Appendix 1. FORTRAN Code for StreamVOC Model

```
program streamvoc
  use msflib
  use dflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  character*255 aboutmessage
  integer i4, i
  logical lret
  external parfilin, Enviro_Params, CalcFlowProfile
!
!  Initialise Winteracter and open the main window.
!  This will also activate the main menu for the program.
!
  call WInitialise(' ')
  wc%flags  = SysMenuOn + MinButton + MaxButton
  wc%x      = -1
  wc%y      = -1
  wc%width  = 0
  wc%height = 0
  wc%menuid = IDR_MENU1
  wc%title  = 'StreamVOC - VOC Source Apportionment in Streams and Rivers'
  call WindowOpen(wc)
  call WBitmapGet(MainWinBitmap,0)
  aboutmessage = 'StreamVOC\nVersion 2.20:  April 10, 2004\n&
Includes gas exchange, mixing, biochemical degradation\n&
Gas exchange at weirs/sills [Gulliver et al. 1998]\n&
Component flux method for SA calculation\n\n&
William Asher\nOHSU-OGIST-EBS\nBeaverton, Oregon'C    !217 characters 4/12/04
!* This is the main loop of the program.  It does nothing but
!* cycle endlessly, allowing the menus to be used.
  do
    call WMessage(itype, mainmessage)
!   Branch depending on type of message.
    select case (itype)
!   Menu Selection, activate dialog, bring to front if already active.
      case (MenuSelect)
        select case (mainmessage%value1)
          case (IDD_LoadParamFile)
            call ParFilIn(lret)
            if (ParamFileOK) call CalcFlowProfile
            if (ProfileWindowOpen) then
              focusprofile = 2
              call ViewRiverProfile(lret)
            endif
          case (IDD_SaveResults)
            call DatFilOut(lret)
          case (IDD_ViewVOCParams)
            call VOC_Params(lret)
          case (IDD_EnviroParams)
             call Enviro_Params(lret)
          case (IDD_HydrogParams)
             call Hydrog_Params(lret)
          case (IDD_RuntimeParameters)
             call Model_Params(lret)
          case (IDD_ViewProfile)
             call ViewRiverProfile(lret)
          case (IDD_StartModel)
             call Start_Model(lret)
          case (IDD_HaltModel)
             call Halt_Model(lret)
          case (IDD_AboutBox)
            msg1 = 'About StreamVOC'
```

```
            call wmessagebox (OKOnly, InformationIcon, CommonOK, &
aboutmessage, msg1)
            call WindowSelect(0)
            call WBitmapPut(MainWinBitmap, 0, 0)
            if (ErrorWindow) then
              call WindowSelect(ErrWinUnit)
              call WBitmapPut(ErrWinBitmap, 0, 0)
            endif
            if (ProfileWindowOpen) then
              focusprofile = 2
              call ViewRiverProfile(lret)
            endif
          case (IDD_Exit)
            exit
        end select
      case (PushButton)
        select case (mainmessage.value1)
          case (ID_OKEnvParams)
            call Enviro_OK()
          case (ID_HydrogOK)
            call Hydrog_OK()
          case (ID_OKVOCParams)
            call VOCParam_OK()
          case (ID_OKModelParams)
            call ModelPar_OK()
        end select
!      Redisplay plot if we get an expose or resize event
      case (Expose,Resize)
        if (mainmessage.win .eq. 0) then
          if (.not. lrunning) then
            call WindowSelect(0)
            call WBitmapPut(MainWinBitmap, 0, 0)
          endif
        elseif (mainmessage.win .eq. ErrWinUnit) then
          if (ErrorWindow) then
            call WindowSelect(ErrWinUnit)
            call WBitmapPut(ErrWinBitmap, 0, 0)
          endif
        elseif (mainmessage.win .eq. ProfileWinUnit) then
          if (ProfileWindowOpen) then
            focusprofile = 2
            call ViewRiverProfile(lret)
          endif
        endif
      case (CloseRequest)
        if (mainmessage.win .eq. 0) then  ! closed main window, exit program
          exit
        elseif (mainmessage.win .eq. ErrWinUnit) then
          call WindowCloseChild(ErrWinUnit)
          ErrorWindow = .false.
        elseif (mainmessage.win .eq. ProfileWinUnit) then
          call WindowCloseChild(ProfileWinUnit)
          ProfileWindowOpen = .false.
        endif
    end select
  end do
  call WindowClose()
end
```

Begin StreamVOC Subroutines and Functions, Listed Alphabetically

```
subroutine CalcFlowProfile()
  use msflib
  use voccom
  implicit none
  integer  ipnts, i, j, k, n, imax, ierr, idist, icnt, iweir
```

```
  real*8 deltaflow, delx, xval, xmid, tempwid, makeup, &
         cfactor, q, ramp, offset
  real*8 fdw, inputinterp, GetSourceNumber, WeirEffic_Calc
  external fdw, inputinterp, GetSourceNumber, WeirEffic_Calc
  real*8 temparr(:,:)
  allocatable temparr
! FlowVolumes contains flow, width, depth info as function of distance
  if (allocated(FlowVolumes)) deallocate(FlowVolumes)
  ierr = iaxes + 2
  allocate (FlowVolumes(ierr, 2+4*Reaches+4*DistributedSources))
!  need distance, VOCInput, and SourceNumber for VOCInputs, that's why
!  it is dimensioned as iaxes+1
  if (allocated(VOCInputs)) deallocate(VOCInputs)
  ierr = iaxes + 1
  allocate (VOCInputs(ierr, 2+4*Reaches+4*DistributedSources))
! set data into array, FlowVolumes
! primary data array for river physical profile!
  FlowVolumes(1,1) = RiverData(1,1)    !distance   km
  FlowVolumes(2,1) = RiverData(1,5)    !flow volume    m^3/sec
  FlowVolumes(3,1) = RiverData(1,3)    !depth    m
  FlowVolumes(4,1) = RiverData(1,4)    !width    m
  imax = 1
  do j = 2, reachpoints-1
    if (RiverData(j,1) .eq. RiverData(j,2)) then
!     dealing with Reach endpoint or pnt. src.
!     if Reach boundary isn't there from dist source
!     need to add a point to FlowVolumes
      if (RiverData(j,1) .gt. FlowVolumes(1,imax)) then
!       increment IMAX to the new point
        imax = imax + 1
!       add in the point for the end of the reach
        FlowVolumes(1,imax) = RiverData(j,1)
        FlowVolumes(2,imax) = FlowVolumes(2,imax-1)
        FlowVolumes(3,imax) = FlowVolumes(3,imax-1)
        FlowVolumes(4,imax) = FlowVolumes(4,imax-1)
      endif
!     set the ramp distance =  river width up to a maximum of 5 m
      if (RiverData(j,4) .gt. 5.0d0) then
        ramp = 5.0d-3
      else
        ramp = 1.0d-3* RiverData(j,4)
      endif
!      increment IMAX to the new point
      imax = imax + 1
!      now add in all the changes in Flow, Depth, Width
      FlowVolumes(1,imax) = RiverData(j,1) + ramp
      FlowVolumes(2,imax) = FlowVolumes(2,imax-1) + RiverData(j,5)  !flow
      FlowVolumes(3,imax) = RiverData(j,3)                          !depth
      FlowVolumes(4,imax) = RiverData(j,4)                          !dwidth
    else
!     RiverData(j,1) .lt. RiverData(j,2) dist. source
!     find out if Dist Src starts at beginning of reach
!     (FlowVolumes(1,imax) is the last point,
!     either pnt src, reach boundary, or dist src)
      if (RiverData(j,1) .gt. FlowVolumes(1,imax)) then
!       start of new dist. src beyond last endpoint, add in start point
        imax = imax + 1
        FlowVolumes(1,imax) = RiverData(j,1)
        FlowVolumes(2,imax) = FlowVolumes(2,imax-1)  ! no change in flow
        FlowVolumes(3,imax) = FlowVolumes(3,imax-1)  ! no change in depth
        FlowVolumes(4,imax) = FlowVolumes(4,imax-1)  ! no change in width
      else
!       dist src starts before the last endpoint, nothing needs to be done
!       else here in case we change method and need it later
        continue
      endif
!     now add in point for dist src endpoint
```

```
      imax = imax + 1
      FlowVolumes(1,imax) = RiverData(j,2)
      FlowVolumes(2,imax) = FlowVolumes(2,imax-1) + RiverData(j,5)
!     add the total increase in flow
      FlowVolumes(3,imax) = FlowVolumes(3,imax-1)  ! no change in depth
      FlowVolumes(4,imax) = FlowVolumes(4,imax-1)  ! no change in width
    endif
  end do
! add on the end point if necessary
! (may already be there if last dist. source ends at end of river)
  if (RiverData(ReachPoints,1) .gt. FlowVolumes(1,imax)) then
    imax = imax + 1
    FlowVolumes(1,imax) = RiverData(j,1)
    FlowVolumes(2,imax) = FlowVolumes(2,imax-1)
    FlowVolumes(3,imax) = FlowVolumes(3,imax-1)
    FlowVolumes(4,imax) = FlowVolumes(4,imax-1)
  endif
! now resize the array to be the correct number of elements
  ProfilePoints = imax
  ierr = iaxes+2
  allocate (temparr(ierr,ProfilePoints))
  do i = 1, ierr
    do j = 1, ProfilePoints
      temparr(i,j) = FlowVolumes(i,j)
    end do
  end do
  deallocate (FlowVolumes)
  allocate(FlowVolumes(ierr,ProfilePoints))
  do i = 1, ierr
    do j = 1, ProfilePoints
      FlowVolumes(i,j) = temparr(i,j)
    end do
  end do
  deallocate (temparr)
! now compute the VOC input profile
  VOCInputs(1,1) = RiverData(1,1)
  VOCInputs(2,1) = 0.0d0
  if (RiverData(1,6) .gt. 0.0d0) then
    VOCInputs(3,1) = 1
    SourceNumber = 1
!   next source will be the second, 1 is initial concentration
  else
    VOCInputs(3,1) = 0
    SourceNumber = 0
!   next source will be the first, no initial concentration
  endif
  imax = 1
  idist = 1
! CFACTOR converts m3*ug/sec-L into g/sec so that div by m.w. gives mol/sec
  cfactor = 1.0d-3
  do i = 2, reachpoints - 1
!   this if-then checks to make sure there is actually a source to include
    if ((RiverData(i,5) .gt. 0.0d0) .and. (RiverData(i,6) .gt. 0.0d0)) then
!     increment the source number to get ready for the next source
      SourceNumber = SourceNumber + 1
!     figure out if we have a distributed or point source
      if (RiverData(i,1) .lt. RiverData(i,2)) then
!       dealing with distributed source
        if (RiverData(i,1) .gt. VOCInputs(1,imax)) then
!         beginning is not at reach boundary
!         set the initial zero level for the new src
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,1)
          VOCInputs(2,imax) = 0.0d0
          VOCInputs(3,imax) = SourceNumber
!         increment imax to add new point for src
          imax = imax + 1
```

```
!         set the source strength as the source beginning
          VOCInputs(1,imax) = RiverData(i,1)
!         see p-51 of STREAMVOC notebook for discussion
          VOCInputs(2,imax) = cfactor * RiverData(i,5) * RiverData(i,6) / &
                  (molweight * (1.0d3*(RiverData(i,2)-RiverData(i,1))))
          VOCInputs(3,imax) = SourceNumber
!         set the source strength at the source end
!         increment imax to add new point for src
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,2)
          VOCInputs(2,imax) = cfactor * RiverData(i,5) * RiverData(i,6) / &
                  (molweight * (1.0d3*(RiverData(i,2)-RiverData(i,1))))
          VOCInputs(3,imax) = SourceNumber
!         bring the source back to zero
!         increment imax to add end point for src
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,2)
          VOCInputs(2,imax) = 0.0d0
          VOCInputs(3,imax) = SourceNumber
        else   ! RiverData(j,1)<VOCInputs(1,imax) and sources overlap
!         because beginning of dist src is end of old one
!         don't need to add zero point
!         it's already there from previous point
!         just need to add point(s) for source strength
!         set the source strength at the beginning of the source
          imax = imax + 1
          VOCInputs(1,imax) = VOCInputs(1,imax-1)
!         see p-51 of STREAMVOC notebook for discussion
!         note that src distance adjusted for previous src. boundary
          VOCInputs(2,imax) = cfactor * RiverData(i,5) * RiverData(i,6) / &
                  (molweight * (1.0d3*(RiverData(i,2)-VOCInputs(1,imax))))
          VOCInputs(3,imax) = SourceNumber
!         set the source strength at the end of the source
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,2)
          VOCInputs(2,imax) = cfactor * RiverData(i,5) * RiverData(i,6) / &
                  (molweight * (1.0d3*(RiverData(i,2)-VOCInputs(1,imax-1))))
!         change line above to IMAX-1 because incremented IMAX for new point
          VOCInputs(3,imax) = SourceNumber
!         now take the source input back to zero
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,2)
          VOCInputs(2,imax) = 0.0d0
          VOCInputs(3,imax) = SourceNumber
        endif
      else
!       end of distributed source entry
!       RiverData(i,1) .eq. RiverData(i,2): confluence
        if (RiverData(i,1) .gt. VOCInputs(1,imax)) then
!         if there was no distributed source or end not at boundary
!         add in zero point
          imax = imax + 1
          VOCInputs(1,imax) = RiverData(i,1)
          VOCInputs(2,imax) = 0.0d0
          VOCInputs(3,imax) = SourceNumber
        endif
!       set the nominal src distance for a point source = max 5 m or width
        if (RiverData(i,4) .gt. 5.0d0) then
          ramp = 5.0d-3
        else
          ramp = 1.0d-3 * RiverData(i,4)
        endif
!        see p-51 of STREAMVOC notebook for discussion
        imax = imax + 1
        VOCInputs(1,imax) = VOCInputs(1,imax-1)
!       either way, this distance will be correct
!       set the source strength at the start of the source
```

```
        VOCInputs(2,imax) = &
            cfactor*(RiverData(i,5)*RiverData(i,6))/(1.0d3*ramp*molweight)
        VOCInputs(3,imax) = SourceNumber
!       now add in the end of the point source RAMP distance after start
        imax = imax + 1
        VOCInputs(1,imax) = VOCInputs(1,imax-1) + ramp
        VOCInputs(2,imax) = &
            cfactor*(RiverData(i,5)*RiverData(i,6))/(1.0d3*ramp*molweight)
        VOCInputs(3,imax) = SourceNumber
!       add the final zero point bringing source to end
        imax = imax + 1
        VOCInputs(1,imax) = VOCInputs(1,imax-1)
        VOCInputs(2,imax) = 0.0d0
        VOCInputs(3,imax) = SourceNumber
      endif   ! end of reach boundary input
    endif    ! end of if-then to make sure there is actually a source
  end do
! add in the endpoint of the river, just in case
  VOCInputs(1,imax+1) = RiverLength
  VOCInputs(2,imax+1) = 0.0d0
  VOCInputs(3,imax+1) = 0.0d0
  VOCInputPoints = imax+1
! now resize the array to be the correct number of elements
  ierr = iaxes + 1
  allocate (temparr(ierr,VOCInputPoints))
  do i = 1, ierr
    do j = 1, VOCInputPoints
      temparr(i,j) = VOCInputs(i,j)
    end do
  end do
  deallocate (VOCInputs)
  allocate(VOCInputs(ierr,VOCInputPoints))
  do i = 1, ierr
    do j = 1, VOCInputPoints
      VOCInputs(i,j) = temparr(i,j)
    end do
  end do
  deallocate (temparr)
! now calculate lookup table for flow velocities
! to be used for timescales and transfer coefficients
! FLOWVELOCITY array defined in VOCCOM as (2,splinepnts), real*8
  MaxVelocity = 0.0d0
  MinVelocity = 500.0d0
  MaxVOCInput = 0.0d0
! deallocate the splined arrays if necessary
  if (allocated(spl_Depth)) deallocate(spl_Depth)
  if (allocated(spl_Width)) deallocate(spl_Width)
  if (allocated(spl_Flow)) deallocate(spl_Flow)
  if (allocated(spl_VOCInputs)) deallocate(spl_VOCInputs)
  if (allocated(spl_Veloc)) deallocate(spl_Veloc)
  if (allocated(spl_Timestep)) deallocate(spl_Timestep)
  if (allocated(spl_xval)) deallocate(spl_xval)
  if (allocated(OldData)) deallocate(OldData)
! figure out how many splined points we're going to need.
! Formula is on P-55 of STREAMVOC notebook 1 and is based on VOCInput array
  splinepnts = Boxpnts * (ProfilePoints - 1)
! allocate splined arrays
  allocate(spl_Depth(splinepnts))
  allocate(spl_Width(splinepnts))
  allocate(spl_Flow(splinepnts))
  allocate(spl_VOCInputs(3,splinepnts))
  allocate(spl_Veloc(2,splinepnts))
  allocate(spl_Timestep(splinepnts))
  allocate(spl_xval(splinepnts))
! reset splinepnts to 0 so that we have correct number of pnts
  splinepnts = 0
  icnt = 1
```

```
  if (WeirNum .gt. 0) then
    iweir = 1
  else
    iweir = 0
  endif
  do i = 1, ProfilePoints-1
    delx = (FlowVolumes(1,i+1) - FlowVolumes(1,i)) / (dble(float(Boxpnts-1))) !   delx in km
    do j = 1, Boxpnts
      xval = FlowVolumes(1,i) + delx * float(j-1)   ! xval in km
      if ((xval .gt. spl_xval(icnt-1)) .or. (xval .eq. 0.0d0)) then
        xmid = xval + 0.5d0 * delx     !xmid in km
        if (xmid .gt. RiverLength) xmid = RiverLength
        spl_Depth(icnt) = fdw(xval,3)
        spl_Width(icnt) = fdw(xval,4)
        spl_Flow(icnt) = fdw(xval,2)  !  flow in m^3/sec
        spl_VOCInputs(1,icnt) = xval  !  calculates VOC input
        spl_VOCInputs(2,icnt) = inputinterp(xval)  ! VOC input in mol/(sec-m)
        spl_VOCInputs(3,icnt) = GetSourceNumber(xval)
!       flow is in m^3/s so velocity in m/s  is flow/(depth(m)*width(m))
!       calculate midpoint velocity as average over interval
        spl_Veloc(1,icnt) = xval
!       calculates midpoint flow velocity and other variables
        spl_Veloc(2,icnt)=fdw(xval,2)/(fdw(xval,3)*fdw(xval,4))
!       timestep is distance in km divided by velocity in m/sec * 0.001 km/m
!       1000 m/km * km * sec/m = sec
        spl_Timestep(icnt) = 1.0d3 * delx / spl_Veloc(2,icnt)
!       save the beginning xvalue of each interval, spl_xval is in km!
        spl_xval(icnt) = xval   ! xval in km
        if ((iweir .gt. 0) .and. (iweir .le. WeirNum)) then
          if (spl_xval(icnt) .ge. WeirPoints(iweir,1)) then
!           calculate specific flow
            q = spl_flow(icnt)/WeirPoints(iweir,2)
            WeirEffic(iweir) = WeirEffic_Calc(WeirType(iweir), q, &
              WeirPoints(iweir,3), WeirPoints(iweir,4), WeirPoints(iweir,5))
            iweir = iweir + 1
          endif
        endif
        if (spl_Veloc(2,icnt) .gt. MaxVelocity) &
              MaxVelocity = spl_Veloc(2,icnt)
        if (spl_Veloc(2,icnt) .lt. MinVelocity) &
              MinVelocity = spl_Veloc(2,icnt)
        if (spl_VOCInputs(2,icnt) .gt. MaxVOCInput) &
              MaxVOCInput = spl_VOCInputs(2,icnt)
        if (spl_VOCInputs(2,icnt) .lt. MinVOCInput) &
              MinVOCInput = spl_VOCInputs(2,icnt)
        icnt = icnt + 1
        splinepnts = splinepnts+1
      endif
    end do !j
  end do !i
!  debugging output used to write the Splined Data sets
!  open (10, file = 'test.dat')
!  write(10, '(a,a)') '      xval        depth      width    flow',&
!  '      x-VOCInput    y-VOCInput     x-Veloc       y-Veloc      Timestep'
!  do i = 1, splinepnts
!    write (10, '(5f12.4,e14.4,f10.1,f12.4,2e14.4)') spl_xval(i), &
!      spl_Depth(i), spl_Width(i), spl_Flow(i),
!     (spl_VOCInputs(j,i), j = 1, 3), (spl_Veloc(j,i), j = 1, 2), &
!      spl_Timestep(i)
!  end do
!  close(10)
! end debugging output
! allocate data array used for plotting data when model runs
  allocate(OldData(2,splinepnts,2))
  return
end subroutine CalcFlowProfile
```

```
subroutine cfunc_comp (t, c, dt)
! CFUNC partitions net flux into Fin and Fout
! Fin = kOA * Cs
! Fout = kOA * Cb
! P125 of River Modelling I lab notebook
  use VOCCOM
  use cfunccom
  implicit none
! local real*8 variables, some are shorthand notation
  integer i
  real*8 t,c(num_eqs),dt(num_eqs), mtinv,dmi, dmiair, dmt, gasout, gasin, &
         airterm, gwloss, area
  real*8 sourceinput(:)
  allocatable sourceinput
  allocate (sourceinput(num_eqs))
  ! c(1) = River flow (m^3/sec)
  ! c(2) = River concentration (mol/m^3)
  ! c(3:num_eqs-1) = concentration fractions of the individual sources
  ! c(num_eqs) = air source conc. fractopm
  ! dt(1) = dQ/dt
  ! dt(2) = dC(total)/dt
  ! dt(3:num_eqs-1) = change in conc. frac. of the individual sources
  ! dt(num_eqs) = change in conc frac of air source
  area = c(1) / depth
  gasout = kl*area*c(2)
  gasin = kl*area*cs
  dt(1) = qprime
! GWLOSS removes VOC mass from river due to flow to groundwater
  if (qprime .lt. 0.0d0) then
    gwloss = qprime * c(2)   ! remove mass if necessary
  else
    gwloss = 0.0d0           ! remember to reset to zero if not needed
  endif
  dt(2) = (gasin - gasout + ii + gwloss)/c(1) - kd*c(2) - c(2)*dt(1)/c(1)
  mtinv = 1.0d0 / (c(1) * c(2))
  dmt = (gasin - gasout - kd*c(2)*c(1) + ii + gwloss)
  sourceinput = 0.0d0   ! array initialization
! need to add two to currentsource to offset for flow and total concentration
  sourceinput(currentsource+2) = ii
  sourceinput(num_eqs) = gasin
  do i = 3, num_eqs    ! i = num_eqs is the atmospheric source
    dmi = sourceinput(i) + c(i) * (gwloss - gasout - kd*c(2)*c(1))
!   change in concentration fraction
    dt(i) = mtinv * (dmi - c(i)*dmt)
  end do
  deallocate (sourceinput)
  return
end subroutine cfunc_comp

subroutine cfunc_net (t, c, dt)
! version of CFUNC that does not partition net flux into Fin and Fout
! does not calculate atmospheric source term if Cs <= Cb
! used in versions of StreamVOC that are pre-June 23, 2005
  use VOCCOM
  use cfunccom
  implicit none
! local real*8 variables, some are shorthand notation
  integer i
  real*8 t, c(num_eqs), dt(num_eqs), mtinv,dmi,dmiair,dmt,gasex,airterm, &
         gwloss, area
  real*8 sourceinput(:)
  allocatable sourceinput
  allocate (sourceinput(num_eqs))
  ! c(1) = River flow (m^3/sec)
  ! c(2) = River concentration (mol/m^3)
  ! c(3:num_eqs-1) = concentration fractions of the individual sources
  ! c(num_eqs) = air source conc. fractopm
```

```
  ! dt(1) = dQ/dt
  ! dt(2) = dC(total)/dt
  ! dt(3:num_eqs-1) = change in conc frac of the individual sources
  ! dt(num_eqs) = change in conc frac of air source
  area = c(1) / depth
  gasex = kl*area*(cs-c(2))
  dt(1) = qprime
! removes VOC mass from river due to loss of flow to groundwater
  if (qprime .lt. 0.0d0) then
    gwloss = qprime * c(2)   ! remove mass if necessary
  else
    gwloss = 0.0d0           ! remember to reset to zero if not needed
  endif
  dt(2) = (gasex + ii + gwloss)/c(1) - kd*c(2) - c(2)*dt(1)/c(1)
  mtinv = 1.0d0 / (c(1) * c(2))
  dmt = (gasex - kd*c(2)*c(1) + ii + gwloss)
  do i = 3, num_eqs
    sourceinput(i) = 0.0d0
  end do
! need to add two to currentsource to offset for flow and total conc
  sourceinput(currentsource+2) = ii
  if (gasex .gt. 0.0d0) then
    airterm = 0.0d0    ! source from atmosphere
    dmiair = gasex + c(num_eqs)*(-kd*c(2)*c(1) + gwloss)
  else
    airterm = gasex    ! sink to atmosphere
    dmiair = c(num_eqs)*(gasex-kd*c(2)*c(1) + gwloss)
  endif
! i = num_eqs is the atmospheric source term, dealt with separately
  do i = 3, num_eqs-1
    dmi = sourceinput(i) + c(i) * (airterm - kd*c(2)*c(1) + gwloss)
!   change in conc frac defined on page 68 of StreamVOC notebook 1
    dt(i) = mtinv * (dmi - c(i)*dmt)
  end do
  dt(num_eqs) = mtinv * (dmiair - c(num_eqs)*dmt)
  deallocate (sourceinput)
  return
end subroutine cfunc_net

real*8 function cs_func(i)
! calculates csat from water temperature
use VOCcom
implicit none
integer numpts, i
real*8 t, time, atmpr, deg_c, airconc, interpolate, sol_calc, local_sol
external interpolate
deg_c = SurfaceTemp
airconc = AtmVOCConc
atmpr = AtmosPress
airconc = airconc * 1.0d-9   !  change ppbv into atmospheres
local_sol = sol_calc(deg_c, SolParam)  !  solubility in mol/m^3-atm
cs_func = local_sol * airconc * atmpr
return
end function cs_func

subroutine DatFilOut(bool_in)
! saves the output data file from the model
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  logical ret, bool_in
  integer ierror
  character*255 Datfile_Out_tmp
  character*60 dlgtitle
  call unusedqq(checked)
```

```
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(ret)
    endif
    return
  endif
!* dialog title
  dlgtitle = 'Save Model Output to ASCII File'
!  reset temporary filename
  Datfile_Out_tmp = ''
!* create dialog
  call WSelectFile(IDS_DataFileString, 13, Datfile_Out_tmp, dlgtitle)
!* check for error
  VOC_File_Sav = .false.
  if (Datfile_Out_tmp .ne. '') then
    Datfile_Out = Datfile_Out_tmp
    call save_VOC(VOC_File_Sav)
  endif
  return
end subroutine DatFilOut

real*8 function diff_calc(tempc, iform)
! function returns D in cm^2/sec which is converted to Sc for calculating kL
! nu used in Sc relation is also cm^2/sec so no need to convert D to SI units
use VOCcom
implicit none
integer iform
real*8 tempc
real*8 mu, nu, sc
select case (iform)
  case(1)
!   Absolute viscosity (mu) in gm-cm/sec calc. from temperature in deg-C.
!   The underlying data is from data from CRC 63rd edition.
!   The polynomial fit was done in the spreadsheet KINVISC.WB1 in QDATA
    mu = 1.7825047d0-5.75921d-2*tempc+1.11378d-3*tempc**2-9.55317d-6*tempc**3
    diff_calc = (4.7199d-07*(tempc+2.7316d2))/(mu*(MolarVolume**0.6d0))
  case(2)
    sc = wd0 + wd1*tempc + wd2*tempc**2 + wd3*tempc**3
!   Kinematic viscosity (nu) in cm^2/sec is calc. from temperature in deg-C.
!   The underlying data is from data from CRC 63rd edition.
!   The polynomial fit was done in the spreadsheet KINVISC.WB1 in QDATA
    nu=1.7826598d-2-5.76464d-4*tempc+1.12266d-5*tempc**2-9.66507d-8*tempc**3
    diff_calc = 0.0d0
    if (sc .ne. 0.0d0) diff_calc = nu/sc
end select
return
end function diff_calc

subroutine dot2space(tline)
implicit none
character*72 tline
integer length, i
length = len_trim(tline)
do i = 1, length
  if (tline(i:i) .eq. '~') tline(i:i) = ' '
end do
return
end subroutine dot2space
```

```
real*8 function dqdt(i)
! calculates derivative of flow volumes w.r.t. time using flow_volume and velocity series
use voccom
implicit none
logical test
integer i
real*8 b, flow1, flow2
! begin subroutine
flow1 = spl_Flow(i)
flow2 = spl_Flow(i+1)
b = (flow2 - flow1) / spl_timestep(i)
dqdt = b
return
end function dqdt

subroutine Enviro_OK()
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  call WDialogUnload()
  menuactive = .false.
  return
end subroutine Enviro_OK

SUBROUTINE Enviro_Params(bool_in)
!*****************************************************************************
!*                                                                           *
!*  This subroutine creates a dialog box that allows the user to view      *
!*  the parameters associated with the enviromental conditions.  It also   *
!*  sets the call back routines for the help and ok buttons.               *
!*                                                                           *
!*****************************************************************************
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  logical lret, bool_in
  integer iret, ierr, iloop
  external Enviro_ok
  call unusedqq(bool_in)
  ierr = 0
  msg0 = ''
  msg1 = ''
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  iloop = 1
    menuactive = .true.
!   initialize the dialog box
    call WDialogLoad(IDD_EnviroParams)
!   write initial values and set subroutines
```

```
    if (iloop .eq. 1) then
      write(err_str(1), '(f7.2)') 100*Rel_Hum
      write(err_str(2), '(f7.2)') WindSpeed
      write(err_str(3), '(f7.2)') AirTemp
      write(err_str(4), '(f7.2)') AtmosPress
      write(err_str(5), '(f7.2)') SurfaceTemp
    endif
    call WDialogPutString(IDC_RelativeHumidity, err_str(1))
    call WDialogPutString(IDC_WindSpeed, err_str(2))
    call WDialogPutString(IDC_AirTemp, err_str(3))
    call WDialogPutString(IDC_AtmPressure, err_str(4))
    call WDialogPutString(IDC_WaterTemp, err_str(5))
!   initiate the dialog window
    call WDialogShow(-1, -1, 0, 2)  ! show modeless dialog IDD_EnviroParams
  return
end subroutine Enviro_Params

subroutine ErrorOutput(vpos, offset)
use msflib
use winteracter
use VOCcom
implicit none
integer vpos, offset
call WindowSelect(ErrWinUnit)
call WindowOutString(0, vpos, msg0)
vpos = vpos + offset
return
end subroutine ErrorOutput

real*8 function fdw(x, icol)
use VOCcom
integer numcols, icol
real*8 x
real*8 interpolate
external interpolate
numpts = ProfilePoints
numcols = 4
fdw = interpolate(FlowVolumes, x, icol, numcols, numpts)
return
end function fdw

real function findmaxval(index)
! index is column of RiverData array to find maximum value
use voccom
integer index, i
findmaxval = 0.0
do i = 1, ReachPoints
  if (findmaxval .lt. sngl(RiverData(i,index))) &
      findmaxval = sngl(RiverData(i,index))
end do
return
end function findmaxval

real*8 function flux_h2o(t)
!note:  flux_h2o unused in streamvoc, left in for future modeling of stream
!       evaporation
use VOCcom
real*8 t
real*8 airt, tempk, degc, logvp, vp, delc, time, u, ka_h2o, flux_mass
numpts = splinepnts
degc = SurfaceTemp
u = WindSpeed
airt = AirTemp
! variable T is time in days from model start, TIME is time in months
time = 1.2d1*(t/3.65d2 - float(int(t/3.65d2)))
! Air-side resistance estimated from H2O relation in Schwarzenbach et al. Env. Org. Chem.
```

```
ka_h2o = 0.01d0*(0.2d0*u+0.3d0)*dsqrt((airt+2.7316d2)/2.9316d2)           !ka_h2o in m/s at
AirTemp
tempk = degc + 2.7316d2
! v.p. of H2O predicted from data from CRC-63rd, fit done in Quattro
logvp = 3.14004128517d1 + -6.788619575d1*(1.0d2/tempk) + -5.00162020852d0*dlog(1.0d-2*tempk)
vp = dexp(logvp)
vpatm = vp/7.60d2
delc = (1.0d0 - rel_hum) * vpatm / (gasconst * tempk) ! delta-C for water
flux_mass = 2.4d1 * 3.6d3 * 1.8d1 * ka_h2o * delc ! water lost in g/day-m^2
flux_h2o = 1.0d-03 * flux_mass  ! height of water loss in mm over timestep
return
end function flux_h2o

real*8 function GetSourceNumber(x)
use VOCcom
integer numcols, numpts, i, icol
real*8 x, VOCin
real*8 interpolate
external interpolate
numpts = VOCInputPoints
numcols = 3
icol = 2
VOCin = interpolate(VOCInputs, x, icol, numcols, numpts)
if (VOCin .gt. 0.0d0) then
  i = 1
  do while (i .le. numpts-1)
    if ((x .ge. VOCInputs(1,i)) .and. (x .le. VOCInputs(1,i+1))) then
      GetSourceNumber = max(VOCinputs(3,i), VOCInputs(3,i+1))
      return
    endif
    i = i + 1
  end do
endif
GetSourceNumber = 0
return
end function GetSourceNumber

subroutine go_vocdrv(arg2)
!*************************************************************************
!*                                                                       *
!*  description for subroutine go_vocdrv(arg2)                           *
!*                                                                       *
!*  used to call the subroutine that does the modeling calculations      *
!*                                                                       *
!*************************************************************************
  use dfmt
  implicit none
  integer(4) arg2
  arg2 = 0
  call vocdrv
  call exitthread(0)    !exit code is 0
  return
end subroutine go_vocdrv

SUBROUTINE Halt_Model(bool_in)
! halts model in mid-run, processing may continue until stop is reached in
! rkintout.for
  use msflib
  use winteracter
  use voccom
  implicit none
  logical(kind=4) bool_in
  integer iret
  call unusedqq(bool_in)
  lrunning = .false.
  menuactive = .false.
  return
end subroutine Halt_Model
```

```
subroutine Hydrog_OK()
!*****************************************************************************
!*  callback routine for when the ok button has been pushed                  *
!*****************************************************************************
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  call WDialogUnload()
  menuactive = .false.
  return
end subroutine Hydrog_OK

subroutine Hydrog_Params(bool_in)
  use msflib
  use winteracter
  use voccom
  implicit none
  INCLUDE 'RESOURCE.FD'
  character*128 dialogtext(:)
  allocatable dialogtext
  integer i, j, iret, index
  logical bool_in, lret
  call unusedqq(bool_in)
  allocate (dialogtext(reachpoints))
  msg0 = ''
  msg1 = ''
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  menuactive = .true.
! initialize the dialog box
  call WDialogLoad(IDD_HydrogParamsList)
  write(err_str(1),'(i5)') Reaches
  write(err_str(2),'(f13.2)') RiverLength
  write(err_str(3),'(i5)') DistributedSources
  index = 3
  do i = 1, ReachPoints
    if (RiverData(i,1) .eq. RiverData(i,2)) then
      iret = 1    ! reach boundary
    else
      iret = 2    ! distributed source
    endif
    if (i .lt. 10) then
      write(dialogtext(i), '(2x,i4,10x,i1,2x,6(5x, e11.4),6x,i4)') &
          i, iret, (RiverData(i,j), j=1,6), int(RiverData(i,7))
    elseif ((i .ge. 10) .and. (i .lt. 100)) then
      write(dialogtext(i), '(1x,i4,10x,i1,2x,6(5x, e11.4),6x,i4)') &
          i, iret, (RiverData(i,j), j=1,6), int(RiverData(i,7))
    else
      write(dialogtext(i), '(i4,10x,i1,2x,6(5x, e11.4),6x,i4)') &
          i, iret, (RiverData(i,j), j=1,6), int(RiverData(i,7))
    endif
```

```
  end do
  index = index + ReachPoints
  call WDialogPutMenu(IDC_HydrogParamList, dialogtext, reachpoints, 1)
  call WDialogPutString(IDC_ReachPoints, err_str(1))
  call WDialogPutString(IDC_TotalLength, err_str(2))
  call WDialogPutString(IDC_DistributedSourcePoints, err_str(3))
! bring up the dialog box
  call WDialogShow(-1, -1, 0, 2)
  deallocate (dialogtext)
  return
end subroutine Hydrog_Params

real*8 function inputinterp(x)
use VOCcom
integer numcols, icol
real*8 x
real*8 interpolate
external interpolate
numpts = VOCInputPoints
numcols = 3
icol = 2
i = 1
inputinterp = interpolate(VOCInputs, x, icol, numcols, numpts)
return
end function inputinterp

real*8 function interpolate(x_series, xval, icol, numcols, numpts)
! interpolation routine for SPLINEPNT-pt time series.
implicit none
integer i, numpts, iday, numcols, icol
real*8 x_series(numcols,numpts), xi, xf, yi, yf, xval
i = 1
do while (i .le. numpts)
  if (xval .eq. x_series(1,i)) then
    interpolate = x_series(icol,i)
    return
  elseif ((xval .gt. x_series(1,i)) .and. (xval .le. x_series(1,i+1))) then
    xi = x_series(1,i)
    xf = x_series(1,i+1)
    yi = x_series(icol,i)
    yf = x_series(icol,i+1)
    interpolate = yi + (yf-yi)*((xval-xi)/(xf-xi))
    return
  endif
  i = i + 1
end do
return
end function interpolate

real*8 function kd_func(i)
! function calculates the biochemical degradation rate of the VOC
use VOCcom
implicit none
integer numpts, i
real*8 ii_kgday, t, time
!, interpolate
!external interpolate
! now just sets kd_func equal to degradation rate since constant but used a
!function in case I want to change to make a function of time
kd_func = DegradationRate
return
end function kd_func

real*8 function kl_func(i)
! calculates kL from river flow velocity, water depth, water temperature, &
! and wind speed
! Calculates Koa assuming liquid and gas-phase rate control
```

```
! Based opn relationships contained in &
! USGS Paper 1589, Transport, Behavior, and fate of Volatile
! Organic Compounds in Streams by D. Rathbun.  pp4-12 & pp 35-40
use VOCcom
implicit none
integer numpts, i
real*8 airt, kl_mps, depth, u, v, degc, sc, diff_calc, nu, phi, psi, &
       ka_h2o_26, k20, kltemp, kvoc, ka_voc, kl_voc, kH, sol_calc, d
external diff_calc, sol_calc
u = WindSpeed
degc = SurfaceTemp
airt = AirTemp
v = spl_veloc(2,i)
depth = spl_depth(i)
kH = sol_calc(degc, SolParam)
d = diff_calc(degc, DiffParam)          !  calculate diffusivity (d) of VOC
!   Kinematic viscosity (nu) in cm^2/sec is calculated from temp in deg-C.
!   The underlying data is from data from CRC 63rd edition.
!   The polynomial fit was done in the spreadsheet KINVISC.WB1 in QDATA
nu = 1.7826598d-2-5.76464d-04*degc+1.12266d-05*degc**2-9.66507d-08*degc**3
sc = nu/d
! Air-side resistance estimated from H2O relation on P 57 of Rathbun, Eq. 57,
! Temp corr using Eq. 58 and refer to VOC using Eqs. 26 and 28
ka_h2o_26 = (4.16d2 + 1.56d2*u)/(2.4d1*3.6d3)     !eq. 57, ka_h2o in m/day @ 26.1 deg-C converted
to m/s
psi = 4.42d0 * MolWeight**(-0.462d0)          ! Eq. 28, correction factor
! correct ka @26 C for H2O to VOC at air temperature
ka_voc = psi * ka_h2o_26 * exp(9.34d-3*(airt - 2.61d1))
! now do some conditional testing to set the correct range for
! parameterization for kL
if ((v .ge. 0.05d0) .and. (depth .gt. 0.274d0)) then
  ! Reaerat coeff K in 1/day from O'Conner & Dobbins (Eq. 55, Rathbun) 20 C
  k20 = 3.93d0 * sqrt(v) * depth**(-1.5d0) / (2.4d1*3.6d3)  ! eq. 55
  kvoc = k20 * (1.024d0)**(degc - 2.0d1)                  ! eq. 56
  phi = 2.52d0 * molarvolume**(-0.301d0)                    ! eq. 27
! eq. 25 and converted to m/sec from 1/sec as in Eq. 16
  kl_voc = depth * phi * kvoc
elseif ((v .ge. 0.04d0) .and. (depth .ge. 0.119d0)) then
  ! Reaer coeff K in 1/day from Owens et al. (Eq. 53, Rathbun) at 20 C
  k20 = 6.92d0 * (v**0.73d0) * depth**(-1.75d0) / (2.4d1*3.6d3)  ! eq. 55
  kvoc = k20 * (1.024d0)**(degc - 2.0d1)                  ! eq. 56
  phi = 2.52d0 * molarvolume**(-0.301d0)                    ! eq. 27
! eq. 25 and converted to m/sec from 1/sec as in Eq. 16
  kl_voc = depth * phi * kvoc
elseif (v .lt. 0.04d0) then
! neither flow formula works, calc kL using U since we are in a pond
! kL in m/s from Wanninkhof et al. (1991; GasEx2 symp. paper on Page 441)
  kl_voc = 0.01d0*0.45d0*dsqrt(600.0d0/sc)*u**1.64d0/3.6d3
endif
! if-then needed for kG param.  can't divide by zero for kl_voc at u=0
if ((u .eq. 0.0d0) .and. (v .lt. 0.04d0)) then
  ! no water-side term, only kA
  kl_mps = ka_voc/(kH*gasconst*(degc+273.16d0))
else
  kl_mps=1.0d0/(1.0d0/kl_voc+1.0d0/(ka_voc/(kH*gasconst*(degc+273.16d0))))
endif
kl_func = kl_mps ! set value for kOL to the function output
return
end function kl_func

real*8 function mass_input(i)
! function calculates the VOC input
use VOCcom
implicit none
integer numpts, i
real*8 delx
! this makes sure we don't multiply a big xvalue times a voc input per meter
```

```
! it really screws up the mass balance
! delx is in km
if (i .gt. 1) then
  delx = min((spl_xval(i+1) - spl_xval(i)), (spl_xval(i) - spl_xval(i-1)))
else
  delx = (spl_xval(i+1) - spl_xval(i))
endif
! spl_VOCInputs is in mol/sec-m so 1.0d3 changes delx(km) to delx(m)
mass_input = 1.0d3 * spl_VOCInputs(2,i+1) * delx / spl_timestep(i)
return
end function mass_input

integer function mass_source(i)
! function calculates the VOC input to the lake from motorboats etc.
use VOCcom
implicit none
integer numpts, i
mass_source = max(int(spl_VOCInputs(3,i)), int(spl_VOCInputs(3,i+1)))
return
end function mass_source

subroutine menuset(ival)
use msflib
use winteracter
use VOCcom
implicit none
include 'resource.fd'
integer ival
! ival = 1: enable   ival = 0: grayed
  call WMenuRoot(IDR_Menu1)
  call WMenuSetState(IDD_SaveResults,1,ival)    ! VOC_Params enabled
  call WMenuSetState(IDD_ViewVOCParams,1,ival)    ! VOC_Params enabled
  call WMenuSetState(IDD_EnviroParams,1,ival)    ! Enviro_params enabled
  call WMenuSetState(IDD_HydrogParams,1,ival)    ! Hyrog_params enabled
  call WMenuSetState(IDD_RuntimeParameters,1,ival)    ! Model_params enabled
  call WMenuSetState(IDD_ViewProfile,1,ival)    ! View_River_Profile
  call WMenuSetState(IDD_StartModel,1,ival)    ! Start_model enabled
  call WMenuSetState(IDD_HaltModel,1,ival)    ! Start_model enabled
return
end subroutine menuset

subroutine menusetrunning(ival)
use msflib
use winteracter
use VOCcom
implicit none
include 'resource.fd'
integer ival
! ival = 1: enable   ival = 0: grayed
  call WMenuRoot(IDR_Menu1)
  call WMenuSetState(IDD_LoadParamFile,1,ival)    ! Load param disabled
  call WMenuSetState(IDD_SaveResults,1,ival)    ! Save_data disabled
  call WMenuSetState(IDD_ViewVOCParams,1,ival)    ! VOC_Params disabled
  call WMenuSetState(IDD_EnviroParams,1,ival)    ! Enviro_params disabled
  call WMenuSetState(IDD_HydrogParams,1,ival)    ! Hyrog_params disabled
  call WMenuSetState(IDD_RuntimeParameters,1,ival)    ! Model_params disabled
  call WMenuSetState(IDD_ViewProfile,1,ival)    ! View_River_Profile
  call WMenuSetState(IDD_StartModel,1,ival)    ! Start_model disabled
  call WMenuSetState(IDD_HaltModel,1,1)        ! Halt model enabled
  call WMenuSetState(IDD_AboutBox,1,ival)    ! About Box
return
end subroutine menusetrunning

subroutine Model_Params(bool_in)
  use msflib
  use winteracter
  use voccom
```

```
  implicit none
  include 'resource.fd'
  logical lret, err, bool_in
  integer iret
  call unusedqq(bool_in)
  msg0 = ''
  msg1 = ''
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  err = .true.
  menuactive = .true.
! initialize the dialog box
  call WDialogLoad(IDD_RuntimeParams)
  write(err_str(2),'(i4)') Boxpnts
  write(err_str(3),'(e15.4)') Tolerance
  err_str(4) = Title
  err_str(5) = Comment(1)
  err_str(6) = Comment(2)
  call WDialogPutString(IDC_NumberOfBoxes, err_str(2))
  call WDialogPutString(IDC_Tolerance, err_str(3))
  call WDialogPutString(IDC_Title, err_str(4))
  call WDialogPutString(IDC_Comment1, err_str(5))
  call WDialogPutString(IDC_Comment2, err_str(6))
! bring up the dialog box
  call WDialogShow(-1, -1, 0, 2)
  return
end subroutine Model_Params

subroutine ModelPar_OK ()
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  call WDialogUnload()
  menuactive = .false.
  return
end subroutine ModelPar_OK

subroutine openerrorwindow
use voccom
use winteracter
if (errorwindow) then
  call WindowRaise(ErrWinUnit)
  call WindowClear()
else
  errorwindowwc%flags  = SysMenuOn + MinButton + MaxButton
  errorwindowwc%x      = 20
  errorwindowwc%y      = 50
  errorwindowwc%width  = 600
  errorwindowwc%height = 800
  errorwindowwc%menuid = 0
  errorwindowwc%title  = 'StreamVOC_Winteracter Parameter File Input Errors'
```

```
  call WindowOpenChild(errorwindowwc, ErrWinUnit)
  errorwindow = .true.
endif
return
end subroutine openerrorwindow

subroutine parfilin(bool_in)
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  logical ret, ts_error, bool_in
  integer ierror, i
  character*255 Parfile_In_TMP
  character*60 dlgtitle
  external read_parfile
  call unusedqq(bool_in)
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(ret)
    endif
    return
  endif
! the next 7 lines added to automatically read data from default.par
!  ret = .true.
!  ierror = 0
!  if(ret .and. (ierror == 0))then
!    Parfile_In = 'default.par'
!    call read_parfile
!  endif
!  return
! remove before production version
  dlgtitle = 'Read Parameter File'
  Parfile_In_TMP = ''
  call WSelectFile(IDS_ParameterFileString, 8, Parfile_In_TMP, dlgtitle)
  if (Parfile_In_TMP .ne. '') then
    Parfile_In = Parfile_In_TMP
     call read_parfile
  endif
end subroutine ParFilIn

subroutine PlotProfile(xleft, xright, ybottom, ytop, deltay, xmin, xmax, &
   ymin,ymax,xdata,ydata1,ydata2,numpnts,numsets,colorvalue,xlabel,ylabel)
integer numpnts, numsets, colorvalue, iticks, strlen
parameter (iticks = 5)
real*4 xdata(numpnts), ydata1(numpnts), ydata2(numpnts)
real*4 xleft, xright, ybottom, ytop, deltay, xmin, xmax, ymin, ymax
real*4 xticks(iticks), yticks(iticks)
character*(*) xlabel, ylabel
  call IGrArea(xleft, ytop-deltay, xright, ytop)
  call IGrUnits(xmin, ymin, xmax, ymax)
  call IPgArea(0.1, 0.15, 0.95, 0.95)
  call IPgNewGraph(numsets, numpnts, '', '', 'X')
  call IPgStyle(1, 0, 0, 0, colorvalue, colorvalue)
! see col256 in wint/demos for key to values
  if (numsets .eq. 2) call IPgStyle(2, 0, 0, 0, colorvalue, colorvalue)
```

```
  call IPgUnits(xmin, ymin, xmax, ymax)
! set the x-axis ticks
  call SetTicks(xticks, iticks, xmin, xmax)
! set the y-axis ticks
  call SetTicks(yticks, iticks, ymin, ymax)
  call IGrCharFont(7)
  call IGrCharSize(1.0, 2.0)
  call IPgXScalePos(0.45)
  call IPgXUserScale(xticks, iticks)
  call IPgYUserScale(yticks, iticks)
  call IPgAxesXY(xmin, ymin)
  call IPgXTickPos(ymin,ymax)
  call IPgXscale('TN')
  call IPgYscaleLeft('TN')
  call IPgXLabel(xlabel,'C')
  call IPgYLabelPos(0.6)
  call IPgYLabelLeft(ylabel,'B9')
  call IPgXYPairs(xData, yData1)
  if (numsets .eq. 2) call IPgXYPairs(xData, yData2)
return
end subroutine PlotProfile

subroutine read_parfile
subroutine read_parfile
  use msflib
  use winteracter
  use VOCcom
  implicit none
  include 'resource.fd'
  integer(kind=4) vpos, offset
  integer(kind=4)iret, i, ierror, ifile, j, isource
  logical lexist, error, f_error, data_ok, diffsol_error, la_error, &
          initialsettings, lret
  character*3 dbs
  character*72 dtitle, dcomment(2), header, error_msg
  character*255 tempfile, FileSave
  integer ipnts, isplit, dReachPoints, dReaches, dDistributedSources, idum,&
          dDiffParam, dSolParam, dWeirType(:), dWeirNum
  real*8 dRiverLength, dSurfaceTemp, dAtmVOCConc, dDegradationRate,dAirTemp,&
         dWindSpeed, dAtmosPress, dInitialFlowRate, dMolWeight, dTol, dMV, &
         dDensity, dSalinity, dsola, dsolb, dwa0, dwa1, dwa2, dwb0, dwb1, &
         dwb2, dwd0, dwd1, dwd2, dwd3, drel_hum, dRiverData(:,:), &
         dDiffusivity, dSolubility, dTimestep, dWeirPoints(:,:)
  real*8 tempc, sc, mu, nu, tempk, sol1, alpha, logalpha
  allocatable dRiverData, dWeirType, dWeirPoints
  external ParFilIn, DatFilOut, VOC_Params, Enviro_Params, Hydrog_Params,&
           Model_Params, Start_model, Halt_model, ViewRiverProfile
  data dbs/'$$$'/
  tempfile = '*.*'
!  set temp MolarVolume variable to -1 so we can see if it's been set already
! Need to do this because
  dMV = 1.0d0
  offset = 250
  vpos = 200
! first reset the error flag and close the error window (if open)
  error = .false.
  ierror = 0
!* open parameter file
  inquire (file=ParFile_In, exist=LExist)
  if (.not. LExist) then
    msg0 = ' Parameter file does not exist!\nRe-enter filename'C
    msg1 = ' ERROR OPENING FILE '
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
```

```
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  if (allocated(dRiverData)) deallocate(dRiverData)
  if (allocated(dWeirPoints)) deallocate(dWeirPoints)
  if (allocated(dWeirType)) deallocate(dWeirType)
  open (ParFilUnit, file=ParFile_In, status='OLD', action = 'READ')
! read in header line, then the ModelFunction switch:
!               1 for Component Fluxes, 0 for net fluxes
  ModelFunction = -1
  WeirFlow = -1
  FlowUnits = -1
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) ModelFunction, WeirFlow, FlowUnits
  if ((ifile .ne. 0) .or. (ModelFunction .eq. -1) .or. &
      (WeirFlow .eq. -1) .or. (FlowUnits .eq. -1)) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write(msg0,'(a,i2,a)')'Input Error #',ierror,': Error Reading Flags'
    call ErrorOutput(vpos, offset)
  endif
  if ((ModelFunction .lt. 0) .and. (ModelFunction .gt. 1)) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  ModelFunction Flag must be 0 or 1'
    call ErrorOutput(vpos, offset)
  endif
  if ((WeirFlow .lt. 0) .and. (WeirFlow .gt. 1)) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  WeirFlow Flag must be 0 or 1'
    call ErrorOutput(vpos, offset)
  endif
  if ((FlowUnits .lt. 0) .and. (FlowUnits .gt. 1)) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  FlowUnits Flag must be 0 or 1'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then total river length (km)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dRiverLength
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  Error reading river length.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then the number of reaches
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dReaches
  if (ifile .ne. 0) then
    error = .true.
```

```
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  Error reading reach points.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then number of distributed source regions
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dDistributedSources
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  Error reading distributed source points.'
    call ErrorOutput(vpos, offset)
  endif
! calculate the number reach points to be read in (=dReaches+dDistSources+1)
  dReachPoints = dReaches + dDistributedSources + 1
  if (dReachPoints .gt. 101) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
           ':  Sum of # of reaches & dist. sources must be <= 100 '
    call ErrorOutput(vpos, offset)
  endif
! allocate river data array in preparation for reading in river physical data
  allocate (dRiverData(dReachPoints, 6))
! read in the two header lines, then read the river physical data into dRiverData array
! #  StartDist. (km) EndDist.(km)  Depth(m)  Width(m)  Flow(cfm)  Conc.(ug/L)
! for distributed source, flow is total increase over source region
! for distributed source, concentration is average conc. over source region
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile)
  do i = 1, dReachPoints
    read (ParFilUnit, *, iostat=ifile) idum, (dRiverData(i,j), j = 1, 6)
    if (ifile .ne. 0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a,i2)') 'Input Error #', ierror, &
            ':  Error reading reach data point #', i
      call ErrorOutput(vpos, offset)
    endif
  end do
! read in header line, then river temperature (C)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dSurfaceTemp
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  Error reading river temperature.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then air temperature (C)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dAirTemp
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
          ':  Error reading air temperature.'
    call ErrorOutput(vpos, offset)
```

```
  endif
! read in header line, then wind speed (m/s)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dWindSpeed
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading wind speed.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then relative humidity
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) drel_hum
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading relative humidity.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then barometric pressure (atm)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dAtmosPress
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading atmospheric pressure.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then atmos. VOC concentration (ppbv)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dAtmVOCConc
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading atmospheric VOC concentration.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then VOC molec. weight (g/mol)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dMolWeight
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading atmospheric pressure.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then diffusivity parameterization
! 1 for Wilke-Chang (requires molar volume at normal boiling point)
! 2 for Wanninkhof polynomial fit
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dDiffParam
  read (ParFilUnit, *, iostat=ifile)
  if (dDiffParam .eq. 1) then
!    read in VOC molar volume at normal boiling point (mL/mol)
    read (ParFilUnit, *, iostat=ifile) dMV
    if (ifile .ne. 0) then
      error = .true.
```

```
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading molar volume.'
      call ErrorOutput(vpos, offset)
    endif
  elseif (dDiffParam .eq. 2) then
!    read in VOC density in g/mL
    read (ParFilUnit, *, iostat=ifile) dDensity
    if (ifile .ne. 0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
            ':  Error reading VOC density.'
      call ErrorOutput(vpos, offset)
    endif
  else
    error = .true.
    diffsol_error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Diffusivity parameterization .ne. (1 .or. 2).'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then solubility parameterization
! 1 for exp(-(A-B/T))
! 2 for Wanninkhof polynomial fit
! 3 for direct entry
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dSolParam
  read (ParFilUnit, *, iostat=ifile)
  if (dSolParam .eq. 1) then
!    read in VOC molar volume at normal boiling point (ml/mol)
    read (ParFilUnit, *, iostat=ifile) dSolA, dSolB
    if (ifile .ne. 0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading exponential solubility coefficients.'
      call ErrorOutput(vpos, offset)
    endif
  elseif (dSolParam .eq. 2) then
!    read in four wanninkhof Sc parameterization coefficients
    read (ParFilUnit,*,iostat=ifile)dwa0,dwa1,dwa2,dwb0,dwb1,dwb2,dSalinity
    if (ifile .ne. 0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading Wanninkof solubility coefficients.'
      call ErrorOutput(vpos, offset)
    endif
  elseif (dSolParam .eq. 3) then
!   read in solubility directly
    read (ParFilUnit, *, iostat=ifile) dSolubility
    if (ifile .ne. 0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading solubility.'
      call ErrorOutput(vpos, offset)
    endif
  else
```

```
    error = .true.
    diffsol_error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Solubility parameterization .ne. (1 .or. 2 .or. 3).'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then degradation rate (1/sec)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dDegradationRate
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading VOC degradation rate.'
    call ErrorOutput(vpos, offset)
  endif
! read in header line, then tolerance for Runge-Kutta
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dTol
  if (ifile .ne. 0) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading Runge-Kutta tolerance.'
    call ErrorOutput(vpos, offset)
  endif
! read title, comment line header and read title and comments, if they exist
  read (parfilunit, *, iostat=ifile)
  title = '     '
  if (ifile .eq. 0) then
    read (parfilunit, '(72a)', iostat=ifile) dtitle
    if (ifile .eq. 0) then
      read (parfilunit, '(72a)', iostat=ifile) dcomment(1)
      if (ifile .eq. 0) then
        read (parfilunit, '(72a)', iostat=ifile) dcomment(2)
      endif
    endif
  endif  ! End of comment line input
!  read in the weir/waterfall information
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile)
  read (ParFilUnit, *, iostat=ifile) dWeirNum
  if ((ifile .ne. 0) .or. (dWeirNum .lt. 0)) then
    error = .true.
    ierror = ierror + 1
    if (ierror .eq. 1) call OpenErrorWindow
    write (msg0, '(a,i2,a)') 'Input Error #', ierror, &
         ':  Error reading number of weir points.'
    call ErrorOutput(vpos, offset)
  endif
! check to see if there are any weirs, if not, go on to check for errors
  if (dWeirNum .gt. 0) then
! allocate river data array in preparationfor reading in river physical data
    allocate (dWeirPoints(dWeirNum,5))
    allocate (dWeirType(dWeirNum))
!    read in the weir information
! #  WeirLocation (km)  L(width)  h  s  Ht  WeirType
!    (see WeirEffic in physicchem.f90 for details)
    read (ParFilUnit, *, iostat=ifile)
    read (ParFilUnit, *, iostat=ifile)
    do i = 1, dWeirNum
      read (ParFilUnit, *, iostat=ifile) idum, (dWeirPoints(i,j), j=1,5), &
          dWeirType(i)
```

```
      if (ifile .ne. 0) then
        error = .true.
        ierror = ierror + 1
        if (ierror .eq. 1) call OpenErrorWindow
        write (msg0, '(a,i2,a,i2)') 'Input Error #', ierror, &
         ':  Error reading reach data point #', i
        call ErrorOutput(vpos, offset)
      endif
    end do
  endif
!  end of weir input information
! read in data output filename if present
  read (ParFilUnit, *, iostat=ifile) tempfile
! close the parameter file, end of data input.
  close (parfilunit)
  ierror = 0
  if (.not. error) then
    if (dRiverLength .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in RIVER LENGTH.  Length must be > 0'
      call ErrorOutput(vpos, offset)
    endif
    if (dRiverData(dReachPoints,1) .ne. dRiverLength) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a,a)') ' Error in river physical data.  ',&
            'Maximum length must be = Total River Length'
      call ErrorOutput(vpos, offset)
    endif
    if (dMolWeight .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in MOLECULAR WEIGHT.  Weight must be >0'
      call ErrorOutput(vpos, offset)
    endif
    if (dSurfaceTemp .lt. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)') &
          ' Error in river temperature.  Temperature must be > 0.0 C'
      call ErrorOutput(vpos, offset)
    endif
    if (dAirTemp .le. -50.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)') &
         ' Error in AIR TEMPERATURE.  Air Temp must be > -50 C'
      call ErrorOutput(vpos, offset)
    endif
    if (dDegradationRate .lt. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in DEGRADATION RATE.  Rate must be >= 0.0 1/s'
      call ErrorOutput(vpos, offset)
    endif
    if (dWindSpeed .le. 0.0d0) then
      ierror = ierror + 1
```

```
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in WIND SPEED.  WIND SPEED must be >= 0'
      call ErrorOutput(vpos, offset)
    endif
    if ((drel_hum .le. 0.0d0) .or. (dRel_Hum .gt. 100.0)) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in RELATIVE HUMIDITY.  0 .GE. RH .LE. 100'
      call ErrorOutput(vpos, offset)
    endif
    if (dAtmosPress .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in ATMOSPHERIC PRESSURE.  PRESSURE must be .GT. 0'
      call ErrorOutput(vpos, offset)
    endif
    if (dAtmVOCConc .lt. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in ATMOS, VOC CONCENTRATION.  CONCENTRATION must be .GE. 0'
      call ErrorOutput(vpos, offset)
    endif
    if (dTol .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in TOLERANCE.  TOLERANCE must be .GT. 0'
      call ErrorOutput(vpos, offset)
    endif
    if (dRiverData(1,1) .ne. 0.0d0) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in river physical data point # 1, distance .NE. 0.0'
      call ErrorOutput(vpos, offset)
    endif
    if ((dRiverData(1,3) .le. 0.0d0) .or. (dRiverData(1,4) .le. 0.0d0) .or. &
        (dRiverData(1,6) .lt. 0.0d0)) then
      error = .true.
      ierror = ierror + 1
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in river phys. data point # 1, depth, width, or conc. <= 0.0'
      call ErrorOutput(vpos, offset)
    endif
    do i = 2, dReachPoints - 1
        if (dRiverData(i,1) .lt. dRiverData(i-1,2)) then
!         Distance must increase from end to beginning points
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a, i2, a)') &
            ' Error in river physical data point #', i, ' distance decreases'
          call ErrorOutput(vpos, offset)
        endif
        if (dRiverData(i,2) .lt. dRiverData(i,1)) then
          error = .true.
```

```
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a, i2, a)') ' Error in data point #', i+1, &
               ' distances decrease from start-end'
          call ErrorOutput(vpos, offset)
        endif
        if ((dRiverData(i,2)-dRiverData(i,1) .gt. 0.0d0) .and. &
            (dRiverData(i,2)-dRiverData(i,1) .lt. &
                      1.0d-3*min(5.0d0, dRiverData(i,4)))) then
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a, i2, a)') ' Error in data point #', i+1, &
               ' dist. src length too small'
          call ErrorOutput(vpos, offset)
        endif
        if ((dRiverData(i,3).le.0.0d0).or.(dRiverData(i,4).le.0.0d0)) then
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a,i2,a)') ' Error in river physical data point #',i,&
            ' depth or width <= 0.0'
          call ErrorOutput(vpos, offset)
        endif
        if (dRiverData(i,3) .le. 0.119d0) then
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a,i2,a)') ' Error in river physical data point #',i,&
                ' depth <= 0.119 m, cannot parameterize kL'
          call ErrorOutput(vpos, offset)
        endif
        if (dRiverData(i,6) .lt. 0.0d0) then
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a,i2,a)') 'Error in river physical data point #',i,&
              ' conc < 0.0'
          call ErrorOutput(vpos, offset)
        endif
        if ((dRiverData(i,1) .lt. dRiverData(i,2)) .and. &
            (dRiverData(i,6) .lt. 0.0d0)) then
          error = .true.
          ierror = ierror + 1
          if (ierror .eq. 1) call OpenErrorWindow
          write (msg0,'(a,i2,a)') 'Error in river physical data point #',i,&
                'distributed source conc < 0.0'
          call ErrorOutput(vpos, offset)
        endif
      end do
!     now check the weir data, if there is any
      if (dWeirNum .gt. 0) then
        do i = 1, dWeirNum
          if ((dWeirType(i) .lt. 1) .or. (dWeirType(i) .gt. 3)) then
            error = .true.
            ierror = ierror + 1
            if (ierror .eq. 1) call OpenErrorWindow
            write (msg0,'(a,i2,a)') ' Error in weir #', i, &
               '  invalid weir type (1, 2, or 3)'
            call ErrorOutput(vpos, offset)
          endif
          if ((dWeirPoints(i,1) .le. 0.0d0) .or. &
              (dWeirPoints(i,1) .ge. dRiverLength)) then
            error = .true.
            ierror = ierror + 1
            if (ierror .eq. 1) call OpenErrorWindow
            write (msg0,'(a,i2,a)') ' Error in weir #', i, '  invalid position'
```

```
          call ErrorOutput(vpos, offset)
        endif
      end do
    endif
!    now check diffusivity and solubility for validity (both must be > 0)
    if (dDiffParam .eq. 2) then
      dMV = dDensity / dMolWeight
    endif
    tempc = dSurfaceTemp
!    Absolute viscosity (mu) in gm-cm/sec is calc. from temper in deg-C.
!    The underlying data is from data from CRC 63rd edition.
!    The polynomial fit was done in the spreadsheet KINVISC.WB1 in QDATA
    mu=1.7825047d0-5.75921d-2*tempc+1.11378d-3*tempc**2-9.55317d-06*tempc**3
    if (dMV .gt. 0.0d0) then
      dDiffusivity = (4.7199d-07*(tempc+2.7316d2))/(mu*(dMV**0.6d0))
    else
      dDiffusivity = -1.0d0
    endif
    if (dDiffusivity .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
      if (ierror .eq. 1) call OpenErrorWindow
      write (msg0,'(a)')
      ' Error in Diffusivity or parameterization.  Diffusivity must be > 0'
      call ErrorOutput(vpos, offset)
    endif
    if (dDiffusivity .gt. 0.1) then
!      diffusivity of VOC in water>larger than diffus. of heat
      error_msg = 'Parameter File Warning'
      msg0 = &
     'Calculated Diffusivity > 1.0 cm^2/sec!\nPlease check parameter file'C
      call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, error_msg)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      if (ProfileWindowOpen) then
        call WindowSelect(ProfileWinUnit)
        call WindowFontColour(NotSet, TextWhiteBold)
        call WindowClearArea(0,0,9999,9999)
        focusprofile = 2
        call ViewRiverProfile(lret)
      endif
    endif
    select case (dSolParam)
      case (1, 2)
        tempk = dSurfaceTemp + 2.7315d2
        if (dSolParam .eq. 1) then
          dSolubility = 1.0d0/dexp(dsolA - dsolB/(tempk))
!         following Robbins et al., mol/m^3-atm
        else
!          calculating Bunsen solubilities assuming salinity = zero
!          using Wanninkhof relation
          logalpha = dwa0 + dwa1*(1.0d2/tempk) + dwa2*dlog(1.0d-2*tempk) + &
            dsalinity * (dwb1 + dwb2*(1.0d-2*tempk) + dwb2*(1.0d-2*tempk)**2)
          alpha = dexp(logalpha)
          sol1 = alpha /(8.20575d-2 * (tempk))   ! mol/L-atm
          dSolubility = 1.0d3*sol1                 ! mol/m^3-atm
        endif
      case (3)
        continue
    end select
    if (dSolubility .le. 0.0d0) then
      ierror = ierror + 1
      error = .true.
```

```
      if (ierror.eq.1) open(ErrWinUnit,file='USER',title='PARAM FILE ERRORS')
      write (msg0,'(a)')
      ' Error in Solubility or parameterization.  Solubility must be > 0'
      call ErrorOutput(vpos, offset)
    endif
    if (0.001*dSolubility .gt. 100.0) then
      error_msg = 'Parameter File Warning'
      msg0 = &
    'Calculated Solubility > 100.0 mol/L-atm!\nPlease check parameter file'C
      call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, error_msg)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      if (ProfileWindowOpen) then
        call WindowSelect(ProfileWinUnit)
        call WindowFontColour(NotSet, TextWhiteBold)
        call WindowClearArea(0,0,9999,9999)
        focusprofile = 2
        call ViewRiverProfile(lret)
      endif
    endif
!    Now check to make sure there aren't more than 25 sources or the data
!    output string will be longer than 255 characters and we'll get an error
    isource = 0
    do i = 1, ReachPoints
      if ((dRiverData(i,5) .gt. 0.0d0) .and. &
          (RiverData(i,6) .gt. 0.0d0)) then
        isource = isource + 1
      endif
    end do
    if (isource .gt. 25) then
      ierror = ierror + 1
      error = .true.
      if (ierror.eq.1) open(ErrWinUnit,file='USER',title='PARAM FILE ERRORS')
      write (msg0,'(a)')
      ' Too many sources.  Total number must be <= 25'
      call ErrorOutput(vpos, offset)
    endif
! now finally try to open the output data file if tempfile .ne. '*.*'
    if ((.not. error) .and. (tempfile .ne. '*.*')) then
      FileSave = DatFile_Out
      DatFile_Out = tempfile
      call Save_VOC
    endif
    if (.not. error) then
!   NEED TO RESET THE DIFFUS AND SOL COEFFICIENTS BEFORE RETURNING
      Density = dDensity
      MolarVolume = dMV
      sola = dsola
      solb = dsolb
      wa0 = dwa0
      wa1 = dwa1
      wa2 = dwa2
      wb0 = dwb0
      wb1 = dwb1
      wb2 = dwb2
      salinity = dsalinity
      DiffParam = dDiffParam
      SolParam = dSolParam
      Diffusivity = dDiffusivity
      Solubility = dSolubility
      rel_hum = drel_hum
      RiverLength = dRiverLength
      SurfaceTemp = dSurfaceTemp
```

```
      AtmVOCConc = dAtmVOCConc
      DegradationRate = dDegradationRate
      AirTemp = dAirTemp
      WindSpeed = dWindSpeed
      AtmosPress = dAtmosPress
      MolWeight = dMolWeight
      Tolerance = dTol
      ReachPoints = dReachPoints
      Reaches = dReaches
      DistributedSources = dDistributedSources
      Title = dTitle
      comment(1) = dcomment(1)
      comment(2) = dcomment(2)
      if (allocated(RiverData)) deallocate(RiverData)
      allocate(RiverData(ReachPoints,7))
      isource = 0
      do i = 1, ReachPoints
        do j = 1, 6
          RiverData(i,j) = dRiverData(i,j)
        end do
!        need to convert flow units if they were entered as cfm
        if (FlowUnits .eq. 1) then
!          Flow entered in param file as cfm so we need to convert to m^3/s
          RiverData(i,5) = dRiverData(i,5)/2118.9d0
        endif
        if ((RiverData(i,5) .gt. 0.0d0) .and. &
            (RiverData(i,6) .gt. 0.0d0)) then
          isource = isource + 1
          RiverData(i,7) = isource
        endif
      end do
      deallocate (dRiverData)
      if (dWeirNum .gt. 0) then
        WeirNum = dWeirNum
        if (allocated(WeirPoints)) deallocate(WeirPoints)
        if (allocated(WeirType)) deallocate(WeirType)
        if (allocated(WeirEffic)) deallocate(WeirEffic)
        allocate(WeirType(WeirNum))
        allocate(WeirPoints(WeirNum,5))
        allocate(WeirEffic(WeirNum))
        WeirType = dWeirType
        WeirPoints = dWeirPoints
      else
        WeirNum = 0
      endif
      if (allocated(dWeirPoints)) deallocate (dWeirPoints)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      call WindowFontColour(NotSet, TextWhiteBold)
      call WindowClearArea(0,0,9999,1900)
      msg0 = 'Parameter file successfully read'
      call WindowOutString(0, 300, msg0)
      msg0 = 'View input parameters enabled'
      call WindowOutString(0, 600, msg0)
      msg0 = 'Model ready to run'
      call WindowOutString(0, 900, msg0)
      call WBitmapGet(MainWinBitmap,0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      call menuset(1)  ! turn menus on
!      some housekeeping stuff
      OpenWindow = .true.    ! tells Plot subroutine to open graphics window
      ParamFileOK = .true.   ! successfuly read parameter file
      if (errorwindow) then
        call WindowCloseChild(ErrWinUnit)
```

```
        errorwindow = .false.
      endif
      return
    else
      errorwindow = .true.
      msg0 = &
'Errors in param file!\nPlease check file\nErrors in error output window'C
      call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      if (ProfileWindowOpen) then
        call WindowSelect(ProfileWinUnit)
        call WindowFontColour(NotSet, TextWhiteBold)
        call WindowClearArea(0,0,9999,9999)
        focusprofile = 2
        call ViewRiverProfile(lret)
      endif
      ParamFileOK = .false.     ! parameter file not read
      call WBitmapGet(ErrWinBitmap,0)
      call menuset(0)  ! set menus off
      return
    endif
  else
    msg0='Errors in param file\nFILE NOT READ!!!!\nPlease check param file.'C
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    errorwindow = .true.
    ParamFileOK = .false.     ! successfuly read parameter file
    call WBitmapGet(ErrWinBitmap,0)
    call menuset(0)  ! set menus off
    return
  endif
end subroutine read_parfile

subroutine save_voc (forcewrite)
!saves an mtbe data file, checks to make sure existing file not overwritten.
  use msflib
  use winteracter
  use voccom
  implicit none
  logical forcewrite    !  if .true. then overwrite any existing file
  integer iret, ierr, ibutton
  integer*2 delval
  logical LExist, lret
!* open Data output file
  inquire (file=DatFile_Out, exist=LExist)
  if ((LExist) .and. (.not. forcewrite)) then
    msg0 = trim(DatFile_out)
    msg1 = ' OVERWRITE FILE? '
    call wmessagebox (YesNo, ExclamationIcon, CommonNo, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
```

```
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    if (WInfoDialog(4) .eq. CommonYes) then
      delval = delfilesqq(DatFile_Out)
      ierr = 0
      open(DatFilUnit, file=DatFile_Out, status='NEW', iostat=ierr)
      if (ierr .ne. 0) then
        write(msg0, '(a,i3)') 'Unknown File I/O Error, IOSTAT value =', ierr
        msg1 = ' ERROR OPENING FILE '
        call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
        call WindowSelect(0)
        call WBitmapPut(MainWinBitmap, 0, 0)
        if (ErrorWindow) then
          call WindowSelect(ErrWinUnit)
          call WBitmapPut(ErrWinBitmap, 0, 0)
        endif
        if (ProfileWindowOpen) then
          focusprofile = 2
          call ViewRiverProfile(lret)
        endif
        return
      endif
      DatOut = .true.
      VOC_File_sav = .true.
    endif
  else
    if (LExist) delval = delfilesqq(DatFile_Out)
    ierr = 0
    open(DatFilUnit, file=DatFile_Out, status='NEW', iostat=ierr)
    if (ierr .ne. 0) then
      write(msg0, '(a,i3)')  ' Unknown File I/O Error, IOSTAT value = ', ierr
      msg1 = ' ERROR OPENING FILE '
      call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      if (ProfileWindowOpen) then
        focusprofile = 2
        call ViewRiverProfile(lret)
      endif
      return
    endif
    DatOut = .true.
    VOC_File_sav = .true.
  endif
  return
end subroutine save_voc

subroutine SetTicks (ticks, iticks, xmin, xmax)
implicit none
integer iticks
real*4 ticks(iticks), xmin, xmax
integer i
  do i = 1, iticks
    ticks(i) = xmin + (xmax-xmin) * float(i-1) / float(iticks-1)
    if (ticks(i) .gt. 1.0) then
      if (xmax .ge. 10.0) then
        ticks(i) = float(floor(ticks(i)))
!       make sure we plot ticks on even numbers
      else
```

```
        ticks(i) = 0.1*float(floor(10.0*ticks(i)))
!       retain the tenth digit in the tick
      endif
    elseif (ticks(i) .gt. 0.10) then
      ticks(i) = 0.01*float(floor(100.0*ticks(i)))
!     round xticks(i) to the nearest tenth
    elseif (ticks(i) .gt. 0.010) then
      ticks(i) = 0.001*float(floor(1000.0*ticks(i)))
!     round xticks(i) to the nearest hundredth
!     add in more elseifs here if we decide to run river streams with
!     intervals smaller than a 0.01 m
    endif
  end do
return
end subroutine SetTicks

real*8 function sol_calc(tempc, iform)
use VOCcom
implicit none
integer iform
real*8 tempc
real*8 tempk, logalpha, alpha, sol1
tempk = tempc + 2.7316d2
select case (iform)
  case(1)
    sol_calc=1.0d0/dexp(solA-solB/(tempk)) ! from Robbins et al., mol/m^3-atm
  case(2)
!   calc Bunsen sol. assuming salinity = zero using Wanninkhof relation
    logalpha = wa0 + wa1*(1.0d2/tempk) + wa2*dlog(1.0d-2*tempk) + &
            salinity * (wb1 + wb2*(1.0d-2*tempk) + wb2*(1.0d-2*tempk)**2)
    alpha = dexp(logalpha)
    sol1 = alpha /(8.20575d-2 * (tempk))   ! mol/L-atm
    sol_calc = 1.0d3*sol1                 ! mol/m^3-atm
  case(3)
    sol_calc = solubility
end select
return
end function sol_calc

subroutine Start_Model(lret)
  use dfmt
  use msflib
  use winteracter
  use voccom
  implicit none
  integer(kind=4)iret
  external go_vocdrv
  integer(kind=4) arg2, threadhandle,threadid, i
  logical ts_error, lret
  ts_error = .false.
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore starting model'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  i = 1
```

```
  if (.not. lrunning) then
    lrunning = .true.
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    call WindowFontColour(NotSet, TextWhiteBold)
    call WindowClearArea(0,0,9999,1900)
    call WindowOutString(0, 300, title)
    call WindowOutString(0, 600, comment(1))
    call WindowOutString(0, 900, comment(2))
!   need to check file I/O before detach thread or it screws up graphics
    if (.not. DatOut) then
      msg0 = 'Model output will not be saved to data file'
    else
      msg0 = 'Model output saved to data file: '//trim(DatFile_out)
    endif
    call WindowOutString(0, 1300, msg0)
    msg1 = 'Model Status:  Running'
    call WindowOutString(0, 1600, msg1)
!   Note:  From DVF example CHAOS.F90 (Launch_ChaosDrive File:Lchchsdr.f90)
!   added the type LOC() around ARG2.  Keeps thread running.
    call menusetrunning(0)
    arg2=0
    threadhandle = createthread(0,0,go_vocdrv,loc(arg2),0,threadid)
  else
    call WindowFontColour(TextRedBold, TextWhiteBold)
    msg0 = &
' Model already running!   Press <Run\\Stop> to terminate before restarting'
    call WindowOutString(0, 1600, msg0)
    call WindowFontColour(TextBlack, TextWhiteBold)
  endif
  return
end subroutine Start_Model

subroutine ViewRiverProfile (bool_in)
  use msflib
  use winteracter
  use voccom
  implicit none
  integer id, callbacktype
  logical bool_in, lret
  integer  retcode, ipnts, i, j, k, ierr
  integer  iscale, iticks
! real constants needed by Winteracter (can't use real*8!)
  real xmin, xmax, ymin, ymax, xticks(5), yticks(5)
  real maxdepth, maxwidth, maxflow, minflow, findmaxval, vocmin, vocmax, &
       minvel, maxvel
  real xwindowleft, xwindowright, deltaywindow, yspacing, ylast
  real xDepth(:), yDepth1(:), yDepth2(:), yWidth1(:), yWidth2(:), yFlow(:), &
       yVelocity(:), xVOCInput(:), yVOCInput(:)
  allocatable xDepth, yDepth1, yDepth2, yWidth1, yWidth2, yFlow, &
       yVelocity, xVOCInput, yVOCInput
  external findmaxval
  call unusedqq(bool_in)
! set these data so they get reinitialized every time we call the subroutine
  xwindowleft = 0.0
  xwindowright = 1.0
  deltaywindow = 0.2
  yspacing = 0.0
  ylast = 1.0
  xmin = 0.0e0
  xmax = sngl(RiverLength)
  iticks = 5
! fill in the tickmark array for the x-axis
  do i = 1, iticks
    xticks(i) = sngl(RiverLength) * float(i-1) / float(iticks-1)
  end do
! allocate the temporary REAL*4 arrays used by Winteracter for data plotting
```

```
  allocate (xDepth(ProfilePoints), yDepth1(ProfilePoints), &
            yDepth2(ProfilePoints), yWidth1(ProfilePoints), &
            yWidth2(ProfilePoints), yFlow(ProfilePoints), &
            yVelocity(ProfilePoints)), (xVOCInput(VOCInputPoints), &
            yVOCInput(VOCInputPoints))
! find the maximum depth and width
  maxdepth = findmaxval(3)   !  RiverData has depth in column 3
  maxwidth = findmaxval(4)   !  RiverData has depth in column 4
  maxflow = 0.0
  minflow = 1.0e6
  maxvel = 0.0
  minvel = 1.0e6
! put depth, width, flow, velocity data in arrays for plotting
  do j = 1, ProfilePoints
    xDepth(j) = sngl(FlowVolumes(1,j))
    yDepth1(j) = 0.0
    yDepth2(j) = sngl(-1.0 * FlowVolumes(3,j))
    yWidth1(j) = -0.5 * FlowVolumes(4,j)
    yWidth2(j) = 0.5 * FlowVolumes(4,j)
    yFlow(j) = sngl(FlowVolumes(2,j))
    yVelocity(j) = yFlow(j)/(35.3*60.0*-1.0*yDepth2(j)*FlowVolumes(4,j))
    if (maxflow .lt. yFlow(j)) maxflow = yFlow(j)
    if (minflow .gt. yFlow(j)) minflow = yFlow(j)
    if (maxvel .lt. yVelocity(j)) maxvel = yVelocity(j)
    if (minvel .gt. yVelocity(j)) minvel = yVelocity(j)
  end do
! put VOC input data in arrays for plotting
! file i/o added 7/30/03 to get VOC Input data for plotting
  open (33, file = 'VOCInputData.dat')
  do j = 1, VOCInputPoints
    xVOCInput(j) = VOCInputs(1,j)
    yVOCInput(j) = VOCInputs(2,j)
    write (33, '(2e15.5)') xVOCInput(j), yVOCInput(j)
  end do
  close (33)
! find log of VOC input maximum and minimum
  vocmin = 100.0
  vocmax = -100.0
  do i = 1, splinepnts
    if (spl_VOCInputs(2,i) .gt. 0.0) then
      if (vocmin .gt. sngl(dlog10(spl_VOCInputs(2,i)))) &
            vocmin = sngl(dlog10(spl_VOCInputs(2,i)))
      if (vocmax .lt. sngl(dlog10(spl_VOCInputs(2,i)))) &
            vocmax = sngl(dlog10(spl_VOCInputs(2,i)))
    endif
  end do
  vocmin = float(int(vocmin)-2)
  vocmax = float(int(vocmax)+1)
! set yVOCInput to hold log(VOC Input data) set zeros to VOCMIN for plotting
  do j = 1, VOCInputPoints
    if (yVOCInput(j) .gt. 0.0) then
      yVOCInput(j) = log10(yVOCInput(j))
    else
      yVOCInput(j) = vocmin
    endif
  end do
! open up the window with the river profile
  if (.not. ProfileWindowOpen) then
    profilewindow%flags  = SysMenuOn + MinButton + MaxButton
    profilewindow%x      = 20
    profilewindow%y      = 20
    profilewindow%width  = 800
    profilewindow%height = 650
    profilewindow%title  = 'Graphical Plots of River Profile Data'
    call WindowOpenChild(ProfileWindow, ProfileWinUnit)
    if (ProfileWinUnit .eq. -1) then
      msg0 = 'Could not open window for display of river profile'
```

```
      msg1 = 'Window Error'
      call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
      call WindowSelect(0)
      call WBitmapPut(MainWinBitmap, 0, 0)
      if (ErrorWindow) then
        call WindowSelect(ErrWinUnit)
        call WBitmapPut(ErrWinBitmap, 0, 0)
      endif
      return
    endif
    ProfileWindowOpen = .true.
  endif
  if (focusprofile .eq. 1) then
    call WindowRaise(ProfileWinUnit)
  elseif (focusprofile .eq. 2) then
    call WindowSelect(ProfileWinUnit)
    focusprofile = 0
  endif
! this is the depth profile plot
  ymin = (-1.0 * MaxDepth - 0.05 * maxdepth)
  ymax = 0.05
  call PlotProfile(xwindowleft, xwindowright, ylast-deltaywindow, ylast, &
                   deltaywindow, xmin, xmax, ymin, ymax, xDepth, yDepth1, &
                   yDepth2, ProfilePoints, 2, 159, 'Distance (km)', &
                   '   Depth-m')
! 159 is bright blue, see col256 in wint/demos
! now plot the width profile
  ylast = ylast - deltaywindow
  ymin = -0.5*MaxWidth - 0.05*Maxwidth
  ymax = 0.5*MaxWidth + 0.05*MaxWidth
  call PlotProfile(xwindowleft, xwindowright, ylast-deltaywindow, ylast, &
       deltaywindow, xmin, xmax, ymin, ymax, xDepth, yWidth1, yWidth2, &
       ProfilePoints, 2, 95, 'Distance (km)', '  Width-m')
! now plot the flow volumes
  ylast = ylast - deltaywindow
  ymin = MinFlow - 0.05 * MinFlow
  ymax = MaxFlow + 0.05 * MaxFlow
  call PlotProfile(xwindowleft, xwindowright, ylast-deltaywindow, ylast,&
deltaywindow, xmin, xmax, ymin, ymax, xDepth, yFlow, &
yFlow, ProfilePoints, 1, 191, 'Distance (km)', &
' Q-m3/s')
! 191 is bright magenta, see col256 in wint/demos
! now plot the velocities
  ylast = ylast - deltaywindow
  ymin = MinVel - 0.05 * MinVel
  ymax = MaxVel + 0.05 * MaxVel
  call PlotProfile(xwindowleft, xwindowright, ylast-deltaywindow, ylast,&
                   deltaywindow, xmin, xmax, ymin, ymax, xDepth, &
                   yVelocity, yVelocity, ProfilePoints, 1, 31, &
                   'Distance (km)', '  Veloc.-m/s')
! 31 is bright red, see col256 in wint/demos
! now plot the VOC input series
  ylast = ylast - deltaywindow
  call PlotProfile(xwindowleft, xwindowright, ylast-deltaywindow, ylast, &
                   deltaywindow, xmin, xmax, vocmin, vocmax, xVOCInput, &
                   yVOCInput, yVOCInput, VOCInputPoints, 1, 127, &
                   'Distance (km)', ' VOC-mol/km')
!  now we can deallocate all the data arrays
  deallocate (xDepth, yDepth1, yDepth2, yWidth1, yWidth2, yFlow, &
              yVelocity, xVOCinput, yVOCInput)
  focusoutput = 1
  focusconcfracoutput = 1
!  this next line will cause the profile window to be brought to the top with each call from the
! main menu.  If you want it to stay in the background with each menu call, set FOCUSPROFILE = 2
  focusprofile = 1
  return
end subroutine ViewRiverProfile
```

```
subroutine VOC_Params (bool_in)
  use msflib
  use winteracter
  use voccom
  implicit none
  INCLUDE 'RESOURCE.FD'
  logical lret, err, bool_in
  integer iret, ierr, iloop
  call unusedqq(bool_in)
  ierr = 0
  msg0 = ''
  msg1 = ''
  if (MenuActive) then
    msg0 = 'Please close open set-up menu\nbefore opening new window'C
    msg1 = 'Window Error'
    call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
    call WindowSelect(0)
    call WBitmapPut(MainWinBitmap, 0, 0)
    if (ErrorWindow) then
      call WindowSelect(ErrWinUnit)
      call WBitmapPut(ErrWinBitmap, 0, 0)
    endif
    if (ProfileWindowOpen) then
      focusprofile = 2
      call ViewRiverProfile(lret)
    endif
    return
  endif
  err = .true.
  menuactive = .true.
! initialize the dialog box
  call WDialogLoad(IDD_VOCParams)
! write the current molecular weight into the dialog box edit field
! write the current init. concentration into the dialog box edit field
! write the current atmos. concentration into the dialog box edit field
  write(err_str(1),'(f9.3)') MolWeight
  write(err_str(2),'(f8.3)') AtmVOCConc
  write(err_str(3),'(e12.4)') Diffusivity
  write(err_str(4),'(e12.4)') Solubility
  write(err_str(5),'(e12.4)') DegradationRate
  call WDialogPutString(IDC_MolWeight, err_str(1))
  call WDialogPutString(IDC_AtmosConc, err_str(2))
  call WDialogPutString(IDC_Diffusivity, err_str(3))
  call WDialogPutString(IDC_Solubility, err_str(4))
  call WDialogPutString(IDC_DegradationRate, err_str(5))
! bring up the dialog box
  call WDialogShow(-1, -1, 0, 2)
  return
end subroutine VOC_Params

subroutine vocdrv
  use dfmt
  use msflib
  use winteracter
  use voccom
  use cfunccom
  implicit none
  include 'resource.fd'
  type(win_message) message
  logical lret
  integer i, j, k, irelab, ifail, iret, numpts, ierr, outputstep, iweir
  real*8 conc(:), work(:,:), sumconcfrac(:)
! conc(num_eqs), work(num_eqs, 20)
  allocatable conc, work, sumconcfrac
  real*8 time_beg, time_end, total_time, clast, convunits, degc, tol, &
         tm, tm0, tm1, &
         gasexchange, degradation, flowconv, concsave, xbeg
```

```
! real*8 external functions
  real*8 cfunc_net, cfunc_comp, dqdt, mass_input, kl_func, kd_func, cs_func
! integer external funtions
  integer mass_source
  external cfunc_net, cfunc_comp, dqdt, mass_input, kl_func, kd_func, &
           cs_func, mass_source, xyplot, xyplot2
! arrays used for plotting data
  real*4 plotarray(2,2,2), plotarray2(:,:,:), total_fraction
  allocatable plotarray2
! allocate array used to hold individ concs from each src and reset to zero
!  SourceNumber one greater than number of actual sources.
!  Extra source is for gas exchange
!  allocate(sourceconc(SourceNumber+1))
!  allocate(sourceconc0(SourceNumber+1))
  allocate(plotarray2(2,2,SourceNumber+1))
  num_eqs = sourcenumber + 3  ! one eq for each src and flow and total conc.
  allocate(conc(num_eqs), work(num_eqs,20), sumconcfrac(num_eqs-3))
  outputstep = 2
  if (WeirNum .gt. 0) then
    iweir = 1   ! start out with the first weir
  else
    iweir = 0
  endif
!  clear out OldData array
  do i = 1, 2
    do j = 1, splinepnts
      do k = 1, 2
        OldData(i,j,k) = 0.0e0
      enddo
    enddo
  enddo
! graphics reinitialization
  PlotInit = .false.     ! reset PlotInit so that axes will be redrawn
  PlotInit2 = .false.    ! reset PlotInit2 so that axes will be redrawn
! set a constant for conversion of concentrations
  convunits = 1.0d3*molweight     ! converts moles/m^3 to ug/L
  if (FlowUnits .eq. 0) then
    flowconv = 1.0d0
  else
    flowconv = 2118.9             ! change m^3/sec to cfm
  endif
! scale factors used of generating the concentration plots
  MaxConc = log10(sngl(max(RiverData(1,6), cs_func(1)*convunits))) + 0.2e0
  if ((RiverData(1,6) .gt. 0.0d0) .and. (cs_func(1) .gt. 0.0d0)) then
    MinConc = log10(sngl(min(RiverData(1,6), cs_func(1)*convunits))) - 0.2e0
  else
    MinConc = -5.0e0
  endif
  MinLength = 0.0e0
  MaxLength = sngl(RiverLength)
! set the initial variables
  total_time = 0.0d0
  plotarray(1,1,1) = 0.0e0
  plotarray(1,1,2) = 0.0e0
  conc(2) = 1.0d-3 * RiverData(1,6) / molweight ! 0.001  ug/L to mol/m^3
  if (conc(2) .gt. 0.0d0) then
    plotarray(2,1,1) = log10(sngl(conc(2) * convunits))
  elseif (cs_func(1) .gt. 0.0e0) then
    plotarray(2,1,1) = log10(sngl(cs_func(1) * convunits))
  else
    plotarray(2,1,1) = -5.0e0
  endif
  if (cs_func(1) .gt. 0.0d0) then
    plotarray(2,1,2) = log10(sngl(cs_func(1) * convunits))
  else
    plotarray(2,1,2) = -5.0e0
  endif
```

```
!  now set up the initial source fractions, src 1=1.0d0, all others=0.0d0
  conc(3) = 1.0d0   ! conc(3) is fraction of source 1
  do i = 4, num_eqs
    conc(i) = 0.0d0
  end do
!  setup the initial values in OldData
  OldPoints = 0
! Now set up the arrays for plotting concentration fractions
  plotarray2(2,1,1) = 1.0e0
  do i = 2, SourceNumber+1
    plotarray2(2,1,i) = 0.0e0
  end do
  do i = 1, SourceNumber+1
    plotarray2(1,1,i) = 0.0e0
  end do
! If saving the data write out the header to the file
  if (DatOut) then
    if (FlowUnits .eq. 0) then
     write (DatFilUnit,'(a\)') '%Dist(km)        C(ug/L)     Flow(m3/s)    V(m/s)     kL(m/s)
VOCInput(mol)    cs(ug/L)   CurrSrc   Initial'
    else
     write (DatFilUnit,'(a\)') '%Dist(km)        C(ug/L)     Flow(cfm)     V(m/s)     kL(m/s)
VOCInput(mol)    cs(ug/L)   CurrSrc   Initial'
    endif
    do j = 1, SourceNumber
      write (datfilunit,'(4x,i2,a\)',iostat=ierr) j, '-Src'
    end do
    do k = 2, SourceNumber
      write (datfilunit,'(1x,i2,a\)',iostat=ierr) k, '-SumSrc'
    end do
    write (datfilunit,'(4x,a)',iostat=ierr) 'Sum-Air'
!   now write the initial data point
    do j = 1, num_eqs-3
      sumconcfrac(j) = 0.0d0
      do k = j+3, 4, -1
        sumconcfrac(j) = sumconcfrac(j) + conc(k)
      end do
    end do
    tm = 0.0d0
    cs = cs_func(1)    ! saturation concentration, f(T, Patm)  (moles/m^3)
    write (datfilunit,'(f12.6,6e13.4,i12,f10.4\)',iostat=ierr) &
      spl_xval(1), conc(2)*convunits, spl_flow(1)*flowconv, spl_veloc(2,1), &
      kl_func(1), tm, cs*convunits, currentsource, conc(3)
    do j = 4, Num_Eqs
      write (datfilunit,'(f10.4\)',iostat=ierr) conc(j)
    end do
    do k = 1, Num_Eqs-4
      write (datfilunit,'(f10.4\)',iostat=ierr) sumconcfrac(k)
    end do
    write (datfilunit,'(f10.4)',iostat=ierr) sumconcfrac(Num_Eqs-3)
   endif
!  begin master loop through the NUMPTS segments of river discretization
    do i = 2,  splinepnts
      time_beg = total_time
      time_end = total_time + spl_timestep(i-1)
      total_time = total_time + spl_timestep(i-1)
      conc(1) = spl_flow(i-1)
!      set values of physico-chemical parameters for use in CFUNC
!     these are passed as parameters in MODULE CFUNCCOM
      kl = kl_func(i-1)    ! gas transfer velocity, f(v, d, w)  (m/s)
      kd = kd_func(i-1)    !  biochemical degradation rate, f(T) (1/s)
      cs = cs_func(i-1)    ! saturation concentration, f(T, Patm)(moles/m^3)
      v = spl_veloc(2,i-1)  ! river velocity (m/s)
      qprime = dqdt(i-1)  ! change in flow with time (m^3/s^2)
!     total VOC input from confluences and
!     distributed sources (moles/sec-sec)
      ii = mass_input(i-1)
```

```
      currentsource = mass_source(i-1)
      j = currentsource
      tm = ii * (time_end - time_beg)
      depth = spl_depth(i-1)
!     set values of parameters needed by D02BAF
      ifail = 0
      tol = tolerance
!     note that TOLERANCE is set by user in dialog menu RUNTIME Parameters
      if ((iweir .gt. 0) .and. (iweir .le. WeirNum)) then
        concsave = conc(2)
        if (WeirPoints(iweir,1) .le. spl_xval(i)) then
          ! the previous segment had a weir
          ! adjust starting concentration to account for aeration by weir
          ! WeirEffic calculated in CalcFlowProfile using WeirEffic
          ! subrotuine in physicochem
          select case (WeirFlow)
            case (0)  ! no flow under weir/dam
              !adjust concentration
              WeirFrac = 1.0d0
              conc(2) = WeirEffic(iweir) * (cs - conc(2)) + conc(2)
            case (1)  ! flow under weir or dam predicted by Darcy's Law
              ! adjust concentration assuming 80% of total flow goes over
              ! weir, 20% goes under
              ! frac come from Darcy's Law assuming K=3000 ft/day (gravel)
              WeirFrac = 0.8d0
              conc(2) = &
                (WeirFrac*conc(1)*(WeirEffic(iweir)*(cs-conc(2))+conc(2))+&
                (1-0d0-WeirFrac)*conc(1)*conc(2))/conc(1)
          end select
          select case (ModelFunction)
            case (0)   ! netflux, only need to change alpha's if Cs > Cb
              if (cs .gt. concsave) then
                do j = 3, num_eqs-1
                  conc(j) = (conc(j)*concsave)/conc(2) !adjust conc fracs
                end do
                conc(num_eqs) = &
                    (conc(num_eqs)*concsave + (conc(2)-concsave))/conc(2)
                !   adjust air fraction
              endif
            case (1)  !component fluxes, see page 134 for discussion
              testsum = 0.0d0
              do j = 3, num_eqs-1
                !adjust concentration fractions for the sources
                conc(j) = &
                    conc(j)*concsave*(1-WeirFrac*WeirEffic(iweir))/conc(2)
                testsum = testsum + conc(j)
              end do
              !adjust air fraction
              conc(num_eqs) = &
                (conc(num_eqs)*concsave*(1-WeirFrac*WeirEffic(iweir)) + &
                 WeirFrac*WeirEffic(iweir)*cs) / conc(2)
              testsum = testsum + conc(num_eqs)
          end select
          iweir = iweir + 1
          !  increment iweir so that next time we don't do it again
        endif
      endif
      tm0 = conc(1) * conc(2)
!     decide which version of CONC to use, NET_FLUX or COMPONENT_FLUX
      select case (ModelFunction)
        case (0)   ! this is for NET FLUX Calculations
          call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                cfunc_net, work, ifail)
        case (1)   ! this is for COMPONENT FLUX Calculations
          call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                cfunc_comp, work, ifail)
        case default  ! defaults to NET FLUX
```

```
          call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                cfunc_net, work, ifail)
      end select
      if (ifail .eq. 3) then
!       IFAIL = 3 means tol is too small, try once with larger value
        tol = 10.0 * tol
        select case (ModelFunction)
          case (0)   ! this is for NET FLUX Calculations
            call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                    cfunc_net, work, ifail)
          case (1)   ! this is for COMPONENT FLUX Calculations
            call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                    cfunc_comp, work, ifail)
          case default  ! defaults to NET FLUX
            call d02baf(time_beg, time_end, num_eqs, conc, tol, &
                    cfunc_net, work, ifail)
        end select
      endif
      if (ifail .ne. 0) then
!        close(22)
        call WindowFontColour(TextBlack, TextWhiteBold)
        msg1 = 'Model Status:  Run halted on error'
        call WindowOutString(0, 1600, msg1)
        lrunning = .FALSE.
        menuactive = .false.
        if (DatOut) then
          close (DatFilUnit)
          DatOut = .false.
          msg0 = 'Closed Output File: '//trim(datfile_out)
          call WindowOutString(0, 1300, msg0)
        endif
!        save final screen into the bitmap for updating
        call WBitmapGet(MainWinBitmap,0)
!        deallocate(sourceconc)
!        deallocate(sourceconc0)
        deallocate(conc)
        deallocate(plotarray2)
        write (msg0, '(a, i2, a)') 'ifail = ', ifail, &
           '  Halt model and check parameters'
        msg1 = 'Error in D02BAF'
        call wmessagebox (OKOnly, ExclamationIcon, CommonOK, msg0, msg1)
        call WindowSelect(0)
        call WBitmapPut(MainWinBitmap, 0, 0)
        if (ErrorWindow) then
          call WindowSelect(ErrWinUnit)
          call WBitmapPut(ErrWinBitmap, 0, 0)
        endif
        if (ProfileWindowOpen) then
          focusprofile = 2
          call ViewRiverProfile(lret)
        endif
        call menuset(1)   ! turn all the menus back on
        call exitthread(0)    !exit code is 0
      endif
!      put new values into OldData
      OldPoints = OldPoints + 1
      OldData(1,OldPoints,1) = sngl(spl_xval(i-1))
      OldData(1,OldPoints,2) = sngl(spl_xval(i-1))
      if (conc(2) .gt. 0.0d0) then
        OldData(2,OldPoints,1) = log10(sngl(conc(2) * convunits))
      else
        OldData(2,OldPoints,1) = -5.0e0
      endif
      if (cs .gt. 0.0d0) then
        OldData(2,OldPoints,2) = log10(sngl(cs * convunits))
      else
        OldData(2,OldPoints,2) = -5.0e0
```

```
      endif
!      Beginning of data plotting routine
      if ((mod(i-1,outputstep) .eq. 0) .and. &
         ((splinepnts - i) .gt. outputstep)) then
        plotarray(1,2,1) = sngl(spl_xval(i+outputstep-1))
        plotarray(1,2,2) = sngl(spl_xval(i+outputstep-1))
        if (conc(2) .gt. 0.0d0) then
          plotarray(2,2,1) = log10(sngl(conc(2) * convunits))
        else
          plotarray(2,2,1) = -5.0e0
        endif
        if (cs_func(i-1) .gt. 0.0d0) then
          plotarray(2,2,2) = log10(sngl(cs_func(i-1) * convunits))
        else
          plotarray(2,2,2) = -5.0e0
        endif
        if ((MaxConc .lt. plotarray(2,2,1)) .or. &
            (MaxConc .lt. plotarray(2,2,2))) then
          PlotInit = .false.
          if (MaxConc .lt. plotarray(2,2,1)) &
             MaxConc = plotarray(2,2,1) + 0.2e0*(MaxConc-MinConc)
          ! conc. in log units,
          if (MaxConc .lt. plotarray(2,2,2)) &
             MaxConc = plotarray(2,2,2) + 0.2e0*(MaxConc-MinConc)
          ! so factor of two increase
        endif
        if ((MinConc .gt. plotarray(2,2,1)) .or. &
            (MinConc .gt. plotarray(2,2,2))) then
          PlotInit = .false.
          if (MinConc .gt. plotarray(2,2,1)) &
             MinConc = plotarray(2,2,1) - 0.2e0*(MaxConc-MinConc)
          ! conc in log units,
          if (MinConc .gt. plotarray(2,2,2)) &
             MinConc = plotarray(2,2,2) - 0.2e0*(MaxConc-MinConc)
          ! so factor of two decrease
        endif
        call xyplot(plotarray)
!        now move new valus of conc frac to old values in plotarray
        do j = 1, 2
          plotarray(1,1,j) = plotarray(1,2,j)
          plotarray(2,1,j) = plotarray(2,2,j)
        end do
!        put the new values into PLOTARRAY2
        total_fraction = 0.0e0
        do j = 1, SourceNumber+1
          plotarray2(1,2,j) = sngl(spl_xval(i+outputstep-1))
          plotarray2(2,2,j) = sngl(conc(j+2))
!         conc(j+2) is source fraction for source j
          total_fraction = total_fraction + plotarray2(2,2,j)
        end do
!    write (22, '(i7, f15.7, e17.7)') i, plotarray2(1,1,1), total_fraction
        call xyplot2(plotarray2, SourceNumber+1)
!       now move new valus of conc frac to old values in plotarray2
        do j = 1, SourceNumber+1
          plotarray2(1,1,j) = plotarray2(1,2,j)
          plotarray2(2,1,j) = plotarray2(2,2,j)
        end do
      endif
!     end of data output if then to plot every outputstepth point
!      write data to file every point if necessary
      if (DatOut) then
        do j = 1, num_eqs-3
          sumconcfrac(j) = 0.0d0
          do k = j+3, 4, -1
            sumconcfrac(j) = sumconcfrac(j) + conc(k)
          end do
        end do
```

```
        write (datfilunit,'(f12.6,6e13.4,i12,f10.4\)',iostat=ierr) &
          spl_xval(i-1), conc(2)*convunits, conc(1)*flowunits, v, kl, &
          tm, cs*convunits, currentsource, conc(3)
        do j = 4, Num_Eqs
          write (datfilunit,'(f10.4\)',iostat=ierr) conc(j)
        end do
        do k = 1, Num_Eqs-4
          write (datfilunit,'(f10.4\)',iostat=ierr) sumconcfrac(k)
        end do
        write (datfilunit,'(f10.4)',iostat=ierr) sumconcfrac(Num_Eqs-3)
      endif
!     check to see if HaltModel has been called
      if (.not. lrunning) then
        msg1 = 'Model Status:  Run halted by user'
        call WindowFontColour(TextRedBold, TextWhiteBold)
        call WindowOutString(0, 1600, msg1)
        call WindowFontColour(TextBlack, TextWhiteBold)
        menuactive = .false.
        if (DatOut) then
          close (DatFilUnit)
          DatOut = .false.
          msg0 = 'Closed Output File: '//trim(datfile_out)
          call WindowOutString(0, 1300, msg0)
        endif
        deallocate(conc, plotarray2)
        call WBitmapGet(MainWinBitmap,0)
!        This is the call that actually terminates the thread
        call menuset(1)   ! turn all the menus back on
        call exitthread(0)    !exit code is 0
      endif
!    end of main do loop over splinepnts
    end do
!  write final data point to file if necessary
  if (DatOut) then
    do j = 1, num_eqs-3
      sumconcfrac(j) = 0.0d0
      do k = j+3, 4, -1
        sumconcfrac(j) = sumconcfrac(j) + conc(k)
      end do
    end do
    write (datfilunit,'(f12.6,6e13.4,i12,f10.4\)',iostat=ierr) &
      spl_xval(splinepnts), conc(2)*convunits, conc(1)*flowunits, v, kl, &
      tm, cs*convunits, currentsource, conc(3)
    do j = 4, Num_Eqs
      write (datfilunit,'(f10.4\)',iostat=ierr) conc(j)
    end do
    do k = 1, Num_Eqs-4
      write (datfilunit,'(f10.4\)',iostat=ierr) sumconcfrac(k)
    end do
    write (datfilunit,'(f10.4)',iostat=ierr) sumconcfrac(Num_Eqs-3)
  endif
!  close(22)
  call WindowFontColour(TextBlack, TextWhiteBold)
  msg1 = 'Model Status:  Run completed'
  call WindowOutString(0, 1600, msg1)
  lrunning = .FALSE.
  menuactive = .false.
  if (DatOut) then
    close (DatFilUnit)
    DatOut = .false.
    msg0 = 'Closed Output File: '//trim(datfile_out)
    call WindowOutString(0, 1300, msg0)
  endif
! save final screen into the bitmap for updating
  call WBitmapGet(MainWinBitmap,0)
!  deallocate(sourceconc)
!  deallocate(sourceconc0)
```

```
  deallocate(conc)
  deallocate(sumconcfrac)
  deallocate(plotarray2)
  call menuset(1)   ! turn all the menus back on
  call exitthread(0)    !exit code is 0
end subroutine vocdrv

subroutine VOCParam_OK()
  use msflib
  use winteracter
  use voccom
  implicit none
  include 'resource.fd'
  call WDialogUnload()
  menuactive = .false.
  return
end subroutine VOCParam_OK

real*8 function WeirEffic_calc(istruct, q, h, s, Ht)
! parameterizations taken from Gulliver et al. JHE, July 1998, 664-671
!  itype = structure type, 1 = sharp-crested weir (i.e., plunging jet_
!                          2 = ogee-crested weir (spillway, no free jet)
!                          3 = gated sill
!  q = specific discharge = Q/L : Q is flow, L is length (m^2/sec)
!  h = head loss across structure (m)
!  s = submergence of gate lip (only used for type 3) (m)
!  Ht = tailwater depth (m)
use voccom
implicit none
! passed variables
integer istruct
real*8 q, h, s, Ht
!local variables
real*8 nu, tempc, R, F, g, e20, effic, d, ft, fd, fall, do2
! nu = kin visc
! tempc = temperature
! R = reynolds number
! F = Froude number
! g = accel of gravity
! e20 = oxygen transfer coefficient at 20 C
! e = weir efficiency for VOC, returned as WeirEffic
! d = molecular diffusivity of VOC
! ft = temperature correction factor for e20
! fd = diffusivity correction factor for e20
! f  = overall correction factor (= ft*fd)
! do2 = diffusivity of oxygen in FW at 20C
! external functions
real*8 diff_calc
external diff_calc
tempc = SurfaceTemp
g = 9.8d0   ! acceleration of gravity
do2 = 1.888d-5   ! in cm^2/sec from scalpha.wb3 workbook
! Kinematic viscosity (nu) in cm^2/sec calc. from temperature in deg-C.
! The underlying data is from data from CRC 63rd edition.
! The polynomial fit was done in the spreadsheet KINVISC.WB1 in QDATA
! The 1.0d-4 factor converts cm^2/sec into m^2/sec
nu = 1.0d-4*(1.7826598d-2 - 5.76464d-4*tempc +  1.12266d-5*tempc**2 - 9.66507d-8*tempc**3)
! Reynolds number
R = q/nu
! Froude number
F = dexp(0.25d0*log(8*g*(h**3)/q**2))  !  (8gh^3/q^2)^1/4
select case (istruct)
  case (1)   !  sharp-crested weir
    e20 = 1.0d0 - (1.0d0/(1+0.24*1.0d-4*(F**1.79)*(R**0.53)))**1.115
  case (2)   !  ogee-crested weir
    e20 = 1.0d0 - dexp(((-0.263d0*h)/(1.0d0+0.215d0*q))-0.203d0*Ht)
  case (3)
```

```
    e20 = 1.0d0 - dexp((-0.0086d0*(h*q/s))-0.118d0)
  case default
    continue
end select
! correction factors taken from Urban et al., October 2001, JHE, 848-859
!temperature correction factor
ft = 1.0d0 + 0.2103d-1*(tempc-2.0d1) + 8.261d-5*(tempc-2.0d1)**2
! diffusivity correction factor
d = diff_calc(tempc, 1)  !  calculate diffusivity of VOC using Wilke-Chang
fd = dsqrt(d/do2)
fall = fd*ft
effic = 1.0d0 - (1.0d0 - e20)**fall
WeirEffic_calc = effic
end function WeirEffic_calc

subroutine xyplot(xyData)
use msflib
use winteracter
use voccom
use cfunccom
implicit none
character*25 dummytitle
character*11 setlabels(2)
data setlabels/'[VOC-total]', '[Atm-Equil]'/
integer  retcode, ipnts, i, j, k
integer  iscale, iticks
real*4 xyData(2,2,2), xData(2), yData1(2), yData2(2), xticks(:), yticks(:), &
       xold(:), yold1(:), yold2(:)
allocatable xticks, yticks, xold, yold1, yold2
parameter (iticks = 5)
if (.not. lrunning) return
allocate (xticks(iticks), yticks(iticks))
ipnts = 2
do i = 1, 2
  xdata(i) = xyData(1,i,1)
  ydata1(i) = xyData(2,i,1)
  ydata2(i) = xyData(2,i,2)
end do
if (.not. Plotinit) then
  call WindowFontColour(NotSet, TextWhite)
  if (xData(1) .eq. 0.0e0) then
    call WindowClearArea(0,1900,9999,9999)   ! reset everything
  else
    call WindowClearArea(0,1900,8499,4999)   ! reset the total conc. plot
  endif
endif
if ((profilewindowopen .or. errorwindow) .and. (focusoutput .eq. 1)) then
  call WindowRaise(0)
  focusoutput = 0
endif
! common calls to set up graphics area for output
! only need to draw the axes once, that is done in the IF-THEN statement
call IGrArea(totalxmin, totalymin, totalxmax, totalymax)
call IGrUnits(minlength, minconc, maxlength, maxconc)
call IPgArea(0.1, 0.15, 0.85, 0.95)
call IPgNewGraph(isets, ipnts, '', '', 'X')
call IPgStyle(1, 0, 0, 0, 191, 191)   ! see col256 in wint/demos color defs
call IPgStyle(2, 0, 0, 0, 159, 159)
call IPgUnits(minlength, minconc, maxlength, maxconc)
! Axis construction, labeling etc. in here
if (.not. PlotInit) then
!  set the x-axis ticks
  call SetTicks(xticks, iticks, minlength, maxlength)
!  set the y-axis ticks
  call SetTicks(yticks, iticks, minconc, maxconc)
  call IGrCharFont(7)
  call IGrCharSize(1.0, 2.0)
```

```
  call IPgXScalePos(0.45)
  call IPgXUserScale(xticks, iticks)
  call IPgYUserScale(yticks, iticks)
  call IPgAxesXY(minlength, minconc)
  call IPgXTickPos(minconc, maxconc)
  call IPgXscale('TN')
  call IPgYscaleLeft('TN')
  call IPgXLabel('Distance (km)','C')
  call IPgYLabelLeft('log[VOC]|ug/L','CV')
  if (xData(1) .gt. 0.0e0) then   ! need to plot OldData
    allocate (xold(oldpoints), yold1(oldpoints), yold2(oldpoints))
    do i = 1, oldpoints
      xold(i) = OldData(1,i,1)
      yold1(i) = OldData(2,i,1)
      yold2(i) = OldData(2,i,2)
    end do
    call IPgNewGraph(isets, oldpoints, '', '', 'X')
    call IPgStyle(1, 0, 0, 0, 191, 191) ! see col256 in wint/demos color defs
    call IPgStyle(2, 0, 0, 0, 159, 159)
    call IPgXYPairs(xold, yold1)
    call IPgXYPairs(xold, yold2)
    deallocate (xold, yold1, yold2)
    call IPgNewGraph(isets, ipnts, '', '', 'X')
    call IPgStyle(1, 0, 0, 0, 191, 191) ! see col256 in wint/demos color defs
    call IPgStyle(2, 0, 0, 0, 159, 159)
  endif
endif
! plot the data
call IPgXYPairs(xData, yData1)
call IPgXYPairs(xData, yData2)
if ((.not. PlotInit) .and. (xData(1) .eq. 0.0e0)) then
  call IPgKeyAll(setlabels,'H')
endif
PlotInit = .true.
deallocate (xticks, yticks)
return
end subroutine xyplot

subroutine xyplot2(xyData, numsets)
use msflib
use dflib
use winteracter
use voccom
use cfunccom
implicit none
character*25 dummytitle
character*8 setlabels(21)
data setlabels/'Src.  1', 'Src.  2', 'Src.  3', 'Src.  4', 'Src.  5', &
               'Src.  6', 'Src.  7', 'Src.  8', 'Src.  9', 'Src. 10', &
               'Src. 11', 'Src. 12', 'Src. 13', 'Src. 14', 'Src. 15', &
               'Src. 16', 'Src. 17', 'Src. 18', 'Src. 19', 'Src. 20', &
               'Src. 21'/
integer  retcode, ipnts, i, j, k, ierr, numsets, iticks
integer  iscale, numpnts
real*4  xyData(2,2,numsets), y11, y12, y21, y22
real*4 xdata(:), ydata(:,:), xticks(:), yticks(:)
allocatable xdata, ydata, xticks, yticks
numpnts = 14     ! changed from 2 for plotting cumulative fractions
if (.not. lrunning) return
setlabels(numsets) = 'Air Src.'
iticks = 5
allocate(xdata(numpnts), ydata(numsets,numpnts))
allocate(xticks(iticks), yticks(iticks))
xdata(1) = xyData(1,1,1)
xdata(2) = xyData(1,2,1)
xdata(3) = xyData(1,2,1)
xdata(4) = xyData(1,1,1)
```

```
xdata(5) = xyData(1,1,1)
xdata(6) = xyData(1,1,1) + 0.111111e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(7) = xyData(1,1,1) + 0.222222e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(8) = xyData(1,1,1) + 0.333333e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(9) = xyData(1,1,1) + 0.444444e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(10) = xyData(1,1,1) + 0.555555e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(11) = xyData(1,1,1) + 0.666666e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(12) = xyData(1,1,1) + 0.777777e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(13) = xyData(1,1,1) + 0.888888e0*(xyData(1,2,1) - xyData(1,1,1))
xdata(14) = xyData(1,2,1)
y11 = 0.0e0
y21 = 0.0e0
do j = 1, numsets
  ydata(j,1) = y11
  ydata(j,2) = y21
  y12 = y11 + xyData(2,1,j)
  y22 = y21 + xyData(2,2,j)
  ydata(j,3) = y22
  ydata(j,4) = y12
  ydata(j,5) = y11
  ydata(j,6) = y12 + 0.111111e0*(y22-y12)
  ydata(j,7) = y11 + 0.222222e0*(y21-y11)
  ydata(j,8) = y12 + 0.333333e0*(y22-y12)
  ydata(j,9) = y11 + 0.444444e0*(y21-y11)
  ydata(j,10) = y12 + 0.555555e0*(y22-y12)
  ydata(j,11) = y11 + 0.666666e0*(y21-y11)
  ydata(j,12) = y12 + 0.777777e0*(y22-y12)
  ydata(j,13) = y11 + 0.888888e0*(y21-y11)
  ydata(j,14) = y22
  y11 = y12
  y21 = y22
end do
if ((profilewindowopen .or. errorwindow) .and. (focusoutput .eq. 1)) then
  call WindowRaise(0)
  focusoutput = 0
endif
! common calls to set up graphics area for output
! only need to draw the axes once, that is done in the IF-THEN statement
call WindowSelect(0)
call IGrArea(totalxmin, fracymin, totalxmax, fracymax)
call IGrUnits(minlength, 0.0e0, maxlength, 1.0e0)
call IPgArea(0.1, 0.15, 0.85, 0.95)
call IPgNewGraph(numsets, numpnts, '', '', 'X')
do i = 1, numsets-1
  call IPgStyle(i, 0, 0, 0, DataSetColor(i), DataSetColor(i))
! see col256 in wint/demos for key to values
end do
call IPgStyle(numsets, 0, 0, 0, 0, 0)   ! Gas Exchange always bright white
call IPgUnits(minlength, 0.0e0, maxlength, 1.0e0)
! Axis construction, labeling etc. in here
if (.not. PlotInit2) then
!  set the x-axis ticks
  call SetTicks(xticks, iticks, minlength, maxlength)
!  set the y-axis ticks
  call SetTicks(yticks, iticks, 0.0e0, 1.0e0)
  call IGrCharFont(7)
  call IGrCharSize(0.75, 1.5)
  call IPgXScalePos(0.45)
  call IPgXUserScale(xticks, iticks)
  call IPgYUserScale(yticks, iticks)
  call IPgAxesXY(minlength, 0.0e0)
  call IPgXTickPos(0.0e0, 1.0e0)
  call IPgXscale('TN')
  call IPgYscaleLeft('TN')
  call IPgXLabel('Distance (km)','C')
  call IPgYLabelLeft('Conc. Fraction','CV')
endif
```

```
! plot the data
do i = 1, numsets
  call IPgXYPairs(xData, yData(i,1:numpnts))
end do
if (.not. PlotInit2) then
  call IPgKeyAll(setlabels,'H')
  PlotInit2 = .true.
endif
write(setlabels(numsets), '(a,i2)') 'Src. ', numsets
deallocate (xdata, ydata, xticks, yticks)
return
end subroutine xyplot2
```

Modules and Compiler Resource Files

```
module voccom
use winteracter
implicit none
!**************************************************************************
!*  DESCRIPTION FOR MODULE VOCCOM                                         *
!*                                                                        *
!*  This module contains data structures used for the model and windows   *
!*  Before you modify any of these names, make sure you change the names of*
!*  the corresponding variables in *all* subroutines.  Modify this module  *
!*  carefully.  DO NOT delete items without ensuring that the program     *
!*  will recompile and link                                               *
!**************************************************************************
! data definition parameters used below
  integer iaxes, isets, num_eqs
!  parameter (num_eqs=4)
  parameter (iaxes=2, isets=2)
! windows messaging variable
  type(win_message) mainmessage  ! windows messaging var from winteracter
  integer itype   !  type of message
  type (win_style) wc, textwindow, profilewindow, fractionwindow, &
                   errorwindowwc
  integer DataSetColor(21)
  data DataSetColor /223, 31, 159, 191, 127, 95, 63, 110, 78, 207, 162, 237,&
                     46, 64, 128, 192, 32, 223, 31, 159, 191/
  logical PlotInit, PlotOldData, PlotInit2
  character*72 Title, comment(2)
  character*90 ctemp
  character*255 msg0, msg1
! scalar constants used
  real*8 MolWeight, ScVal, HVal, rel_hum, Solubility, Diffusivity, &
      MolarVolume, Density, DegradationRate, wa0, wa1, wa2, wb0, wb1, wb2, &
      salinity, wd0, wd1, wd2, wd3, sola, solb, RiverLength, &
      MaxVelocity, MinVelocity, SurfaceTemp, AirTemp, WindSpeed, &
      AtmosPress, AtmVOCConc, MaxVOCInput, MinVOCInput
  real*8 Tolerance, gasconst
  integer ReachPoints, Reaches, DistributedSources, ProfilePoints, &
      VOCInputPoints, splinepnts, oldpoints, SourceNumber, Boxpnts, &
      WeirNum
  integer diffparam, solparam, savediffparam, savesolparam
  integer DatFilUnit, ParFilUnit
! ModelFuntion flag is used to choose between
!                 NETFLUX and COMPONENTFLUX functions in DE Solver
! WeirFlow flag is used to choose between
!                 flow under weir and no flow under weir
! FlowUnits is used to choose between
!                 m^3/s and cfm in the param files
       integer ModelFunction, WeirFlow, FlowUnits
! single precision constants for Winteracter graphics
  real*4 MaxConc, MinConc, totalxmin, totalxmax, totalymin, totalymax, &
         fracymin, fracymax, minlength, maxlength
! booleans used for program control
```

```
  logical LRunning, DatOut, OpenWindow, MenuActive, ParamFileOK
! Splined array variables used in program.
  real*8  spl_Depth(:), spl_Width(:), spl_Flow(:), spl_xval(:), &
          spl_Timestep(:), spl_Veloc(:,:), spl_VOCInputs(:,:)
  allocatable  spl_Depth, spl_Width, spl_Flow, spl_xval, spl_Timestep, &
               spl_Veloc, spl_VOCInputs
! allocatable array variables used for setting up the splined data.
  real*8  RiverData(:,:), FlowVolumes(:,:), VOCInputs(:,:), WeirPoints(:,:),&
          WeirEffic(:)
  integer WeirType(:)
  allocatable RiverData, FlowVolumes, VOCInputs, WeirPoints, WeirEffic, &
              WeirType
! allocatable array used for plotting data
  real*4 OldData(:,:,:)
  allocatable OldData
! data for constants
  data tolerance /1.0d-8/
  data gasconst /8.20575d-5/      ! R in m3 atm/K mole
  data ParamFileOK /.false./
  data totalxmin, totalxmax, totalymin, totalymax, fracymin, fracymax &
        /0.0e0,    1.0e0,      0.5e0,     0.8e0,     0.0e0,    0.5e0/
  data Boxpnts/400/
! definitions moved to VOCCOM from ERRORCOM for ease of file management
  character*128 err_str(33)
  logical winfunc, ErrorWindow, ProfileWindowOpen, ConcFracWindowOpen, &
          TotalConcWindowOpen
  integer totalconcwinunit, profilewinunit, concfracwinunit, errwinunit, &
          focusprofile, focusoutput, focusconcfracoutput, &
          ErrWinBitmap, MainWinBitmap
  data totalconcwinunit, errwinunit, profilewinunit, concfracwinunit &
       /16, 17, 18, 19/
  data focusoutput, focusconcfracoutput, focusprofile /0, 1, 1/
  data ModelFunction /0/   ! selects NETFLUX version
  data TotalConcWindowOpen, ProfileWindowOpen, ConcFracWindowOpen, &
       ErrorWindow /.false., .false., .false., .false./
! variables involved with the various input files
  character(len=255) parfile_out, parfile_in, datfile_out
  logical Par_File_Read, Par_File_Sav, VOC_File_Sav
  data Par_File_Sav /.true./
  parameter (winfunc = .true.)
  parameter (parfilunit = 14)
  parameter (datfilunit = 15)
END MODULE VOCCOM


module cfunccom
  integer currentsource
  real*8 kl, cs, d, w, v, qprime, ii, kd, depth
!  real*8 sourceconc(:), sourceconc0(:)
!  allocatable sourceconc, sourceconc0
end module cfunccom
//main_win.rc   Note:  specific to Microsoft Developer Studio 6.0
//                     and Compaq Visual Fortran 6.6D
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS
```

```
//////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

//////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_VOCParams DIALOG DISCARDABLE  0, 0, 179, 115
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "VOC Parameters"
FONT 10, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",ID_OKVOCParams,66,97,40,14
    GROUPBOX        "Diffusivity (cm^2/sec)",IDC_STATIC,3,30,81,26
    GROUPBOX        "Solubility (mol/m^3-atm)",IDC_STATIC,92,30,82,26
    GROUPBOX        "Molecular Weight (g/mol)",IDC_STATIC,3,2,81,26
    GROUPBOX        "Degradation Rate (1/sec)",IDC_STATIC,49,62,81,26
    GROUPBOX        "Atmos. Conc. (ppb)",IDC_STATIC,92,2,82,26
    LTEXT           "",IDC_MolWeight,9,13,51,10,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_AtmosConc,100,14,51,10,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_Diffusivity,9,41,51,10,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_Solubility,100,42,51,10,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_DegradationRate,55,74,51,10,SS_SUNKEN | WS_BORDER
END

IDD_EnviroParams DIALOG DISCARDABLE  0, 0, 158, 117
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Environmental Parameters"
FONT 10, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",ID_OKEnvParams,58,96,40,14
    GROUPBOX        "Atmos. Press. (atm)",IDC_STATIC,78,35,73,24
    GROUPBOX        "Wind Speed (m/s)",IDC_STATIC,7,35,68,24
    GROUPBOX        "Air Temperature (C)",IDC_STATIC,7,6,68,24
    GROUPBOX        "Relative Humidity (%)",IDC_STATIC,42,64,73,24
    LTEXT           "",IDC_AirTemp,16,16,51,9,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_WindSpeed,16,45,51,9,SS_CENTERIMAGE | SS_SUNKEN |
                    WS_BORDER
    LTEXT           "",IDC_AtmPressure,88,45,51,9,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_RelativeHumidity,52,74,51,9,SS_SUNKEN | WS_BORDER
    GROUPBOX        "Water Temperature (C)",IDC_STATIC,78,6,73,24
    LTEXT           "",IDC_WaterTemp,90,16,51,9,SS_CENTERIMAGE | SS_SUNKEN |
                    WS_BORDER
END

IDD_RuntimeParams DIALOG DISCARDABLE  0, 0, 246, 112
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Runtime Parameters"
FONT 10, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",ID_OKModelParams,96,94,50,14
    LTEXT           "Runge-Kutta Tolerance",IDC_STATIC,2,3,74,8
    LTEXT           "Simulation Title",IDC_STATIC,2,29,50,10
    LTEXT           "Comments",IDC_STATIC,2,55,35,10
    LTEXT           "",IDC_Comment1,2,65,240,12,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_Comment2,2,79,240,12,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_Title,2,40,240,12,SS_SUNKEN | WS_BORDER
    LTEXT           "",IDC_Tolerance,2,14,69,11,SS_SUNKEN | WS_BORDER
    LTEXT           "Number of Boxes per Reach",IDC_STATIC,92,3,85,8
    EDITTEXT        IDC_NumberOfBoxes,92,14,65,12,ES_AUTOHSCROLL
```

```
END

IDD_HydrogParamsList DIALOG DISCARDABLE  0, 0, 349, 159
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Hydrological Parameters"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",ID_HydrogOK,149,138,50,14
    LISTBOX         IDC_HydrogParamLIST,7,49,335,84,LBS_SORT |
                    LBS_USETABSTOPS | LBS_NOINTEGRALHEIGHT | LBS_NOSEL |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT          "No.      Type      Start (km)         End (km)           Z (m)
W (m)                Q(cfm)               [VOC] (ug/L)    No.",
                    IDC_STATIC,7,40,335,8
    LTEXT           "Reach\nPoint",IDC_STATIC,7,24,22,15
    LTEXT           "Source",IDC_STATIC,310,31,25,7
    LTEXT           "Reach",IDC_STATIC,29,32,22,9
    CTEXT           "River Length: ",IDC_STATIC,7,8,42,8
    CTEXT           "Segments: ",IDC_STATIC,141,8,35,8
    CONTROL         "Distributed Sources:",IDC_STATIC,"Static",
                    SS_LEFTNOWORDWRAP | WS_GROUP,248,8,63,10
    CTEXT           "Static",IDC_TotalLength,49,7,40,10,SS_CENTERIMAGE |
                    SS_SUNKEN | WS_BORDER
    CTEXT           "Static",IDC_ReachPoints,176,7,32,10,SS_CENTERIMAGE |
                    SS_SUNKEN | WS_BORDER
    CTEXT           "Static",IDC_DistributedSourcePoints,313,8,29,10,
                    SS_CENTERIMAGE | SS_SUNKEN | WS_BORDER
END

/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_VOCParams, DIALOG
    BEGIN
        LEFTMARGIN, 3
        RIGHTMARGIN, 174
        TOPMARGIN, 2
        BOTTOMMARGIN, 111
    END

    IDD_EnviroParams, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 151
        TOPMARGIN, 6
        BOTTOMMARGIN, 110
    END

    IDD_RuntimeParams, DIALOG
    BEGIN
        LEFTMARGIN, 2
        RIGHTMARGIN, 242
        TOPMARGIN, 3
        BOTTOMMARGIN, 108
    END

    IDD_HydrogParamsList, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 342
        TOPMARGIN, 7
        BOTTOMMARGIN, 152
```

```
    END
END
#endif    // APSTUDIO_INVOKED

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END
2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END
3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END
#endif    // APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDI_ICON1               ICON    DISCARDABLE     "icon1.ico"

#ifndef _MAC
////////////////////////////////////////////////////////////////////////////
//
// Version
//
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x40004L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "APL-UW\0"
            VALUE "FileDescription", "StreamVOC\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "StreamVOC\0"
            VALUE "LegalCopyright", "Copyright © 2002\0"
            VALUE "OriginalFilename", "streamvoc.exe\0"
            VALUE "ProductName", "StreamVOC\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
```

```
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END
#endif    // !_MAC

/////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "&File "
    BEGIN
        MENUITEM "&Read Parameter File",        IDD_LoadParamFile
        MENUITEM "&Save VOC Model Results",     IDD_SAVERESULTS, GRAYED
        MENUITEM "E&xit",                       IDD_Exit
    END
    POPUP "&View Parameters"
    BEGIN
        MENUITEM "&VOC Parameters",             IDD_ViewVOCParams, GRAYED
        MENUITEM "&Environmental Parameters",   IDD_EnviroParams, GRAYED
        MENUITEM "&Hydrographical Parameters",  IDD_HydrogParams, GRAYED
        MENUITEM "&Runtime Parameters",         IDD_RuntimeParameters
        , GRAYED
        MENUITEM "&Display River Profile",      IDD_ViewProfile, GRAYED
    END
    POPUP "&Run"
    BEGIN
        MENUITEM "&Start Model",                IDD_STARTMODEL, GRAYED
        MENUITEM "&Halt Model",                 IDD_HaltModel, GRAYED
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About SteamVOC",             IDD_AboutBox
    END
END

/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_ParameterFileString "Parameter Files (*.par)|*.par|"
    IDS_DataFileString      "ASCII Data Files(*.dat)|*.dat|ASCII Text Files (*.txt)|*.txt|"
END
#endif    // English (U.S.) resources

/////////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
/////////////////////////////////////////////////////////////////////////////

#endif    // not APSTUDIO_INVOKED

// END main_win.rc
```

```
// resource.h header file
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by main_win.rc
//
#define IDS_ParameterFileString         1
#define IDS_DataFileString              2
#define IDD_LoadParamFile               101
#define IDD_MTBEParams                  102
#define IDD_VOCParams                   102
#define IDD_MeteorParams                103
#define IDD_EnviroParams                103
#define IDD_TimeSeriesEntry             105
#define IDD_RuntimeParams               110
#define IDD_ResetParams                 111
#define IDD_HydrogParams                112
#define IDD_DiffParam                   113
#define IDD_Exit                        113
#define IDD_SolParam                    114
#define IDI_ICON1                       115
#define IDR_MENU1                       124
#define IDD_HydrogParamsList            127
#define IDC_MTBEInputSeries             1014
#define IDC_WindSpeed                   1017
#define IDC_AirTemp                     1018
#define IDC_AtmPressure                 1019
#define IDC_WaterTemp                   1020
#define IDC_Tolerance                   1053
#define IDC_Inflow                      1054
#define IDC_OutputDistance              1054
#define IDC_CallDiffParam               1055
#define IDC_MixedLayer                  1061
#define IDC_SurfaceTemp                 1062
#define IDC_LakeDepth                   1063
#define IDC_LakeArea                    1064
#define IDC_ReachPointEntry             1064
#define IDC_AtmMTBEConc                 1065
#define IDC_DistributedSourceEntry      1065
#define IDC_MolWeight                   1066
#define IDC_InitialConc                 1067
#define IDC_Diffusivity                 1067
#define IDC_AtmosConc                   1068
#define IDC_DiffButtWilk                1069
#define IDC_Solubility                  1069
#define IDC_DiffButtWann                1070
#define IDC_Wank_a0                     1071
#define IDC_Wank_a1                     1072
#define IDC_Wank_a2                     1073
#define IDC_Wank_a3                     1074
#define IDC_MolarVolume                 1075
#define IDC_SolButtWann                 1076
#define IDC_Wank_a0_sol                 1077
#define IDC_Wank_a1_sol                 1078
#define IDC_Wank_a2_sol                 1079
#define IDC_Wank_a3_sol                 1080
#define IDC_Wank_Salinity               1080
#define IDC_SolButtRobbins              1081
#define IDC_RobbinsA                    1082
#define IDC_RobbinsB                    1083
#define IDC_CallSolParam                1084
#define IDC_Comment1                    1085
#define IDC_DegradationRate             1085
#define IDC_Title                       1087
#define IDC_Comment2                    1088
#define IDC_UOK3                        1089
#define IDC_UMonthly1                   1090
```

```
#define IDC_UWeekly1                    1091
#define IDC_TAOK3                       1092
#define IDC_TAMonthly1                  1093
#define IDC_TAWeekly1                   1094
#define IDC_PAOK3                       1095
#define IDC_PAMonthly1                  1096
#define IDC_PAWeekly1                   1097
#define IDC_TotalTime                   1098
#define IDC_OutputTimestep              1099
#define IDC_ViewRiverProfile            1162
#define IDC_CurrentWorkDir              1163
#define IDC_DataDirectory               1164
#define IDC_TimeSeriesFilename          1165
#define IDC_FileStatus                  1166
#define IDC_FileStatus2                 1167
#define IDC_LocalTitle                  1185
#define IDC_RiverPhysicalData1          1191
#define IDC_RiverPhysicalData2          1192
#define ID_OKEnvParams                  1192
#define IDC_RiverPhysicalData3          1193
#define IDC_InflowHeightMonthly         1194
#define IDC_ReachPoints1                1194
#define IDC_TotalLength                 1194
#define IDC_InflowHeightWeekly          1195
#define IDC_RiverPhysicalData4          1195
#define IDC_InflowHeightDaily           1196
#define IDC_RiverPhysicalData5          1196
#define IDC_InflowHeightOK              1197
#define IDC_RiverPhysicalData6          1197
#define IDC_InflowHeightOK2             1198
#define IDC_RiverPhysicalData7          1198
#define IDC_InflowHeightWeekly1         1199
#define IDC_RiverPhysicalData8          1199
#define IDC_InflowHeightMonthly1        1200
#define IDC_RiverPhysicalData9          1200
#define IDC_InflowHeightOK3             1201
#define IDC_RiverPhysicalData10         1201
#define IDC_OutflowHeightWeekly1        1202
#define IDC_RiverPhysicalData11         1202
#define IDC_OutflowHeightMonthly1       1203
#define IDC_RiverPhysicalData12         1203
#define IDC_OutflowHeightOK3            1204
#define IDC_RiverPhysicalData13         1204
#define IDC_RiverPhysicalData14         1205
#define IDC_OutflowHeightOK2            1206
#define IDC_RiverPhysicalData15         1206
#define IDC_EpiLossRate                 1207
#define IDC_RiverPhysicalData16         1207
#define IDC_EpiLossWeekly1              1208
#define IDC_RiverPhysicalData17         1208
#define IDC_EpiLossMonthly1             1209
#define IDC_RiverPhysicalData18         1209
#define IDC_EpiLossOK3                  1210
#define IDC_RiverPhysicalData19         1210
#define IDC_HypLossRate                 1211
#define IDC_RiverPhysicalData20         1211
#define IDC_HypLossOK3                  1212
#define IDC_DistributedSourceData1      1212
#define ID_OKHydroParams                1212
#define IDC_HypLossMonthly1             1213
#define IDC_DistributedSourceData2      1213
#define ID_OKModelParams                1213
#define IDC_HypLossWeekly1              1214
#define IDC_DistributedSourceData3      1214
#define ID_OKVOCParams                  1214
#define IDC_EpiLossMonthly              1215
#define IDC_DistributedSourceData4      1215
```

```
#define IDC_NumberOfBoxes               1215
#define IDC_EpiLossWeekly               1216
#define IDC_DistributedSourceData5      1216
#define IDC_EpiLossDaily                1217
#define IDC_DistributedSourceData6      1217
#define IDC_EpiLossOK2                  1218
#define IDC_DistributedSourceData7      1218
#define IDC_EpiLossOK                   1219
#define IDC_DistributedSourceData8      1219
#define IDC_HypLossMonthly              1220
#define IDC_DistributedSourceData9      1220
#define IDC_HypLossWeekly               1221
#define IDC_DistributedSourceData10     1221
#define IDC_HypLossDaily                1222
#define IDC_DistributedSourcePoints2    1222
#define IDC_ReachPoints                 1222
#define IDC_HypLossOK2                  1223
#define IDC_DistributedSourcePoints3    1223
#define IDC_DistributedSourcePoints     1223
#define IDC_HypLossOK                   1224
#define IDC_HydrogParamLIST             1224
#define IDC_Wank_b0_sol                 1225
#define ID_HydrogOK                     1225
#define IDC_Wank_b1_sol                 1226
#define IDC_Wank_b2_sol                 1227
#define IDC_DataUnits                   1228
#define IDC_MonthlyTitle                1229
#define IDC_GraphData                   1232
#define IDC_LoadData                    1233
#define IDC_RDErrorMessage              1235
#define IDC_ScDay                       1236
#define IDC_ScVal                       1237
#define IDC_CalcSc                      1238
#define IDC_HDay                        1239
#define IDC_HVal                        1240
#define IDC_CalcH                       1241
#define IDC_RelativeHumidity            1242
#define IDD_STARTMODEL                  40004
#define IDD_HaltModel                   40005
#define IDD_ViewVOCParams               40006
#define IDD_SAVERESULTS                 40007
#define IDD_RuntimeParameters           40008
#define IDD_ViewProfile                 40009
#define IDD_AboutBox                    40010
// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        128
#define _APS_NEXT_COMMAND_VALUE         40011
#define _APS_NEXT_CONTROL_VALUE         1226
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
//end resource.h


// resource.fd
!MS$FREEFORM
! Microsoft Developer Studio generated include file.
! Used by main_win.rc
!
      integer, parameter :: IDS_ParameterFileString         = 1
      integer, parameter :: IDS_DataFileString              = 2
      integer, parameter :: IDD_LoadParamFile               = 101
      integer, parameter :: IDD_MTBEParams                  = 102
      integer, parameter :: IDD_VOCParams                   = 102
```

```
integer, parameter :: IDD_MeteorParams               = 103
integer, parameter :: IDD_EnviroParams               = 103
integer, parameter :: IDD_TimeSeriesEntry            = 105
integer, parameter :: IDD_RuntimeParams              = 110
integer, parameter :: IDD_ResetParams                = 111
integer, parameter :: IDD_HydrogParams               = 112
integer, parameter :: IDD_DiffParam                  = 113
integer, parameter :: IDD_Exit                       = 113
integer, parameter :: IDD_SolParam                   = 114
integer, parameter :: IDI_ICON1                      = 115
integer, parameter :: IDR_MENU1                      = 124
integer, parameter :: IDD_HydrogParamsList           = 127
integer, parameter :: IDC_MTBEInputSeries            = 1014
integer, parameter :: IDC_WindSpeed                  = 1017
integer, parameter :: IDC_AirTemp                    = 1018
integer, parameter :: IDC_AtmPressure                = 1019
integer, parameter :: IDC_WaterTemp                  = 1020
integer, parameter :: IDC_Tolerance                  = 1053
integer, parameter :: IDC_Inflow                     = 1054
integer, parameter :: IDC_OutputDistance             = 1054
integer, parameter :: IDC_CallDiffParam              = 1055
integer, parameter :: IDC_MixedLayer                 = 1061
integer, parameter :: IDC_SurfaceTemp                = 1062
integer, parameter :: IDC_LakeDepth                  = 1063
integer, parameter :: IDC_LakeArea                   = 1064
integer, parameter :: IDC_ReachPointEntry            = 1064
integer, parameter :: IDC_AtmMTBEConc                = 1065
integer, parameter :: IDC_DistributedSourceEntry     = 1065
integer, parameter :: IDC_MolWeight                  = 1066
integer, parameter :: IDC_InitialConc                = 1067
integer, parameter :: IDC_Diffusivity                = 1067
integer, parameter :: IDC_AtmosConc                  = 1068
integer, parameter :: IDC_DiffButtWilk               = 1069
integer, parameter :: IDC_Solubility                 = 1069
integer, parameter :: IDC_DiffButtWann               = 1070
integer, parameter :: IDC_Wank_a0                    = 1071
integer, parameter :: IDC_Wank_a1                    = 1072
integer, parameter :: IDC_Wank_a2                    = 1073
integer, parameter :: IDC_Wank_a3                    = 1074
integer, parameter :: IDC_MolarVolume                = 1075
integer, parameter :: IDC_SolButtWann                = 1076
integer, parameter :: IDC_Wank_a0_sol                = 1077
integer, parameter :: IDC_Wank_a1_sol                = 1078
integer, parameter :: IDC_Wank_a2_sol                = 1079
integer, parameter :: IDC_Wank_a3_sol                = 1080
integer, parameter :: IDC_Wank_Salinity              = 1080
integer, parameter :: IDC_SolButtRobbins             = 1081
integer, parameter :: IDC_RobbinsA                   = 1082
integer, parameter :: IDC_RobbinsB                   = 1083
integer, parameter :: IDC_CallSolParam               = 1084
integer, parameter :: IDC_Comment1                   = 1085
integer, parameter :: IDC_DegradationRate            = 1085
integer, parameter :: IDC_Title                      = 1087
integer, parameter :: IDC_Comment2                   = 1088
integer, parameter :: IDC_UOK3                       = 1089
integer, parameter :: IDC_UMonthly1                  = 1090
integer, parameter :: IDC_UWeekly1                   = 1091
integer, parameter :: IDC_TAOK3                      = 1092
integer, parameter :: IDC_TAMonthly1                 = 1093
integer, parameter :: IDC_TAWeekly1                  = 1094
integer, parameter :: IDC_PAOK3                      = 1095
integer, parameter :: IDC_PAMonthly1                 = 1096
integer, parameter :: IDC_PAWeekly1                  = 1097
integer, parameter :: IDC_TotalTime                  = 1098
integer, parameter :: IDC_OutputTimestep             = 1099
integer, parameter :: IDC_ViewRiverProfile           = 1162
integer, parameter :: IDC_CurrentWorkDir             = 1163
```

```
integer, parameter :: IDC_DataDirectory              = 1164
integer, parameter :: IDC_TimeSeriesFilename         = 1165
integer, parameter :: IDC_FileStatus                 = 1166
integer, parameter :: IDC_FileStatus2                = 1167
integer, parameter :: IDC_LocalTitle                 = 1185
integer, parameter :: IDC_RiverPhysicalData1         = 1191
integer, parameter :: IDC_RiverPhysicalData2         = 1192
integer, parameter :: ID_OKEnvParams                 = 1192
integer, parameter :: IDC_RiverPhysicalData3         = 1193
integer, parameter :: IDC_InflowHeightMonthly        = 1194
integer, parameter :: IDC_ReachPoints1               = 1194
integer, parameter :: IDC_TotalLength                = 1194
integer, parameter :: IDC_InflowHeightWeekly         = 1195
integer, parameter :: IDC_RiverPhysicalData4         = 1195
integer, parameter :: IDC_InflowHeightDaily          = 1196
integer, parameter :: IDC_RiverPhysicalData5         = 1196
integer, parameter :: IDC_InflowHeightOK             = 1197
integer, parameter :: IDC_RiverPhysicalData6         = 1197
integer, parameter :: IDC_InflowHeightOK2            = 1198
integer, parameter :: IDC_RiverPhysicalData7         = 1198
integer, parameter :: IDC_InflowHeightWeekly1        = 1199
integer, parameter :: IDC_RiverPhysicalData8         = 1199
integer, parameter :: IDC_InflowHeightMonthly1       = 1200
integer, parameter :: IDC_RiverPhysicalData9         = 1200
integer, parameter :: IDC_InflowHeightOK3            = 1201
integer, parameter :: IDC_RiverPhysicalData10        = 1201
integer, parameter :: IDC_OutflowHeightWeekly1       = 1202
integer, parameter :: IDC_RiverPhysicalData11        = 1202
integer, parameter :: IDC_OutflowHeightMonthly1      = 1203
integer, parameter :: IDC_RiverPhysicalData12        = 1203
integer, parameter :: IDC_OutflowHeightOK3           = 1204
integer, parameter :: IDC_RiverPhysicalData13        = 1204
integer, parameter :: IDC_RiverPhysicalData14        = 1205
integer, parameter :: IDC_OutflowHeightOK2           = 1206
integer, parameter :: IDC_RiverPhysicalData15        = 1206
integer, parameter :: IDC_EpiLossRate                = 1207
integer, parameter :: IDC_RiverPhysicalData16        = 1207
integer, parameter :: IDC_EpiLossWeekly1             = 1208
integer, parameter :: IDC_RiverPhysicalData17        = 1208
integer, parameter :: IDC_EpiLossMonthly1            = 1209
integer, parameter :: IDC_RiverPhysicalData18        = 1209
integer, parameter :: IDC_EpiLossOK3                 = 1210
integer, parameter :: IDC_RiverPhysicalData19        = 1210
integer, parameter :: IDC_HypLossRate                = 1211
integer, parameter :: IDC_RiverPhysicalData20        = 1211
integer, parameter :: IDC_HypLossOK3                 = 1212
integer, parameter :: IDC_DistributedSourceData1     = 1212
integer, parameter :: ID_OKHydroParams               = 1212
integer, parameter :: IDC_HypLossMonthly1            = 1213
integer, parameter :: IDC_DistributedSourceData2     = 1213
integer, parameter :: ID_OKModelParams               = 1213
integer, parameter :: IDC_HypLossWeekly1             = 1214
integer, parameter :: IDC_DistributedSourceData3     = 1214
integer, parameter :: ID_OKVOCParams                 = 1214
integer, parameter :: IDC_EpiLossMonthly             = 1215
integer, parameter :: IDC_DistributedSourceData4     = 1215
integer, parameter :: IDC_NumberOfBoxes              = 1215
integer, parameter :: IDC_EpiLossWeekly              = 1216
integer, parameter :: IDC_DistributedSourceData5     = 1216
integer, parameter :: IDC_EpiLossDaily               = 1217
integer, parameter :: IDC_DistributedSourceData6     = 1217
integer, parameter :: IDC_EpiLossOK2                 = 1218
integer, parameter :: IDC_DistributedSourceData7     = 1218
integer, parameter :: IDC_EpiLossOK                  = 1219
integer, parameter :: IDC_DistributedSourceData8     = 1219
integer, parameter :: IDC_HypLossMonthly             = 1220
integer, parameter :: IDC_DistributedSourceData9     = 1220
```

```
      integer, parameter :: IDC_HypLossWeekly              = 1221
      integer, parameter :: IDC_DistributedSourceData10    = 1221
      integer, parameter :: IDC_HypLossDaily               = 1222
      integer, parameter :: IDC_DistributedSourcePoints2   = 1222
      integer, parameter :: IDC_ReachPoints                = 1222
      integer, parameter :: IDC_HypLossOK2                 = 1223
      integer, parameter :: IDC_DistributedSourcePoints3   = 1223
      integer, parameter :: IDC_DistributedSourcePoints    = 1223
      integer, parameter :: IDC_HypLossOK                  = 1224
      integer, parameter :: IDC_HydrogParamLIST            = 1224
      integer, parameter :: IDC_Wank_b0_sol                = 1225
      integer, parameter :: ID_HydrogOK                    = 1225
      integer, parameter :: IDC_Wank_b1_sol                = 1226
      integer, parameter :: IDC_Wank_b2_sol                = 1227
      integer, parameter :: IDC_DataUnits                  = 1228
      integer, parameter :: IDC_MonthlyTitle               = 1229
      integer, parameter :: IDC_GraphData                  = 1232
      integer, parameter :: IDC_LoadData                   = 1233
      integer, parameter :: IDC_RDErrorMessage             = 1235
      integer, parameter :: IDC_ScDay                      = 1236
      integer, parameter :: IDC_ScVal                      = 1237
      integer, parameter :: IDC_CalcSc                     = 1238
      integer, parameter :: IDC_HDay                       = 1239
      integer, parameter :: IDC_HVal                       = 1240
      integer, parameter :: IDC_CalcH                      = 1241
      integer, parameter :: IDC_RelativeHumidity           = 1242
      integer, parameter :: IDD_STARTMODEL                 = 40004
      integer, parameter :: IDD_HaltModel                  = 40005
      integer, parameter :: IDD_ViewVOCParams              = 40006
      integer, parameter :: IDD_SAVERESULTS                = 40007
      integer, parameter :: IDD_RuntimeParameters          = 40008
      integer, parameter :: IDD_ViewProfile                = 40009
      integer, parameter :: IDD_AboutBox                   = 40010
//end resource.fd
```