```
#############################################################################
#
#  Central model
#
#############################################################################
#############################################################################
#     UNITS
#
#     load              MW
#     dispatch          MW
#     Flow              cumec
#     cost        $/MWh
#     conversion factor MW/cumec
#
#############################################################################


#-------------------------------------------------------------------
#                       Time
#-------------------------------------------------------------------
param N_D default 7;    # number of days.
param hours_in_tp >=0, default 0.5;       # hours in a trading period
param seconds_in_hour >=0, default 3600;        # seconds in an hour

param TP{1..N_D} default 48; # number of trading periods in each day of the week
param TP_2 default 48;

param w_dy{1..N_D}; # week day
param wk{1..N_D}; # week
param m_dy{1..N_D}; # month day
param mnth{1..N_D}; # month
param yr{1..N_D}; # year

param T;          # number of trading periods in one week
param T_2 := if wk[1]=51 then 48*8 else 48*7;   # TPs in the second week
set TIME := 1..(T+T_2); # two weeks
param dy{t in TIME}:=  floor((t-1)/TP_2) + 1  ; # week day for each TP

#-------------------------------------------------------------------
#                       Island
#-------------------------------------------------------------------
set ISLAND ;                                # ISLAND

#-------------------------------------------------------------------
#                       Demand
#-------------------------------------------------------------------
set NODES ordered;       # nodes
set NODES_ISLAND within {NODES,ISLAND} ;  # position of nodes in island

param loads_posneg {NODES, TIME}, default 0; # negative values for embedded generation
param loads {n in NODES, t in TIME} := max(0,loads_posneg[n,t]) ; #demand to be non-negative

# Demand scales, lower bounds, upper bounds and demands not scaled
param demand_scale_node{NODES,TIME} ;
param demand_scale_lb{NODES}; param demand_scale_ub{NODES}; param demand_scale_lb_mw{NODES};

# Lost load
set LL_SECTOR ;                             # sector
set LL_SEGMENT ;                            # segment
param LL_cost{LL_SECTOR,LL_SEGMENT};            # cost in $/MWh
param LL_sec_dist{LL_SECTOR,ISLAND};            # sector distribution in each island
param LL_seg_bound{LL_SEGMENT};                 # proportion of segment

var LostLoad {NODES,TIME,LL_SECTOR,LL_SEGMENT} >=0, default 0;   # LL in MW

#-------------------------------------------------------------------
#                       Generation
#-------------------------------------------------------------------
set GENERATORS;
set NODES_GEN within {NODES, GENERATORS};  # generators supplying NODES

set OFFERING_GENS within GENERATORS;
set FIXED_GENS within GENERATORS;
set EMBEDDED_GENS within GENERATORS:=GENERATORS diff OFFERING_GENS diff FIXED_GENS;
```

```
param gcap {OFFERING_GENS} default 99999; # capacity of generator
param MWloss{OFFERING_GENS,TIME}, default 0; # capacity derating due to maintenance outage

param fixedDispatch{GENERATORS,TIME} default 0; # historical dispatch

# Hydro
set HYDROS within GENERATORS diff EMBEDDED_GENS;  # hydro generator
set AVE_HYDROS within HYDROS; # fixed hydro generators

param conv_fac {HYDROS diff AVE_HYDROS} >=0; # conversion factor, MW/cumec
param conv_fac_2 {HYDROS diff AVE_HYDROS} >=0; # for second week

set SCALING := 1..19 ;  # scaling for conv_fac for stations. Generators in the same station
has the same scaling
set HYDROS_SCALING within {HYDROS inter OFFERING_GENS,SCALING};
param scaling_fac_r{SCALING, t in TIME}, default 1;
param scaling_fac {s in SCALING, t in TIME} := 1/scaling_fac_r[s,t];

# Thermal
set THERMALS within GENERATORS diff EMBEDDED_GENS;

set OFFERING_THERMALS;
set FUEL ordered;
set THERMALS_FUEL within {OFFERING_THERMALS,FUEL};
param therm_heat_rate {OFFERING_THERMALS} >= 0; # GJ/MWh
param fuel_cost {FUEL} >= 0;  # cost to run thermal plant in $/GJ
param fuel_cost_2 {FUEL} >= 0;      # for second week

param therm_start_year{THERMALS} , default 0;
param therm_start_week{THERMALS} , default 0;
param therm_end_year{THERMALS} , default 3000;
param therm_end_week{THERMALS} , default 100;
set UNAVAIL_THERMALS within THERMALS := {j in THERMALS :
    yr[1] < therm_start_year[j]
or (yr[1] = therm_start_year[j] and wk[1] < therm_start_week[j])
or (yr[1] = therm_end_year[j] and wk[1] > therm_end_week[j])
or  yr[1] > therm_end_year[j]
}
;

var Dispatch {j in GENERATORS, TIME} >=0;  # rate of dispatch at each time step in MW
var Generation {NODES, TIME} >=0;               # total power injection into a bus (MW)

#---------------------------------------------------------------------
#                       Transmission
#---------------------------------------------------------------------
set LINK within {NODES, NODES}; # tranmission lines in network
set HVDC within LINK;           # HVDC lines

param X {(i,j) in LINK} ; # Line reactance (MW)

param U {(i,j) in LINK} ; # Line capacity(MW)

param npieces {LINK}; # number of segments in the piecewise linear function for each link
param breakpoint {(i,j) in LINK,1..npieces[i,j]-1};   # breakpoints for each of the loss
functions
param slope {(i,j) in LINK,1..npieces[i,j]};          # slopes for each of the loss function
segments
param HVDCBreakPoint {(i,j) in HVDC, 1..npieces[i,j]-1};    # breakpoints for HVDC
param HVDCSlope {(i,j) in HVDC, 1..npieces[i,j]};     # slopes for HVDC
# one less breakpoint than slope, because of assuming that zero is always a breakpoint

param U_eff {(i,j) in LINK, t in TIME} >=0, default U[i,j]; # U - derating
param HVDC_caploss{HVDC,TIME} >=0, default 0;

var P {(i,j) in LINK, t in TIME} >=0, <=U_eff[i,j,t], default 0 ;     # power flow
var FS {(i,j) in LINK, t in TIME} >=0, <=U_eff[i,j,t], default 0 ;    # sent power
var FR {(i,j) in LINK, t in TIME} >=0, <=U_eff[i,j,t], default 0 ;    # received power
var W {(i,j) in LINK, t in TIME} >=0, default 0;                  # loss
var Theta {NODES, TIME} >=0, default 0;                    # voltage angle for loop flows

#---------------------------------------------------------------------
#                       Reservoir and junction
#---------------------------------------------------------------------
set RES ordered;
```

```
set RES_LAKE within RES ;
set RES_LAKE_LARGE within RES_LAKE ;
set RES_LAKE_SMALL := RES_LAKE diff RES_LAKE_LARGE ;
set RES_ART within RES;

set JUNCTIONS;
set TRIB_JUNCTIONS within JUNCTIONS;

param res_cap{RES};
param specific_energy{RES} ;
param specific_energy_max := max{r in RES_LAKE}specific_energy[r]

param inter_state {r in RES} >=0; # Inter stage states of reservoirs, final state of prev
stage, ini state of this stage
param his_fin_state{RES} >= 0;      # historical final state in the first week

var State{r in RES,TIME} >=0, <=res_cap[r] ; # State of reservoir at the end of each period
var fin_state {r in RES_LAKE}>=0;   # final state in the first week

#---------------------------------------------------------------------
#                       Inflow
#---------------------------------------------------------------------
param inflow_posneg{r in RES, t in TIME} default 0 ;
param inflow_trib_posneg{jn in TRIB_JUNCTIONS, t in TIME} default 0 ;
param inflow{r in RES, t in TIME} := max(0,inflow_posneg[r,t]) ;
param inflow_trib{jn in TRIB_JUNCTIONS, t in TIME} := max(0,inflow_trib_posneg[jn,t]) ;

param inflow_scale{RES_LAKE union TRIB_JUNCTIONS, t in TIME},  default 1;

#---------------------------------------------------------------------
#                       Flow
#---------------------------------------------------------------------
set HYDRO_NODES := RES union {'SEA'} union {HYDROS diff AVE_HYDROS} union JUNCTIONS;
set ARCS within {HYDRO_NODES,HYDRO_NODES};

# Bound
param lb {ARCS} default 0 >=0;
param ub {(i,j) in ARCS} default 1000000 >= lb[i,j];
param ub_2 {(i,j) in ARCS} default 1000000 >= lb[i,j];
param night_st>=0;
param night_fn>=0;
param lb_night{(i,j) in ARCS}>=0;

set ARCS_LB := {(i,j) in ARCS : lb[i,j]>0 or lb_night[i,j]>0} ;
set ARCS_UB := {(i,j) in ARCS : ub[i,j]>0 and ub[i,j]<1000000} ;

# Delay in flow
param delay {(i,j) in ARCS}, default 0;
param FlowDelayed{(i,j) in ARCS,TIME} >=0, default 0 ;

# Penalty cost
param WaterInfeaCost >= 0, default 15000;       # Penalty cost on water balance violation,
$/MWh
param WaterInfeaCostB >= 0, default 500;  # Penalty cost on LB and UB violation $/MWh

# Spill
set SPILL_ARCS within ARCS default {};
param SpillRate{SPILL_ARCS} >= 0;
param SpillRate_2{SPILL_ARCS} >= 0;

param SpillStation {SPILL_ARCS} , default 1 ;

var FlowLeave{(i,j) in ARCS,TIME} >=0 ;
var FlowArrive{(i,j) in ARCS,TIME} >=0 ;

var CorrPos{RES,TIME} >=0, default 0 ;
var CorrNeg{RES,TIME} >=0, default 0 ;

var CorrLB{ARCS_LB,TIME} >=0, default 0 ; # Penalty variable for flow LB violation
var CorrUB{ARCS_UB,TIME} >=0, default 0 ; # Penalty variable for flow UB violation

var Spill{SPILL_ARCS,TIME};
var SpillCumec{SPILL_ARCS,TIME};
var SpillCubic ;
```

```
var HydroCubic ;

#---------------------------------------------------------------------
#                       Cut
#---------------------------------------------------------------------
param stage_total >=0 , default 52;       # stages
param nCut {1..stage_total,1..1} >= 0 ;           # number of cuts in each stage
param alpha {t in 1..stage_total,j in 1..1,1..nCut[t,j]}, default 0;          # cut
coefficients
param beta  {RES,t in 1..stage_total,j in 1..1,1..nCut[t,j]}, default 0;
param stage ;           # current stage in 1...stage_total, to be defined in run file

var gamma    ;          # future cost

#---------------------------------------------------------------------
#                       Cost
#---------------------------------------------------------------------
var thermal_fuel_cost; var lostload_cost; var WaterInfeas_cost; var flowb_cost;
var spill_cost; var nationalload_cost;
var thermal_fuel_cost_2; var lostload_cost_2; var WaterInfeas_cost_2; var flowb_cost_2; var
spill_cost_2;
var future_cost;

#---------------------------------------------------------------------
#                       Constraints for energy
#---------------------------------------------------------------------
# Generation
subject to EMB {j in EMBEDDED_GENS, t in TIME: t<=T+TP_2} :
           Dispatch[j,t] = 0
;

subject to FIXED { j in FIXED_GENS, t in TIME: t<=T+TP_2}:
            Dispatch[j,t] = max(0,fixedDispatch[j,t])
;

subject to OFFERING2{j in OFFERING_GENS, t in TIME: t<=T+TP_2}:
           Dispatch[j,t]<=gcap[j] - MWloss[j,t]
;

subject to Dispatch_Constraint_Thermal
      {j in UNAVAIL_THERMALS, t in TIME: t<=T+TP_2} :
           Dispatch[j,t] <= 0
;

subject to Dispatch_Defn_Hydro {(i,s) in HYDROS_SCALING, t in TIME: t<=T+TP_2} :
      Dispatch[i,t] = sum {(i,j) in ARCS} FlowLeave[i,j,t] * conv_fac[i] / scaling_fac[s,t]
;

subject to SpillDef {(i,j) in SPILL_ARCS, t in TIME: t<=T+TP_2}:
          Spill[i,j,t] = FlowLeave[i,j,t] * SpillRate[i,j]
;

# Meet demand
subject to Supply{i in NODES, t in TIME: t<=T+TP_2}:
           Generation[i,t] = sum {(i,j) in NODES_GEN} Dispatch[j,t]
;

subject to Users {i in NODES, t in TIME: t<=T+TP_2}:
            Generation[i,t] + sum {(i,j) in LINK} (FR[j,i,t] - FS[i,j,t])
            >=
            ((loads[i,t]- demand_scale_lb_mw[i]) * demand_scale_node[i,t] + demand_scale_lb_mw
[i])
            - sum{sec in LL_SECTOR,seg in LL_SEGMENT}LostLoad[i,t,sec,seg]
;

subject to LostLoad_bound {(i,isl) in NODES_ISLAND,sec in LL_SECTOR, seg in LL_SEGMENT, t in
TIME: t<=T+TP_2}:
            LostLoad[i,t,sec,seg] <=
                  ((loads[i,t]- demand_scale_lb_mw[i]) * demand_scale_node[i,t] +
demand_scale_lb_mw[i])
                        * LL_sec_dist[sec,isl]*LL_seg_bound[seg]
;

# Transmission
subject to VoltageAngle {(i,j) in LINK diff HVDC, t in TIME: t<=T+TP_2}:
```

```
          P[i,j,t] - P[j,i,t] = (Theta [j,t] - Theta [i,t])/X[i,j]
;

subject to LossDefinition {(i,j) in LINK diff HVDC, t in TIME: t<=T+TP_2}:
          W[i,j,t] >= <<{p in 1..npieces[i,j]-1} breakpoint[i,j,p]; {p in 1..npieces[i,j]}
slope[i,j,p]>> P[i,j,t]
;

subject to LossDefinitionHVDC {(i,j) in HVDC, t in TIME: t<=T+TP_2}:
          W[i,j,t] >= <<{p in 1..npieces[i,j]-1} HVDCBreakPoint[i,j,p]; {p in 1..npieces
[i,j]} HVDCSlope[i,j,p]>> P[i,j,t]
;

subject to ReceivedPower {(i,j) in LINK, t in TIME: t<=T+TP_2}:
          FR[i,j,t] = P[i,j,t] -  0.5 * W[i,j,t]
;

subject to SentPower {(i,j) in LINK, t in TIME: t<=T+TP_2}:
          FS[i,j,t] = P[i,j,t] +  0.5 * W[i,j,t]
;

subject to HVDC_outage {(i,j) in HVDC, t in TIME: t<=T+TP_2}:
           FS[i,j,t] <= U[i,j] - HVDC_caploss[i,j,t]
;

#-----------------------------------------------------------------------
#                       Constraints for water
#-----------------------------------------------------------------------
subject to StateEqn {r in RES, t in TIME: t<=T+TP_2} :
    State[r,t] =
        ( if t=1 then inter_state[r] else 0 )
      + ( if t>1 then State[r,t-1] else 0 )
      + seconds_in_hour*hours_in_tp *
      (
            (if r in RES_LAKE then inflow[r,t]*inflow_scale[r,t])
            + sum {(i,r) in ARCS} FlowArrive[i,r,t]
            - sum {(r,i) in ARCS} FlowLeave[r,i,t]
            - CorrPos[r,t] + CorrNeg[r,t]
      )
;

subject to StateEqnFin {r in RES_LAKE} :
      fin_state[r] = State[r,T]
;

subject to Def_FlowArrive{(i,j) in ARCS,t in TIME: t<=T+TP_2}:
      FlowArrive[i,j,t]=(if t<=delay[i,j] then FlowDelayed[i,j,t] else FlowLeave[i,j,t-delay
[i,j]])
;

subject to Balance {j in HYDRO_NODES diff (RES union{'SEA'}), t in TIME: t<=T+TP_2} :
    sum {(i,j) in ARCS} FlowArrive[i,j,t] + (if j in TRIB_JUNCTIONS then inflow_trib[j,t]
*inflow_scale[j,t] else 0)
      = sum {(j,i) in ARCS} FlowLeave[j,i,t]
;

subject to Flow_lb {(i,j) in ARCS, t in TIME: t<=T+TP_2} :
    FlowLeave[i,j,t] + (if (i,j) in ARCS_LB then CorrLB[i,j,t] else 0) >=
      if (i,j) in ARCS_LB then
      max(lb[i,j], (if t-(dy[t]-1)*48<=night_fn or t-(dy[t]-1)*48>=night_st then lb_night[i,j]
else 0))
      else 0
;

subject to Flow_ub {(i,j) in ARCS_UB, t in TIME: t<=T+TP_2} :
    FlowLeave[i,j,t] - CorrUB[i,j,t]  <=  ub[i,j]
;


subject to SpillCumec_cal {(i,j) in SPILL_ARCS,t in TIME:t<=T} :
      SpillCumec[i,j,t] = SpillStation[i,j]*Spill[i,j,t]/SpillRate[i,j]
;

subject to SpillCubic_cal :
      SpillCubic = sum{(i,j) in SPILL_ARCS,t in TIME:t<=T} SpillCumec[i,j,t]
```

```
*seconds_in_hour*hours_in_tp
;

subject to HydroCubic_cal :
      HydroCubic = sum{(i,s) in HYDROS_SCALING,t in TIME:t<=T} sum {(i,j) in ARCS} FlowLeave
[i,j,t]*seconds_in_hour*hours_in_tp
;

#-----------------------------------------------------------------------
#                       Constraints for cost
#-----------------------------------------------------------------------
# Objective
minimize total_cost:
      thermal_fuel_cost + lostload_cost + WaterInfeas_cost + flowb_cost + spill_cost +
      thermal_fuel_cost_2 + lostload_cost_2 + WaterInfeas_cost_2 + flowb_cost_2 + spill_cost_2
      + future_cost
;

# first week cost
subject to thermal_fuel_cost_cal :
      thermal_fuel_cost = sum {(j,f) in THERMALS_FUEL} sum {t in TIME:t<=T} hours_in_tp *
Dispatch[j,t]
            * ( if yr[1]=2007 and wk[1]<=21 and j='GEN.Thermal.Huntly.gas' then 0 else
therm_heat_rate[j]*fuel_cost[f] )
;

subject to lostload_cost_cal :
      lostload_cost = sum{i in NODES, t in TIME:t<=T} hours_in_tp *
      sum{sec in LL_SECTOR,seg in LL_SEGMENT}LostLoad[i,t,sec,seg]*LL_cost[sec,seg]
;

subject to waterinfeas_cost_cal  :
      WaterInfeas_cost = WaterInfeaCost * sum{r in RES, t in TIME:t<=T} (  specific_energy_max
* hours_in_tp * (CorrPos[r,t]+CorrNeg[r,t])  )
;

subject to flowlb_cost_cal :
      flowb_cost = WaterInfeaCostB * sum{(i,j) in ARCS_LB,t in TIME:t<=T} (
specific_energy_max * hours_in_tp * CorrLB[i,j,t] )
                  + WaterInfeaCostB * sum{(i,j) in ARCS_UB,t in TIME:t<=T} (
specific_energy_max * hours_in_tp * CorrUB[i,j,t] )
;

subject to nationalload_cost_cal :
      nationalload_cost = sum {i in NODES, t in TIME:t<=T} hours_in_tp *
                              ((loads[i,t]- demand_scale_lb_mw[i]) * demand_scale_node[i,t] +
demand_scale_lb_mw[i])
;

subject to spill_cost_cal :
      spill_cost = sum{(i,j) in SPILL_ARCS, t in TIME:t<=T} Spill[i,j,t] * hours_in_tp
;

# second week cost
subject to thermal_fuel_cost_cal_2 :
      thermal_fuel_cost_2 = sum {(j,f) in THERMALS_FUEL} sum {t in TIME:t>T and t<=T+TP_2}
hours_in_tp * Dispatch[j,t]
            * ( if yr[1]=2007 and wk[1]<=20 and j='GEN.Thermal.Huntly.gas' then 0 else
therm_heat_rate[j]*fuel_cost_2[f] )
;

subject to lostload_cost_cal_2 :
      lostload_cost_2 = sum{i in NODES, t in TIME:t>T and t<=T+TP_2} hours_in_tp *
      sum{sec in LL_SECTOR,seg in LL_SEGMENT}LostLoad[i,t,sec,seg]*LL_cost[sec,seg]
;

subject to waterinfeas_cost_cal_2  :
      WaterInfeas_cost_2 = WaterInfeaCost * sum{r in RES, t in TIME:t>T and t<=T+TP_2} (
specific_energy_max * hours_in_tp * (CorrPos[r,t]+CorrNeg[r,t])  )
;

subject to flowlb_cost_cal_2 :
      flowb_cost_2 = WaterInfeaCostB * sum{(i,j) in ARCS_LB,t in TIME:t>T and t<=T+TP_2}
                  ( specific_energy_max * hours_in_tp * CorrLB[i,j,t] )
                  + WaterInfeaCostB * sum{(i,j) in ARCS_UB,t in TIME:t>T and t<=T+TP_2}
```

```
                  ( specific_energy_max * hours_in_tp * CorrUB[i,j,t] )
;

subject to spill_cost_cal_2 :
      spill_cost_2 = sum{(i,j) in SPILL_ARCS, t in TIME:t>T and t<=T+TP_2} Spill[i,j,t] *
hours_in_tp
;

subject to future_cost_cal :
      future_cost = gamma
;

# Cut definition, 1 is the Markov state if Markov model is used
subject to Cut_Defn {k in 1..nCut[stage,1]}:
      gamma  >= alpha[stage,1,k] + sum {r in RES_LAKE_LARGE}(beta[r,stage,1,k] * State[r,T+TP_
2] );


###############################################################################
#
#                               End
#
###############################################################################
```