# APPENDIX E: User Defined Solution Control

You can control the I/O, monitor the energies and other solution norms of interest, and shut down the problem whenever you please with user subroutine `uctrl1` in dyn21.F. The arguments are defined in the listing provided below. This subroutine is called each time step and does not need any control card to operate.

```
      subroutine uctrl1 (numnp,ndof,time,dt1,dt2,prtc,pltc,frci,prto,
     . plto,frco,vt,vr,at,ar,ut,ur,xmst,xmsr,irbody,rbdyn,usrhv,
     . messag,totalm,cycle,idrint,mtype,lrb,nrba,rbcor,x,rbv,nrbn,
     . nrb,xrb,yrb,zrb,axrb,ayrb,azrb,dtx,nmmat,rba,fvalnew,fvalold,
     . fvalmid,fvalnxt)
c
c*************************************************************
c|  Livermore Software Technology Corporation  (LSTC)         |
c|  ------------------------------------------------------    |
c|  Copyright 1987-2008 Livermore Software Tech.  Corp        |
c|  All rights reserved                                       |
c*************************************************************
c
c     user subroutine for solution control.  It is called at the
c     beginning of time step n+1.
c
c
c     note:  ls-dyna uses an internal numbering system to
c            accommodate arbitrary node numbering.   to access
c            information for user node n, address array location m,
c            m = lqf8(n,1).  to obtain user node number, n,
c            corresponding to array address m, set n = lqfinv(m,1)
c
c     arguments:
c          numnp = number of nodal points
c          ndof = number of degrees of freedom per node
c          time = current solution time at n+1
c          dt1  = time step size between time n-1 and n
c          dt2  = time step size between time n and n+1
c          prtc = output interval for taurus time history data
c          pltc = output interval for taurus state data
c          frci = output interval for taurus interface force data
c          prto = output time for time history file
c          plto = output time for state data
c          frco = output time for force data
c          vt(3,numnp) = nodal translational velocity vector
c          vr(3,numnp) = nodal rotational velocity vector.  this
c                        array is defined if and only if ndof = 6
c          at(3,numnp) = nodal translational acceleration vector
c          ar(3,numnp) = nodal rotational acceleration vector.  this
c                        array is defined if and only if ndof = 6
c          ut(3,numnp) = nodal translational displacement vector
c          ur(3,numnp) = nodal rotational displacement vector.  this
c                        array is defined if and only if ndof = 6
c          xmst(numnp) = reciprocal of nodal translational masses
c          xmsr(numnp) = reciprocal of nodal rotational masses.  this
c                        array is defined if and only if ndof = 6
```

```
c          fvalold     = array for storing load curve values at time n
c          fvalnew     = array for setting load curve values at time n+1
c                         only load curves with 0 input points may be user
c                         defined.  When the load curve is user set, the
c                         value at time n must be stored in array fvalold.
c          fvalmid     = array for predicting load curve values at time n+3/2
c          fvalnxt     = array for predicting load curve values at time n+2
c                         for some applications it is necessary to predict the
c                        load curve values at time n+2, a time that is not known,
c                         this for instance for boundary prescribed motion.  In
c                         this case the load curve values at time n+3/2 need to
c                         be predicted in fvalmid, and fvalold should be set to
c                         fvalnew and fvalnew should be set to fvalnxt.  See
c                         coding below.
c
c          irbody      = 0 if no rigid bodies
c          rbdyn(numnp)=flag for rigid body nodal points
c                         if deformable node then set to 1.0
c                         if rigid body node then set to 0.0
c                         defined if and only if rigid bodies are present
c                         i.e., irbody.ne.0 if no rigid bodies are
c                         present
c          usrhv(lenhv)=user defined history variables that are stored
c                         in the restart file.  lenhv = 100+7*nummat where
c                         nummat is the # of materials in the problem.
c                         array usrhv is updated only in this subroutine.
c          messag      = flag for dyna3d which may be set to:
c                        'sw1.' ls-dyna3d terminates with restart file
c                        'sw3.' ls-dyna3d writes a restart file
c                        'sw4.' ls-dyna3d writes a plot state
c           totalm     = total mass in problem
c           cycle      = cycle number
c           idrint     = flag for dynamic relaxation phase
c                         .ne.0: dynamic relaxation in progress
c                         .eq.0: solution phase
c          mtype(*)    = material type for each part in the model
c          lrb(*)      = lead rigid body for each rigid body
c                        If part n is rigid and is a lead or has not
c                        been merged then lrb(n)==n.
c          nrba(*)     = starting index in nrb(*) of the nodes for each
c                        rigid body
c          rbcor(3,*)  = rigid body cg coordinates
c          x(3,*)      = Node coordinate array
c          rbv(6,*)    = rigid body cg velocity
c          nrbn(*)     = # nodes in each rigid body
c          nrb(*)      = List of all rigid body nodes
c          xrb(*)      = RB scratch array as long as longest nrbn() value
c          yrb(*)      = RB scratch array as long as longest nrbn() value
c          zrb(*)      = RB scratch array as long as longest nrbn() value
c          axrb(*)     = RB scratch array as long as longest nrbn() value
c          ayrb(*)     = RB scratch array as long as longest nrbn() value
c          azrb(*)     = RB scratch array as long as longest nrbn() value
c          dtx         = (dt1+dt2)*0.5 except at time 0 when it is = dt2
c          nmmat       = number of parts in the model
c          rba(6,*)    = rigid body cg acceleration
c          fvalnew     = array for setting load curve values at time n+1
c                        only load curves with 0 input points may be user
c                        defined.  When the load curve is user set, the
c                        value at time n must be stored in array fvalold.
c          fvalold     = array for storing load curve values at time n
```

```
c          fvalmid     = array for predicting load curve values at time n+3/2
c          fvalnxt     = array for predicting load curve values at time n+2
c                         for some applications it is necessary to predict the
c                        load curve values at time n+2, a time that is not known,
c                         this for instance for boundary prescribed motion.  In
c                         this case the load curve values at time n+3/2 need to
c                         be predicted in fvalmid, and fvalold should be set to
c                         fvalnew and fvalnew should be set to fvalnxt.  See
c                         coding below.
c
      include 'ptimes.inc'
c
c     prtims(1-37)=output intervals for ascii files
c
c      ascii files:
c           ( 1)-cross section forces
c           ( 2)-rigid wall forces
c           ( 3)-nodal data
c           ( 4)-element data
c           ( 5)-global data
c           ( 6)-discrete elements
c           ( 7)-material energies
c           ( 8)-noda interface forces
c           ( 9)-resultant interface forces
c           (10)-smug animator
c           (11)-spc reaction forces
c           (12)-nodal constraint resultant forces
c           (13)-airbag statistics
c           (14)-avs database
c           (15)-nodal force groups
c           (16)-output intervals for nodal boundary conditions
c           (17)-(32) unused at this time
c           (37)-auto tiebreak damage output
c
c     prtlst(32)=output times for ascii files above.  when solution time
c                exceeds the output time a print state is dumped.
c
      common/rbkeng/enrbdy,rbdyx,rbdyy,rbdyz
c
c      total rigid body energies and momentums:
c           enrbdy = rigid body kinetic energy
c           rbdyx = rigid body x-momentum
c           rbdyy = rigid body y-momentum
c           rbdyz = rigid body z-momentum
c
      common/swmke/swxmom,swymom,swzmom,swkeng
c
c      total stonewall energies and momentums:
c           swxmom = stonewall x-momentum
c           swymom = stonewall y-momentum
c           swzmom = stonewall z-momentum
c           swkeng = stonewall kinetic energy
c
      common/deengs/deeng
c
c      deeng = total discrete element energy
c
      common/bk28/summss,xke,xpe,tt,xte0,erodeke,erodeie,selie,selke,
     . erodehg
c
```

```
c      xpe  = total internal energy in the finite elements
c
       common/sprengs/spreng
c
c      spreng = total spr energy
c
       character*(*) messag
       integer cycle
       real*8 x
       dimension vt(3,*),vr(3,*),at(3,*),ar(3,*),
      . xmst(*),xmsr(*),rbdyn(*),usrhv(*),mtype(*),lrb(*),nrba(*),
      . rbcor(3,1),x(*),rbv(6,*),nrbn(*),nrb(*),xrb(*),yrb(*),
      . zrb(*),axrb(*),ayrb(*),azrb(*),rba(6,*),fvalnew(*),
      . fvalold(*),fvalmid(*),fvalnxt(*)
       real*8 ut(3,*),ur(3,*)
c
c      sample momentum and kinetic energy calculations
c
c      remove all comments in column 1 below to activate
       i = 1
       if (i.eq.1) return
c      return
cc
cc
cc     initialize kinetic energy, xke, and x,y,z momentums.
cc
c      xke = 2.*swkeng+2.*enrbdy
c      xm = swxmom+rbdyx
c      ym = swymom+rbdyy
c      zm = swzmom+rbdyz
cc
c      numnp2 = numnp
c      if (ndof.eq.6) then
c        numnp2 = numnp+numnp
c      endif
c      write(iotty,*)ndof
cc
cc
cc     no rigid bodies present
cc
c      if (irbody.eq.0) then
cc       note in blank comment vr follows vt.  this fact is used below.
c        do 10 n = 1,numnp2
c        xmsn = 1./xmst(n)
c        vn1 = vt(1,n)
c        vn2 = vt(2,n)
c        vn3 = vt(3,n)
c        xm = xm+xmsn*vn1
c        ym = ym+xmsn*vn2
c        zm = zm+xmsn*vn3
c        xke = xke+xmsn*(vn1*vn1+vn2*vn2+vn3*vn3)
c   10   continue
cc
cc
cc     rigid bodies present
cc
c      else
cc       nodal accerations for rigid bodies
cc
c        do 12 n = 1,nmmat
```

```
c        if (mtype(n).ne.20.or.lrb(n).ne.n) go to 12
c        lrbn = nrba(n)
c        call stvlut(rbcor(1,n),x,vt,at,ar,vr,rbv(1,n),dt2,
c    .   nrbn(n),nrb(lrbn),xrb,yrb,zrb,axrb,ayrb,azrb,dtx)
c
c        rigid body nodal accelerations
c
c        if (ndof.eq.6) then
c          call rbnacc(nrbn(n),nrb(lrbn),rba(4,n),ar)
c        endif
c
c  12    continue
cc
c        do 20 n = 1,numnp
c        xmsn = 1./xmst(n)
c        vn1 = rbdyn(n)*vt(1,n)
c        vn2 = rbdyn(n)*vt(2,n)
c        vn3 = rbdyn(n)*vt(3,n)
c        xm = xm+xmsn*vn1
c        ym = ym+xmsn*vn2
c        zm = zm+xmsn*vn3
c        xke = xke+xmsn*(vn1*vn1+vn2*vn2+vn3*vn3)
c   20   continue
c        if (ndof.eq.6) then
c          do 30 n = 1,numnp
c          xmsn = 1./xmsr(n)
c          vn1 = rbdyn(n)*vr(1,n)
c          vn2 = rbdyn(n)*vr(2,n)
c          vn3 = rbdyn(n)*vr(3,n)
c          xm = xm+xmsn*vn1
c          ym = ym+xmsn*vn2
c          zm = zm+xmsn*vn3
c          xke = xke+xmsn*(vn1*vn1+vn2*vn2+vn3*vn3)
c   30     continue
c        endif
c      endif
cc
cc     total kinetic energy
c      xke=.5*xke
cc     total internal energy
c      xie = xpe+deeng+spreng
cc     total energy
c      xte = xke+xpe+deeng+spreng
cc     total x-rigid body velocity
c      xrbv = xm/totalm
cc     total y-rigid body velocity
c      yrbv = ym/totalm
cc     total z-rigid body velocity
c      zrbv = zm/totalm
       return
       end
```