```
Subroutine Type5995
! TRNSYS Type 5995: Lumped capacitance model - Modified by Daniel Marques to turn into a model of 1-D heat conduction in a PCM slab
!
! This component calculates the transient behavior of a PCM slab, modeled with the finit difference method employing heat capacity method to deal
! with phase change problem. It considers the PCM slab is like a wall. We employ the implicit scheme to solve 1-D heat conduction equation.
!
! Copyright © 2024 University of Aveiro, Portugal, Daniel Marques. All rights reserved.

!export this subroutine for its use in external DLLs.
!DEC$ATTRIBUTES DLLEXPORT :: TYPE5995

!
 !Use Statements
 Use TrnsysConstants
 Use TrnsysFunctions
!

!Variable Declarations
Implicit None !force explicit declaration of local variables

!
!Define a derived type to hold multiple parameters - this is done to set the type of data each node will store
!real(8) is the same as Douple Precision to declare variables as floating-point numbers
type :: NodeData
    Double Precision :: temperature_init
    Double Precision :: specific_heat
    Double Precision :: thermal_conductivity
    Double Precision :: liq_fract
    Double Precision :: temperature_final
end type NodeData

!Define a derived type to hold multiple parameters - this is done to set the type of data each boundary condition needs
type :: BCsData
    Double Precision :: environment_temperature
    Double Precision :: heat_transfer_coeff
end type BCsData

!Declare an array of NodeData with n elements - this is used to have an array of multiple NodeData structures
```

```
39  type(NodeData), allocatable :: NodesArray(:) !type(NodeData) :: NodesArray(11)
40  !Declare an allocatable array of BCsData with a size determined at runtime
41  type(BCsData) :: BCsArray(2) !previously I had: type(BCsData), allocatable ::
      BCsArray(:)
42
43  !------------------------------------------------------------------------------
      ----------------------------------------
44  ! Declare all other values needed for the code
45
46  Integer :: ii,ee,nn
47  Double Precision Time,Timestep
48  Integer CurrentUnit,CurrentType,mode,nBCs,ninp,jj,AA,BB
49  Double Precision
      density,thickness,mass_PCM,volume,Cp_PCM,K_PCM,area,T_0,Q_PCM,T_inf,E_in,T_i,T_
      f,T_bar, &
50
    Cp_solid,Cp_liquid,K_solid,K_liquid,T_solidus,T_liquidus,Fusion_Enthalpy,Liquid_F
    raction,E_out,U_inf,deltaX,E_out_abs,volume_i
51  Logical found_end
52   Double Precision, allocatable :: aa_TA(:),bb_TA(:),cc_TA(:),dd_TA(:),xx_TA(:) !
       code to declare the arrays needed for the Thomas Algorithm for FDM - implicit
       scheme
53   Double Precision, allocatable :: c_prime(:), d_prime(:) !also needed fot the
       Thomas Algorithm
54
55  Double Precision :: temp !variable used also for Thomas Algorithm
56
57  !------------------------------------------------------------------------------
      ----------------------------------------
58  !Get the Global Trnsys Simulation Variables
59   Time=getSimulationTime()
60   Timestep=getSimulationTimeStep()
61   CurrentUnit = getCurrentUnit()
62   CurrentType = getCurrentType()
63
64  !------------------------------------------------------------------------------
      ----------------------------------------
65
66  !------------------------------------------------------------------------------
      ----------------------------------------
67  !Set the Version Number for This Type
68   If(getIsVersionSigningTime()) Then
69       Call SetTypeVersion(17)
70       Return
71   Endif
72  !------------------------------------------------------------------------------
      ----------------------------------------
73
74  !------------------------------------------------------------------------------
      ----------------------------------------
75  !Do All of the Last Call Manipulations Here
76   If(getIsLastCallofSimulation()) Then
```

```
 77
 78  !------------------------------------------------------------------------------
       ----------------------------------------
 79  ! Deallocate BCsArray to ensure the code is not consuming memory resources in the
       computer anymore - this is just a good practice
 80   ! deallocate(BCsArray)
 81  ! Deallocating the Thomas Algorithms arrays to ensure we stop consuming memory
       resources
 82   deallocate(aa_TA,bb_TA,cc_TA,dd_TA,xx_TA,c_prime,d_prime)
 83   ! Deallocate NodesArray
 84   deallocate(NodesArray)
 85
 86      Return
 87   Endif
 88  !------------------------------------------------------------------------------
       ----------------------------------------
 89
 90  !------------------------------------------------------------------------------
       ----------------------------------------
 91  !Perform Any "End of Timestep" Manipulations That May Be Required
 92   If(getIsEndOfTimestep()) Then
 93      Do ii=1,nn
 94          Call SetStaticArrayValue(ii,getStaticArrayValue(ii))
 95      End Do
 96
 97      !I have decided to comment and avoid this Report - this was from the older
         version of Type59
 98      !If (getIsIncludedInSSR()) Then
 99         !Call updateReportMinMax(1,getOutputValue(1))
100         !Call updateReportIntegral(1,getInputValue(2))
101         !Call updateReportIntegral(2,getOutputValue(2))
102      !EndIf
103      Return
104   Endif
105  !------------------------------------------------------------------------------
       ----------------------------------------
106
107  !------------------------------------------------------------------------------
       ----------------------------------------
108  !Do All of the "Very First Call of the Simulation Manipulations" Here
109   If (getIsFirstCallofSimulation()) Then
110      ninp = getNumberOfInputs() !set the number of INPUTS to the number found in
         the deck
111      nn = getParameterValue(14) ! Retrieve nn from the proforma parameters
112
113  !  Tell the TRNSYS Engine How This Type Works
114      Call SetNumberofParameters(14)
115      Call SetNumberofInputs(ninp)
116      Call SetNumberofDerivatives(0)
117      Call SetNumberofOutputs(5+nn)
118      Call SetIterationMode(1)
119      Call SetNumberStoredVariables(nn,0)
```

```
120
121  ! Set the Correct Input and Output Variable Types
122      Call SetInputUnits(1,'PW1')
123      Do jj=3,ninp,2
124      Call SetInputUnits(jj-1,'TE1')
125      Call SetInputUnits(jj,'HT1')
126      End Do
127      Call SetOutputUnits(1,'TE1')
128      Call SetOutputUnits(2,'PW1')
129      Call SetOutputUnits(3,'DM1')
130      Call SetOutputUnits(4,'AR1')
131      Call SetOutputUnits(5,'EN1')
132      Do jj=6,nn+5
133      Call SetOutputUnits(jj,'TE1')
134      End Do
135
136   !  Set up this Type's entry in the SSR - also decided to comment and avoid this
       part of the code - it was from the original type59
137      !If (getIsIncludedInSSR()) Then
138         !Call setNumberOfReportVariables(2,1,4,0) !(nInt,nMinMax,nValues,nText)
139      !EndIf
140
141      Return
142   EndIf
143  !------------------------------------------------------------------------------
       ---------------------------------------
144
145  !------------------------------------------------------------------------------
       ---------------------------------------
146  !Do All of the "Start Time" Manipulations Here - There Are No Iterations at the
       Intial Time
147   If (getIsStartTime()) Then
148
149  !  Read in the Values of the Parameters from the Input File
150      mode = jfix(getParameterValue(1) + 0.1)
151      density = getParameterValue(2)
152      thickness = getParameterValue(3)
153      mass_PCM = getParameterValue(4)
154      Cp_solid = getParameterValue(5)
155      Cp_liquid = getParameterValue(6)
156      T_0 = getParameterValue(7)
157      K_solid = getParameterValue(8)
158      K_liquid = getParameterValue(9)
159      Fusion_Enthalpy = getParameterValue(10)
160      T_solidus = getParameterValue(11)
161      T_liquidus = getParameterValue(12)
162      nBCs = getParameterValue(13)
163      nn = getParameterValue(14) !initializing this variable for the Thomas
           algoritm equal to the number of nodes
164
165  !  Check the Parameters for Problems
166      if (mode /= 1) Call FoundBadParameter(1,'Fatal','The mode must be 1.')
```

```
167     if (density <= 0.0 ) Call FoundBadParameter(2,'Fatal','The density must be greater than 0.')
168     if (thickness <= 0.0 )  Call FoundBadParameter(3,'Fatal','The thickness of the PCM slab must be greater than 0.')
169     if (mass_PCM <= 0.0 ) Call FoundBadParameter(4,'Fatal','The mass of PCM must be greater than 0.')
170     if (Cp_solid <= 0.0 ) Call FoundBadParameter(5,'Fatal','The specific heat solid must be greater than 0.')
171     if (Cp_liquid <= 0.0 ) Call FoundBadParameter(6,'Fatal','The specific heat liquid must be greater than 0.')
172     if (K_solid <= 0.0 ) Call FoundBadParameter(8,'Fatal','The thermal conductivity solid must be greater than 0.')
173     if (K_liquid <= 0.0 ) Call FoundBadParameter(9,'Fatal','The thermal conductivity liquid must be greater than 0.')
174     if (Fusion_Enthalpy <= 0.0 ) Call FoundBadParameter(10,'Fatal','The fusion latent heat must be greater than 0.')
175     if (nBCs <= 0.0 ) Call FoundBadParameter(13,'Fatal','The number of boundary conditions must be greater than 0.')
176
177 !  Set up the SSR array - Decided to comment the whole piece of code - it was originally from the type59
178     !If (getIsIncludedInSSR()) Then
179        !Call initReportValue(1,'Density',density,'kg/m3')
180        !Call initReportValue(2,'Volume',volume,'m3')
181        !Call initReportValue(3,'Specific Heat',Cp_PCM,'kJ/kg-K')
182        !Call initReportValue(4,'Surface Area',area,'m2')
183        !Call initReportMinMax(1,'Temperature','C')
184        !Call initReportIntegral(1,'Energy Input','kJ/hr','kJ')
185        !Call initReportIntegral(2,'Heat Transfer to Surroundings','kJ/hr','kJ')

186     !EndIf
187
188 !Allocating the vairable nn to the arrays (different classes of arrays)
189
190 allocate(NodesArray(nn)) !allocating the array of nodes with the size nn= number of nodes decided in the proforma
191 allocate(aa_TA(nn),bb_TA(nn),cc_TA(nn),dd_TA(nn),xx_TA(nn),c_prime(nn),d_prime(nn)) !allocating the arrays with the size nn = number of nodes
192
193 ! Initialize the array elements - since this is an initialization i placed it in the group of tasks to performe at getisstarttime()
194 Do ii=1,nn
195     NodesArray(ii)%temperature_init = 0.0
196     NodesArray(ii)%specific_heat = 0.0
197     NodesArray(ii)%thermal_conductivity = 0.0
198     NodesArray(ii)%liq_fract = 0.0
199     NodesArray(ii)%temperature_final = 0.0
200 End Do
201
202 ! Initialize all other arrays
203     aa_TA = 0.0
204     bb_TA = 0.0
```

```
205      cc_TA = 0.0
206      dd_TA = 0.0
207      xx_TA = 0.0
208      c_prime = 0.0
209      d_prime = 0.0
210      BCsArray%environment_temperature = 0.0
211      BCsArray%heat_transfer_coeff = 0.0
212
213   !  Set the Initial Values of the Outputs
214      Call SetOutputValue(1,0.d0)
215      Call SetOutputValue(2,0.d0)
216      Call SetOutputValue(3,0.d0)
217      Call SetOutputValue(4,0.d0)
218      Call SetOutputValue(5,0.d0)
219      Do jj=6,nn+5
220      Call SetOutputValue(jj,0.d0)
221      End Do
222
223   !  Set Initial Values of Storage Variables - I use this to assign to the staticarrayvalues the T_0 (initial temperature of the PCM)
224   Do ii=1,nn
225      Call SetStaticArrayValue(ii,T_0)
226   End Do
227
228   !  Initialize cumulative energy at the start of the simulation - this is useful for an ouput of the type
229      E_out = 0.0
230   !  Initialize temp variable at the start of the simulation - this is a variable that will be used in the Thomas Algorithm for solving the equations' system from the implicit scheme
231      temp = 0.0
232
233      Return
234    Endif
235   !------------------------------------------------------------------------------------------
236
237   !------------------------------------------------------------------------------------------
238   !ReRead the Parameters if Another Unit of This Type Has Been Called Last
239    If(getIsReReadParameters()) Then
240      mode = jfix(getParameterValue(1) + 0.1)
241      density = getParameterValue(2)
242      thickness = getParameterValue(3)
243      mass_PCM = getParameterValue(4)
244      Cp_solid = getParameterValue(5)
245      Cp_liquid = getParameterValue(6)
246      T_0 = getParameterValue(7)
247      K_solid = getParameterValue(8)
248      K_liquid = getParameterValue(9)
249      Fusion_Enthalpy = getParameterValue(10)
250      T_solidus = getParameterValue(11)
```

```
251      T_liquidus = getParameterValue(12)
252      nBCs = getParameterValue(13)
253    Endif
254  !------------------------------------------------------------------------------
       ----------------------------------------
255
256  !------------------------------------------------------------------------------
       ----------------------------------------
257  !Perform one time calculations
258  volume = mass_PCM/density
259  area = volume/thickness
260  deltaX = thickness/(nn-1)
261  volume_i = area*deltaX
262  !------------------------------------------------------------------------------
       ----------------------------------------
263
264  !------------------------------------------------------------------------------
       ----------------------------------------
265  !Retrieve the Stored Variables - doing this to make sure that the Array of Nodes
       is always being updated with the new Initial temperature values accurately
       stored in staticarrayvalues
266  Do AA=1,nn
267    NodesArray(AA)%temperature_init = getStaticArrayValue(AA)
268  End Do
269  !------------------------------------------------------------------------------
       ----------------------------------------
270
271  !------------------------------------------------------------------------------
       ----------------------------------------
272  !Allocate the rest of the initial values of the properties to the nodes, (Cp, K,
       fL), given the values it retrived of initial temperature in previous Do loop
273
274  Do AA=1,nn
275      If (NodesArray(AA)%temperature_init < T_solidus) Then
276          NodesArray(AA)%specific_heat = Cp_solid
277          NodesArray(AA)%thermal_conductivity = K_solid
278          NodesArray(AA)%liq_fract = 0.0
279      Else If (NodesArray(AA)%temperature_init > T_liquidus) Then
280          NodesArray(AA)%specific_heat = Cp_liquid
281          NodesArray(AA)%thermal_conductivity = K_liquid
282          NodesArray(AA)%liq_fract = 1.0
283      Else
284          NodesArray(AA)%specific_heat = (Cp_liquid+Cp_solid)/2.0 +
               Fusion_Enthalpy/(T_liquidus-T_solidus)
285          NodesArray(AA)%thermal_conductivity = (NodesArray(AA)%temperature_init-
               T_solidus)/(T_liquidus-T_solidus)*K_liquid + (1-(NodesArray(AA)%
               temperature_init-T_solidus)/(T_liquidus-T_solidus))*K_solid
286          NodesArray(AA)%liq_fract = (NodesArray(AA)%temperature_init-T_solidus)/
               (T_liquidus-T_solidus)
287      Endif
288  End Do
289  !------------------------------------------------------------------------------
```

```
      ---------------------------------------
290
291   !------------------------------------------------------------------------------ ↵
        ---------------------------------------
292   ! Allocate the BCsArray based on the value of nBCs
293   ! allocate(BCsArray(nBCs))
294   !------------------------------------------------------------------------------ ↵
        ---------------------------------------
295
296   !------------------------------------------------------------------------------ ↵
        ---------------------------------------
297   ! Get the Current Inputs to the Model
298    E_in = getInputValue(1) ! variable where a heat gain or removal (W) to apply to ↵
         all PCM domain is stored
299    Do ee = 1,nBCs
300       T_inf = getInputValue(ee*2)
301       U_inf = getInputValue(ee*2+1)
302       BCsArray(ee)%environment_temperature = T_inf
303       BCsArray(ee)%heat_transfer_coeff = U_inf
304    End Do
305   !------------------------------------------------------------------------------ ↵
        ---------------------------------------
306
307   !------------------------------------------------------------------------------ ↵
        ---------------------------------------
308   ! Performing calculations based on the FDM to calculate the final temperatures of ↵
         each node
309
310   ! Defining the coefficients associated to nodes i-1 (aa), i (bb), i+1 (cc) and    ↵
        right-hand side of the equations (dd) to initialize and populate the arrays    ↵
        with size nn
311
312   ! Surface node #1 - exterior boundary condition
313   aa_TA(1) = 0.0
314   bb_TA(1) = area * (BCsArray(1)%heat_transfer_coeff + (NodesArray(1)%             ↵
        thermal_conductivity + NodesArray(2)%thermal_conductivity)/(2*deltaX) +        ↵
        (density * NodesArray(1)%specific_heat * deltaX / (2*Timestep)))
315   cc_TA(1) = -(NodesArray(1)%thermal_conductivity + NodesArray(2)%                 ↵
        thermal_conductivity) * area / (2*deltaX)
316   dd_TA(1) = (density * NodesArray(1)%specific_heat * area * deltaX * NodesArray   ↵
        (1)%temperature_init) / (2*Timestep) + (BCsArray(1)%heat_transfer_coeff * area ↵
        * BCsArray(1)%environment_temperature) + E_in/(2*(nn-1))
317
318   ! Interior nodes #2 - #nn-1
319   Do ii=2,nn-1
320       aa_TA(ii) = -(NodesArray(ii)%thermal_conductivity + NodesArray(ii-1)%        ↵
          thermal_conductivity) * volume_i / (deltaX*deltaX*2)
321       bb_TA(ii) = volume_i * (density*NodesArray(ii)%specific_heat/Timestep +      ↵
          (2*NodesArray(ii)%thermal_conductivity+NodesArray(ii+1)%                     ↵
          thermal_conductivity+NodesArray(ii-1)%thermal_conductivity)/                 ↵
          (deltaX*deltaX*2))
322       cc_TA(ii) = -(NodesArray(ii)%thermal_conductivity + NodesArray(ii+1)%        ↵
```

```
          thermal_conductivity)*volume_i/(deltaX*deltaX*2)
323       dd_TA(ii) = density * NodesArray(ii)%specific_heat * volume_i * NodesArray
            (ii)%temperature_init / Timestep + E_in/(nn-1)
324   End Do
325
326   ! Surface node #nn - interior boundary condition
327   aa_TA(nn) = -(NodesArray(nn)%thermal_conductivity + NodesArray(nn-1)%
        thermal_conductivity) * area / (2*deltaX)
328   bb_TA(nn) = area * (BCsArray(2)%heat_transfer_coeff + (NodesArray(nn)%
        thermal_conductivity + NodesArray(nn-1)%thermal_conductivity)/(2*deltaX) +
        (density * NodesArray(nn)%specific_heat * deltaX / (2*Timestep)))
329   cc_TA(nn) = 0.0
330   dd_TA(nn) = (density * NodesArray(nn)%specific_heat * area * deltaX * NodesArray
        (nn)%temperature_init) / (2 * Timestep) + (BCsArray(2)%heat_transfer_coeff *
        area * BCsArray(2)%environment_temperature) + E_in/(2*(nn-1))
331
332   ! Thomas algorithm step 1 - Forward sweep
333   c_prime(1) = cc_TA(1)/bb_TA(1)
334   d_prime(1) = dd_TA(1)/bb_TA(1)
335
336   Do AA = 2,nn
337       temp = bb_TA(AA) - aa_TA(AA)*c_prime(AA-1)
338       c_prime(AA) = cc_TA(AA)/temp
339       d_prime(AA) = (dd_TA(AA) - aa_TA(AA)*d_prime(AA-1))/temp
340   End Do
341
342   ! Thomas algorithm step 2 - Backward substitution
343   xx_TA(nn) = d_prime(nn)
344   Do AA = (nn-1),1,-1
345       xx_TA(AA) = d_prime(AA) - c_prime(AA) * xx_TA(AA+1)
346   End Do
347   !------------------------------------------------------------------------------
        ----------------------------------------
348
349   !------------------------------------------------------------------------------
        ----------------------------------------
350   ! Set the values in storage - after calculations
351   Do BB=1,nn
352   NodesArray(BB)%temperature_final = xx_TA(BB)
353   Call SetStaticArrayValue(BB,NodesArray(BB)%temperature_final)
354   End Do
355   !------------------------------------------------------------------------------
        ----------------------------------------
356
357   !------------------------------------------------------------------------------
        ----------------------------------------
358   ! Perform calculations after solving the FDM implicit scheme at each time step
359
360   ! Calculate average temperature_final of our PCM domain
361   T_f = 0.0 !Initialize T_f = 0.0
362   Liquid_Fraction = 0.0 !initialize Liquid_Fraction
363
```

```
364  Do BB=1,nn
365  T_f = T_f + NodesArray(BB)%temperature_final
366
367  ! calculate the average liquid fraction of the all domain based on average
       temperature of all nodes
368  If (NodesArray(BB)%temperature_final < T_solidus) Then
369          Liquid_Fraction = Liquid_Fraction + 0.0
370      Else If (NodesArray(BB)%temperature_final > T_liquidus) Then
371          Liquid_Fraction = Liquid_Fraction + 1.0
372      Else
373          Liquid_Fraction = Liquid_Fraction + (NodesArray(BB)%temperature_final-
             T_solidus)/(T_liquidus-T_solidus)
374  Endif
375  End Do
376
377  T_f = T_f/nn !calculates the average of final temperatures for all the nodes
378  Liquid_Fraction = Liquid_Fraction/nn
379
380  ! calculate the absorbed/released energy during each time step
381  Q_PCM = 0.0 ! initialize
382  ! Loop through all nodes to calculate the total heat variation
383  ! node 1 first - surface node
384  Q_PCM = Q_PCM + density*area*deltaX*NodesArray(1)%specific_heat*(NodesArray(1)%
       temperature_final-NodesArray(1)%temperature_init)/(2*Timestep)
385  ! all other interior nodes then
386  Do BB=2,nn-1
387  Q_PCM = Q_PCM + density*area*deltaX*NodesArray(BB)%specific_heat*(NodesArray(BB)%
       temperature_final-NodesArray(BB)%temperature_init)/Timestep
388  End Do
389  ! final node - other surface node
390  Q_PCM = Q_PCM + density*area*deltaX*NodesArray(nn)%specific_heat*(NodesArray(nn)%
       temperature_final-NodesArray(nn)%temperature_init)/(2*Timestep)
391  ! Q_PCM now contains the total heat variation for the entire PCM domain
392
393  ! Calculate the cumulative energy absorbed/released by the PCM (E_out)
394  E_out = E_out + Q_PCM*Timestep
395
396  ! Calculate the absolute value of E_out
397  E_out_abs = ABS(E_out)
398
399  !------------------------------------------------------------------------------
       ---------------------------------------
400
401  !------------------------------------------------------------------------------
       ---------------------------------------
402  ! Set the Outputs from this Model
403   Call SetOutputValue(1,T_f)
404   Call SetOutputValue(2,Q_PCM)
405   Call SetOutputValue(3,Liquid_Fraction)
406   Call SetOutputValue(4,area)
407   Call SetOutputValue(5,E_out_abs) !Output the absolute value
408   Do jj=6,nn+5
```

```
409       Call SetOutputValue(jj,NodesArray(jj-5)%temperature_final)
410     End Do
411    !------------------------------------------------------------------------------
        ----------------------------------------
412
413     Return
414    End
```