

# Module Interface Specification for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 17, 2019

# 1 Revision History

Date	Version	Notes
2019/11/25	1.0	First Version
Date 2	1.1	Notes

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/sharyuwu/optimum-tilt-of-solar-panels/blob/master/docs/SRS/SRS.pdf>

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

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>1</b>
<b>6</b>	<b>MIS of Control Module</b>	<b>3</b>
6.1	Module . . . . .	3
6.2	Uses . . . . .	3
6.3	Syntax . . . . .	3
6.3.1	Exported Constants . . . . .	3
6.3.2	Exported Access Programs . . . . .	3
6.4	Semantics . . . . .	3
6.4.1	State Variables . . . . .	3
6.4.2	Environment Variables . . . . .	4
6.4.3	Assumptions . . . . .	4
6.4.4	Access Routine Semantics . . . . .	4
6.4.5	Local Functions . . . . .	5
<b>7</b>	<b>Sun Catcher Type Module</b>	<b>7</b>
7.1	Module . . . . .	7
7.2	Uses . . . . .	7
7.3	Exported Types . . . . .	7
7.4	Syntax . . . . .	7
7.4.1	Exported Constants . . . . .	7
7.4.2	Exported Access Programs . . . . .	7
7.5	Semantics . . . . .	7
7.5.1	State Variables . . . . .	7
7.5.2	Environment Variables . . . . .	7
7.5.3	Assumptions . . . . .	8
<b>8</b>	<b>Day ADT Module</b>	<b>9</b>
8.1	Template Module . . . . .	9
8.2	Uses . . . . .	9
8.3	Exported Types . . . . .	9
8.4	Syntax . . . . .	9
8.4.1	Exported Constants . . . . .	9
8.4.2	Exported Access Programs . . . . .	9

8.5	Semantics . . . . .	9
8.5.1	State Variables . . . . .	9
8.5.2	Environment Variables . . . . .	10
8.5.3	Assumptions . . . . .	10
8.5.4	Access Routine Semantics . . . . .	10
8.5.5	Local Functions . . . . .	11
<b>9</b>	<b>MIS of Input Parameters Module</b>	<b>12</b>
9.1	Module . . . . .	12
9.2	Uses . . . . .	12
9.3	Syntax . . . . .	12
9.3.1	Exported Constants . . . . .	12
9.3.2	Exported Access Programs . . . . .	12
9.4	Semantics . . . . .	12
9.4.1	State Variables . . . . .	12
9.4.2	Environment Variables . . . . .	12
9.4.3	Assumptions . . . . .	12
9.4.4	Access Routine Semantics . . . . .	13
9.4.5	Local Functions . . . . .	14
<b>10</b>	<b>MIS of Input Verification Module</b>	<b>15</b>
10.1	Module . . . . .	15
10.2	Uses . . . . .	15
10.3	Syntax . . . . .	15
10.3.1	Exported Constants . . . . .	15
10.3.2	Exported Access Programs . . . . .	15
10.4	Semantics . . . . .	15
10.4.1	State Variables . . . . .	15
10.4.2	Environment Variables . . . . .	15
10.4.3	Assumptions . . . . .	15
10.4.4	Access Routine Semantics . . . . .	15
10.4.5	Local Functions . . . . .	16
<b>11</b>	<b>MIS of Table-layout Module</b>	<b>17</b>
11.1	Module . . . . .	17
11.2	Uses . . . . .	17
11.3	Syntax . . . . .	17
11.3.1	Exported Constants . . . . .	17
11.3.2	Exported Access Programs . . . . .	17
11.4	Semantics . . . . .	17
11.4.1	State Variables . . . . .	17
11.4.2	Environment Variables . . . . .	17
11.4.3	Assumptions . . . . .	17

11.4.4	Access Routine Semantics . . . . .	17
11.4.5	Local Functions . . . . .	18
<b>12</b>	<b>MIS of Optimum Tilt Angle Module</b>	<b>19</b>
12.1	Module . . . . .	19
12.2	Uses . . . . .	19
12.3	Syntax . . . . .	19
12.3.1	Exported Constants . . . . .	19
12.3.2	Exported Access Programs . . . . .	19
12.4	Semantics . . . . .	19
12.4.1	State Variables . . . . .	19
12.4.2	Environment Variables . . . . .	19
12.4.3	Assumptions . . . . .	20
12.4.4	Access Routine Semantics . . . . .	20
12.4.5	Local Functions . . . . .	20
<b>13</b>	<b>MIS of Solar Energy Absorption Module</b>	<b>21</b>
13.1	Module . . . . .	21
13.2	Uses . . . . .	21
13.3	Syntax . . . . .	21
13.3.1	Exported Constants . . . . .	21
13.3.2	Exported Access Programs . . . . .	21
13.4	Semantics . . . . .	21
13.4.1	State Variables . . . . .	21
13.4.2	Environment Variables . . . . .	21
13.4.3	Assumptions . . . . .	21
13.4.4	Access Routine Semantics . . . . .	21
13.4.5	Local Functions . . . . .	22
<b>14</b>	<b>MIS of Calculation</b>	<b>23</b>
14.1	Module . . . . .	23
14.2	Uses . . . . .	23
14.3	Syntax . . . . .	23
14.3.1	Exported Constants . . . . .	23
14.3.2	Exported Access Programs . . . . .	23
14.4	Semantics . . . . .	23
14.4.1	State Variables . . . . .	23
14.4.2	Environment Variables . . . . .	23
14.4.3	Assumptions . . . . .	23
14.4.4	Access Routine Semantics . . . . .	23
14.4.5	Local Functions . . . . .	24
<b>15</b>	<b>Appendix</b>	<b>25</b>

### 3 Introduction

The following document details the Module Interface Specifications for Sun Catcher[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 <https://github.com/sharyuwu/optimum-tilt-of-solar-panels>. [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 ?, with the addition that template modules have been adapted from ?. The mathematical notation comes from Chapter 3 of ?. 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 Sun Catcher.

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 $(-\infty, \infty)$
bool	$\mathbb{B}$	a statement of True or False
sequence	[ ]	a sequence of the same type

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 Verification Module Output Parameters Module Solar Energy Absorption Module Optimum Tilt Angle Module Sun Intensity Equation Module Zenith Angle Equation Module
Software Decision Module	Day ADT Module Table-layout Module Sun Catcher Type Module

Table 1: Module Hierarchy



## 6 MIS of Control Module

### 6.1 Module

Control

### 6.2 Uses

SunCatTy

DayT

InputPara (Section 9)

[Please leave space between the name and the reference. You should also identify what type of reference it is, like you would for a Figure or a Table. That is, you should say (Section 9). —SS]

InputVer (Section 10)

OutputPara (Section ??)

TiltAng (Section 12)

Energy (Section 13)

### 6.3 Syntax

#### 6.3.1 Exported Constants

filename = "analemma.txt"

#### 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
setInput	-	-	-
getOneAng	-	-	-
[main? — SS][changed name — Author]			
getTwoAng	-	-	-
writeOutput	-	-	-

### 6.4 Semantics

#### 6.4.1 State Variables

[You use this type, but I don't think you import (use) the module that defines it —SS]  
[Change DayT as Day ADT —Author] [State Variable dayS and dayE only exit in Input

[Why are these two state variables here? I thought you were going to use the Input Parameters module? —SS] [Now that I think about it, given that you are planning on implementing with Haskell, you might want to avoid having modules with state information. —SS] [Yes, Agree. But I will still keep some of my state variable inside the document for the purpose the easy understanding. Then implement in a slightly different way. —Author]

#### 6.4.2 Environment Variables

#### 6.4.3 Assumptions

#### 6.4.4 Access Routine Semantics

setInput ( ):

- transition: Implement the InputPara and the environment variables for Output by following steps.

Get ( $\Phi_P$ : DegreeT,  $P_{A_h}$ : real,  $P_{A_w}$ : real,  $year_{Start}$ : natural number,  $month_{Start}$ : natural number,  $day_{Start}$ : natural number,  $year_{End}$ : natural number,  $month_{End}$ : natural number,  $day_{End}$ : natural number) from users' input.

load\_anale\_data (filename)  
InputPara.init ( )

# Verify the input values using Input Verification Module  
verifiedLat ( InputPara.getla ( ) )  
verifiedP ( InputPara.getph ( ), InputPara.getpw ( ) )  
verifiedD (InputPara.getdayS, InputPara.getdayE)

getOneAng ( ):

- transition:Implied the one optimum cut by following steps

# Get the zenith angle between dayS and dayE  
 $\theta_{S_{date}} = \text{Calculation.getzen ( InputPara.getdec, localDaySandPer ( ), localDaySand- DayE ( ) )}$

# Get the optimum tilt angle using  $\theta_{S_{date}}$   
 $\theta_T = \text{getilt ( } \theta_{S_{date}}, \text{ localsunIn}(\theta_{S_{date}}) \text{ )}$

# Get the estimated solar absorption using Optimum Tilt Angle Module

```

 $P_E = \text{getenergy}(\text{InputPara.getpw}, \text{InputPara.getph}, \text{localtiltSunIn}(\text{localsunIn}(\theta_{S_{\text{date}}}, \theta_T, \theta_T), \theta_{S_{\text{date}}}))$ 

```

```

# Output  $P_E$  and  $\theta_T$  to the Table-layout Module
addressresult( $\theta_T, P_E$ )

```

getTwoAng ( ):

- transition: Implied the two optimum cut by following steps

```

# Get the zenith angle between dayS and  $\frac{\text{dayE} + \text{dayS}}{2}$  and  $\frac{\text{dayE} + \text{dayS}}{2}$  to dayE
zenithSet =  $\{\theta_{S_{\text{date}}} [1..|\frac{\theta_{S_{\text{date}}}}{2}|]\} \cup \{\theta_{S_{\text{date}}} [|\frac{\theta_{S_{\text{date}}}}{2}|..\theta_{S_{\text{date}}}]\}$ 

```

```

# Get the optimum tilt angle using  $\theta_{S_{\text{date}}}$ 
 $\theta_T = \theta_T \cup (\text{zenList} : \text{DegreeT}[ ] \mid \text{zenList} \in \text{zenithSet} \bullet \text{getilt}(\text{zenList}, \text{local-} \\ \text{sunIn}(\text{zenList})) )$ 

```

```

# Get the estimated solar absorption using Optimum Tilt Angle Module

```

```

 $P_E = P_E \cup (\text{zenList} : \text{DegreeT}[ ] \mid \text{zenList} \in \text{zenithSet} \bullet \text{getenergy}(\text{InputPara.getpw}, \\ \text{InputPara.getph}, \text{localtiltSunIn}(\text{localsunIn}(\text{zenList}, \theta_T, \theta_T), \text{zenList})))$ 

```

```

# Output  $P_E$  and  $\theta_T$  to the Table-layout Module
addressresult( $\theta_T, P_E$ )

```

setInput ( ):

- output: out :=  
 # Write the output values in the file using the Table-layout Module  
 display ( )

#### 6.4.5 Local Functions

```

# The days between start day and perhelion
localDaySandPer: Integer
localDaySandPer = InputPara.getdayS. perhelion

```

```

# The days between start day and end day
localDaySandDayE: Integer

```

```

localDaySandDayE =InputPara.getdayE.countDiff (InputPara.getdayS)

# Calculate the total sun intensity for the base case
localsunIn: a sequence of DegreeT  $\rightarrow$  real
localsunIn (zen[ ]) = Calculation.sumSunIn( 1.35, zen)
# Get the optimum sun intensity
localtiltSunIn: DegreeT  $\times$  real  $\rightarrow$  real
localtiltSunIn (degree, intensity ) = TiltAngle.getiltInten (intensity, degree )
[It is nice to have a newpage between modules. —SS][OK —Author]

```

## 7 Sun Catcher Type Module

### 7.1 Module

SunCatTy

### 7.2 Uses

N/A

### 7.3 Exported Types

DegreeT =  $\mathbb{R}$

### 7.4 Syntax

#### 7.4.1 Exported Constants

N/A

#### 7.4.2 Exported Access Programs

N/A

### 7.5 Semantics

N/A

#### 7.5.1 State Variables

N/A

#### 7.5.2 Environment Variables

N/A

### 7.5.3 Assumptions

N/A

## 8 Day ADT Module

### 8.1 Template Module

DayT

### 8.2 Uses

SunCatTy<sup>7</sup>

### 8.3 Exported Types

DayT = ? [It seems confusing to me that you have a DayT, that is not an ADT, and a DayDurT that is an ADT. Couldn't you just have an ADT for DayT? —SS]

### 8.4 Syntax

#### 8.4.1 Exported Constants

#### 8.4.2 Exported Access Programs

Name	In	Out	Exceptions
new DayT	$\mathbb{N}, \mathbb{N}, \mathbb{N}$	DayT	invalid_argument
getnext	-	DayT	-
perihelion	-	Integer	-
countDiff	DayT	Integer	-
$\leq$	DayT	$\mathbb{B}$	-
$>$	DayT	$\mathbb{B}$	-

[The input to the DayDurT could just be DayT, DayT; you don't really need to make the input a tuple. —SS] [Agree —Author]

### 8.5 Semantics

DayT: natural number

#### 8.5.1 State Variables

stateDay : DayT [You have these same state variables in another module —SS]

### 8.5.2 Environment Variables

N/A

### 8.5.3 Assumptions

[It would be neat if you defined a comparison access program to your ADT so that you have greater than implemented. As it is, saying greater than is ambiguous. —SS] [Change the expression of getdiff to countDiff. It uses compare function to check if day1 is greater than day2 —Author]

### 8.5.4 Access Routine Semantics

new DayT (y, m, d):

- transition:  $\text{stateDay} = (m = 1 \vee m = 2 \Rightarrow \text{MonthOneTwo}(y, m, d) \mid \text{True} \Rightarrow \text{MonthNotOneTwo}(y, m, d))$

[You don't actually say otherwise using our notation. You could just say True. You also don't have a dummy variable. I don't really know how to read this expression. —SS]

- output:  $\text{out} := \text{self}$
- exception:

getnext ( ):

- transition:
- output:  $\text{out} := \text{The next day of the stateDay according to the Gregorian Calender.}$
- exception:

perihelion ( ):

- transition:
- output:  $\text{out} := (\text{stateDay.m} = 12 \wedge \text{stateDay.d} \geq 21 \Rightarrow \text{countDiff}(\text{new DayT}(\text{stateDay.y}, 12, 21)) \mid \text{True} \Rightarrow \text{countDiff}(\text{new DayT}(\text{stateDay.y} - 1, 12, 21)))$
- exception:

countDiff ( day ):

- transition:



- output:  $\text{out} := (\text{day} \leq \text{stateDay} \Rightarrow 1 + \text{countDiff}(\text{getNext}(\text{day1})) \mid \text{day} > \text{stateDay} \Rightarrow 0)$

- exception:

$\leq (\text{day})$ :

- transition:

- output:  $\text{out} := (\text{day is placed before than stateDay Gregorian Calender} \vee \text{in day and day is the same day} \Rightarrow \text{True} \mid \text{False})$

- exception:

$> (\text{day})$ :

- transition:

- output:  $\text{out} := (\text{day is placed before than stateDay Gregorian Calender} \vee \text{day and stateDay is the same day} \Rightarrow \text{True} \mid \text{False})$

- exception:

[You shouldn't have a transition and an output. I don't actually see why you need the `daduraL` state variable. I would think you could calculate the duration as needed and simply output it? —SS]

### 8.5.5 Local Functions

$\text{calculateB}: \mathbb{N} \rightarrow \mathbb{N}$

$\text{calculateB}(a) = 2 - a + (a / 4)$

$\text{MonthOneTwo}: \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$\text{MonthOneorTwo}(y, m, d) = 365.25 \times (y - 1) + 30.6001 \times (m + 13) + d - \text{calculateB}(y / 100) + 1720995$

$\text{MonthNotOneTwo}: \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$\text{MonthNotOneorTwo}(y, m, d) = 365.25 \times y + 30.6001 \times (m + 1) + d - \text{calculateB}(y / 100) + 1720995$

## 9 MIS of Input Parameters Module

### 9.1 Module

InputPara

### 9.2 Uses

HardH

Day ADT Module (Section 8)

### 9.3 Syntax

#### 9.3.1 Exported Constants

#### 9.3.2 Exported Access Programs

Name	In	Out	Exceptions
fromKeyBoard	-	-	Key_error
loadAnaleFile	string	-	File_error
getla	-	DegreeT	-
getph	-	real	-
getpw	-	real	-
getdayS	-	DayT	-
getdayE	-	DayT	-
getdec	-	a sequence of real	-

### 9.4 Semantics

#### 9.4.1 State Variables

latitude: DegreeT

dayS: DayT [You have these state variables already in other modules. —SS][Delete dayS and dayE in other module —Author]

dayE: DayT

panH: real

panW: real

declination: a sequence [366] of real

#### 9.4.2 Environment Variables

key: Input variables from keyboard.

#### 9.4.3 Assumptions

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

#### 9.4.4 Access Routine Semantics

fromKeyBoard ( ):

- transition: Get the values from users' input.  
latitude, panH, panW = HardH.  $\Phi_P$ , HardH.  $P_{A_h}$ , HardH.  $P_{A_w}$   
  
dayS := new DayT (HardH.  $year_{Start}$ , HardH.  $month_{Start}$ , HardH.  $day_{Start}$ )  
  
dayE := new DayT ( HardH.  $year_{End}$ , HardH.  $month_{End}$ , HardH.  $day_{End}$ )
- output:
- exception: If the data type of captured values do not match with the parameters' data type  $\Rightarrow$  Key\_error

loadAnaleFile ( fileName ):

- transition: declination [0..366] := read data from the file analemma.txt.  
The text file has the following format, where declination\_i denotes the angle of sun declination for day 1 to day 366. All data values are separate into rows, where each row has a value. There is 366 rows in the file.

declination\_0  
declination\_1  
declination\_2  
.  
.  
.  
declination\_366

- output:
- exception: If the file is not found  $\Rightarrow$  File\_error

getla ( ):

- transition:
- output: latitude
- exception:

getph ( ):

- transition:

- output: panH
- exception:

getpw ( ):

- transition:
- output: panW
- exception:

getdayS ( ):

- transition:
- output: dayS
- exception:

getdayE ( ):

- transition:
- output: dayE
- exception:

getdec ( ):

- transition:
- output: declination
- exception:

#### 9.4.5 Local Functions

## 10 MIS of Input Verification Module

### 10.1 Module

InputVer

### 10.2 Uses

InputPara<sup>9</sup>

### 10.3 Syntax

#### 10.3.1 Exported Constants

#### 10.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifiedLat	DegreeT	$\mathbb{B}$	-
verifiedP	real, real	$\mathbb{B}$	-
verifiedD	DayT, DayT,	$\mathbb{B}$	-

[Rather than simply rely on exceptions, you could have verify return a Boolean. —SS]

### 10.4 Semantics

#### 10.4.1 State Variables

#### 10.4.2 Environment Variables

# Display warning message on the screen  
screen: Hardware.screen

#### 10.4.3 Assumptions

The input values is input from InputPara

#### 10.4.4 Access Routine Semantics

verifiedLat (latitude):

- transition:
- output:  $(\text{latitude} > 90 \vee \text{latitude} < -90 \Rightarrow \text{screen.display} \text{ —“ The degree of the latitude can’t be greater than 90 or smaller than -90. ” } \mid \text{True})$
- exception:  $\text{exc} :=$

verifiedP (ph, pw):

- transition:
- output: ( ph < 0  $\vee$  pw < 0  $\Rightarrow$  screen.display —“ The height and the weight of the panel can’t be negative. ” | True)
- exception: exc :=

verifiedD (ds, de):

- transition:
- output: ( ds < de  $\Rightarrow$  screen.display —“ The end day can’t be smaller than the start day. ” | True)
- exception: exc :=

#### 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 Comparison"

#### 11.3.2 Exported Access Programs

Name	In	Out	Exceptions
addresult	a set of DegreeT, a set of real	-	-
setable	a sequece of a tuple of ( DegreeT, real, integer )	-	-
display	-	-	-

### 11.4 Semantics

#### 11.4.1 State Variables

mainTable: a set of { tuples of ( cutTime: natural number, energy: real) }

tiltTable: a set of { sets of { angle: DegreeT} }

#### 11.4.2 Environment Variables

screen: Write tables (output variables) in the file. [\[You should define an environment variable for the screen. —SS\]](#)

#### 11.4.3 Assumptions

#### 11.4.4 Access Routine Semantics

addresult ( angle, energy ):

- transition:  $\text{mainTable} = \text{mainTable} \cup \{ |\text{angle}|, \text{localAverage}(\text{energy}) \}$   
 $\text{tiltTable} = \text{tiltTable} \cup \{ \text{angle} \}$  [I don't know how to read this. —SS] [Change the expression —Author]
- output:
- exception:

display ( ):

- transition:
- output:  $\text{out} := \text{display}$  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  $\text{resultL.e}[0]$  and  $\text{resultL.e}[1..|\text{resultL}|]$  followed by a symbol %

Tilt Angle	$\text{resultL.t}[0]$	...	$\text{resultL.t}[ \text{resultL} ]$
Energy Absorption	$\text{resultL.e}[0]$	...	$\text{resultL.e}[ \text{resultL} ]$
Energy Competition	$\text{resultL.diff}[0]\%$	...	$\text{resultL.diff}[ \text{resultL} ]\%$

- exception:

#### 11.4.5 Local Functions

$\text{localAverage} : \text{a set of real} \rightarrow \text{real}$

$\text{localAverage}(\text{energy}) = + (i : \text{real} \mid i \in \text{energy} \bullet i) / |\text{energy}|$



## 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
getilt	DegreeT[ ], real	DegreeT	-
getiltInten	DegreeT, real	real	-
[accessProg —SS]	-	-	-

### 12.4 Semantics

#### 12.4.1 State Variables

tiltDegree: DegreeT

[Do you really need state variables? I think you could use an input-output module here. The same comment applies elsewhere. —SS]

[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] getilt (zenithL[ ], sunIn ):

- transition:
- output:  $\text{out} := \text{tiltDegree}$ , such that  
 $\forall(\text{zen}: \text{DegreeT} \mid j \in \text{zenithL} \bullet \text{tiltDegree} = \text{localMax}(\text{tiltDegree}, \text{zen}, \text{sunIn}))$  [This is a complicated expression. I think you aren't quite following our notation. Also, for a complicated expression like this, it helps to add local functions. —SS]
- exception:

getiltInten (degree, sunIn):

- transition:
- output:  $\text{Calculation.sglSunIn}(\text{degree}, \text{sunIn})$
- 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

# Get two tuples, then return a tuple.

localMax:  $\text{DegreeT} \times \text{DegreeT} \times \text{real} \rightarrow \text{DegreeT}$

localMax(origDegree, newDegree, sunIn) = ( getiltInten (origDegree, sunIn)  $\geq$  getiltInten (newDegree, sunIn)  $\Rightarrow$  origDegree | True  $\Rightarrow$  newDegree )

[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<sup>9</sup>

ZenithAng??

TiltAng<sup>12</sup>

### 13.3 Syntax

#### 13.3.1 Exported Constants

#### 13.3.2 Exported Access Programs

Name	In	Out	Exceptions
getenergy	real, real, real, DegreeT [ ]	a set of real	-

### 13.4 Semantics

#### 13.4.1 State Variables

energyL: a sequence of real

#### 13.4.2 Environment Variables

N/A

#### 13.4.3 Assumptions

#### 13.4.4 Access Routine Semantics

getenergy ( pw, ph, maxinten, zen[ ] ):

- transition:
- output: localEnergy ( localSunIn ( zen, maxinten ),pw, ph)
- exception:

### 13.4.5 Local Functions

localSunIn: a sequence of DegreeT  $\times$  real  $\rightarrow$  a set of real

localSunIn (zen[ ], maxinten) =  $\cup$  (i: DegreeT | i  $\in$  z  $\bullet$  { SunInten.single(maxinten, i) })

localEnergy: a set of real  $\times$  real  $\times$  real  $\rightarrow$  a sequence of real

localEnergy (sunIn, pw, ph) =  $\cup$  ( i: integer | i  $\in$  sunIn  $\bullet$  { pw  $\times$  ph  $\times$  18.7  $\times$  0.75  $\times$  i }t)

## 14 MIS of Calculation

### 14.1 Module

Calculation

### 14.2 Uses

### 14.3 Syntax

#### 14.3.1 Exported Constants

$I_S := 1.35$

#### 14.3.2 Exported Access Programs

Name	In	Out	Exceptions
getzen	DegreeT[ ], integer, integer	-	-
sumSunIn	DegreeT[ ], real	real	-
sglSunIn	DegreeT, real	real	-

### 14.4 Semantics

#### 14.4.1 State Variables

#### 14.4.2 Environment Variables

#### 14.4.3 Assumptions

#### 14.4.4 Access Routine Semantics

getzen(zenList[ ], i , diff):

- transition:
- output: zenList[i.. i+diff]
- exception:

sumSunIn ( zenList[ ], energy ):

- output: out := + (zen: DegreeT | zen ∈ zenList •

$$I_S \cdot \left( \frac{1.00}{\text{energy}} \right)^{\text{sec}(\text{zen})}$$

- exception:  $\text{exc} := |\text{zenList}| = 0 \Rightarrow \text{sequence\_empty}$

`sglSunIn (zen, energy):`

- output: `out :=`

$$I_S \cdot \left( \frac{1.00}{\text{energy}} \right)^{\text{sec}(\text{zen})}$$

- exception:

#### 14.4.5 Local Functions

[Your MIS seems more complicated than it has to be. The purpose of all the state variables is not clear. I suggest you review all of the modules to see if they are necessary and whether you really need the state variables. You also probably do not need so many modules. You could combine the modules that export calculations into a “Calculation” module. Also, you might want to explain what is going on in words, in case your math doesn’t say what you think it is saying. —SS]

## 15 Appendix

[Extra information if required —SS]