

Module Interface Specification for Solar Water Heating Systems Incorporating Phase Change Material

Brooks MacLachlan and Spencer Smith

February 14, 2017

Contents

1	Introduction	4
2	Notation	4
3	Module Decomposition	5
4	MIS of Control Module	7
4.1	Module	7
4.2	Uses	7
4.3	Syntax	7
4.3.1	Exported Access Programs	7
4.4	Semantics	7
4.4.1	State Variables	7
4.4.2	Environment Variables	7
4.4.3	Access Routine Semantics	8
5	MIS of Input Parameters Module	9
5.1	Module	9
5.2	Uses	9
5.3	Syntax	9
5.3.1	Exported Access Programs	9
5.3.2	Assumptions	9

5.4	Semantics	9
5.4.1	State Variables	9
5.4.2	Access Routine Semantics	10
5.5	Considerations	12
6	MIS of Input Format Module	13
6.1	Module	13
6.2	Uses	13
6.3	Syntax	13
6.4	Exported Access Programs	13
6.5	Semantics	13
6.5.1	Environment Variables	13
6.5.2	Assumptions	13
6.5.3	Access Routine Semantics	13
6.5.4	Local Functions	14
7	MIS of Input Verification Module	16
7.1	Module	16
7.2	Uses	16
7.3	Syntax	16
7.3.1	Exported Access Programs	16
7.4	Semantics	16
7.4.1	Assumptions	16
7.4.2	Access Routine Semantics	16
7.5	Considerations	18
8	MIS of Temperature ODEs Module	19
8.1	Module	19
8.2	Uses	19
8.3	Syntax	19
8.3.1	Exported Access Programs	19
8.4	Semantics	19
8.4.1	State Variables	19
8.4.2	Assumptions	19
8.4.3	Access Routine Semantics	19

9	MIS of ODE Solver Module	21
9.1	Module	21
9.2	Uses	21
9.3	Syntax	21
9.3.1	Exported Constants	21
9.3.2	Exported Access Programs	21
9.4	Semantics	21
9.4.1	State Variables	21
9.4.2	Access Routine Semantics	22
10	MIS of Energy Module	23
10.1	Module	23
10.2	Uses	23
10.3	Syntax	23
10.3.1	External Access Programs	23
10.4	Semantics	23
10.4.1	State Variables	23
10.4.2	Assumptions	23
10.4.3	Access Routine Semantics	24
10.4.4	Local Functions	24
11	MIS of Output Verification Module	26
11.1	Module	26
11.2	Uses	26
11.3	Syntax	26
11.3.1	Exported Access Programs	26
11.4	Semantics	26
11.4.1	State Variables	26
11.4.2	Environment Variables	26
11.4.3	Local Variables	26
11.4.4	Assumptions	26
11.4.5	Access Routine Semantics	27
11.4.6	Local Functions	27
12	MIS of Plotting Module	29
12.1	Module	29
12.2	Uses	29
12.3	Syntax	29

12.3.1	Exported Access Programs	29
12.4	Semantics	29
12.4.1	State Variables	29
12.4.2	Environment Variables	29
12.4.3	Assumptions	29
12.4.4	Access Routine Semantics	29
13	MIS of Output Module	30
13.1	Module	30
13.2	Uses	30
13.3	Syntax	30
13.3.1	Exported Constants	30
13.3.2	Exported Access Program	30
13.4	Semantics	30
13.4.1	State Variables	30
13.4.2	Environment Variables	30
13.4.3	Access Routine Semantics	31
14	Appendix	32

1 Introduction

The following document details the Module Interface Specifications for the implemented modules in a program simulating a Solar Water Heating System with Phase Change Material. It is intended to ease navigation through the program for design and maintenance purposes.

Complementary documents include the System Requirement Specifications and Module Guide.

2 Notation

The structure of the MIS for modules comes from [Hoffman and Strooper \(1995\)](#), with the addition that template modules have been adapted from [Ghezzi et al. \(2003\)](#). The mathematical notation comes from Chapter 3 of [Hoffman and Strooper \(1995\)](#). For instance, the symbol $:=$ is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by SWHS.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	\mathbb{Z}	a number without a fractional component in $(-\infty, \infty)$
natural number	\mathbb{N}	a number without a fractional component in $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$

The specification of SWHS uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, SWHS uses functions, which are defined by the data types of their inputs and outputs. Functions are described by showing their input data types separated by multiplication symbols on the left side of an arrow, and their output data type on the right side.

3 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Input Parameters Module Input Verification Module Output Format Module Output Verification Module Temperature ODEs Module Energy Equations Module Control Module
Software Decision Module	Sequence Data Structure Module ODE Solver Module Plotting Module

Table 1: Module Hierarchy

4 MIS of Control Module

4.1 Module

main

4.2 Uses

parameters (Section 5), load_params (Section 6), verify_params (Section 7), temperature (Section 8), ODE Solvers Module (Section 9), energy (Section 10), verify_output (Section 11), plot (Section 12), output (Section 13)

4.3 Syntax

4.3.1 Exported Access Programs

Name	In	Out	Exceptions
main	string	-	-

4.4 Semantics

4.4.1 State Variables

time: array of \mathbb{R}

tempW: array of \mathbb{R}

tempP: array of \mathbb{R}

latHeat: array of \mathbb{R}

eW: array of \mathbb{R}

eP: array of \mathbb{R}

eTot: array of \mathbb{R}

4.4.2 Environment Variables

win: 2D array of pixels displayed on the screen

4.4.3 Access Routine Semantics

main(s): transition: $time, tempW, tempP, latHeat, eW, eP, eTot, win := results[0], results[1], results[2], results[3], eW1||eW2||eW3, eP1||eP2||eP3, (\forall i \in [0..|post(eW)| - 1]) (post(eW[i]) + post(eP[i]))$, Prints information about the melting of PCM.

exception: none

5 MIS of Input Parameters Module

5.1 Module

Param

5.2 Uses

N/A

5.3 Syntax

5.3.1 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getL	-	\mathbb{R}	-
get_diam	-	\mathbb{R}	-
getVp	-	\mathbb{R}	-
...
get_tau_w_noPCM	-	\mathbb{R}	-
setL	\mathbb{R}	-	-
set_diam	\mathbb{R}	-	-
setVp	\mathbb{R}	-	-
...
set_tau_w_noPCM	\mathbb{R}	-	-

5.3.2 Assumptions

Parameters will not be used before it has been initialized. That is, init will be called before any other access programs.

5.4 Semantics

5.4.1 State Variables

L: \mathbb{R}

diam: \mathbb{R}

Vp: \mathbb{R}

A_p : \mathbb{R}
 ρ_p : \mathbb{R}
 T_{melt} : \mathbb{R}
 C_{ps} : \mathbb{R}
 C_{pl} : \mathbb{R}
 H_f : \mathbb{R}
 A_c : \mathbb{R}
 T_c : \mathbb{R}
 ρ_w : \mathbb{R}
 C_w : \mathbb{R}
 h_c : \mathbb{R}
 h_p : \mathbb{R}
 T_{init} : \mathbb{R}
 t_{step} : \mathbb{R}
 t_{final} : \mathbb{R}
 $AbsTol$: \mathbb{R}
 $RelTol$: \mathbb{R}
 $Constol$: \mathbb{R}
 V_t : \mathbb{R}
 M_w : \mathbb{R}
 τ_w : \mathbb{R}
 η : \mathbb{R}
 M_p : \mathbb{R}
 τ_{ps} : \mathbb{R}
 τ_{pl} : \mathbb{R}
 E_{pmelt_init} : \mathbb{R}
 E_{p_melt3} : \mathbb{R}
 M_{w_noPCM} : \mathbb{R}
 $\tau_{w_no_PCM}$: \mathbb{R}

5.4.2 Access Routine Semantics

`init()`:

- transition: $L, diam, V_p, A_p, \dots, \tau_{w_no_PCM} := 0, 0, 0, 0, \dots, 0$
- output: $out := self$
- exception: none

getL():

- output: *out* := L
- exception: none

get_diam():

- output: *out* := diam
- exception: none

getVp():

- output: *out* := Vp
- exception: none

getAp():

- output: *out* := Ap
- exception: none

...

get_tau_w_no_PCM():

- output: *out* := tau_w_no_PCM
- exception: none

setL(x):

- transition: L := x
- exception: none

set_diam(x):

- transition: diam := x
- exception: none

setVp(x):

- transition: Vp := x

- exception: none

setAp(x):

- transition: $Ap := x$
- exception: none

...

set_tau_w_no_PCM(x):

- transition: $\text{tau_w_no_PCM} := x$
- exception: none

5.5 Considerations

As implied above, there is a getter and a setter for each state variable. All state variables are initially initialized to zero. Some programming language implementations will likely use the dot notation to access the fields of this template module, rather than actually write out all of the getters and setters.

6 MIS of Input Format Module

6.1 Module

Load_params

6.2 Uses

Param (Section 5)

6.3 Syntax

6.4 Exported Access Programs

Name	In	Out	Exceptions
load_params	string	-	-

6.5 Semantics

6.5.1 Environment Variables

f : sequence of string $\#f[i]$ is the i th string in the text file f

6.5.2 Assumptions

The input string corresponds to an existing filename. The name will be relative to the current directory. The input file is assumed to be formatted correctly. The file contains the string equivalents of the numeric values for each input parameter in order, each on a new line. The order is the same as in the table in R1 of the SRS. Any comments in the input file should be denoted with a '#' symbol.

6.5.3 Access Routine Semantics

load_params(s):

- transition: The filename s is first associated with the file f . File f is then used to modify the state of Param (Section 5) as follows:
 1. Param.init()

2. Read data sequentially from f to populate the state variables of Param from L to ConsTol.
3. Calculate the derived quantities in Param as follows:
 - Param.setVt(calcVt(Param.getL(), Param.get_diam()))
 - Param.setMw(calcMw(Param.getVp(), Param.get_rho_w(), Param.getVt()))
 - Param.set_tau_w(calcTauw(Param.getMw(), Param.getC_w(), Param.get_hc(), Param.getAc()))
 - Param.set_eta(calcEta(Param.get_hp(), Param.getAp(), Param.get_hc(), Param.getAc()))
 - Param.setMp(calcMp(Param.get_rho_p(), Param.getVp()))
 - Param.set_tau_ps(calcTaups(Param.getMp(), Param.getC_ps(), Param.get_hp(), Param.getAp()))
 - Param.set_tau_pl(calcTaupl(Param.getMp(), Param.getC_pl(), Param.get_hp(), Param.getAp()))
 - Param.setEpmelt_init(calcEpmeltinit(Param.getC_ps(), Param.getMp(), Param.getTmelt(), Param.getTinit()))
 - Param.setEp_melt3(calcEpmelt3(Param.getHf(), Param.getMp()))
 - Param.setMw_noPCM(calcMwno(Param.get_rho_w, Param.getVt()))
 - Param.set_tau_no_PCM(calcTauwnoPCM(Param.getMw_noPCM(), Param.getC_w(), Param.get_hc(), Param.getAc()))

- exception: none

6.5.4 Local Functions

calcVt: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

calcVt(L, d) $\equiv \pi \times L \times (\frac{d}{2})^2$

calcMw: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

calcMw(V_p, ρ_w, V_t) $\equiv \rho_w(V_t - V_p)$

calcTauw: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

calcTauw(m_w, C_w, h_c, A_c) $\equiv \frac{m_w C_w}{A_c h_c}$

calcEta: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

calcEta(h_p, A_p, h_c, A_c) $\equiv \frac{h_p A_p}{h_c A_c}$

$$\text{calcMp}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcMp}(\rho_p, V_p) \equiv \rho_p V_p$$

$$\text{calcTaups}: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcTaups}(M_p, C_{ps}, h_p, A_p) \equiv \frac{M_p C_{ps}}{h_p A_p}$$

$$\text{calcTaupl}: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcTaupl}(M_p, C_{pl}, h_p, A_p) \equiv \frac{M_p C_{pl}}{h_p A_p}$$

$$\text{calcEpmeltinit}: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcEpmeltinit}(C_{ps}, M_p, T_{\text{melt}}, T_{\text{init}}) \equiv C_{ps} M_p (T_{\text{melt}} - T_{\text{init}})$$

$$\text{calcEpmelt3}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcEpmelt3}(H_f, M_p) \equiv H_f M_p$$

$$\text{calcMwnoPCM}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcMwnoPCM}(\rho_w, V_t) \equiv \rho_w V_t$$

$$\text{calcTauwnoPCM}: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{calcTauwnoPCM}(M_{w\text{noPCM}}, C_w, h_c, A_c) \equiv \frac{M_{w\text{noPCM}} C_w}{h_c A_c}$$

7 MIS of Input Verification Module

7.1 Module

verify_params

7.2 Uses

Param (Section 5)

7.3 Syntax

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
verify_valid	-	-	badLength, badDiam, badPCMVolume, badPCMAAndTankVol, badPCMArea, badPCMDensity, badMeltTemp, badCoilAndInitTemp, badCoilTemp, badPCMHeatCapSolid, badPCMHeatCapLiquid, badHeatFusion, badCoilArea, badWaterDensity, badWaterHeatCap, badCoilCoeff, badPCMCoeff, badInitTemp, badFinalTime, badInitAndMeltTemp
verify_recommend	-	-	-

7.4 Semantics

7.4.1 Assumptions

All of the fields Param have been assigned values before any of the access routines for this module are called.

7.4.2 Access Routine Semantics

verify_valid():

- transition: none

- exceptions: $\text{exc} := ($
 $\text{Param.getL()} \leq 0 \Rightarrow \text{badLength} \mid$
 $\text{Param.get_diam()} \leq 0 \Rightarrow \text{badDiam} \mid$
 $\text{Params.get_Vp()} \leq 0 \Rightarrow \text{badPCMVVolume} \mid$
 $\text{Params.getVp()} \geq \text{Params.Vt} \Rightarrow \text{badPCMAndTankVol} \mid$
 $\text{Params.getAp()} \leq 0 \Rightarrow \text{badPCMArea} \mid$
 $\text{Params.get_rho_p()} \leq 0 \Rightarrow \text{badPCMDensity} \mid$
 $\text{Params.getTmelt()} \leq 0 \Rightarrow \text{badMeltTemp} \mid$
 $\text{Params.getTmelt()} \geq \text{Params.getTc()} \Rightarrow \text{badMeltTemp} \mid$
 $\text{Params.getTc()} \leq \text{Params.getTinit()} \Rightarrow \text{badCoilAndInitTemp} \mid$
 $\text{Params.getTc()} \geq 100 \vee \text{Params.getTc()} \leq 0 \Rightarrow \text{badCoilTemp} \mid$
 $\text{Params.getC_ps()} \leq 0 \Rightarrow \text{badPCMHeatCapSolid} \mid$
 $\text{Params.getC_pl()} \leq 0 \Rightarrow \text{badPCMHeatCapLiquid} \mid$
 $\text{Params.getHf()} \leq 0 \Rightarrow \text{badHeatFusion} \mid$
 $\text{Params.getAc()} \leq 0 \Rightarrow \text{badCoilArea} \mid$
 $\text{Params.get_rho_w()} \leq 0 \Rightarrow \text{badWaterDensity} \mid$
 $\text{Params.getC_w()} \leq 0 \Rightarrow \text{badWaterHeatCap} \mid$
 $\text{Params.get_hc()} \leq 0 \Rightarrow \text{badCoilCoeff} \mid$
 $\text{Params.get_hp()} \leq 0 \Rightarrow \text{badPCMCoeff} \mid$
 $\text{Params.getTinit()} \leq 0 \vee \text{Params.getTinit()} \geq 100 \Rightarrow \text{badInitTemp} \mid$
 $\text{Params.get_tfinal()} \leq 0 \Rightarrow \text{badFinalTime} \mid$
 $\text{Params.getTinit()} \geq \text{Params.getTmelt()} \Rightarrow \text{badInitAndMeltTemp})$

$\text{verify_recommend}()$:

- transition: none
- exceptions: $\text{exc} := ($
 $\text{Params.getL()} < 0.1 \vee \text{Params.getL()} > 50 \Rightarrow \text{warnLength} \mid$
 $\text{Params.getdiam()} / \text{Params.getL()} < 0.002 \vee \text{Params.getdiam()} / \text{Params.getL()} > 200 \Rightarrow \text{warnDiam} \mid$
 $\text{Params.getVp()} < \text{Params.getVt()} \times 10^{-6} \Rightarrow \text{warnPCMVVol} \mid$
 $\text{Params.getVp()} > \text{Params.getAp()} \vee \text{Params.getAp} > (2/0.001) \times \text{Params.getVp()} \Rightarrow \text{warnVolArea} \mid$
 $(\text{Params.get_rho_p()} \leq 500) \vee (\text{Params.get_rho_p()} \geq 20000) \Rightarrow \text{warn-PCMDensity} \mid \dots)$
Need to continue for the rest of the example - tabular form? # Should add a module (Configuration Module) to store symbolic constants

7.5 Considerations

See Appendix (Section 14) for the complete list of exceptions and associated error messages.

- exception: none

9 MIS of ODE Solver Module

9.1 Module

ODE Solver Module

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

MaxStep: natural number

N: natural number

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
solve	function, array of \mathbb{R} , array of \mathbb{R} , function, \mathbb{R} , \mathbb{R}	array of \mathbb{R} (<i>N</i> of them)	ODE_BAD_INPUT, ODE_MAXSTEP, ODE_ACCURACY

9.4 Semantics

9.4.1 State Variables

results: array of \mathbb{R} (*N* of them)

9.4.2 Access Routine Semantics

`solve(f, domain, ics, events, abstol, reltol)` output: *out* := *results*, where
results holds the solution to the ODE system generated by the solver.

exceptions: *exc* := (Invalid input parameters \Rightarrow ODE_BAD_INPUT | *MaxStep* steps taken and no solution found \Rightarrow ODE_MAXSTEP | *reltol* and *abstol* not satisfied for a step \Rightarrow ODE_ACCURACY)

10 MIS of Energy Module

10.1 Module

energy

10.2 Uses

Param (Section 5)

10.3 Syntax

10.3.1 External Access Programs

Name	In	Out	Exceptions
energy1Wat	array of \mathbb{R} , parameters	array of \mathbb{R}	-
energy1PCM	array of \mathbb{R} , parameters	array of \mathbb{R}	-
energy2Wat	array of \mathbb{R} , parameters	array of \mathbb{R}	-
energy2PCM	array of \mathbb{R} , parameters	array of \mathbb{R}	-
energy3Wat	array of \mathbb{R} , parameters	array of \mathbb{R}	-
energy3PCM	array of \mathbb{R} , parameters	array of \mathbb{R}	-

10.4 Semantics

10.4.1 State Variables

$eW1$: array of \mathbb{R}

$eP1$: array of \mathbb{R}

$eW2$: array of \mathbb{R}

$eP2$: array of \mathbb{R}

$eW3$: array of \mathbb{R}

$eP3$: array of \mathbb{R}

10.4.2 Assumptions

All of the fields of the input parameters structure have been assigned a value.
The values have been properly constrained.

10.4.3 Access Routine Semantics

energy1Wat($Tw1, params$):	transition:	$(\forall i \in [0.. Tw1 - 1]) (eW1[i] := \text{watEnergy}(Tw1[i], params))$
	output:	$out := eW1$
	exception:	none
energy1PCM($Tp1, params$):	transition:	$(\forall i \in [0.. Tp1 - 1]) (eP1[i] := \text{pcmEnergy1}(Tp1[i], params))$
	output:	$out := eP1$
	exception:	none
energy2Wat($Tw2, params$):	transition:	$(\forall i \in [0.. Tw2 - 1]) (eW2[i] := \text{watEnergy}(Tw2[i], params))$
	output:	$out := eW2$
	exception:	none
energy2PCM($Qp2, params$):	transition:	$(\forall i \in [0.. Qp2 - 1]) (eP2[i] := \text{pcmEnergy2}(Qp2[i], params))$
	output:	$out := eP2$
	exception:	none
energy3Wat($Tw3, params$):	transition:	$(\forall i \in [0.. Tw3 - 1]) (eW3[i] := \text{watEnergy}(Tw3[i], params))$
	output:	$out := eW3$
	exception:	none
energy3PCM($Tp3, params$):	transition:	$(\forall i \in [0.. Tp3 - 1]) (eP3[i] := \text{pcmEnergy3}(Tp3[i], params))$
	output:	$out := eP3$
	exception:	none

10.4.4 Local Functions

watEnergy: $\mathbb{R} \times \text{parameters} \rightarrow \mathbb{R}$

$\text{watEnergy}(Tw, params) \equiv params.C_w \times params.Mw \times (Tw - params.Tinit)$

pcmEnergy1: $\mathbb{R} \times \text{parameters} \rightarrow \mathbb{R}$

$$\text{pcmEnergy1}(Tp, \text{params}) \equiv \text{params}.C_{ps} \times \text{params}.Mp \times (Tp - \text{params}.T_{init})$$

$$\text{pcmEnergy2}: \mathbb{R} \times \text{parameters} \rightarrow \mathbb{R}$$

$$\text{pcmEnergy2}(Qp, \text{params}) \equiv \text{params}.E_{\text{pmelt_init}} + Qp$$

$$\text{pcmEnergy3}: \mathbb{R} \times \text{parameters} \rightarrow \mathbb{R}$$

$$\text{pcmEnergy3}(Tp, \text{params}) \equiv \text{params}.E_{\text{pmelt_init}} + \text{params}.E_{\text{p_melt3}} + \text{params}.C_{pl} \times \text{params}.Mp \times (Tp - \text{params}.T_{\text{melt}})$$

11 MIS of Output Verification Module

11.1 Module

verify_output

11.2 Uses

Param (Section 5)

11.3 Syntax

11.3.1 Exported Access Programs

Name	In	Out	Exceptions
verify_output	array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , parameters	-	-

11.4 Semantics

11.4.1 State Variables

expEPCM: array of \mathbb{R}

expEWat: array of \mathbb{R}

errorWater: \mathbb{R}

errorPCM: \mathbb{R}

11.4.2 Environment Variables

win: 2D array of pixels displayed on the screen

11.4.3 Local Variables

11.4.4 Assumptions

All of the fields of the input parameters structure have been assigned a value. The values have been properly constrained. The input arrays are not empty.

11.4.5 Access Routine Semantics

`verify_output($t, Tw, Tp, Ew, Ep, params$):` `transition:` $expEPCM,$ $expEWat,$
 $errorWater,$ $errorPCM,$
 $win := (\forall i \in [1..|t| - 1])$
 $(expectedEp(traprule(delta(t[i - 1], t[i]), Tw[i], Tp[i],$
 $Tw[i - 1], Tp[i - 1]), params)),$
 $(\forall i \in [1..|t| - 1]) (expectedEw$
 $(expectedEc(traprule(delta(t[i - 1], t[i]),$
 $params.Tc, Tw[i],$
 $params.Tc, Tw[i - 1]),$
 $params), post(expEPCM))),$
 $error(sum(post(expEWat),$
 $Ew[|Ew| - 1]),$
 $error(sum(post(expEPCM),$
 $Ep[|Ep| - 1]), (errorWater >$
 $ConsTol \vee errorPCM >$
 $ConsTol \Rightarrow \text{Prints warning}$
 $\text{message(s)})$
`exception:` $(errorWater > ConsTol \Rightarrow$
 $warnWaterError \mid errorPCM >$
 $ConsTol \Rightarrow warnPCMError)$
 These exceptions do not terminate the program.

11.4.6 Local Functions

$$\begin{aligned} \text{delta: } \mathbb{R} \times \mathbb{R} &\rightarrow \mathbb{R} \\ \text{delta}(t1, t2) &\equiv t2 - t1 \end{aligned}$$
$$\begin{aligned} \text{traprul e: } & \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \\ \text{traprul e}(t, A1, B1, A2, B2) & \equiv t \times (A1 - B1 + A2 - B2)/2 \end{aligned}$$
$$\begin{aligned} \text{expectedEc}: \mathbb{R} \times \text{parameters} &\rightarrow \mathbb{R} \\ \text{expectedEc}(c, \text{params}) &\equiv \text{params}.hc \times \text{params}.Ac \times c \end{aligned}$$
$$\text{expectedEp}: \mathbb{R} \times \text{parameters} \rightarrow \mathbb{R}$$

$$\text{expectedEp}(p, \text{params}) \equiv \text{params.hp} \times \text{params.Ap} \times p$$

$$\text{expectedEw}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{expectedEw}(Ec, Ep) \equiv Ec - Ep$$

$$\text{sum}: \text{array of } \mathbb{R}\text{s} \rightarrow \mathbb{R}$$

$$\text{sum}(a) \equiv \sum_{i=0}^{|a|-1} a[i]$$

$$\text{error}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{error}(\text{exp}, \text{act}) \equiv \frac{|\text{exp} - \text{act}|}{\text{act}} \times 100$$

12 MIS of Plotting Module

12.1 Module

plot

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
plot	array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , string	-	-

12.4 Semantics

12.4.1 State Variables

plotFilename: string

12.4.2 Environment Variables

directory: The current directory of files from which the program is run.

12.4.3 Assumptions

The input arrays are all of the same size.

12.4.4 Access Routine Semantics

plot($t, Tw, Tp, Ew, Ep, filename$): transition: *directory*: writes a .png file named *plotFilename* containing the graphs of the simulation results.
exception: none

13 MIS of Output Module

13.1 Module

output

13.2 Uses

Param (Section 5)

13.3 Syntax

13.3.1 Exported Constants

max_width: integer

13.3.2 Exported Access Program

Name	In	Out	Exceptions
output	string, array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , array of \mathbb{R} , parameters	-	-

13.4 Semantics

13.4.1 State Variables

outFilename: string

13.4.2 Environment Variables

directory: The current directory of files from which the program is run.

13.4.3 Access Routine Semantics

`output(params, t, Tw, Tp, Ew, Ep, ETot, filename):` transition: *directory:* writes
a .txt file named
outFilename con-
taining the input
parameters, calcu-
lated parameters,
and results of the
simulation.
exception: none

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseeer.ist.psu.edu/428727.html>.

14 Appendix

Table 2: Possible Exceptions

Message ID	Error Message
badLength	Error: Tank length must be > 0
badDiam	Error: Tank diameter must be > 0
badPCMVolume	Error: PCM volume must be > 0
badPCMAndTankVol	Error: PCM volume must be $<$ tank volume
badPCMArea	Error: PCM area must be > 0
badPCMDensity	Error: ρ_p must be > 0
badMeltTemp	Error: T_{melt} must be > 0 and $< T_c$
badCoilAndInitTemp	Error: T_c must be $> T_{init}$
badCoilTemp	Error: T_c must be > 0 and < 100
badPCMHeatCapSolid	Error: C_{ps} must be > 0
badPCMHeatCapLiquid	Error: C_{pl} must be > 0
badHeatFusion	Error: H_f must be > 0
badCoilArea	Error: A_c must be > 0
badWaterDensity	Error: ρ_w must be > 0
badWaterHeatCap	Error: C_w must be > 0
badCoilCoeff	Error: h_c must be > 0
badPCMCoeff	Error: h_p must be > 0
badInitTemp	Error: T_{init} must be > 0 and < 100
badFinalTime	Error: t_{final} must be > 0
badInitAndMeltTemp	Error: T_{init} must be $< T_{melt}$
ODE_ACCURACY	<i>reltol</i> and <i>abstol</i> were not satisfied by the ODE solver for a given solution step.
ODE_BAD_INPUT	Invalid input to ODE solver
ODE_MAXSTEP	ODE solver took <i>MaxStep</i> steps and did not find solution
warnLength	Warning: It is recommended that $0.1 \leq L \leq 50$
warnDiam	Warning: It is recommended that $0.002 \leq D/L \leq 200$

warnPCMVOL	Warning: It is recommended that V_p be $\geq 0.0001\%$ of V_t
warnVolArea	Warning: It is recommended that $V_p \leq A_p \leq (2/0.001) * V_p$
warnPCMDensity	Warning: It is recommended that $500 < \rho_p < 20000$
warnPCMHeatCapSolid	Warning: It is recommended that $100 < C_{ps} < 4000$
warnPCMHeatCapLiquid	Warning: It is recommended that $100 < C_{pl} < 5000$
warnCoilArea	Warning: It is recommended that $A_c \leq \pi * (D/2) \wedge 2$
warnWaterDensity	Warning: It is recommended that $950 < \rho_w \leq 1000$
warnWaterHeatCap	Warning: It is recommended that $4170 < C_w < 4210$
warnCoilCoeff	Warning: It is recommended that $10 < h_c < 10000$
warnPCMCoeff	Warning: It is recommended that $10 < h_p < 10000$
warnFinalTime	Warning: It is recommended that $0 < t_{final} < 86400$
warnWaterError	Warning: There is greater than $x\%$ relative error between the energy in the water output and the expected output based on the law of conservation of energy. (Where x is the value of <i>ConsTol</i>)
warnPCMError	Warning: There is greater than $x\%$ relative error between the energy in the PCM output and the expected output based on the law of conservation of energy. (Where x is the value of <i>ConsTol</i>)
