100)
      parameter (XMAX = 1.0)
      parameter (YMAX = 1.0)
      parameter (N2DUMP = 20)
      parameter (T2DUMP = 99999.)
c     Note that because of the fixed timestep, N, XMAX, and DT are not
c              independent.
c
c      NXZONES = number of real computational zones in the x-direction, set
c               by an m4 definition at beginning of code.
c      NYZONES = number of real computational zones in the y-direction, set
c               by an m4 definition at beginning of code.
c        NXZONES and NYZONES should be changed by changing the values of
c         256, 256, and 256 at beginning.
c      EOSGAM = the equation of state gamma
c      DT = the fixed timestep
c      XMAX = the righthand edge x-value of the grid.
c                                   The x-grid runs from (0,XMAX)
c      YMAX = the righthand edge y-value of the grid.
c                                   The y-grid runs from (0,YMAX)
c      N2DUMP is the frequency of image dumps in iterations
c      T2DUMP is the frequency of image dumps in program time.
c
c     Do not change this 1.:
      parameter (NBDY = 5)
c
c=======================================================================
c
c     The constant  256  gives the maximum length of a row in the
c     x-direction which will be encountered.
c     The constant  256  gives the maximum length of a column in the
c     y-direction which will be encountered.
c
c
c=======================================================================

      dimension  xl( 1-NBDY:NXZONES+NBDY+1)
      dimension  yl( 1-NBDY:NYZONES+NBDY+1)
      dimension ddd( 1:4, 1-NBDY:NXZONES+NBDY, 1-NBDY:NYZONES+NBDY )
      character*1 achar
c
c     ddd    = the 3-D array into which the 5 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c
c      xl    = the locations of the lefthand zone interfaces
c      yl    = the locations of the bottom zone interfaces
c     ddd    = the 3-D array into which the 4 principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c     ddd(1,i,j) = rrho(i,j)
c     ddd(2,i,j) = pp(i,j)
c     ddd(3,i,j) = uux(i,j)
```

```
c     ddd(4,i,j) = uuy(i,j)
c     rrho   = the zone-averaged densities
c     pp     = the zone-averaged pressures
c     uux    = the zone-averaged x-velocities
c     uuy    = the zone-averaged y-velocities
c
c
c     smlrho = a trivial value of density, used to protect divides.
c     smalle = a trivial value of energy, used to protect divides.
c     smallp = a trivial value of pressure, used to protect divides.
c     smallu = a trivial value of velocity, used to protect divides.
c
      smlrho = 1.0e-06
      smalle = 1.0e-06
      smallp = 1.0e-06
      smallu = 1.0e-06
c
c
      ndump = N2DUMP
      tdump = T2DUMP

c          nstop is the maximum number of iterations to take for the entire
c             calculation
c          tstop is the maximum program time for the entire calculation, here
c             assuming a constant timestep DT
c          ndump is the frequency of image dumps in iteratiopns
c          tdump is the frequency of image dumps in program time.
c
c     initialize the grid:
      xlimit = XMAX
      ylimit = YMAX
c
      call setgrid(  xl,  yl, xlimit, ylimit, nxzones, nyzones, nbdy )
c

      print *,' Enter 1 if you intend to restart from file ''RESTART'''
      print *,' Enter anything else to start the problem with the initial'
      print *,' conditions.'
      read(5,100)  achar
100   format(a1)
      if (achar .eq. "1" ) then
         print *,' restarting'
          call rs_read (ddd, time, ncycle, numdmp, 4, nxzones, nyzones, nbdy )
          print *,' ncycle=',ncycle
          print *,' numdmp=',numdmp
      else
c         initialize counters
          ncycle = 0
          time = 0.0
       numdmp = 0
c
c         initialize the hydrodynamic quantities:
c
          call set2dshock( ddd, nxzones, nyzones, nbdy )
c
      endif
c
c   Request the number of timesteps to complete
       print *,' Please enter number of timesteps to perform.'
       read *,nstop
c
c   Adjust nstop so that it is a multiple of ndump.  This avoids waste.
c
      nstop = ndump * ( (nstop +ndump -1)/ndump )
      print *,' Thank you.  The program'
      print *,' will stop after ',nstop,' timesteps.'
c
c   add the number of completed timesteps ncycle
      nstop = nstop + ncycle
      tstop = float (nstop) * DT

c
c   CONTROL performs the entire hydrodynamic calculation.
c
      dtime = DT
      gamma = EOSGAM

      call cntrol ( xl,  yl, ddd, gamma, dtime, smlrho, smalle, smallp,
     ^             smallu, ncycle, numdmp, time, ndump, tdump, nstop, tstop,
     ^            nxzones, nyzones, nbdy)


c     write a restart dump before exiting.
      call rs_write ("Driverdirect2dg0c_v02", ddd, time, ncycle, numdmp, 4,
     ^              nxzones, nyzones, nbdy )


      stop
      end
```

The call to `rs_write` has been modified to include the name of the program to illustrate the addition of comments to a restart dump.

```
      subroutine rs_write (progrm, ddd, time, ncycle, numdmp, nv,
     ^                    nx, ny, nbdy )
c
c   This subroutine creats a minimal restart dump,  which contains
c       all the information necessary to restart the program from the point
c       at which the restart dump was written.
c
c      progrm = is the name of the program

      character*(*) progrm
      dimension ddd( 1:NV, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c      ddd    = the 3-D array into which the NV principal 2-D arrays of
c               basic physical variables are interleaved in order to
c               optimize memory bandwidth on microprocessor systems.
c               These are, at a minimum:
c      ddd(1,i,j) = rrho(i,j) = the zone-averaged densities.
c      ddd(2,i,j) = pp(i,j)   = the zone-averaged pressures.
c      ddd(3,i,j) = uux(i,j)  = the zone-averaged x-velocities.
c      ddd(4,i,j) = uuy(i,j)  = the zone-averaged y-velocities.
c
c
c      numdmp = the sequential number of the nextimage 0->whatever, numdmp
c       NV = number of different zone averaged quantities in DDD
c       nx = number of real computational zones in the x-direction, set
c                by an m4 definition at beginning of code.
c       ny = number of real computational zones in the y-direction, set
c                by an m4 definition at beginning of code.
c
c       nbdy = the number fake zones necessary for boundary conditions.
c       ncycle = the number of completed timesteps
c       time = the program time.
c
       character*80 comment
c
c       Open the file "RESTART"
      iunit = 34
      call ppm98_binopen_trunc ("RESTART", iunit)

c       construct a comment which includes the program name
      comment = "This was created by " // progrm
      print *, comment
      call ppm98_write_comment (comment, iunit )
c      the time of creation
      call ppm98_timestamp ( iunit )
c
c       write out the scalars first
c
      call ppm98_rsi4write_vector0d ("NCYCLE: number of iterations done",
     6                               ncycle, iunit)
      call ppm98_rsi4write_vector0d ("NUMDMP: number of image files",
     6                               numdmp, iunit)
      call ppm98_rsr4write_vector0d ("TIME: program time",
     6                               time, iunit)
c
c      now write out ddd
      call ppm98_rsr4write_vector3d ("DDD: zone averages",
     6     0, nv, 0, nbdy, nx, nbdy, nbdy, ny,nbdy,  ddd, iunit)
c
c
c   Save the PPMLIB internal constants which control interpolation
c    and diffusion, etc.
c
      call ppm98_save_constants(iunit)

      call ppm98_binclose (iunit)
      return
      end
```

As before, the subroutine `rs_write` opens the file “RESTART.” Next however, an ASCII character string is constructed which includes the name of the driver as passed to this subroutine by the calling routine. The full line included in the restart dump reads

```
This was created by Driverdirect2dg0c_v02
```

Next, the date and time of hte file creation is recored as a comment in the restart dump by a call to **ppm98_timestamp**. Note, this time cannot be any more accurate that the system clock. Next the integer scalers `ncycle` and `numdmp` are written by calls to **ppm98_rsi4write_vector0d**. (The “rsi4” portion of the subroutine name indicates that is is for restart dumps and pertains to FORTRAN type `integer*4`. The “0d” indicates a “0”-dimensional vector.) The real scaler `time` are written by call to **ppm98_rsr4write_vector0d**. (The “rsr4” portion of the subroutine name indicates that is is for restart dumps and pertains to FORTRAN type `real*4`.) In addition to including to the value of the scaler, these routines allow the user to include a short description of the quantity. Likewise, the array of zone averages ddd is written by a call to **ppm98_rsr4write_vector3d**, which is the appropriate routine for a 3-dimensional vector of FORTRAN type real*4. Before closing the file, a call is made to the subroutine ppm98_save_constants. This call is unique to the interpolation and hydrodynamics routine of PPMLIB. The behaviour of these routines routine are controlled by a small number of internal constants, which are user settable to some degree. A call to this subroutine adds the values of these internal constants to the restart dump. This is recommended even if the user uses the default values a safe programming practice.

To read the restart dump, the comments (the ‘‘timestamp’’ is considered a comment) are read by calls to **ppm98_read_comment** which will print the comment to standard out. Then the data are read with the corresponding PPMLIB routines to read from restart dumps. As each quantity is read, the description stored with it is printed to stand out.

```
      subroutine rs_read (ddd, time, ncycle, numdmp, nv, nx, ny, nbdy )
c
c   This subroutine reads a minimal restart dump.  A restart dump contains
c      all the information necessary to restart the program from the point
c      at which the restart dump was written.
c
      dimension ddd( 1:NV, 1-NBDY:NX+NBDY, 1-NBDY:NY+NBDY )
c
c     ddd    = the 3-D array into which the NV principal 2-D arrays of
c              basic physical variables are interleaved in order to
c              optimize memory bandwidth on microprocessor systems.
c              These are, at a minimum:
c     ddd(1,i,j) = rrho(i,j) = the zone-averaged densities.
c     ddd(2,i,j) = pp(i,j)   = the zone-averaged pressures.
c     ddd(3,i,j) = uux(i,j)  = the zone-averaged x-velocities.
c     ddd(4,i,j) = uuy(i,j)  = the zone-averaged y-velocities.
c
c
c      NV = number of different zone averaged quantities in DDD
c      nx = number of real computational zones in the x-direction, set
c                by an m4 definition at beginning of code.
c      ny = number of real computational zones in the y-direction, set
c                by an m4 definition at beginning of code.
c
c      nbdy = the number fake zones necessary for boundary conditions.
c     numdmp = the sequential number of the nextimage 0->whatever, numdmp
c      ncycle = the number of completed timesteps
c      time = the program time.
      character*80 comment
c
c      Open the file "RESTART"
      iunit = 34
      call ppm98_binopen ("RESTART", iunit)
c
c      read the comments
      call ppm98_read_comment (comment, iunit )
      call ppm98_read_comment (comment, iunit )
c
c      read the scalars first
      call ppm98_rsi4read_vector0d ( ncycle, iunit)
      call ppm98_rsi4read_vector0d ( numdmp, iunit)
      call ppm98_rsr4read_vector0d ( time, iunit)

c     now read ddd
c
      call ppm98_rsr4read_vector3d ( 0, nv, 0, nbdy, nx, nbdy,
     ^         nbdy, ny,nbdy,  ddd, iunit)

      call ppm98_read_constants(iunit)

      call ppm98_binclose (iunit)
      return
      end
```

From these examples, it is obvious that the reading and writing of restart daumps must be customized for each program. Only the user can know in advance the necessary quantities to be stored. It should be noted however, that the comments which appear as ASCII text at the beginning of each restart are easily readable by the standard UNIX utilities ‘‘cat’’ or ‘‘more.’’ These comments, of which the user may include an unlimited number, may be used to provide a detailed listing of the contents of the restart dump. In principle, even the source code for the driver could be included at this stage to ensure that the user would be able to access the restart dump contents in the future.

b. Restart Dump Summary

This section concludes with a listing of the names of available PPMLIB routines related to restart dumps, with a brief description of their function.

I. Routines to open and close files:

PPM98_BINCLOSE Closes a previously opened file on disk

PPM98_BINOPEN Opens a file on disk for reading or writing

PPM98_BINOPEN_TRUNC Opens a file on disk for writing. If a file of this name already exists, its contents are destroyed.

II. Routines to read:

PPM98_BINREAD Reads a specified number of bytes from disk from a file without record marks.

PPM98_BINWRITE Writes a specified number of bytes to disk without record marks.

PPM98_RSI2READ_VECTOR0D Reads an INTEGER*2 scalar and displays the stored comment (if any.)

PPM98_RSI2READ_VECTOR1D Reads an INTEGER*2 1-D vector and displays the stored comment (if any.)

PPM98_RSI2READ_VECTOR2D Reads an INTEGER*2 2-D vector and displays the stored comment (if any.)

PPM98_RSI2READ_VECTOR3D Reads an INTEGER*2 3-D vector and displays the stored comment (if any.)

PPM98_RSI2READ_VECTOR4D Reads an INTEGER*2 4-D vector and displays the stored comment (if any.)

PPM98_RSI4READ_VECTOR0D Reads an INTEGER*4 scalar and displays the stored comment (if any.)

PPM98_RSI4READ_VECTOR1D Reads an INTEGER*4 1-D vector and displays the stored comment (if any.)

PPM98_RSI4READ_VECTOR2D Reads an INTEGER*4 2-D vector and displays the stored comment (if any.)

PPM98_RSI4READ_VECTOR3D Reads an INTEGER*4 3-D vector and displays the stored comment (if any.)

PPM98_RSI4READ_VECTOR4D Reads an INTEGER*4 4-D vector and displays the stored comment (if any.)

PPM98_RSI8READ_VECTOR0D Reads an INTEGER*8 scalar and displays the stored comment (if any.)

PPM98_RSI8READ_VECTOR1D Reads an INTEGER*8 1-D vector and displays the stored comment (if any.)

PPM98_RSI8READ_VECTOR2D Reads an INTEGER*8 2-D vector and displays the stored comment (if any.)

PPM98_RSI8READ_VECTOR3D Reads an INTEGER*8 3-D vector and displays the stored comment (if any.)

PPM98_RSI8READ_VECTOR4D Reads an INTEGER*8 4-D vector and displays the stored comment (if any.)

PPM98_RSR4READ_VECTOR0D Reads a REAL*4 scalar and displays the stored comment (if any.)

PPM98_RSR4READ_VECTOR1D Reads a REAL*4 1-D vector and displays the stored comment (if any.)

PPM98_RSR4READ_VECTOR2D Reads a REAL*4 2-D vector and displays the stored comment (if any.)

PPM98_RSR4READ_VECTOR3D Reads a REAL*4 3-D vector and displays the stored comment (if any.)

PPM98_RSR4READ_VECTOR4D Reads a REAL*4 4-D vector and displays the stored comment (if any.)

PPM98_RSR8READ_VECTOR0D Reads a REAL*8 scalar and displays the stored comment (if any.)

PPM98_RSR8READ_VECTOR1D Reads a REAL*8 1-D vector and displays the stored comment (if any.)

PPM98_RSR8READ_VECTOR2D Reads a REAL*8 2-D vector and displays the stored comment (if any.)

PPM98_RSR8READ_VECTOR3D Reads a REAL*8 3-D vector and displays the stored comment (if any.)

PPM98_RSR8READ_VECTOR4D Reads a REAL*8 4-D vector and displays the stored comment (if any.)

PPM98_RSR16READ_VECTOR0D Reads a REAL*16 scalar and displays the stored comment (if any.)

PPM98_RSR16READ_VECTOR1D Reads a REAL*16 1-D vector and displays the stored comment (if any.)

PPM98_RSR16READ_VECTOR2D Reads a REAL*16 2-D vector and displays the stored comment (if any.)

PPM98_RSR16READ_VECTOR3D Reads a REAL*16 3-D vector and displays the stored comment (if any.)

PPM98_RSR16READ_VECTOR4D Reads a REAL*16 4-D vector and displays the stored comment (if any.)

PPM98_READ_CONSTANTS Reads the values of the PPMLIB internal constants from the file.

III. Routines to write:

PPM98_BINWRITE Writes a specified number of bytes to disk without record marks.

PPM98_WRITE_COMMENT Writes a line of text to disk as a comment.

PPM98_WRITE_TIMESTAMP Records in the restart dump the date and time of file creation as a comment.

PPM98_RSI2WRITE_VECTOR0D Writes an INTEGER*2 scalar and a comment (if desired.)

PPM98_RSI2WRITE_VECTOR1D Writes an INTEGER*2 1-D vector and and a comment (if desired.)

PPM98_RSI2WRITE_VECTOR2D Writes an INTEGER*2 2-D vector and and a comment (if desired.)

PPM98_RSI2WRITE_VECTOR3D Writes an INTEGER*2 3-D vector and and a comment (if desired.)

PPM98_RSI2WRITE_VECTOR4D Writes an INTEGER*2 4-D vector and and a comment (if desired.)

PPM98_RSI4WRITE_VECTOR0D Writes an INTEGER*4 scalar and a comment (if desired.)

PPM98_RSI4WRITE_VECTOR1D Writes an INTEGER*4 1-D vector and and a comment (if desired.)

PPM98_RSI4WRITE_VECTOR2D Writes an INTEGER*4 2-D vector and and a comment (if desired.)

PPM98_RSI4WRITE_VECTOR3D Writes an INTEGER*4 3-D vector and and a comment (if desired.)

PPM98_RSI4WRITE_VECTOR4D Writes an INTEGER*4 4-D vector and and a comment (if desired.)

PPM98_RSI8WRITE_VECTOR0D Writes an INTEGER*8 scalar and a comment (if desired.)

PPM98_RSI8WRITE_VECTOR1D Writes an INTEGER*8 1-D vector and and a comment (if desired.)

PPM98_RSI8WRITE_VECTOR2D Writes an INTEGER*8 2-D vector and and a comment (if desired.)

PPM98_RSI8WRITE_VECTOR3D Writes an INTEGER*8 3-D vector and and a comment (if desired.)

PPM98_RSI8WRITE_VECTOR4D Writes an INTEGER*8 4-D vector and and a comment (if desired.)

PPM98_RSR4WRITE_VECTOR0D Writes a REAL*4 scalar and a comment (if desired.)

PPM98_RSR4WRITE_VECTOR1D Writes a REAL*4 1-D vector and a comment (if desired.)

PPM98_RSR4WRITE_VECTOR2D Writes a REAL*4 2-D vector and a comment (if desired.)

PPM98_RSR4WRITE_VECTOR3D Writes a REAL*4 3-D vector and a comment (if desired.)

PPM98_RSR4WRITE_VECTOR4D Writes a REAL*4 4-D vector and a comment (if desired.)

PPM98_RSR8WRITE_VECTOR0D Writes a REAL*8 scalar and a comment (if desired.)

PPM98_RSR8WRITE_VECTOR1D Writes a REAL*8 1-D vector and a comment (if desired.)

PPM98_RSR8WRITE_VECTOR2D Writes a REAL*8 2-D vector and a comment (if desired.)

PPM98_RSR8WRITE_VECTOR3D Writes a REAL*8 3-D vector and a comment (if desired.)

PPM98_RSR8WRITE_VECTOR4D Writes a REAL*8 4-D vector and a comment (if desired.)

PPM98_RSR16WRITE_VECTOR0D Writes a REAL*16 scalar and a comment (if desired.)

PPM98_RSR16WRITE_VECTOR1D Writes a REAL*16 1-D vector and a comment (if desired.)

PPM98_RSR16WRITE_VECTOR2D Writes a REAL*16 2-D vector and a comment (if desired.)

PPM98_RSR16WRITE_VECTOR3D Writes a REAL*16 3-D vector and a comment (if desired.)

PPM98_RSR16WRITE_VECTOR4D Writes a REAL*16 4-D vector and a comment (if desired.)

PPM98_STORE_CONSTANTS Writes the values of the PPMLIB internal constants to the file.