```fortran
#include"cppdefs.h"
!-----------------------------------------------------------------------
!BOP
!
! !MODULE: eqstate --- the equation of state \label{sec:eqstate}
!
! !INTERFACE:
   MODULE eqstate
!
! !DESCRIPTION:
!  Computes the density, $\mean{\rho}$, and buoyancy from the
!  salinity, $S$, the temperature, $\Theta$, and the thermodynamic
!  pressure, $P$, according to an \emph{equation of state},
!  \begin{equation}
!    \label{DefEOS}
!     \mean{\rho} = \hat{\rho} (S,\Theta,P)
!     \point
!  \end{equation}
!
!  The following remark on the thermodynamic interpretation of
!  density, temperature, and pressure is useful here.  If $\Theta$ is
!  identified with the in-situ temperature, and $P$ with the in-situ
!  pressure, then $\mean{\rho}$ will be the in-situ density.  On the
!  other hand, if $P$ is identified with the surface pressure, and
!  $\Theta$ with the potential temperature, the same equation of
!  state, \eq{DefEOS}, will yield $\mean{\rho}$ as the potential
!  density. Note that the quantity {\tt sigma\_t} found in the GOTM output
!  is simply computed from  $\mean{\rho}$ - 1000 kg m$^{-3}$, and may
!  therefore adopt different meanings.
!
!
!  At present, two different models for the equation of state ("modes"),
!  and four different "methods" how to evalute the equation of state
!  are implemented.
!
!  Modes:
!  \begin{enumerate}
!     \item The UNESCO equation of state according to \cite{FofonoffMillard83}
!     \item The \cite{JACKETTea05} equation of state
!  \end{enumerate}
!  Methods:
!  \begin{enumerate}
!     \item the full equation of state --- including pressure effects
!     \item the full equation of state --- without pressure effects
!     \item the linearised equation of state
!     \item a general linear form of the equation of state
!  \end{enumerate}

!USES:
   IMPLICIT NONE

!  default: all is private.
   private
!
! !PUBLIC MEMBER FUNCTIONS:
   public init_eqstate,eqstate1,eos_alpha,eos_beta,unesco,rho_feistel

   interface init_eqstate
      module procedure init_eqstate_nml
      module procedure init_eqstate_yaml
   end interface

!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
!EOP
!
!  private data memebers
   integer, public           :: eq_state_method, eq_state_mode
   REALTYPE                  :: T0,S0,p0,dtr0,dsr0
   logical                   :: init_linearised
!
!-----------------------------------------------------------------------

   contains

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Read the namelist {\tt eqstate}
!
! !INTERFACE:
   subroutine init_eqstate_nml(namlst)
!
! !DESCRIPTION:
!  Here, the namelist {\tt eqstate} in the namelist file {\tt gotmrun.nml}
!  is read.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   integer, intent(in)                  :: namlst
!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
```

```
!EOP
!
! !LOCAL VARIABLES:
   namelist /eqstate/ eq_state_mode,eq_state_method,T0,S0,p0,dtr0,dsr0
!
!-----------------------------------------------------------------------
!BOC
   LEVEL1 'init_eqstate_nml'

   init_linearised = .true.
   read(namlst,nml=eqstate,err=80)
   LEVEL2 'done.'
   return
   80 FATAL 'I could not read "eqstate" namelist'
   stop 'init_eqstate'
   end subroutine init_eqstate_nml
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Read the yaml configuration {\tt eqstate}
!
! !INTERFACE:
   subroutine init_eqstate_yaml(branch)
!
! !DESCRIPTION:
!
! !USES:
   use yaml_settings
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   class (type_settings), intent(inout) :: branch
!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
!EOP
!
! !LOCAL VARIABLES:
   integer, parameter :: rk = kind(_ONE_)
   class (type_settings), pointer :: twig
!
!-----------------------------------------------------------------------
!BOC
   LEVEL1 'init_eqstate_yaml'
   call branch%get(eq_state_mode, 'method', 'method', default=2, &
                   options=(/option(1, 'UNESCO', 'UNESCO'), option(2, 'Jackett et al. (2005)', 'Jackett')/))
   call branch%get(eq_state_method, 'form', 'formulation', &
                   options=(/option(1, 'full', 'full'), option(2, 'full with buoyancy frequency based on surface pressure', 'full-pot'), &
                   option(3, 'linearized at T0,S0,p0', 'linear'), option(4, 'linearized at T0,S0,p0,dtr0,dsr0', 'linear_custom')/), default=1)
   twig => branch%get_child('linear')
   call twig%get(T0, 'T0', 'reference temperature', 'Celsius', &
                 minimum=-2._rk, default=10._rk)
   call twig%get(S0, 'S0', 'reference salinity', 'psu', &
                 minimum=0._rk, default=35._rk)
   call twig%get(p0, 'p0', 'reference pressure', 'Pa', &
                 default=0._rk)
   call twig%get(dtr0, 'dtr0', 'thermal expansion coefficient', 'kg/m^3/K', &
                 default=-0.17_rk)
   call twig%get(dsr0, 'dsr0', 'saline expansion coefficient', 'kg/m^3/psu', &
                 default=0.78_rk)

   init_linearised = .true.
   LEVEL2 'done.'
   end subroutine init_eqstate_yaml
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Select an equation of state
!
! !INTERFACE:
   REALTYPE function eqstate1(S,T,p,g,rho_0)
!
! !DESCRIPTION:
!  Calculates the in-situ buoyancy according to the selected method.
!  {\tt S} is salinity $S$ in psu, {\tt T} is
!  potential temperature $\theta$ in $^{\circ}$C (ITS-90), {\tt p} is
!  gauge pressure (absolute pressure - 10.1325 bar), {\tt g} is the
!  gravitational acceleration in m\,s$^{-2}$ and {\tt rho\_0} the reference
!  density in kg\,m$^{-3}$. {\tt eqstate1} is the in-situ-density
!  in kg\,m$^{-3}$.
!  For {\tt eq\_state\_method}=1, the UNESCO equation of state is used,
!  for {\tt eq\_state\_method}=2, the \cite{JACKETTea05} equation
!  of state is used. Here, some care is needed, since the UNESCO equation
!  used bar for pressure and the \cite{JACKETTea05} uses dbar for pressure.
!  For values of
!  {\tt eq\_state\_method} ranging from 1 to 4, one of the following methods
!  will be used.
!
!   \begin{enumerate}
!      \item the full equation of state for sea water
!            including pressure dependence.
```

```
!     \item the equation of state for sea water
!           with the pressure evaluated at the sea surface as
!           reference level. This is the choice
!           for computations based on potential temperature and density.
!     \item a linearised equation of state.
!           The parameters {\tt T0},
!           {\tt S0} and {\tt p0} have to be specified in the namelist.
!     \item a linear equation of state with prescribed {\tt rho0}, {\tt T0},
!           {\tt S0}, {\tt dtr0}, {\tt dsr0} according to
!           \begin{equation}
!              \label{eosLinear}
!              \rho = \rho_0 + \text{\tt dtr0} (T - T_0)
!                            + \text{\tt dsr0} (S - S_0)
!               \point
!           \end{equation}
!   \end{enumerate}

!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE,intent(in)                 :: S,T,p
   REALTYPE,optional,intent(in)        :: g,rho_0
!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
!EOP
!
! !LOCAL VARIABLES:
   REALTYPE                  :: x
   REALTYPE, save            :: rh0,dtr,dsr
   REALTYPE                  :: dTT,dSS
   logical                   :: press
!
!-----------------------------------------------------------------------
!BOC
   select case (eq_state_mode)
      case(1)
         select case (eq_state_method)
            case (1)
               press=.true.
               x=unesco(S,T,p,press)
            case (2)
               press=.false.
               x=unesco(S,T,p,press)
            case (3)
               if (init_linearised) then
                  press=.true.   ! This allows for choosing potentials other than p=0
                  dTT=0.001
                  dSS=0.001
                  rh0= unesco(S0,T0,p0,press)
                  dtr=(unesco(S0,T0+0.5*dTT,p0,press)        &
                      -unesco(S0,T0-0.5*dTT,p0,press))/dTT
                  dsr=(unesco(S0+0.5*dSS,T0,p0,press)        &
                      -unesco(S0-0.5*dSS,T0,p0,press))/dSS
                  init_linearised=.false.
               end if
               x=rh0+dtr*(T-T0)+dsr*(S-S0)
            case (4)
               x=rho_0+dtr0*(T-T0)+dsr0*(S-S0)
            case default
         end select
      case(2)
         select case (eq_state_method)
            case (1)
               press=.true.
               x=rho_feistel(S,T,p*10.,press)
            case (2)
               press=.false.
               x=rho_feistel(S,T,p*10.,press)
            case (3)
               if (init_linearised) then
                  press=.true.   ! This allows for choosing potentials other than p=0
                  dTT=0.001
                  dSS=0.001
                  rh0= rho_feistel(S0,T0,p0*10.,press)
                  dtr=(rho_feistel(S0,T0+0.5*dTT,p0*10.,press)        &
                      -rho_feistel(S0,T0-0.5*dTT,p0*10.,press))/dTT
                  dsr=(rho_feistel(S0+0.5*dSS,T0,p0*10.,press)        &
                      -rho_feistel(S0-0.5*dSS,T0,p0*10.,press))/dSS
                  init_linearised=.false.
               end if
               x=rh0+dtr*(T-T0)+dsr*(S-S0)
            case (4)
               x=rho_0+dtr0*(T-T0)+dsr0*(S-S0)
            case default
         end select
      case default
   end select

   eqstate1=-g*(x-rho_0)/rho_0

   return
   end function eqstate1
!EOC
```

```
!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Compute thermal expansion coefficient
!
! !INTERFACE:
   REALTYPE function eos_alpha(S,T,p,g,rho_0)
!
! !DESCRIPTION:
!  Computes the thermal expansion coefficient defined by
!  \begin{equation}
!   \label{eosAlpha}
!        \alpha =
!         - \dfrac{1}{\rho_0}
!        \left( \partder{\rho_{is}}{T} \right)_S
!  =
!        \dfrac{1}{g}
!        \left( \partder{B_{is}}{T} \right)_S
!        \comma
!  \end{equation}
!  where $B_{is}$ denotes the in-situ buoyancy. The computation is carried
!  out by a finite difference approximation of \eq{eosAlpha},
!  requiring two evaluations of the equation of state.
!  Note, that comparing \eq{eosAlpha} with \eq{eosLinear} it follows that
!  $\alpha = - \text{\tt dtr0}/\rho_0$.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE,intent(in)                 :: S,T,p
   REALTYPE,optional,intent(in)        :: g,rho_0
!
! !REVISION HISTORY:
!  Original author(s): Lars Umlauf
!
!EOP
!
! !LOCAL VARIABLES:
!
   REALTYPE,parameter                :: delta = 0.01
   REALTYPE                          :: buoy_a,buoy_b
!-----------------------------------------------------------------------
!BOC

      buoy_a    = eqstate1(S,T+0.5*delta,p,g,rho_0)
      buoy_b    = eqstate1(S,T-0.5*delta,p,g,rho_0)

      eos_alpha =  (buoy_a - buoy_b) / (g*delta)

   end function eos_alpha
!EOC


!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Compute saline contraction coefficient
!
! !INTERFACE:
   REALTYPE function eos_beta(S,T,p,g,rho_0)
!
! !DESCRIPTION:
!  Computes the saline contractioncoefficient defined by
!  \begin{equation}
!   \label{eosBeta}
!        \beta =
!         \dfrac{1}{\rho_0}
!        \left( \partder{\rho_{is}}{S} \right)_T
!  =
!        - \dfrac{1}{g}
!        \left( \partder{B_{is}}{S} \right)_T
!        \comma
!  \end{equation}
!  where $B_{is}$ denotes the in-situ buoyancy. The computation is carried
!  out by a finite difference approximation of \eq{eosBeta},
!  requiring two evaluations of the equation of state.
!  Note, that comparing \eq{eosBeta} with \eq{eosLinear} it follows that
!  $\beta = \text{\tt dsr0}/\rho_0$.
!
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE,intent(in)                 :: S,T,p
   REALTYPE,optional,intent(in)        :: g,rho_0
!
! !REVISION HISTORY:
!  Original author(s): Lars Umlauf
!
!EOP
!
! !LOCAL VARIABLES:
!
   REALTYPE,parameter                :: delta = 0.01
   REALTYPE                          :: buoy_a,buoy_b
!-----------------------------------------------------------------------
```

```
!BOC

      buoy_a    =   eqstate1(S+0.5*delta,T,p,g,rho_0)
      buoy_b    =   eqstate1(S-0.5*delta,T,p,g,rho_0)

      eos_beta = -(buoy_a - buoy_b) / (g*delta)

   end function eos_beta
!EOC


!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: The UNESCO equation of state
!
! !INTERFACE:
   REALTYPE function unesco(S,T,p,UNPress)
!
! !DESCRIPTION:
!  Computes the in-situ density in \eq{DefEOS} according to the
!  UNESCO equation of state for sea water (see \cite{FofonoffMillard83}).
!  The pressure
!  dependence can be switched on ({\tt UNPress=.true.}) or off
!  ({\tt UNPress=.false.}). {\tt S} is salinity $S$ in psu, {\tt T} is
!  potential temperature $\theta$ in $^{\circ}$C (ITS-90), {\tt p} is
!  gauge pressure (absolute pressure - 10.1325 bar) and
!  {\tt  unesco} is the in-situ density in kg\,m$^{-3}$.
!  The check value is {\tt unesco(35,25,1000) = 1062.53817} .
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE, intent(in)                :: S,T,p
   LOGICAL, intent(in)                 :: UNPress
!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
!EOP
!
! !LOCAL VARIABLES:
   REALTYPE                  :: x,K
   REALTYPE                  :: T2,T3,T4,T5,S15,S2,S3,p2
!
!-----------------------------------------------------------------------
!BOC
   T2 = T*T
   T3 = T*T2
   T4 = T2*T2
   T5 = T*T4
   S15= S**1.5
   S2 = S*S
   S3 = S*S2

   x=999.842594+6.793952e-02*T-9.09529e-03*T2+1.001685e-04*T3
   x=x-1.120083e-06*T4+6.536332e-09*T5
   x=x+S*(0.824493-4.0899e-03*T+7.6438e-05*T2-8.2467e-07*T3)
   x=x+S*5.3875e-09*T4
   x=x+sqrt(S3)*(-5.72466e-03+1.0227e-04*T-1.6546e-06*T2)
   x=x+4.8314e-04*S2

     if ((UNPress).and.(p.gt.0)) then
     p2=p*p
     K= 19652.21                                             &
       +148.4206     *T          -2.327105    *T2        &
       +  1.360477E-2*T3         -5.155288E-5 *T4        &
       +  3.239908      *p       +1.43713E-3  *T *p      &
       +  1.16092E-4 *T2*p       -5.77905E-7  *T3*p      &
       +  8.50935E-5    *p2      -6.12293E-6  *T *p2     &
       +  5.2787E-8  *T2*p2                              &
       + 54.6746             *S  -0.603459    *T    *S   &
       +  1.09987E-2 *T2     *S  -6.1670E-5   *T3   *S   &
       +  7.944E-2           *S15+1.6483E-2   *T    *S15 &
       -  5.3009E-4  *T2     *S15+2.2838E-3      *p *S   &
       -  1.0981E-5  *T *p   *S  -1.6078E-6   *T2*p *S   &
       +  1.91075E-4    *p   *S15-9.9348E-7      *p2*S   &
       +  2.0816E-8  *T *p2*S    +9.1697E-10  *T2*p2*S
     x=x/(1.-p/K)
   end if

   unesco=x
   return
   end function unesco
!EOC
!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: The \cite{JACKETTea05} equation of state
!
! !INTERFACE:
   REALTYPE function rho_feistel(s,th,p,UNPress)
!
! !DESCRIPTION:
!  Computes the in-situ density in \eq{DefEOS} according to the
!  \cite{JACKETTea05} equation of state for sea water, which is based
!  on the Gibbs potential developed by \cite{FEISTEL03}. The pressure
```

```
!  dependence can be switched on ({\tt UNPress=.true.}) or off
!  ({\tt UNPress=.false.}). {\tt s} is salinity $S$ in psu, {\tt th} is
!  potential temperature $\theta$ in $^{\circ}$C (ITS-90), {\tt p} is
!  gauge pressure (absolute pressure - 10.1325 dbar) and
!  {\tt  rho\_feistel} is the in-situ density in kg\,m$^{-3}$.
!  The check value is {\tt rho\_feistel(20,20,1000) = 1017.728868019642} .
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE, intent(in)                 :: s,th,p
   LOGICAL, intent(in)                  :: UNPress
!
! !REVISION HISTORY:
!  Original author(s): Hans Burchard & Karsten Bolding
!
!EOP
!
! !LOCAL VARIABLES:
   REALTYPE                  :: th2,sqrts,pth,anum,aden
!
!-----------------------------------------------------------------------
!BOC

th2 = th*th
sqrts = sqrt(s)


anum =          9.9984085444849347d+02 +    &
           th*( 7.3471625860981584d+00 +    &
           th*(-5.3211231792841769d-02 +    &
           th*  3.6492439109814549d-04)) +  &
            s*( 2.5880571023991390d+00 -    &
           th*  6.7168282786692355d-03 +    &
            s*  1.9203202055760151d-03)

aden =          1.0000000000000000d+00 +    &
           th*( 7.2815210113327091d-03 +    &
           th*(-4.4787265461983921d-05 +    &
           th*( 3.3851002965802430d-07 +    &
           th*  1.3651202389758572d-10))) + &
            s*( 1.7632126669040377d-03 -    &
           th*( 8.8066583251206474d-06 +    &
          th2*  1.8832689434804897d-10) +   &
        sqrts*( 5.7463776745432097d-06 +    &
          th2*  1.4716275472242334d-09))



if((UNPress).and.(p.gt.0.d0)) then

    pth = p*th

    anum = anum +        p*( 1.1798263740430364d-02 +   &
                       th2*  9.8920219266399117d-08 +   &
                         s*  4.6996642771754730d-06 -   &
                         p*( 2.5862187075154352d-08 +   &
                       th2*  3.2921414007960662d-12))

    aden = aden +        p*( 6.7103246285651894d-06 -   &
                  pth*(th2*  2.4461698007024582d-17 +   &
                         p*  9.1534417604289062d-18))

end if


rho_feistel = anum/aden


!       Note:   this function should always be run in double precision
!               (since rho is returned rather than sigma = rho-1.0d3)

   return
   end function rho_feistel
!EOC

   end module eqstate

!-----------------------------------------------------------------------
! Copyright by the GOTM-team under the GNU Public License - www.gnu.org
!-----------------------------------------------------------------------
```