Module Interface Specification for Sun Catcher

Sharon (Yu-Shiuan) Wu

November 23, 2019

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at . . . . [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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 $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Sun Catcher.

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

The specification of Sun Catcher 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, Sun Catcher uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 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	Control Module Input Parameters Module Input Verify Module Output Parameters Module Solar Energy Absorption Module Optimum Tilt Angle Module Sun Intensity Equation Module Zenith Angle Equation Module Sun Declination Module Days Module
Software Decision Module	Count days Module Table-layout Module Sequence Data Structure Module

Table 1: Module Hierarchy

6 Sun Catcher Type Module

6.1 Module

SunCatTy

6.2 Uses

N/A

6.3 Exported Types

DegreeT = $\mathbb{R}$
DayT = a tuple of (d:$\mathbb{N}$, m:$\mathbb{N}$, y:$\mathbb{N}$)
AnalemmaT = a tuple of (x: $\mathbb{R}$,y: $\mathbb{R}$,z: $\mathbb{R}$)

6.4 Syntax

6.4.1 Exported Constants

N/A

6.4.2 Exported Access Programs

N/A

6.5 Semantics

N/A

6.5.1 State Variables

N/A

6.5.2 Environment Variables

N/A

6.5.3 Assumptions

For DayT the day will always lie between 1 and 31, the month will always lie between 1 and 12, and year will always be greater than 1.

7 Day Duration ADT Module

7.1 Template Module

DayDurTy

7.2 Uses

SunCatTy

7.3 Exported Types

DayDurT = ?

7.4 Syntax

7.4.1 Exported Constants

7.4.2 Exported Access Programs

Name	In	Out	Exceptions
new DayDurT	a tuple of (DayT, DayT)	DayDurT	invalid_argument
getdurlist	-	a sequence of integer	-
setlist	DayT	-	stop_recursive
setperihelion	DayT	-	-
getnext	-	DayT	-
Countdiff:	a tuple of (DayT, DayT)	integer	-

7.5 Semantics

7.5.1 State Variables

dayS : DayT
dayE : DayT
periheD: DayT
daduraL: a sequence of integer
dateL: a sequence of DayT

7.5.2 Environment Variables

N/A

7.5.3 Assumptions

In getdiff (day1, day2), assume that day 2 is always greater than day 1.

7.5.4 Access Routine Semantics

new DayDurT (s,e):

- transition: dayS, dayE, periheD.d, periheD.m := s, e, 21, 12;

  dateL := || (s = e ⇒ ⟨ e ⟩ | otherwise ⇒ ⟨ s, setlist (getnext ( )) ⟩)
- output: out := self
- exception: exc := (s >e ⇒ invalid_argument)

setlist (day):

- transition: dateL := || (day = dayE ⇒ ⟨ dayE ⟩ | otherwise ⇒ ⟨ day, setlist (getnext ( )) ⟩)
- output:
- exception: exc := (day = dateL [0] ⇒ stop_recursive)

getdurlist ( ):

- transition: daduraL := || (d: DayT | d ∈ dateL • ⟨ Countdiff (periheD, d) ⟩ where setperihelion (d))
- output: out := daduraL
- exception:

setperihelion (day):

- transition: periheD.y := (day.m = 12 ∧ day.d ≥ 21 ⇒ day.y | otherwise ⇒ day.y - 1)
- output:
- exception:

Countdiff (day1, day2):

- transition:
- output: out := + (day1 = day2 ⇒ 0 | otherwise ⇒ Countdiff (getnext (day1), day2) + 1)
- exception:

getnext ( ):

- transition:
- output: dayS := (dayS < dayE ⇒ The next day of dayS according to the calendar | otherwise ⇒ dateL [0])
- exception:

8 MIS of Input Parameters Module

8.1 Module

InputPara

8.2 Uses

HardH

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	Key_error
getla	-	DegreeT	-
getph	-	real	-
getpw	-	real	-
getds	-	DayT	-
getde	-	DayT	-

8.4 Semantics

8.4.1 State Variables

latitude: DegreeT
dayS: DayT

dayE: DayT
panH: real
panW real

8.4.2 Environment Variables

key: Input variables from keyboard.

8.4.3 Assumptions

When users chick the submit bottom from system interface, InputPara.init ( ) implement.

8.4.4 Access Routine Semantics

init ( ):

- transition: latitude, panH, panW, dayS, dayE := HardH.la, HardH.pn, HardH.pw, (HardH.ds, HardH.ms, HardH.ys), (HardH.de, HardH.me, HardH.ye)
- output:
- exception:

getla ( ):

- transition:
- output: latitude
- exception:

getph ( ):

- transition:
- output: panH
- exception:

getpw ( ):

- transition:
- output: panW
- exception:

getds ( ):

- transition:

- output: dayS
- exception:

getde ( ):

- transition:
- output: dayE
- exception:

8.4.5 Local Functions

9 MIS of Input Verify Module

9.1 Module

InputVer

9.2 Uses

InputPara

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getverlan	-	bool	-
getverpan	-	bool	-
getverday	-	bool	-

9.4 Semantics

9.4.1 State Variables

latitude: DegreeT
dayS: DayT
dayE: DayT
panH: real
panW real

9.4.2 Environment Variables

9.4.3 Assumptions

InputPara will always implement before InputVer

9.4.4 Access Routine Semantics

init ( ):

- transition: latitude,panH, panW, dayS, dayE := InputPara.la, InputPara.ph, InputPara.pw, InputPara.ds, InputPara.de
- output:
- exception:

getverlan ( ):

- transition:
- output: out := (latitude $\leq$ 90 $\wedge$ latitude $\geq$ - 90 $\Rightarrow$ True | otherwise $\Rightarrow$ False)
- exception:

getverpan ( ):

- transition:
- output: out := (panH $>$ 0 $\wedge$ panW $>$ 0 $\Rightarrow$ True | otherwise $\Rightarrow$ False)
- exception:

getverday ( ):

- transition:
- output: out := (if dayS and dayE is a valid date in Gregorian calendar $\Rightarrow$ True | otherwise $\Rightarrow$ False)
- exception:

9.4.5 Local Functions

10 MIS of Output Parameters Module

10.1 Module

OutputPara

10.2 Uses

Table

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
addresult	a tulple of (DegreeT, real, integer)	-	-
getresult	-	a sequence of a tulple of (DegreeT, real, integer)	-
getdiff	real, real	integer	-

10.4 Semantics

10.4.1 State Variables

resultL: a sequence of a tulple of (t: DegreeT, e: real, diff: integer)

10.4.2 Environment Variables

10.4.3 Assumptions

10.4.4 Access Routine Semantics

init ( ):

- transition: resultL := { }
- output:
- exception:

addresult (angle, energy):

- transition: resultL := $\|$ ($|\text{resultL}| = 0 \Rightarrow \langle$ (angle, energy, 100) $\rangle$ | angle $\notin$ resultL.t $\Rightarrow \langle$ (angle, energy, getdiff (resultL.e, energy)) $\rangle$)
- output:
- exception:

getresult ( ):

- transition:
- output: out := resultL
- exception:

getdiff (e1, e2):

- transition:
- output: out := $\frac{e2}{e1} \times 100$
- exception:

10.4.5 Local Functions

11 MIS of Table-layout Module

11.1 Module

Table

11.2 Uses

OutputPara
HardH

11.3 Syntax

11.3.1 Exported Constants

hRow1: "Tilt Angle" hRow2: "Energy Absorption" hRow3: "Energy Competition"

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
display	-	-	-

11.4 Semantics

11.4.1 State Variables

resultL: a sequence of a tulple of (t: DegreeT, e: real, diff: integer)

11.4.2 Environment Variables

Output variables will display on the screen.

11.4.3 Assumptions

11.4.4 Access Routine Semantics

init ( ):

- transition: resultL := OutPara.getresult ( )
- output:
- exception:

display ( ):

- transition:
- output: out := display a table that shows the result on the screen.

  The text file has the following format, where the first row following heading: hRow1, denotes the result of optimal tilt angle; the second row following heading, hRow2, denotes the result of estimating energy absorption; and the third row following heading, hRow3, denotes the result of different energy absorption between resultL.e [0] and resultL.e [1.. | resultL|] followed by a symbol %

Tilt Angle	resultL.t [0]	...	resultL.t [| resultL|]
Energy Absorption	resultL.e [0]	...	resultL.e [| resultL|]
Energy Competition	resultL.diff [0] %	...	resultL.diff [| resultL|] %

- exception:

11.4.5 Local Functions

12 MIS of Optimum Tilt Angle Module

[Use labels for cross-referencing —SS]

12.1 Module

TitleAng [Short name for the module —SS]

12.2 Uses

SunInten
ZenithAng

12.3 Syntax

12.3.1 Exported Constants

$I_S := 1.35$

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getilt	-	DegreeT	-
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

sunIn: real
zenithL: a sequence of DegreeT
maxsunIn: a tuple of (i: real, d: DegreeT)

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

N/A [This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS] init( ):

- transition: sunIn, zenithL,maxsunIn.i, maxsunIn.d := SunInten.sum(I_S), ZenithAng.getzen ( ),
- output:
- exception:

[accessProg —SS] getilt ( ):

- transition: maxsunIn := (j: integer | j ∈ [0.. |zenithL|] • j = 0 ⇒ (SunInten.single (sunIn, zenithL [0]), zenithL [0]) | ifMax (maxsunIn.i, SunInten.single (sunIn, zenithL [j])) ≠ maxsunIn.i ⇒ (SunInten.single (sunIn, zenithL [j]), zenithL [j]))
- output: out := maxsunIn.d
- exception:

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

ifMax: real × real → real
ifMax(x, y) = (x ≥ y ⇒ x | otherwise ⇒ y)

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of Solar Energy Absorption Module

13.1 Module

Energy

13.2 Uses

InputPara, ZenithAng, TiltAng

13.3 Syntax

13.3.1 Exported Constants

$I_S := 1.35$

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getsunIn	-	a sequence of real	-
getenergy	-	a sequence of real	-
getavenergy	-	real	-

13.4 Semantics

13.4.1 State Variables

sunInL: a sequence of real
energyL: a sequence of real
zenL: a sequence of DegreeT
maxAng: real
avenergy: real
panW: real
panH: real

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

ZenithAngle has been fully implement before implement Energy.init ( ).
Energy.init ( ) is called first before any other exported access programs.
Energy.getsunIn ( ) is always called before Energy.getenergy ( ).

13.4.4 Access Routine Semantics

init ( ):

- transition: panW, panH, maxAng, zenL := InputPara.getpw ( ), InputPara.getph ( ), TitleAng.getilt ( ), ZenithAngle.getzen ( )
- output:

- exception:

getsunIn ( ):

- transition: sunInL := ‖ (i: integer | i ∈ [0.. |zenL|] • ⟨ SunInten.single(maxAng, zenL[i]) ⟩)
- output: out := sunInL
- exception:

getenergy ( ):

- transition: energyL := ‖ (i: integer | i ∈ [0.. |sunInL|] • ⟨ panW × panH × 18.7 × 0.75 × sunInL [i] ⟩)
- output: out := energyL
- exception:

getavenergy ( ):

- transition: avenergy := + (i: integer | i ∈ [0.. |energyL|] • energyL[i]) / |energyL|
- output: out := avenergy
- exception:

13.4.5 Local Functions

14 MIS of Sun Intensity Equation Module

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

14.1 Module

SunInten

14.2 Uses

ZenithAngle

14.3 Syntax

14.3.1 Exported Constants

$I_S := 1.35$

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
sum	real	real	-
single	a tuple of (real, DegreeT)	real	-

14.4 Semantics

14.4.1 State Variables

zenL: a sequence of DegreeT

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

SunInten.init ( ) is called first before any other exported access programs
ZenithAngle.init ( ) has been fully implement before implement SunInten.init ( )

14.4.4 Access Routine Semantics

init ( ):

- transition : zenL := ZenithAng.getzen ( )
- output: out :
- exception:

sum (i):

- output: out := + (i: integer | i ∈ [0.. |zenL|] •

$$I_S \cdot \frac{1.00}{i}^{sec(zenL[i])})$$

- exception:

single (i, d):

- output: out :=

$$I_S \cdot \frac{1.00}{i}^{sec(d)}$$

- exception:

14.4.5 Local Functions

15 MIS of Zenith Angle Equation Module

15.1 Module

ZenithAng

15.2 Uses

DayDurTy
SunCatTy

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	AnalammaT	-	-
getzen	-	a sequence of DegreeT	-
getfiltdec	-	a sequence of DegreeT	-
getdec	-	a sequence of DegreeT	-

15.4 Semantics

15.4.1 State Variables

zenS: a sequence of DegreeT
decS: a sequence of DegreeT
day: DayDurTy
dayL: a sequence of integer
latitude: real
analeL: a sequence of AnalemmaT

15.4.2 Environment Variables

15.4.3 Assumptions

Read File Module will always be implemented before Zenith angle Module.
InputPara Module will always be implemented before Zenith angle Module.
ZenithAng.init ( ) is called before any other access program.
ZenithAng.getdec ( ) is called before ZenithAng.getzen ( ).

15.4.4 Access Routine Semantics

[accessProg —SS]init (a):

- transition: analeL := || (i: integer | i ∈ [0.. 365] • (a.x [i], a.y [i], a.z [i]))
  day := new DayDurT (InputPara.days, InputPara.daye)
  dayL, latitude := day.getdurlist, InputPara.getlat
- output:
- exception:

getzen ( ):

- transition: zenS := || (i: integer | i ∈ [0.. 365] • latitude × decS [i] ≥ 0 ⇒ ⟨ latitude - decS [i] ⟩ | otherwise ⇒ ⟨ latitude + decS [i] ⟩)
- output: out := zenS
- exception:

getdec ( ):

- transition: decS := || (i: integer | i ∈ [0.. |dayL|] •

$$\langle \arcsin \frac{analeL.z[dayL[i]]}{\sqrt{analeL.x^2[dayL[i]]+analeL.y^2[dayL[i]]+analeL.z^2[dayL[i]]}} \rangle)$$

- output: out := decS
- exception:

15.4.5 Local Functions

16 MIS of Read File Module

16.1 Module

ReadFile

16.2 Uses

ZenithAng

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
load_anale_data	string	-	-

16.4 Semantics

16.4.1 State Variables

16.4.2 Environment Variables

anale_data: File listing Analemma data

16.4.3 Assumptions

The input file will match the given specification.

16.4.4 Access Routine Semantics

load_anale_data (s):

- transition: read data from the file anale_data associated with the string s. Use this data to update the state of the ZenithAng module. load_anale_data (s) will

  first initialize ZenithAng (ZenithAng.init ( )) before populating ZenithAng with analemma data that follows the types in SunCatTy.

  The text file has the following format, where x_i, y_i z_i denotes the vector that locates Earth relative to the Sun on the day i since perihelion.

  All data values in a row are separated by commas. Rows are separated by a new line. The data shown below is for a total of 366 days.

x_0, y_0, z_0
x_1, y_1, z_1
x_2, y_2, z_2
.
.
.
x_365, y_365, z_365

- output:
- exception:

16.4.5 Local Functions

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://citeseer.ist.psu.edu/428727.html.

17 Appendix

[Extra information if required —SS]