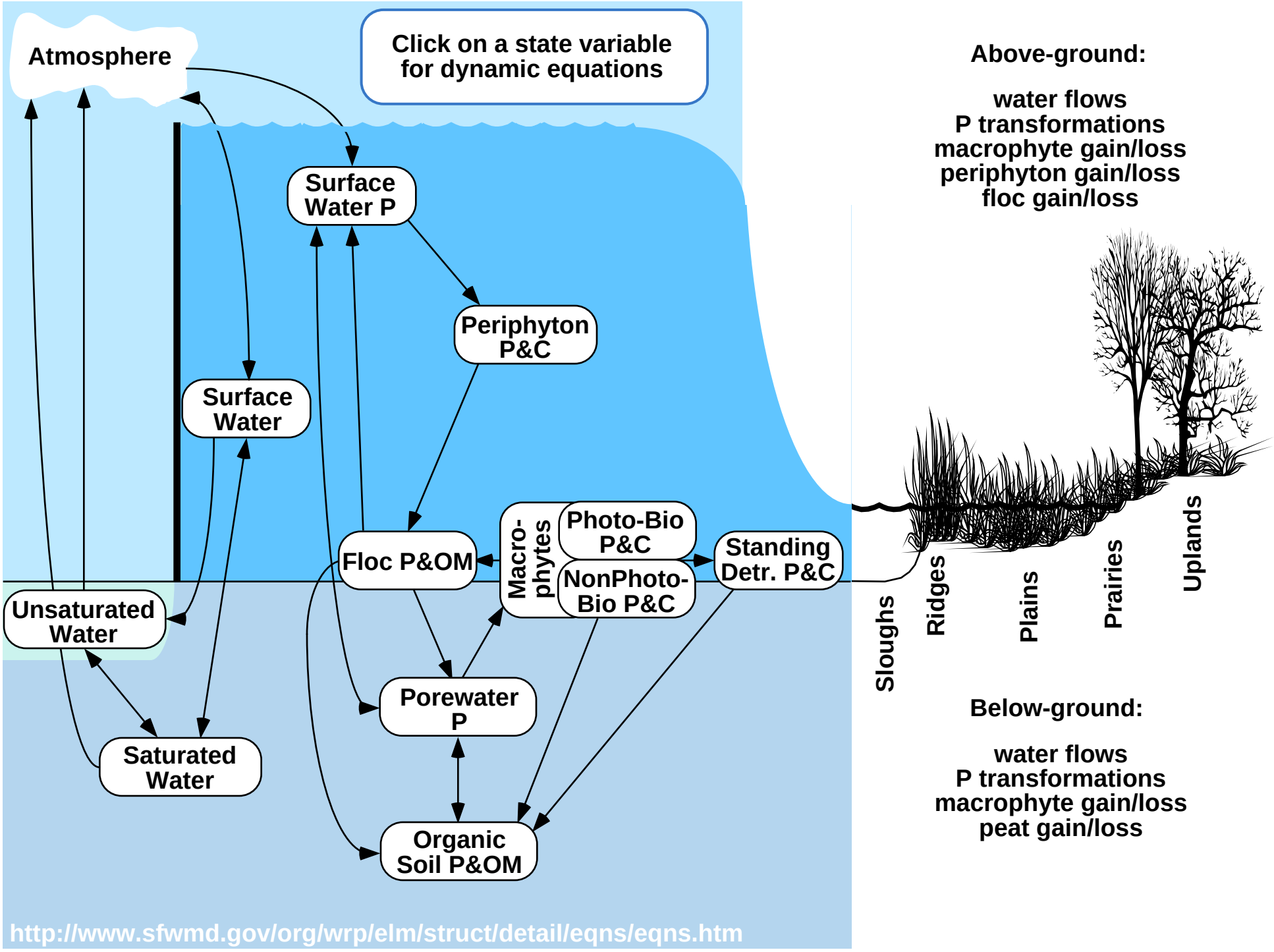
Atmosphere
Click on a state variable for dynamic equations
Surface Water P
Periphyton P&C
Surface Water
Floc P&OM
Macro-phytes
Photo-Bio P&C
NonPhoto-Bio P&C
Standing Detr. P&C
Unsaturated Water
Porewater P
Saturated Water
Organic Soil P&OM
http://www.sfwmd.gov/org/wrp/elm/struct/detail/eqns/eqns.htm
Above-ground:
water flows
P transformations
macrophyte gain/loss
periphyton gain/loss
floc gain/loss
Sloughs
Ridges
Plains
Prairies
Uplands
Below-ground:
water flows
P transformations
macrophyte gain/loss
peat gain/loss

Algae/Periphyton

2 communities, oligotrophic, or calcareous& eutrophic, or non-calcareous

Production Control Functions

Light

turbidity-effects (~no effect)

macrophyte shading

% light saturation

control function

Water

dryout effect

```
{
 algae/periphyton - dynamics of C and P of two communities of periphyton

 NC_ALG[cellLoc] == carbon mass of the NonCalcareous (more appropriately, the eutrophic, or non-native)
        periphyton community (gC/m^2)
 C_ALG[cellLoc] == carbon mass of the Calcareous (more appropriately, the oligotrophic, or native) periphyton
        community (gC/m^2)

 NC_ALG_P[cellLoc] == phosphorus mass of NonCalcareous (more appropriately, the eutrophic, or non-native)
        periphyton community (gP/m^2)
 C_ALG_P[cellLoc] == phosphorus mass of Calcareous (more appropriately, the oligotrophic, or native) periphyton
        community (gP/m^2)

Parameter definitions:
    global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
    habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}
{ light, water, temperature controls apply to both calc and non-calc }
ALG_LIGHT_EXTINCT  =  alg_light_ext_coef { changed light extinction from dynamic to fixed parm now, with no
        suspended organic matter }

{ algal self-shading implicit in density-dependent constraint function later }
ALG_INCID_LIGHT  = SOLRADGRD*Exp(-MAC_LAI*ALG_SHADE_FACTOR)
Z_extinct = SURFACE_WAT*ALG_LIGHT_EXTINCT
I_ISat = ALG_INCID_LIGHT/ALG_LIGHT_SAT
{ averaged over whole water column (based on Steele '65) }
ALG_LIGHT_CF   =
        if ( Z_extinct > 0.0 )
        then ( 2.718/Z_extinct * (Exp(-I_ISat * Exp(-Z_extinct)) - Exp(-I_ISat)) )
        else (I_ISat*Exp(1.0-I_ISat))

{ low-water growth constraint ready for something better based on data }
ALG_WAT_CF   =
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Temperature

control function

```
        if ( SURFACE_WAT>0.0 )
        then ( 1.0 )
        else ( 0.0)

{ Jorgensen 1976 }
ALG_TEMP_CF  = exp(-2.3*abs((H2O_TEMP-ALG_TEMP_OPT)/(ALG_TEMP_OPT-5.0)))

min_litTemp = Min(ALG_LIGHT_CF,ALG_TEMP_CF);
```

Nutrients

available P

```
{ the 2 communities have same form of growth response to avail phosphorus; avail PO4 is calc'd from TP, and is
        always at least 10% of TP conc }
{ using regression for predicting PO4 from TP }
PO4Pconc = Max(TP_SFWT_CONC_MG*PO4toTP + PO4toTPint, 0.10 * TP_SFWT_CONC_MG); { mg/L }
```

non-calc. control func.
calcar. control func.

```
NC_ALG_NUT_CF  = Exp(-alg_uptake_coef * Max(NC_ALG_KS_P-PO4Pconc, 0.0)/NC_ALG_KS_P)  { mg/L }
C_ALG_NUT_CF  = Exp(-alg_uptake_coef * Max(C_ALG_KS_P-PO4Pconc, 0.0)/C_ALG_KS_P)  { mg/L }
```

Total

non-calc. control func.
calcar. control func.

```
{ the form of the control function assumes that at very low P conc, the microbial assemblage scavenges P, allowing
        periph a minimum nutrient availability }
NC_ALG_PROD_CF  = Min(min_litTemp,ALG_WAT_CF)*Max(NC_ALG_NUT_CF, alg_alkP_min)
C_ALG_PROD_CF   = Min(min_litTemp,ALG_WAT_CF)*Max(C_ALG_NUT_CF, alg_alkP_min)
```

Mortality & respiration

```
NC_ALG_AVAIL_MORT  = Max(NC_ALG-ALG_REFUGE,0)
C_ALG_AVAIL_MORT  = Max(C_ALG-ALG_REFUGE,0)
```

non-calc. respiration potential

```
NC_ALG_RESP_POT   =
        if ( UNSAT_DEPTH>algMortDepth )
        then ( 0.0)
        else ( ALG_RC_RESP*ALG_TEMP_CF*NC_ALG_AVAIL_MORT )
```

calc. respiration potential

```
C_ALG_RESP_POT   =
        if ( UNSAT_DEPTH>algMortDepth )
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

	then (0.0) else (ALG_RC_RESP*ALG_TEMP_CF *C_ALG_AVAIL_MORT)
non-calc. respiration actual	NC_ALG_RESP = if (NC_ALG_RESP_POT*DT>NC_ALG_AVAIL_MORT) then (NC_ALG_AVAIL_MORT/DT) else (NC_ALG_RESP_POT)
calc. respiration actual	C_ALG_RESP = if (C_ALG_RESP_POT*DT>C_ALG_AVAIL_MORT) then (C_ALG_AVAIL_MORT/DT) else (C_ALG_RESP_POT)
	NC_ALG_PR1 = NC_ALG_RESP C_ALG_PR1 = C_ALG_RESP
calc.respiration response to P	{ this is the threshold control function that increases calcareous periph mortality (loss of calc. sheath?) as P increases } { moved the threshold response from respiration to mortality, as the respiration increase allowed the P:C to increase to great excess } C_ALG_thresh_CF = Min(exp(alg_R_accel*Max(TP_SFWT_CONC_MG- C_ALG_threshTP,0.0)/C_ALG_threshTP), 100.0)
non-calc. mortality potential	NC_ALG_MORT_POT = if (UNSAT_DEPTH>algMortDepth) then (NC_ALG_AVAIL_MORT*ALG_RC_MORT_DRY) else (NC_ALG_AVAIL_MORT*ALG_RC_MORT)
calc. mortality potential	C_ALG_MORT_POT = if (UNSAT_DEPTH>algMortDepth) then (C_ALG_AVAIL_MORT*ALG_RC_MORT_DRY) else (C_ALG_thresh_CF * C_ALG_AVAIL_MORT*ALG_RC_MORT)
non-calc. mortality	NC_ALG_MORT = if ((NC_ALG_MORT_POT+NC_ALG_PR1)*DT>NC_ALG_AVAIL_MORT)

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

actual

```
        then ( (NC_ALG_AVAIL_MORT-NC_ALG_PR1*DT)/DT )
        else ( NC_ALG_MORT_POT)
```

calc. mortality actual

```
C_ALG_MORT   =
        if ( (C_ALG_MORT_POT+C_ALG_PR1)*DT>C_ALG_AVAIL_MORT )
        then ( (C_ALG_AVAIL_MORT-C_ALG_PR1*DT)/DT )
        else ( C_ALG_MORT_POT)
```

Gross primary production

```
{ gross production of the 2 communities (gC/m2/d, NOT kgC/m2/d) }
{ density constraint contains both noncalc and calc, competition effect accentuated by calc algae }
NC_ALG_GPP  =  NC_ALG_PROD_CF*ALG_RC_PROD*NC_ALG*Max( (1.0-
        (AlgComp*C_ALG+NC_ALG)/ALG_MAX),0.0);
```

non-calc GPP

```
C_ALG_GPP   =  C_ALG_PROD_CF *ALG_RC_PROD* C_ALG*Max( (1.0-
        (C_ALG+NC_ALG)/ALG_MAX),0.0);
```

calc. GPP

```
{ P uptake is dependent on available P and relative to a maximum P:C ratio for the tissue (g C/m^2/d * P:Cmax *
        dimless * dimless = gP/m2/d (NOT kg) )}
NC_ALG_GPP_P = NC_ALG_GPP *ALG_PC * NC_ALG_NUT_CF* Max(1.0-NC_ALG_PC/ALG_PC, 0.0);
```

non-calc. P uptake

calc. P uptake

```
C_ALG_GPP_P =  C_ALG_GPP * ALG_PC * C_ALG_NUT_CF * Max(1.0-C_ALG_PC/ALG_PC, 0.0);

{ check for available P mass (the nutCF does not) }
PO4P = Min(PO4Pconc * SFWT_VOL,  1000.0*TP_SF_WT); {g P available (from conc. in mg/l = g/m3) }
```

check P mass in water & determine excess demand

```
reduc  =
        if ( (NC_ALG_GPP_P+C_ALG_GPP_P) > 0)
        then (PO4P / ( (NC_ALG_GPP_P+C_ALG_GPP_P)*CELL_SIZE*DT) )
        else (1.0)
        { can have high conc, but low mass of P avail, in presence of high peri biomass and high demand }
        { reduce the production proportionally if excess demand is found }

if (reduc < 1.0) then
(
NC_ALG_GPP = NC_ALG_GPP*reduc
NC_ALG_GPP_P = NC_ALG_GPP_P*reduc
```

reduce GPP if needed

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Update state variables

non-calc carbon update
calc. carbon update

non-calc. P loss with mortality

calc. P loss with mortality

non-calc phosphorus update
calc. phosphorus update

```
C_ALG_GPP = C_ALG_GPP*reduc
C_ALG_GPP_P = C_ALG_GPP_P*reduc
)

{ state variables calc'd (gC/m2, NOT kgC/m2) }
NC_ALG (t) = NC_ALG (t - dt)+(NC_ALG_GPP- NC_ALG_RESP - NC_ALG_MORT) * dt;
C_ALG (t) = C_ALG (t - dt) +(C_ALG_GPP - C_ALG_RESP  - C_ALG_MORT) * dt;

{  now calc the P  associated with the C fluxes (GPP_P already calc'd) }
mortPot =  NC_ALG_MORT * NC_ALG_PC;
NC_ALG_MORT_P  =
        if (mortPot*DT>NC_ALG_P)
        then (NC_ALG_P/DT)
        else (mortPot)
mortPot =  C_ALG_MORT * C_ALG_PC;
C_ALG_MORT_P  =
        if (mortPot*DT>C_ALG_P)
        then (C_ALG_P/DT)
        else (mortPot)


{ state variables calc'd (gP/m2, NOT kgP/m2) }
NC_ALG_P (t) = NC_ALG_P (t - dt) +(NC_ALG_GPP_P - NC_ALG_MORT_P) * dt;
C_ALG_P (t) = C_ALG_P (t - dt) +(C_ALG_GPP_P - C_ALG_MORT_P) * dt;

NC_ALG_PC  =
        if (NC_ALG>0.0)
        then (NC_ALG_P/ NC_ALG)
        else ( ALG_PC * 0.03) { default to 3% of  max P:C }
C_ALG_PC  =
        if (C_ALG>0.0)
        then (C_ALG_P/ C_ALG)
        else ( ALG_PC * 0.03 ) { default to 3% of max P:C }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Macrophytes

dynamics of macrophye community/habitat

Production Control Functions

- Light
- Temperature
- Water
- Nutrients
- Salinity

```
{
Macrophytes - dynamics (growth, mortality, etc) of macrophyte community/habitat
  MAC_NOPH_BIOMAS[cellLoc] == carbon mass of live Non-Photosynthetic tissues of macrophytes (kgC/m^2)
  MAC_PH_BIOMAS[cellLoc] == carbon mass of live Photosynthetic tissues of macrophytes (kgC/m^2)

  mac_nph_P[cellLoc] == phosphorus mass of live Non-Photosynthetic tissues of macrophytes (kgP/m^2)
  mac_ph_P[cellLoc] == phosphorus mass of live Photosynthetic tissues of macrophytes (kgP/m^2)

  mac_nph_OM[cellLoc] == organic matter mass of live Non-Photosynthetic tissues of macrophytes (kgOM/m^2)
  mac_ph_OM[cellLoc] == organic matter mass of live Photosynthetic tissues of macrophytes (kgOM/m^2)

spatial (cell) location defines habitat ( arrayOf[HAB[cellLoc]] );
habitat type can switch when global dynamic function (cell_dyn1) calls succession function (F_Hab_SwA())

Parameter definitions:
     global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
     habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}

{ various control functions for production calc }

MAC_LIGHT_CF  = SOLRADGRD/MAC_LIGHTSAT*Exp(1.0-SOLRADGRD/MAC_LIGHTSAT);
MAC_TEMP_CF  = exp(-2.3*abs((AIR_TEMP-MAC_TEMPOPT)/(MAC_TEMPOPT-5.0)))
MAC_WATER_CF  = Min(MAC_WATER_AVAIL_CF, Max(1.0-Max((SURFACE_WAT-
        MAC_WAT_TOLER)/MAC_WAT_TOLER,0.0),0.0));
        { earlier version included effect of po4 sfwat  }
MAC_NUT_CF  = Exp(-mac_uptake_coef * Max(MAC_KSP-TP_SEDWT_CONCACTMG, 0.0)/MAC_KSP) ; {
        mg/L }
MAC_SALT_CF  = 1.0; { tmp, removed this ( MAC_SALIN_THRESH>0.0 ) ? (  Max( 1.0-Max(SAL_MEAN-
        MAC_SALIN_THRESH,0.0)/MAC_SALIN_THRESH,0.0) ) : ( Max(1.0-SAL_MEAN,0.0)); }

min_litTemp = Min(MAC_LIGHT_CF, MAC_TEMP_CF);
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Total

Net primary production

NPP potential

```
MAC_PROD_CF  = Min(min_litTemp,MAC_WATER_CF)*MAC_NUT_CF *MAC_SALT_CF;


{ net primary production, kg C/m2/d }
PHBIO_NPP  = PHBIO_RCNPP*MAC_PROD_CF*MAC_PH_BIOMAS * (1.0-
        MAC_TOT_BIOM/MAC_MAX_BIO);
```

P uptake with NPP

```
{ P uptake is dependent on available P and relative to a maximum P:C ratio for the tissue (kg C/m^2/d * P:Cmax *
        dimless = kgP/m2/d ) }
NPP_P = PHBIO_NPP  * PHBIO_PC  * Max(MAC_NUT_CF*2.0,1.0) * Max(1.0-mac_ph_PC/PHBIO_PC, 0.0);
```

check P mass in water & determine excess demand

```
{ check for available P mass that will be taken up from sed water in active zone (nutCF does not) }
reduc  =
        if (NPP_P > 0.0)
        then (TP_SED_WT_AZ / ( NPP_P*CELL_SIZE*DT) )      { oct99 within-plant variable stoichiometry }
        else (1.0)
        { reduce the production proportionally if excess demand is found }
        { can have high conc, but low mass of P avail, in presence of high plant biomass and high demand }
```

reduce NPP if needed

```
if (reduc < 1.0) {
PHBIO_NPP = PHBIO_NPP*reduc
NPP_P = NPP_P*reduc }
```

Mortality & other loasses

PhotoBio

```
{ losses from photobio }
phbio_ddep = Max(1.0-Max( (PHBIO_SAT-MAC_PH_BIOMAS) /(PHBIO_SAT-PHBIO_REFUGE),0.0),0.0);
PHBIO_AVAIL  = MAC_PH_BIOMAS*phbio_ddep;
```

fire loss (zero fire)

```
PHBIO_FIRE   =
        if ( PHBIO_AVAIL*FIRE_SPECIFIC_RATE*DT>PHBIO_AVAIL )
        then ( PHBIO_AVAIL/DT )
        else ( PHBIO_AVAIL*FIRE_SPECIFIC_RATE)
PHBIO_PR1  = PHBIO_FIRE;
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

translocation potential

translocation to NPHbio

NonPhotoBio

fire loss (zero fire)

translocation potential

translocation to NPHbio

```
MAC_PHtoNPH_Init = PHBIO_MAX / NPHBIO_MAX ; {if habitat type changes }
MAC_PHtoNPH  =
        if (MAC_NOPH_BIOMAS>0.0)
        then ( MAC_PH_BIOMAS / MAC_NOPH_BIOMAS)
        else (MAC_PHtoNPH_Init)

NPHBIO_TRANSLOC_POT   =
        if (MAC_PHtoNPH>MAC_PHtoNPH_Init)
        then (exp(Mac_transl_rc *(MAC_PHtoNPH-MAC_PHtoNPH_Init)) - 1.0)
        else (0.0)
NPHBIO_TRANSLOC   =
        if ( (NPHBIO_TRANSLOC_POT+PHBIO_PR1)*DT >PHBIO_AVAIL )
        then ( (PHBIO_AVAIL-PHBIO_PR1*DT)/DT )
        else ( NPHBIO_TRANSLOC_POT)

{ losses from non-photobio  }
nphbio_ddep = Max(1.0-Max((NPHBIO_SAT-MAC_NOPH_BIOMAS)/(NPHBIO_SAT-
        NPHBIO_REFUGE),0.0),0.0);
NPHBIO_AVAIL  = MAC_NOPH_BIOMAS*nphbio_ddep;
NPHBIO_FIRE   =
        if ( NPHBIO_AVAIL*FIRE_SPECIFIC_RATE*NPHBIO_ABVBEL*DT >NPHBIO_AVAIL )
        then ( NPHBIO_AVAIL/DT )
        else ( NPHBIO_AVAIL*FIRE_SPECIFIC_RATE*NPHBIO_ABVBEL)
NPHBIO_PR1  = NPHBIO_FIRE;

PHBIO_TRANSLOC_POT   =
        if (MAC_PHtoNPH<MAC_PHtoNPH_Init)
        then (exp(Mac_transl_rc *(MAC_PHtoNPH_Init-MAC_PHtoNPH)) - 1.0)
        else (0.0)
PHBIO_TRANSLOC   =
        if ( (PHBIO_TRANSLOC_POT+NPHBIO_PR1)*DT >NPHBIO_AVAIL )
        then ( (NPHBIO_AVAIL-NPHBIO_PR1*DT)/DT )
        else ( PHBIO_TRANSLOC_POT)
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

mortality

```
NPHBIO_PR2  = NPHBIO_PR1+PHBIO_TRANSLOC;

NPHBIO_PR3  = NPHBIO_PR2;
NPHBIO_MORT_POT  = NPHBIO_AVAIL*PHBIO_RCMORT * (1.0 + Max(1.0-
        MAC_PH_BIOMAS/PHBIO_MAX,0.0) )/2.0; {oct99 decreased mort w/ increasing photobio }
NPHBIO_MORT   =
        if ( (NPHBIO_PR3+NPHBIO_MORT_POT)*DT>NPHBIO_AVAIL )
        then ( (NPHBIO_AVAIL-NPHBIO_PR3*DT)/DT )
        else ( NPHBIO_MORT_POT)
```

Update state variables

Carbon

NonPhoto carbon update

```
{ calc nonphotosynthetic biomass state var (kgC/m2) }

MAC_NOPH_BIOMAS (t) = MAC_NOPH_BIOMAS (t - dt)+(NPHBIO_TRANSLOC - NPHBIO_MORT -
        PHBIO_TRANSLOC - NPHBIO_FIRE ) * dt;
```

PhotoBio mortality

```
{ further loss calcs for photobio }
PHBIO_PR2  = NPHBIO_TRANSLOC+PHBIO_PR1;
PHBIO_MORT_POT  = PHBIO_RCMORT *PHBIO_AVAIL *(1.0 + (1.0-MAC_WATER_AVAIL_CF) )/2.0;
PHBIO_MORT   =
        if ( (PHBIO_MORT_POT+PHBIO_PR2)*DT>PHBIO_AVAIL )
        then ( (PHBIO_AVAIL-PHBIO_PR2*DT)/DT )
        else ( PHBIO_MORT_POT)
```

Photo carbon update

```
{ calc nonphotosynthetic biomass state var (kgC/m2) }

MAC_PH_BIOMAS (t) = MAC_PH_BIOMAS (t - dt)+(PHBIO_TRANSLOC + PHBIO_NPP - PHBIO_FIRE -
        PHBIO_MORT - NPHBIO_TRANSLOC) * dt;

{ book-keeping calcs used in other modules }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

"book-keeping"

Phosphorus & Organic Matter

(Note: the OM must be updated to accomodate habitat succession (& changes in the C:OM fixed parm)

```
MAC_TOT_BIOM  = MAC_PH_BIOMAS+MAC_NOPH_BIOMAS;
MAC_REL_BIOM   =
        if ( MAC_TOT_BIOM > 0.0 )
        then MAC_TOT_BIOM/MAC_MAX_BIO
        else 0.0001
MAC_HEIGHT  = MAC_REL_BIOM^0.33*MAC_MAXHT;
MAC_LAI  = MAC_REL_BIOM*MAC_MAXLAI;
{ note that an "effective" LAI that accounts for water depth is calc'd in hydro module }

{  now calc the P and organic matter associated with the C fluxes }
        {  phbio_npp_P = PHBIO_NPP * PHBIO_PC;  }    { habitat-specfic stoichiometry }
phbio_npp_P = NPP_P     { oct99 within-plant variable stoichiometry }
phbio_npp_OM = PHBIO_NPP / PHBIO_CTOOM { habitat-specfic stoichiometry }

phbio_fire_P = PHBIO_FIRE * mac_ph_PC
phbio_fire_OM = PHBIO_FIRE / mac_ph_CtoOM;

phbio_mort_P = PHBIO_MORT * mac_ph_PC
phbio_mort_OM = PHBIO_MORT / mac_ph_CtoOM

phbio_transl_P = PHBIO_TRANSLOC * mac_nph_PC
phbio_transl_OM = PHBIO_TRANSLOC / mac_nph_CtoOM

nphbio_transl_P = NPHBIO_TRANSLOC * mac_ph_PC
nphbio_transl_OM = NPHBIO_TRANSLOC / mac_ph_CtoOM

nphbio_fire_P = NPHBIO_FIRE * mac_nph_PC
nphbio_fire_OM = NPHBIO_FIRE / mac_nph_CtoOM

nphbio_mort_P = NPHBIO_MORT * mac_nph_PC
nphbio_mort_OM = NPHBIO_MORT / mac_nph_CtoOM

{ state vars: now calc the P and OM associated with those C state vars }
```

a code "un-generator" parsed these eqn's from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

NonPhoto phosphorus update

```
mac_nph_P (t) = mac_nph_P(t - dt)+(nphbio_transl_P - nphbio_mort_P- phbio_transl_P - nphbio_fire_P ) * dt;
mac_nph_PC  =
        if ( (MAC_NOPH_BIOMAS > 0.0) and (mac_nph_P > 0.0))
        then (mac_nph_P / MAC_NOPH_BIOMAS)          {dec99 these second mass checks not needed now (were
        probably not good to start with) }
        else 0.3 * NPHBIO_PC { default to 0.3 of max for habitat }
```

NonPhoto Organic Matter update

```
mac_nph_OM (t) = mac_nph_OM(t - dt)+(nphbio_transl_OM - nphbio_mort_OM- phbio_transl_OM -
        nphbio_fire_OM ) * dt;
mac_nph_CtoOM  =
        if ( (mac_nph_OM > 0.0) and (MAC_NOPH_BIOMAS>0.0))
        then (MAC_NOPH_BIOMAS / mac_nph_OM)
        else NPHBIO_CTOOM
```

Photo phosphorus update

```
mac_ph_P (t) = mac_ph_P(t - dt)+(phbio_transl_P + phbio_npp_P - phbio_mort_P- nphbio_transl_P - phbio_fire_P )
        * dt;
mac_ph_PC  =
        if ( (MAC_PH_BIOMAS > 0.0) and (mac_ph_P>0.0))
        then (mac_ph_P / MAC_PH_BIOMAS)
        else 0.3 * PHBIO_PC { default to 0.3 of max for habitat }
```

Photo Organic Matter update

```
mac_ph_OM (t) = mac_ph_OM(t - dt)+(phbio_transl_OM + phbio_npp_OM - phbio_mort_OM- nphbio_transl_OM -
        phbio_fire_OM ) * dt;
mac_ph_CtoOM  =
        if ( (mac_ph_OM > 0.0) and (MAC_PH_BIOMAS>0.0))
        then (MAC_PH_BIOMAS / mac_ph_OM)
        else PHBIO_CTOOM
```

Update pore water TP

total pore water

```
{******* phosphorus removed from water here ***********}
TP_SEDWT_UPTAKE  = phbio_npp_P*CELL_SIZE;
{ recalc P in sed water state variable (kg P) }
TP_SED_WT (t) = TP_SED_WT (t - dt)+ (TP_DNFLOW + TP_SED_MINER - TP_UPFLOW - TP_SORBTION -
```

phosphorus

active zone pore water phosphorus

```
        TP_SEDWT_UPTAKE) * dt;
TP_SED_CONC  =
        if (HYD_SED_WAT_VOL>0.0)
        then (TP_SED_WT / HYD_SED_WAT_VOL)
        else (0.0)

        { this is the active zone, where uptake, sorption, and mineralization take place }
TP_SED_WT_AZ = TP_SED_CONC * TP_Act_to_Tot * HYD_DOM_ACTWAT_VOL;
TP_SEDWT_CONCACT  =
        if ( HYD_DOM_ACTWAT_PRES > 0.0)
        then ( TP_SED_WT_AZ/HYD_DOM_ACTWAT_VOL )
        else (TP_SED_CONC) { g/L }
```

Soil

dynamics of Deposited Organic Matter & P

Inputs

macrophytes
standing detritus
floc

Outputs

fire (zero)

P (quality) control function on floc decomposition

temperature control function

Aerobic/anaerobic

```
{
 Organic Soil: Deposited Organic Matter and Deposited Organic Phosphorus dynamics

    DEPOS_ORG_MAT[cellLoc] == mass of Deposited Organic Matter (DOM/AOM) of soil zone, w/o floc layer
        (kg OM/m^2)

    DOP[cellLoc] ==  mass of Deposited Organic Phosphorus of soil zone, w/o floc layer, w/o P sorbed (kg P/m^2)

Parameter definitions:
     global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
     habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}
{ inputs of organic matter (kg OM/m2)}
{  non-photo goes either to stand det or deposited OM, photo goes either to stand det or floc }
DOM_fr_nphBio = DOM_NPHBIO_PROP*nphbio_mort_OM;
DOM_FR_STDET  = stdet_toDOM_OM;
DOM_FR_FLOC  =  FLOC_DEPO ;

{ losses of organic matter (kg OM/m2) }
{ fire is zero }
DOM_FIRE   =
          if ( DEPOS_ORG_MAT*FIRE_SPECIFIC_RATE*DT>DEPOS_ORG_MAT )
          then ( DEPOS_ORG_MAT/DT )
          else ( DEPOS_ORG_MAT*FIRE_SPECIFIC_RATE)
DOM_PR1  = DOM_FIRE;

DOM_QUALITY_CF  =
Min(Exp(-DOM_decomp_coef * Max(DOM_DECOMP_POPT-TP_SEDWT_CONCACTMG, 0.0)/
          DOM_DECOMP_POPT),1.0) ; { mg/L }
DOM_TEMP_CF = exp(-2.3*abs((H2O_TEMP-DOM_DECOMP_TOPT)/(DOM_DECOMP_TOPT-5.0)))

DOM_SED_AEROB_Z  = Min(Max(UNSAT_DEPTH,DOM_AEROBTHIN),DOM_MAXDEPTH);
DOM_SED_ANAEROB_Z  = DOM_MAXDEPTH-DOM_SED_AEROB_Z;
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

depths

Potential decomposition

Actual decomposition

Update state variable

Organic Matter

update organic matter of soil

Soil depth and land surface elevation

Phosphorus

P loss with decomposition

Update P in soil

```
        { beta3 is the calib parm read from Driver.parm }
DOM_DECOMP_POT  =
        beta3*DOM_RCDECOMP*DOM_QUALITY_CF*DOM_TEMP_CF*DEPOS_ORG_MAT
        *(Min(DOM_SED_AEROB_Z/DOM_MAXDEPTH,1.0)*DOM_MOIST_CF
        +DOM_DECOMPRED*Min(DOM_SED_ANAEROB_Z/DOM_MAXDEPTH,1.0) );
DOM_DECOMP   =
        if ( (DOM_DECOMP_POT+DOM_PR1)*DT>DEPOS_ORG_MAT )
        then ( (DEPOS_ORG_MAT-DOM_PR1*DT)/DT )
        else ( DOM_DECOMP_POT)

{ calc state var (kg OM/m2) }

DEPOS_ORG_MAT  = DEPOS_ORG_MAT +( DOM_fr_nphBio + DOM_FR_STDET + DOM_FR_FLOC -
        DOM_DECOMP - DOM_FIRE) * dt;

{ soil elevation }
DOM_Z =  DEPOS_ORG_MAT / DOM_BD ; { (m) organic depth  }
SED_ELEV  = DOM_Z+Inorg_Z+SED_INACT_Z;  { total elevation (m) }

{ P DOM stoich (kg P /m2) }
DOP_nphBio = DOM_NPHBIO_PROP*nphbio_mort_P;
DOP_STDET = stdet_toDOM_P ;
DOP_FLOC = FlocP_DEPO;

DOP_DECOMP =  DOM_DECOMP * DOM_P_OM;
{ not accounting for fire and consumer effects (neither operating now) }

{ calc state var of total P in soil (NOT including dissolved in pore water or sorbed) (kgP/m2) }

DOP  = DOP  +( DOP_nphBio + DOP_STDET + DOP_FLOC - DOP_DECOMP) * dt; { kgP/m2 }

{ now the P ratio }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

P:OM ratio

Total mass of P in soil
Soil TP concentration (mass/mass)

Soil TP concentration (mass/volume)

TP in water

TP mineralized

update TP in porewater

update TP in porewater of biologically active zone

```
DOM_P_OM  =
        if (DEPOS_ORG_MAT>0.0)
        then ( DOP / DEPOS_ORG_MAT)
        else (0.0) { kgP/kgOM }
TPsoil = DOP*CELL_SIZE + TP_SORB; { kg TP in soil }
TPtoSOIL  =
        if ((DEPOS_ORG_MAT*CELL_SIZE + DIM)>0.0)
        then (  TPsoil / (DEPOS_ORG_MAT*CELL_SIZE + DIM) )
        else (0.0) { kgP/kgsoil }
TPtoVOL  =
        if (CELL_SIZE * DOM_Z>0.0)
        then (TPsoil / (CELL_SIZE * DOM_Z) )
        else (0.0) { kgP/m3 soil }

{ now the P gain in sed water with decomp; a small proportion goes into surface water P (below) }
TP_sedMin  =  (1.0 - DOM_AEROBTHIN  / DOM_MAXDEPTH ) * DOP_DECOMP * CELL_SIZE;

{ calc P in sed water state variables (kg P) }
TP_SED_WT = TP_SED_WT + (TP_sedMin) * dt;

{ this is the active zone, where uptake, sorption, and mineralization take place }
TP_SED_WT_AZ = TP_SED_WT_AZ + (TP_sedMin) * dt

TP_SED_CONC  =
        if (HYD_SED_WAT_VOL>0.0)
        then (TP_SED_WT / HYD_SED_WAT_VOL)
        else (0.0)
TP_SEDWT_CONCACT  =
        if ( HYD_DOM_ACTWAT_PRES > 0.0)
        then ( TP_SED_WT_AZ/HYD_DOM_ACTWAT_VOL )
        else (TP_SED_CONC) { g/L }

{ now store the ratio of the conc in the active zone relative to total, prior to horiz fluxes
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

TP in surface water	****code in transition***** } TP_Act_to_Tot = 1.0 / TP_CONC_GRAD; { now the P gain in surface water with decomp in the very thin upper layer of the soil } { if there is no surface water present, assume that this relative contribution will be an additional sorbed component that is introduced to surface water column immediately upon hydration with surface water } TP_sfMin = DOM_AEROBTHIN / DOM_MAXDEPTH * DOP_DECOMP * CELL_SIZE; { calc P in surface water state variable (kg P) } TP_SF_WT = TP_SF_WT + (TP_sfMin) * dt; TP_SFWT_CONC = if (SFWT_VOL > 0.0) then (TP_SF_WT/SFWT_VOL) else (0.0) { used in P fluxes for mass balance }

Floc

dynamics of Floc P and Organic Matter (soil-surface water interface)

Inputs

Periphyton/Algal mortality

Photosynthetic Macrophyte mortality

Surface water

Particulate P settling from water column

Particulate OM settling from water column

Outputs

P (quality) control function on floc decomposition

```
{
FLOCculent organic matter dynamics
  FLOC[cellLoc] == mass of organic flocculent material at the interface between soil and surface water (kg
        OM/m^2)
  FlocP[cellLoc] == mass of phosphorus in the flocculent material at the interface between soil and surface water (kg
        P/m^2)

this routine originally was Suspended Organic Matter; was dramatically modified to instead
represent the organic matter in the flocculent sediment layer

Parameter definitions:
     global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
     habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}

{ inputs (kg OM / m2)  }
{ all periphyton mortality goes to floc }
FLOC_FR_ALGAE  =  (C_ALG_MORT+NC_ALG_MORT)/ALG_C_TO_OM*0.001 ;
{  non-photo goes either to stand det or deposited OM, photo goes either to stand det or floc }
Floc_fr_phBio  = DOM_PHBIO_PROP*phbio_mort_OM;

        { all settling from water column goes to floc }
FlocP_settl = TP_settl / CELL_SIZE; { kg P/m2; }
        { the particulate P settling is assumed a fixed Redfield P:OM ratio }
Floc_settl =   FlocP_settl / TP_P_OM


{ outputs (kg OM / m2) }
{ use the avg conc of sed and sf water here }
FLOC_DECOMP_QUAL_CF  =  Exp(-DOM_decomp_coef * Max(DOM_DECOMP_POPT-
        (TP_SFWT_CONC_MG+TP_SEDWT_CONCACTMG)/2.0, 0.0)/DOM_DECOMP_POPT)  { mg/L }
{ decomp in surface water has higher rate than in soil, assuming this stock is of much higer substrate quality than the
        total soil layer }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Moisture control function

Potential decomposition

Actual decomposition

Incorporate (“deposit”) floc into soil variable(s)

Update state variable

Floc OM update

Floc depth

```
{ beta3 is the calib parm read from Driver.parm }
DOM_MOIST_CF   =
        if (UNSAT_DEPTH>DOM_AEROBTHIN)
        then ( Max(UNSAT_MOIST_PRP,0.0) )
        else ( 1.0 )
FLOC_DECOMP_POT  = beta3 * 10.0*DOM_RCDECOMP* FLOC*DOM_TEMP_CF *
        FLOC_DECOMP_QUAL_CF * DOM_MOIST_CF;
FLOC_DECOMP   =
        if ( (FLOC_DECOMP_POT)*DT>FLOC )
        then ( (FLOC)/DT )
        else ( FLOC_DECOMP_POT)
FLOC_PR1  = FLOC_DECOMP;

{ the incorporation of the floc layer into the "true" soil layer occurs at a rate dependent on the floc depth under
        flooded conditions, then constant rate under dry conditions }
FLOC_DEPO_POT   =
        if ( SURFACE_WAT > DetentZ )
        then ( FLOC_Z/FlocMax * FLOC*Floc_rcSoil )
        else ( FLOC*Floc_rcSoil)
FLOC_DEPO   =
        if ( (FLOC_DEPO_POT+FLOC_PR1)*DT>FLOC )
        then ( (FLOC-FLOC_PR1*DT)/DT )
        else ( FLOC_DEPO_POT)
{ calc the state var (kg OM / m2) }

FLOC  = FLOC +( Floc_settl + Floc_fr_phBio + FLOC_FR_ALGAE - FLOC_DECOMP - FLOC_DEPO ) * dt;

{ the depth of floc is dependent on a fixed floc bulk density }
FLOC_Z =  FLOC / Floc_BD ;

{ book-keeping calcs used in other modules }
{  FLOC_C_AVAIL  = FLOC/CELL_SIZE*FLOC_CTOOM; }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Floc phosphorus

periphyton P

macrophyte P

P loss with decomposition

P loss to “true” soil

Floc P update

update P:OM ratio

TP in water

TP in pore (sediment) water

TP in porewater of biologically active zone

```
{ Floc phosphorus (kg P / m2)  }
FlocP_FR_ALGAE  =  (NC_ALG_MORT_P+ C_ALG_MORT_P)*0.001; { kg P/m2 }
FlocP_PhBio = DOM_PHBIO_PROP*phbio_mort_P ;

FlocP_DECOMP_pot =  FLOC_DECOMP * FlocP_OM;
FlocP_DECOMP   =
        if ( (FlocP_DECOMP_pot)*DT>FlocP )
        then ( (FlocP)/DT )
        else ( FlocP_DECOMP_pot)
FlocP_DEPO_pot =  FLOC_DEPO * FlocP_OM;
FlocP_DEPO   =
        if ( (FlocP_DEPO_pot+FlocP_DECOMP)*DT>FlocP )
        then ( (FlocP-FlocP_DECOMP*DT)/DT )
        else ( FlocP_DEPO_pot)

{ calc the state var (kg P / m2) }

FlocP  = FlocP +( FlocP_settl + FlocP_PhBio + FlocP_FR_ALGAE- FlocP_DECOMP - FlocP_DEPO ) * dt;

FlocP_OM  =
        if ( FLOC>0.0)
        then (FlocP/FLOC)
        else (0.0) { kgP/kgOM }

{ now the P gain in sediment pore water with decomp - 90% goes to porewater, 10% to sfwat }
TP_SED_MINER   =  0.90 * FlocP_DECOMP * CELL_SIZE

{ calc P in sed water state variables (kg P) }
TP_SED_WT  = TP_SED_WT  + ( TP_SED_MINER ) * dt

{ this is the active zone, where uptake, sorption, and mineralization take place }
TP_SED_WT_AZ = TP_SED_WT_AZ  + ( TP_SED_MINER ) * dt
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

TP in surface water	TP_SED_CONC =
	if (HYD_SED_WAT_VOL>0.0)
	then (TP_SED_WT / HYD_SED_WAT_VOL)
	else (0.0)
	TP_SEDWT_CONCACT =
	if (HYD_DOM_ACTWAT_PRES > 0.0)
	then (TP_SED_WT_AZ/HYD_DOM_ACTWAT_VOL)
	else (TP_SED_CONC)
	TP_SEDWT_CONCACTMG = TP_SEDWT_CONCACT*1000.0;
	
	{ now the P gain in surface water with decomp - 90% goes to porewater, 10% to sfwat }
	TP_SFWT_MINER = 0.10*FlocP_DECOMP * CELL_SIZE;
	
	{ calc P in surface water state variable (kg P) }
	
	TP_SF_WT = TP_SF_WT + (TP_SFWT_MINER) * dt;
	TP_SFWT_CONC =
	if (SFWT_VOL > 0.0)
	then (TP_SF_WT/SFWT_VOL)
	else (0.0) { used in P fluxes for mass balance }

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Hydrology

dynamics of surface, unsaturated, and saturated water storages

Meteorological forcings

Rain

Evaporation – intermediate calcs

Evap parms/eqns from Christiansen (1968)

Saturation deficit

```
{
 Hydrology (vertical fluxes only)

 SURFACE_WAT[cellLoc] == water storage above the land surface elevation (meters)
 UNSAT_WATER[cellLoc] == water storage in the pore spaces of the unsaturated zone (meters)
 SAT_WATER[cellLoc] == water storage in the pore spaces of the saturated zone (meters)

 (the horizontal solution, horizFlow function, makes other vert calcs to
 integrate the horiz flows within the three vertical zones of water storage)

Parameter definitions:
     global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
     habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}

{  note that rainfall during a time step is added to surface water storage and available }
{ for runoff (horizFlow) before the calc of infiltration & ET associated with that new input }
{ (infiltration/ET etc will be of avail water the next time step after a rainfall event and horiz flows) }


        { data arrays were created in cell_dyn1 }
SF_WT_FROM_RAIN  = (HYD_PRECIP_DAY*0.0254)


{******* meteorological calcs: temperature, cloudiness, humidity (&dew point temp), and wind speed spatially
        distributed }
        { read in, interpolate (cell_dyn1) daily data on avg air temp, avg wind spd, cloudiness, and dew point temp }
VapPres_DPT =   610.78*Exp(17.269*DewPt/(DewPt+237.3));
VAP_PRESS_SAT = 610.78*Exp(17.269*AIR_TEMP/(AIR_TEMP+237.3));
HUMIDITY =  Min( VapPres_DPT / VAP_PRESS_SAT, 1.0);
VAP_SAT_DEFICIT = 1.0 - HUMIDITY;
WIND_SPEED_CANOPY = WIND_SPEED*0.666;
hum2 = HUMIDITY*HUMIDITY;
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

solar radiation parms/eqns from Nikolov & Zeller (1992)

Ground solar radiation (cal/cm2/d)

Capacities & potentials

Unsaturated capacity

Unsat moisture

relative thickness of surface & unsaturated

unsaturated hydraulic conductivity

field capacity

Potential evap and

```
PAN_CH = 1.035+0.240*hum2/0.36-0.275*hum2*HUMIDITY/0.216 ;
PAN_CT = 0.463+0.425*(AIR_TEMP/20.0)+0.112*AIR_TEMP*AIR_TEMP/400.0;
PAN_CW =  0.672+0.406*(WIND_SPEED_CANOPY/6.7)-
        0.078*WIND_SPEED_CANOPY*WIND_SPEED_CANOPY/44.89 ;
SOLRAD274 = SOLRADATMOS*(SOLBETA-SOLOMEGA*
        (( CLOUDY>0.0 ) ?
        ( CLOUDY ) : ( 0.0)))-SOLALPHA;
SOLRADGRD = SOLRAD274+((SOLRADATMOS+1.0)-SOLRAD274)*SOLALTCORR;
H2O_TEMP = AIR_TEMP;

{******* determine new unsat potentials }
SAT_WT_HEAD  = SAT_WATER/HYD_POROSITY;
UNSAT_DEPTH  = SED_ELEV-SAT_WT_HEAD;
UNSAT_CAP  =  UNSAT_DEPTH*HYD_POROSITY;

UNSAT_MOIST_PRP   =
        if ( UNSAT_CAP>0.0 )
        then ( Min(UNSAT_WATER/UNSAT_CAP,1.0) )
        else ( 1.0)
        { determining the pathway of flow of surface water depending on depth of an unsat zone relative to the
        surface water  }
SAT_VS_UNSAT  = 1/Exp(100.0*Max((SURFACE_WAT-UNSAT_DEPTH),0.0));
UNSAT_HYD_COND_CF  = GRAPH(UNSAT_MOIST_PRP) (0.0, 0.0), (0.1, 0.02), (0.20, 0.04), (0.30, 0.07),
        (0.40, 0.125), (0.50, 0.215), (0.60, 0.345), (0.70, 0.575), (0.80, 0.855), (0.90, 0.985), (1, 1)
        { field capacity = porosity - specific yield; spec yield= proportion of total soil vol that represents water that
        can be moved by gravity }
field_cap = (HYD_POROSITY-HYD_SPEC_YIELD);
        { unsat_avail is proportion of water in pore space available for gravitational flow (above field capacity) }
        { e.g., when moisture prop in pore space <= field_cap/pore_space, no percolation }
        { using old moisture proportion (hasn't changed unless unsat zone was replaced by sat water) }
UNSAT_AVAIL  = Max(UNSAT_MOIST_PRP-(field_cap)/HYD_POROSITY,0.0);
UNSAT_WT_POT  = Max(UNSAT_CAP-UNSAT_WATER,0.0);
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

transpiration

potential canopy transpiration

potential evaporation

effective Leaf Area Index

Total potential transpiration

capillary root withdrawal function

Potential & actual

```
{******* now determine the potential total transpiration and evaporation  }
HYD_CANOPY_TRANSP  = 0.75*DAYLENGTH
        * (MAC_MAXCANOPCOND*3600.0*0.018*0.001)* VAP_SAT_DEFICIT/1E5;
        { beta2 is a global calibration parameter (near 1.0, read from Driver.parm) for evaporation }
HYD_EVAP_CALC  = 0.482*SOLRADGRD/585.0
        *PAN_CW*PAN_CT*PAN_CH*0.01*HYD_PAN_K*beta2;
        {  effective LAI accounts for leaf area submerged }
LAI_eff  =
        if (MAC_HEIGHT>0.0)
        then (Max(1.0 - SURFACE_WAT/MAC_HEIGHT, 0.0)*MAC_LAI)
        else (0.0)
        { total (canopy-driven and evap-driven) potential transpiration without water limitation }
f_LAI_eff = exp(-LAI_eff);

HYD_TOT_POT_TRANSP  = (HYD_CANOPY_TRANSP*(1.0-MAC_CANOPDECOUP)
        + (HYD_EVAP_CALC*MAC_CANOPDECOUP) )
        *(1.0-f_LAI_eff);

        { 0-1 control function of saturated water available to roots -  capillary draw when roots don't quite reach down
        to water table }
SatWat_Root_CF =  Exp(-10.0* Max(UNSAT_DEPTH-NPHBIO_ROOTDEPTH,0.0) );
        { 0-1 proportion representing available water for determining plant growth stress: no stress to macrophyte
        root system if unsat depth < root depth, otherwise  function of moisture proportion within root zone }
HYD_WATER_AVAIL   =
        if (UNSAT_DEPTH > NPHBIO_ROOTDEPTH )
        then ( Max(UNSAT_MOIST_PRP, SatWat_Root_CF) )
        else ( 1.0 ) { dec98 }
MAC_WATER_AVAIL_CF  = GRAPH(HYD_WATER_AVAIL) (0, 0.005), (0.10, 0.01), (0.20, 0.04), (0.30, 0.10),
        (0.40, 0.28), (0.50, 0.715), (0.60, 0.865), (0.70, 0.935), (0.80, 0.975), (0.90, 0.995), (1, 1)

{******* next calc the potential and actual flows }
{ unsatLoss(1) unsat to sat percolation }
{unsat to sat flow here only includes percolation (rising water table accomodated in update after horiz fluxes) }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

flows

percolation

transpiration potential from unsat storage

transpiration from unsat storage

Recharge rate is globally zero (currently)

no effect (zero recharge)

```
UNSAT_PERC  = Min(HYD_RCINFILT*UNSAT_HYD_COND_CF,UNSAT_AVAIL*UNSAT_WATER);
UNSAT_TO_SAT_FL   =
        if ( (UNSAT_PERC)*DT > UNSAT_WATER )
        then ( UNSAT_WATER/DT )
        else ( UNSAT_PERC)
UNSAT_PR1  = UNSAT_TO_SAT_FL;
{ unsatLoss(2) unsat to atmosph transpiration }
HYD_UNSAT_POT_TRANS   =
        if (UNSAT_DEPTH > NPHBIO_ROOTDEPTH )
        then ( HYD_TOT_POT_TRANSP*MAC_WATER_AVAIL_CF )
        else (0.0) { no unsat transp if roots are in saturated zone, dec98 }
UNSAT_TRANSP   =
        if ( (HYD_UNSAT_POT_TRANS+UNSAT_PR1)*DT>UNSAT_WATER )
        then ( (UNSAT_WATER-UNSAT_PR1*DT)/DT )
        else ( HYD_UNSAT_POT_TRANS)

{ satLoss(1) sat to deep aquifer recharge **RATE IS ALWAYS ZERO RIGHT NOW ****}
SAT_WT_RECHG   =
        if ( HYD_RCRECHG*HYD_SPEC_YIELD/HYD_POROSITY*DT>SAT_WATER )
        then ( SAT_WATER/DT )
        else ( HYD_RCRECHG*HYD_SPEC_YIELD/HYD_POROSITY) { !!!! THIS RATE IS ALWAYS ZERO }

{ sat to surf upflow  when gwater exceeds soil capacity due to lateral inflows accomodated in gwFluxes }

{ satLoss(2) sat to unsat with lowering water table due to recharge loss ONLY (horiz outflows accomodated in
        gwFluxes)       (leaves field capacity amount in unsat zone)}
SAT_TO_UNSAT_FL   =
        if ( SAT_WT_RECHG*field_cap/HYD_POROSITY*DT > SAT_WATER )
        then ( (SAT_WATER)/DT )
        else ( SAT_WT_RECHG*field_cap/HYD_POROSITY)
SAT_WT_PR1  = SAT_TO_UNSAT_FL;
{ satLoss(3) sat to atmosph }
HYD_SAT_POT_TRANS  = HYD_TOT_POT_TRANSP*SatWat_Root_CF; {dec98}
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

transpiration potential & actual from saturated storage

```
SAT_WT_TRANSP   =
        if ( (HYD_SAT_POT_TRANS+SAT_WT_PR1)*DT > SAT_WATER )
        then ( (SAT_WATER-SAT_WT_PR1*DT)/DT )
        else ( HYD_SAT_POT_TRANS)
```

surface water downflow to saturated under "deep" pond conditions

```
{ sfwatLoss(1) sf to sat }
{ downflow to saturated assumed to occur in situations with small
unsat layer overlain by significant surface water (SAT_VS_UNSAT), with immediate hydraulic connectivity betweent
        the two storages }
SF_WT_TO_SAT_DOWNFLOW   =
        if ( (1.0-SAT_VS_UNSAT)*UNSAT_WT_POT*DT>SURFACE_WAT )
        then ( SURFACE_WAT/DT )
        else ( (1.0-SAT_VS_UNSAT)*UNSAT_WT_POT)
```

surface infiltration to unsaturated

```
{ sfwatLoss(2) sf to unsat infiltration (sat_vs_unsat splits these losses to groundwater, but downflow to sat is given
        priority) }
SF_WT_POT_INF   =
        if ( (SAT_VS_UNSAT*HYD_RCINFILT+SF_WT_TO_SAT_DOWNFLOW)*DT>SURFACE_WAT )

        then ( (SURFACE_WAT-SF_WT_TO_SAT_DOWNFLOW*DT)/DT )
        else ( SAT_VS_UNSAT*HYD_RCINFILT)
SF_WT_INFILTRATION   =
        if ( SF_WT_POT_INF*DT > (UNSAT_WT_POT-SF_WT_TO_SAT_DOWNFLOW*DT) )
        then ((UNSAT_WT_POT-SF_WT_TO_SAT_DOWNFLOW*DT)/DT )
        else ( SF_WT_POT_INF)
SFWAT_PR1  = SF_WT_INFILTRATION+SF_WT_TO_SAT_DOWNFLOW;
```

actual evaporation

```
{ sfwatLoss(3) sf to atmosph }
SF_WT_EVAP   =
        if ( (f_LAI_eff*HYD_EVAP_CALC+SFWAT_PR1 )*DT>SURFACE_WAT )
        then ( (SURFACE_WAT-SFWAT_PR1*DT)/DT )
        else ( f_LAI_eff*HYD_EVAP_CALC)

{******* then update the state variable storages }
```

a code "un-generator" parsed these eqn's from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Update state variables

Unsaturated storage

```
{ unsat loss priority:  percolation,  transpiration }
{ calc unsaturated storage state var (m) }

UNSAT_WATER (t) = UNSAT_WATER (t - dt) +(SAT_TO_UNSAT_FL + SF_WT_INFILTRATION -
        UNSAT_TO_SAT_FL - UNSAT_TRANSP) * dt;
```

Saturated storage

```
{ sat loss priority:  recharge to deep aquifer, re-assign to unsat with lowered water table, transpiration }
{ calc saturated storage state var (m) }

SAT_WATER (t) = SAT_WATER (t - dt)+(UNSAT_TO_SAT_FL + SF_WT_TO_SAT_DOWNFLOW -
        SAT_WT_TRANSP - SAT_TO_UNSAT_FL - SAT_WT_RECHG) * dt;
```

Surface storage

```
{ sfwat loss priority: downflow to replace groundwater loss, infiltration to unsat, evaporation }
{ calc surface storage state var (m) }

SURFACE_WAT (t) = SURFACE_WAT (t - dt)+(SF_WT_FROM_RAIN  - SF_WT_EVAP -
        SF_WT_INFILTRATION - SF_WT_TO_SAT_DOWNFLOW) * dt;
```

Update heads, capacities

Saturated

Unsaturated

Biologically "active" zone volume (not depth)

```
{******* lastly, update of head calcs, unsat capacity, moisture proportion, etc. in order to calc water in diff storages
        for solute concentrations }
SAT_WT_HEAD  = SAT_WATER/HYD_POROSITY;
UNSAT_DEPTH  = Max(SED_ELEV-SAT_WT_HEAD,0.0);
UNSAT_CAP  =  UNSAT_DEPTH*HYD_POROSITY;
UNSAT_MOIST_PRP   =
        if ( UNSAT_CAP>0.0 )
        then ( Min(UNSAT_WATER/UNSAT_CAP,1.0) )
        else ( 1.0)
HYD_DOM_ACTWAT_VOL  = ( Min(DOM_MAXDEPTH,UNSAT_DEPTH)*UNSAT_MOIST_PRP +
        Max(DOM_MAXDEPTH-UNSAT_DEPTH, 0.0)*HYD_POROSITY ) *CELL_SIZE;
        { flag for presence of small amount of water storage in the active zone must be present }
HYD_DOM_ACTWAT_PRES   =
        if ( HYD_DOM_ACTWAT_VOL > CELL_SIZE*0.01 )
        then ( 1.0 )
```

a code "un-generator" parsed these eqn's from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Manning's N calculation relative to water depth, plant density, plant height	else (0.0) HYD_SED_WAT_VOL = (SAT_WATER+UNSAT_WATER)*CELL_SIZE SFWT_VOL = SURFACE_WAT*CELL_SIZE HydTotHd = SAT_WT_HEAD+SURFACE_WAT; { only used for reporting purposes } { at the square of xx% of the macrophyte's height, the manning's n calc will indicate the macrophyte *starting* to bend over, starting to offer increasingly less resistance } mann_height = (mann_height_coef*MAC_HEIGHT)*(mann_height_coef*MAC_HEIGHT); {oct99 } N_density = Max(MAC_MAXROUGH * MAC_REL_BIOM, MAC_MINROUGH); { manning's n for overland (horiz) flow } HYD_MANNINGS_N = Max(-Abs((N_density-MAC_MINROUGH) *(2.0^(1.0-SURFACE_WAT/mann_height)-1.0)) + N_density,MAC_MINROUGH);

Phosphorus

dynamics of TP in surface water, pore (ground) waters, and sorbed to soil

Update concentrations

Surface water TP conc after horizontal flows

Surface water bioavailable P conc after horizontal flows

Total pore (sediment) water TP conc after horizontal flows

TP mass & concentration in biologically active zone

Inputs/Outputs

atmospheric deposition (rain only here)

```
{
Phosphorus dynamics in water and sorbed to soil

TP_SF_WT[cellLoc] == mass of P in surface water (kg P)
TP_SED_WT[cellLoc] == mass of P in sediment/soil (pore) water (kg P)
TP_SORB[cellLoc] == mass of P sorbed to soil (kg P)

Parameter definitions:
     global parameters in GlobalParms.xls Excel sheet (text export=GlobalParms)
     habitat-specific parameters in HabParms.fmp FileMakerPro database (text export=HabParms)
}

{ calc concentrations after horiz fluxes }
TP_SFWT_CONC   =
          if ( SFWT_VOL > 0.0 )
          then ( TP_SF_WT/SFWT_VOL )
          else ( 0.0) { used in P fluxes for mass balance }
          {oct99 using regression for predicting PO4 from TP }
PO4Pconc =  Max( TP_SFWT_CONC*PO4toTP + 0.001*PO4toTPint,0.0);  { g/l  (note that intercept may be <0)}
          { after spatial (horiz) fluxes, recalc the active zone P mass based on old active/total ratio }
TP_SED_CONC  =
          if (HYD_SED_WAT_VOL>0.0)
          then (TP_SED_WT / HYD_SED_WAT_VOL)
          else (0.0)
          { this is the active zone, where uptake, sorption, and mineralization take place }
TP_SED_WT_AZ = TP_SED_CONC * TP_Act_to_Tot * HYD_DOM_ACTWAT_VOL;
TP_SEDWT_CONCACT  =
          if (HYD_DOM_ACTWAT_PRES > 0.0)
          then ( TP_SED_WT_AZ/HYD_DOM_ACTWAT_VOL )
          else ( TP_SED_CONC)

{ inputs to surf  P (kg P) }
TP_FR_RAIN  = SF_WT_FROM_RAIN*CELL_SIZE*TP_IN_RAIN*0.001;
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

groundwater upflow potential

groundwater upflow actual

Sorption/desorption (input/output)

```
{ calc various loss and/or vertical exchange potentials (kg P) }

TP_UPFLOW_POT  = SAT_SURF_UP_FL*CELL_SIZE*TP_SEDWT_CONCACT
        + Max((TP_SEDWT_CONCACT-PO4Pconc)
        *TP_DIFFCOEF*8.64/TP_DIFFDEPTH*CELL_SIZE,0.0); { was multiplied by depth dec99 }
TP_UPFLOW   =
        if ( (TP_UPFLOW_POT)*DT>TP_SED_WT_AZ )
        then ( (TP_SED_WT_AZ)/DT )
        else ( TP_UPFLOW_POT)
TP_SEDWT_PR1  = TP_UPFLOW;

TP_K  = Max(TP_K_SLOPE*TP_SORBCONC+TP_K_INTER,0.0);
{fix rate oct99}
TP_SORB_POT  =
        ( HYD_DOM_ACTWAT_PRES>0.0 ) ?
        (  0.001
        *(TP_K*(Max(TP_SEDWT_CONCACT,0.0)^0.8 )
        *0.001*(DEPOS_ORG_MAT*CELL_SIZE+DIM)-TP_SORB ) ) :
        ( 0.0);
if (TP_SORB_POT>0.0) {
TP_SORBTION  =
        ( (TP_SORB_POT+TP_SEDWT_PR1)*DT>TP_SED_WT_AZ ) ?
        ( (TP_SED_WT_AZ-TP_SEDWT_PR1*DT)/DT ) :
        ( TP_SORB_POT); }
else { { neg sorption, loss from sorb variable}
TP_SORBTION   =
        if ( (-TP_SORB_POT)*DT>TP_SORB )
        then ( (-TP_SORB)/DT )
        else ( TP_SORB_POT)
}
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

downflow to groundwater potential

downflow to groundwater actual

Update state variables

TP in pore (sediment) water

TP in porewater of biologically active zone

TP in surface water

TP sorbed to soil

Update concentrations

```
TP_DNFLOW_POT  = (SF_WT_INFILTRATION+SF_WT_TO_SAT_DOWNFLOW)
        *CELL_SIZE*TP_SFWT_CONC  { oct99 was the state var, NOT conc!!!, dec99 now replaced with TPconc
        }
        + Max((PO4Pconc-TP_SEDWT_CONCACT)
        *TP_DIFFCOEF*8.64/TP_DIFFDEPTH*CELL_SIZE,0.0) ;{ was multiplied by depth dec99 }
TP_DNFLOW   =
        if ( ( TP_DNFLOW_POT)*DT > TP_SF_WT )
        then ( ( TP_SF_WT)/DT )
        else ( TP_DNFLOW_POT)
{ calc P in sed water state variables (kg P) }

TP_SED_WT (t) = TP_SED_WT (t - dt) + (TP_DNFLOW + TP_SED_MINER - TP_UPFLOW - TP_SORBTION -
        TP_SEDWT_UPTAKE) * dt;
{ this is the active zone, where uptake, sorption, and mineralization take place }
TP_SED_WT_AZ = TP_SED_CONC * TP_Act_to_Tot * HYD_DOM_ACTWAT_VOL;

{ calc P in surface water state variable (kg P) }

TP_SF_WT (t) = TP_SF_WT (t - dt)  +  (TP_UPFLOW + TP_FR_RAIN + TP_SFWT_MINER -
        TP_SFWT_UPTAK - TP_DNFLOW - TP_settl) * dt;

{ calc P sorbed-to-soil state variable (kg P) }
TP_SORB (t) = TP_SORB (t - dt)+(TP_SORBTION) * dt;

TP_SED_CONC  =
        if (HYD_SED_WAT_VOL>0.0)
        then (TP_SED_WT / HYD_SED_WAT_VOL)
        else (0.0) { g/L }
TP_SEDWT_CONCACT  =
        if ( HYD_DOM_ACTWAT_PRES > 0.0)
        then ( TP_SED_WT_AZ/HYD_DOM_ACTWAT_VOL )
        else (TP_SED_CONC) { g/L }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

```
TP_SORBCONC  =
        if ((DEPOS_ORG_MAT*CELL_SIZE + DIM)>0.0)
        then (  TP_SORB*1000.0 / (DEPOS_ORG_MAT*CELL_SIZE + DIM) )
        else (0.0) { gP/kgsoil }



TP_SFWT_CONC   =
        if ( SFWT_VOL > 0.0 )
        then ( TP_SF_WT/SFWT_VOL )
        else ( 0.0) { g/L used in P fluxes for mass balance }
```

Particulate settling of TP

```
{ the following is an empirical method to calculate settling of particulate P out of the water column
into the sediments.  It uses the fixed ratio of PO4 to TP, but allows for a decreasing proportion of
TP to be in (large) particulate form as TP concentration drops below a chosen threshold - the sum of the TP is
        considered to be dissolved plus large particulate plus small particulate (that cannot settle out) }
        { mass (kg) of particulate fraction of TP, available for settling to sediments }
        { using regression for predicting PO4 from TP }
```

"dissolved", or non-particulate P

```
PO4Pconc =  Max(TP_SFWT_CONC_MG*PO4toTP + PO4toTPint,0.0);  { mg/l (note that intercept may be <0)}
nonPO4Pconc = Max(TP_SFWT_CONC_MG-PO4Pconc,0.0); { non-PO4 conc, mg/l }
```

particulate P estimate

```
TPpartic = nonPO4Pconc * (1.0-exp(-nonPO4Pconc/TPpart_thresh)) *0.001 * SFWT_VOL ; { kg particulate P }
```

particulate TP settling specific rate

```
        { settling rate, 1/d }
TPsettlRat  =
        if ( SURFACE_WAT > DetentZ )
        then (settlVel/SURFACE_WAT)
        else 0.0
```

particulate TP settling potential

```
        { potential settling of particulate TP (kg/d) }
TP_settl_pot = TPsettlRat * TPpartic;
```

particulate TP settling

```
TP_settl   =
        if ( ( TP_settl_pot)*DT > TPpartic )
        then ( (TPpartic)/DT )
```

a code "un-generator" parsed these eqn's from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.

Update state variable

TP in surface water

update concentration

```
        else ( TP_settl_pot)
{ calc P in surface water state variable (kg P) }
TP_SF_WT (t) = TP_SF_WT (t - dt)  +  (TP_UPFLOW + TP_FR_RAIN + TP_SFWT_MINER - TP_SFWT_UPTAK
        - TP_DNFLOW - TP_settl) * dt;

{ various book-keeping calcs used in other modules }
{ conc surf and sed wat = kgP/m3=gP/L, another var calc for mg/L }
TP_SFWT_CONC   =
        if ( SFWT_VOL > 0.0 )
        then ( TP_SF_WT/SFWT_VOL )
        else ( 0.0) { used in P fluxes for mass balance }
```

a code “un-generator” parsed these eqn’s from UnitMod.c; Ex.: converted function calls, comments {in double-braces}, removed C syntax, array notation, etc.