The Anatomy of ESATAN and ESARAD Thermal Model Files

Kan Yang

NASA Goddard Space Flight Center

Objectives

Goal: To facilitate conversion between ESA-based thermal model files and Thermal Desktop in interagency collaborations

(i.e. what if you got a file from a vendor or agency that used ESATAN? How would you go about converting it?)

This short course will:

➔ Provide a basic overview of ESATAN syntax

➔ Provide equivalent statements in Thermal Desktop/TSS* and SINDA to aid in the translation from ESATAN format

*some commonly used thermal modeling tools

Who is this presentation for?

➔ People who are familiar with Thermal Desktop/TSS and SINDA but unfamiliar with ESATAN

All equivalent Thermal Desktop statements to ESATAN-TMS/ESARAD will be presented in green boxes

All equivalent TSS statements to ESATAN-TMS/ESARAD will be presented in red boxes

All equivalent SINDA statements to the ESATAN thermal file will be presented in blue boxes

Overview

- ESARAD language (GMM)

  Geometric Math Model to define the surfaces, properties, nodal distributions for radiation modeling and to calculate the radiation couplings

- ESATAN Thermal model (TMM)

  Thermal Math Model to define the node data, heat sources, conduction couplings, and solver control variables for finite difference solution routines

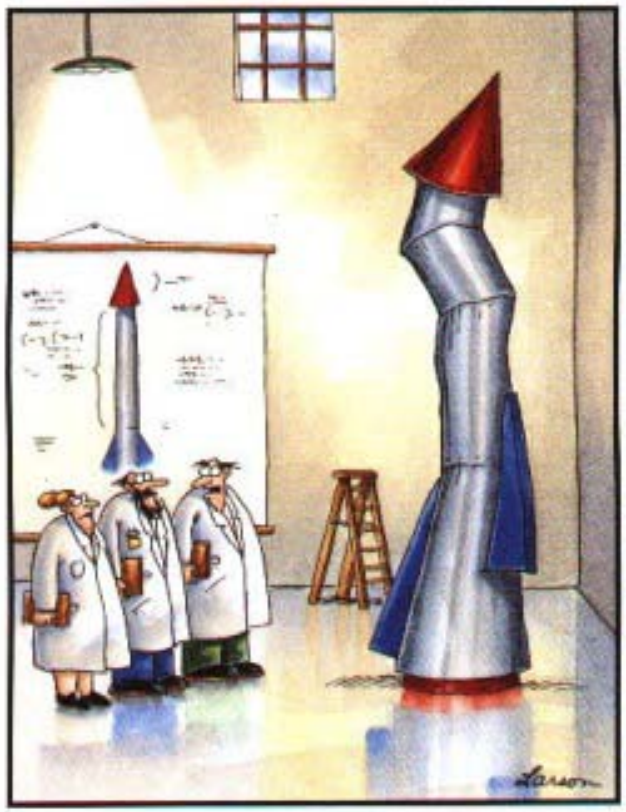
- ESATAN-TMS Workbench GUI (demonstration)

  Graphics interface which outputs in ESARAD language; processes geometry and creates thermal inputs

  NOTE: this is a later addition. Initially, there were only ESARAD and ESATAN files. ESATAN-TMS captures both in a single user environment.

Disclaimer

Presenter is by no means an expert at ESATAN/ESARAD…



"It's time we face reality, my friends. ...
We're not exactly rocket scientists."

What is ESATAN?

- European Space Agency (ESA) –directed software package developed by ITP Engines UK Ltd. under ESA Contract
  - Developed as a standard set of tools that were controllable and modifiable by ESA to address its specific modeling needs (alternative to SINDA, but has many similar properties to SINDA)
- Major features of ESATAN include:
  - Modular structure: input enables model to be described as the sum of many sub-models (each sub-model is a complete model separately suitable for analysis)
  - Easy syntax for combining submodels (node merging or inter-model conductances)
  - Implicit submodel definitions by including previously-defined submodels
  - Mortran language to enable Fortran-like commands with model data or operations

Equivalent Files

Function	ESATAN	Thermal Desktop/SINDA	TSS/SINDA
Graphical User Interface (GUI) / GMM Generator	ESATAN-TMS Workbench (.erg)	Autocad drawing file for Thermal Desktop (.dwg)	SPACE3D / TSS .tssma, .tssgm, .tssop files
Radiation (Radk) Calculations	ESARAD .erk: instructions to execute ray trace ➔ Radk files	RadCAD ➔ TD Case set manager radiation analysis tab ➔ .rdk or .k files	TSS ➔ Ray trace produces .rk folder and .hr folder
Thermal Analysis (TMM)	ESATAN Thermal file (.d)	SINDA .inp and .cc files	SINDA .inp files

Warning! Lots of (relatively) dry material ahead…

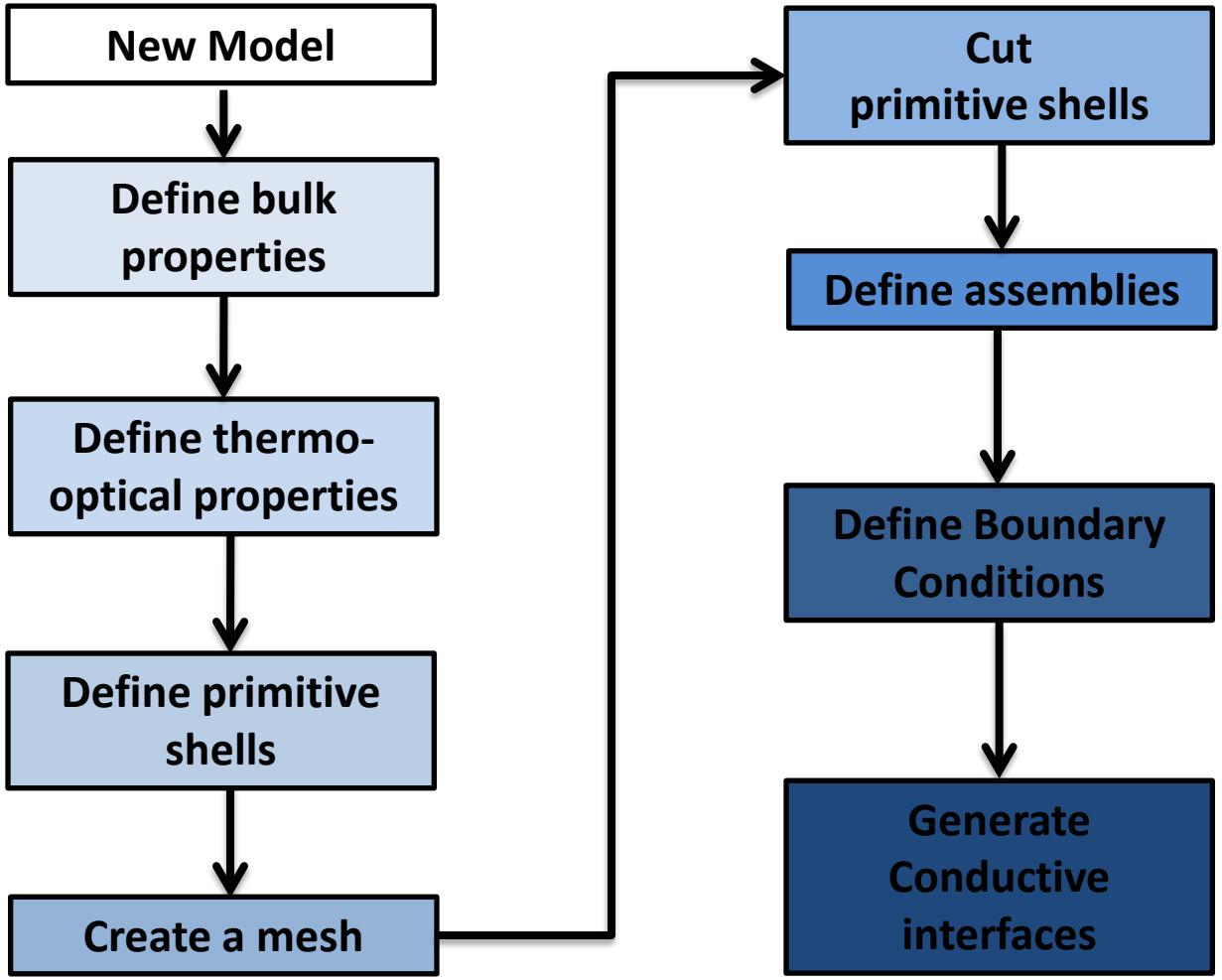
I will try to make the discussion about syntax as painless as possible…

… but you may be excused if your brain feels full

"Mr. Osborne, may I be excused? My brain is full."

ESARAD Language

ESARAD Geometric modeling process



New Model

- The ESARAD language is used to generate the geometry (GMM) and to operate the programs / modules

  ➔ESATAN-TMS GUI can be used to generate Workbench language statements (records all commands executed from GUI in .log files)

- What does a typical ESARAD file look like?
  - Text file with similar syntax to C/C++
  - Comments are denoted with /*…*/ or #
  - All lines must end with ;

New Model

- Basic model construct (stored in .erg esarad geometry file):

  BEGIN_MODEL *modelname*

  (other language statements)

  END_MODEL

GMM defined in Thermal Desktop .dwg file

GMM defined in TSS .tssgm file

Define bulk properties

- ESARAD allows definition of the following bulk properties:
  - Density
  - Specific Heat
  - Thermal conductivity
- ESARAD syntax:

  BULK *propertyname*;

  propertyname = [*density*, *specific heat*, *conductivity*];

Bulk properties can be redefined in the Thermal Desktop "Edit Material Property Data" window

Bulk properties in TSS .tssma file:

material *materialname*

color =
density =
units =
spec_heat_table =
absorb_coef =
units =
.
.

Define Thermo-optical properties

- Optical properties are defined with the following data type:

  OPTICAL *name*;

  name = [*IR emissivity*, *IR reflectivity*, *IR transmissivity*,
  solar absorptivity, *solar reflectivity*,
  solar transmissivity, *IR specular reflectivity*,
  solar specular reflectivity]

- Example:

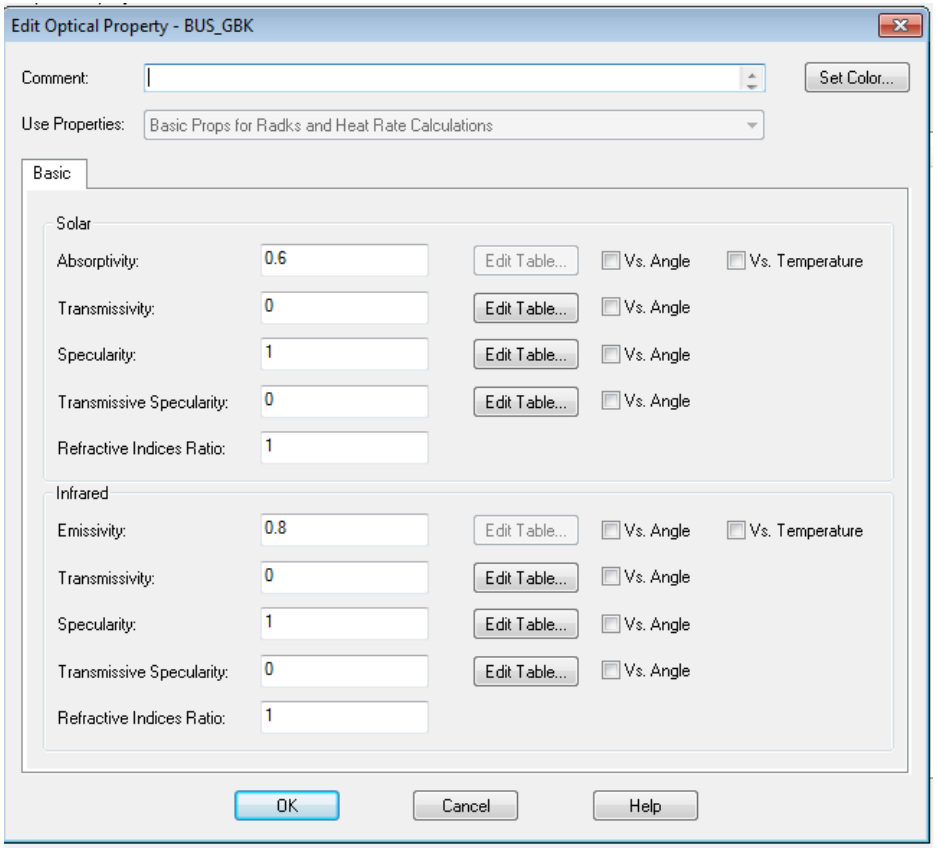
  OPTICAL OPTVAR1;

  OPTVAR1 = [0.6,0.2,0.1,0.1,0.5,0.2,0.1,0.2];

  NOTE: specular reflectivity and diffuse reflectivity defined in values, not percentages (like TSS or Thermal Desktop)

Define Thermo-optical properties

Bulk properties can be redefined in the Thermal Desktop “Edit Optical Property Data” window



Define Thermo-optical properties

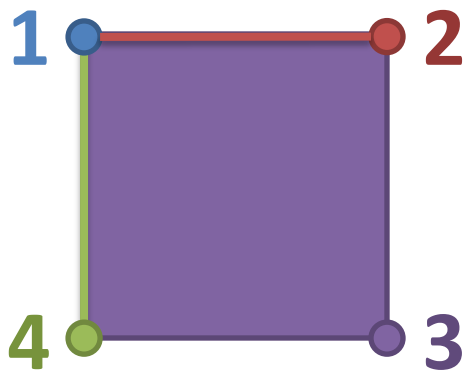
Optical properties in TSS .tssop file:

property *opticalpropertyname*

ir_eps =
ir_trans =
ir_spec =
ir_tspec =
ir_refract =
sol_eps =
sol_trans =
sol_spec =
sol_tspec =
sol_refract =
color =

Define primitive shells

- Define by points:



Defined with Right-handed coord. system:

1: Top/out side
2: Bottom/in side

```
SHELL RECT_1;
RECT_1 = SHELL RECTANGLE (
point1 = [-2.0, -1.0, 0.0],
point2 = [-1.0, -1.0, 0.0],
point4 = [-2.0, -2.0, 0.0],
thick = 0.0;
opt1 = GOLD_BOL,
opt2 = BLACKBODY,
nodes1 = 1,
nodes2 = 1,
nbase1 = 1,
nbase2 = 18,
label = "AVIONICS_12",
side1 = "ACTIVE",
side2 = "INACTIVE"
);
```

Optical properties

Node subdivision

Starting node no.

Surface name

If surface is active/inactive in radiation calcuations

- Other shell types: triangle, triangular prism, sphere, quadrilateral, paraboloid, cylinder, box, cone, disk

Define primitive shells

- For rectangles, the third point is placed in-plane in accordance with other three points; if the user-defined fourth point is not orthogonal to the first two points, ESARAD moves the point

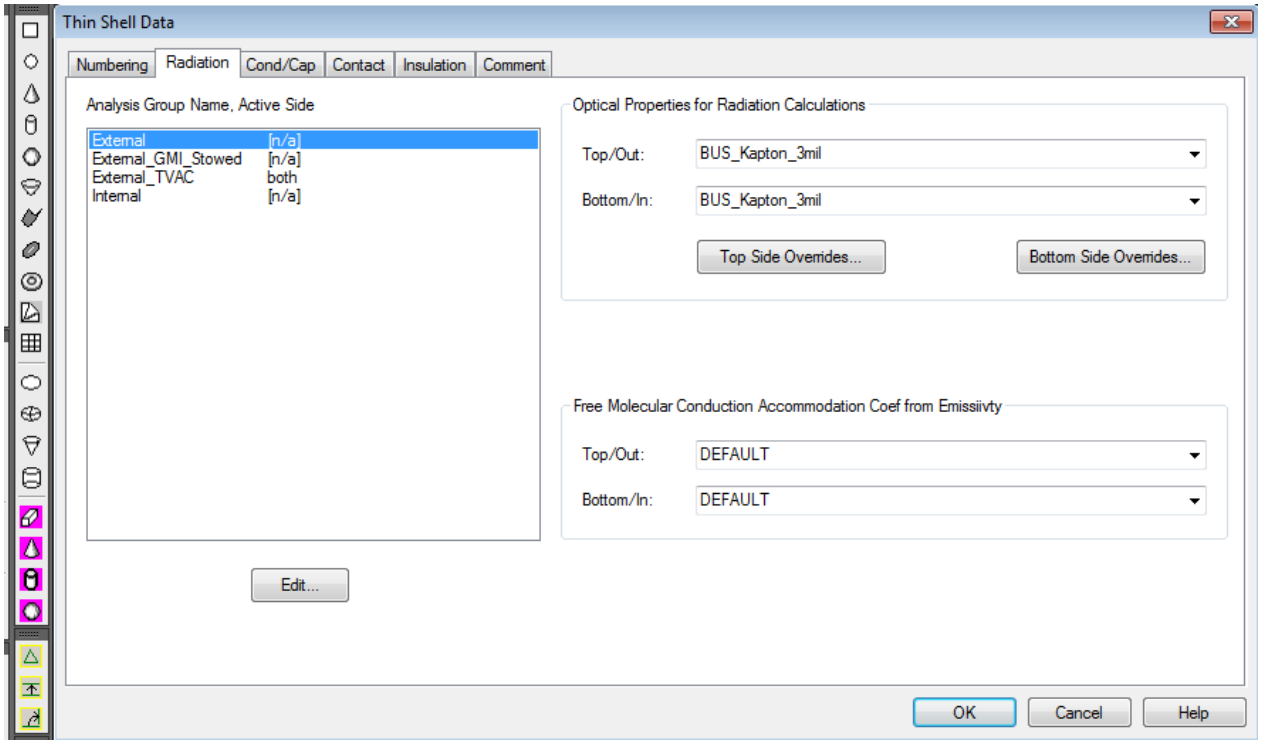
- For quadrilateral, if the fourth point is not in the plane formed by the first three points, ESARAD projects the fourth point onto the plane (model check) and the position of the projected point is used for the calculation

- Can also define shells by parameters (specify dimensions of surface first, then translate/rotate to correct location) or by origin/direction

Define primitive shells

Surfaces can be defined and assigned properties in the Thermal Desktop .dwg file

Examples of surfaces supported in Desktop: Rectangle, Disk, Cone, Cylinder, Sphere, Paraboloid, Parabolic Trough, Torus, Polygon, Ellipsoid, Elliptic Cone, Elliptic Cylinder, Brick, Solid Cone, Solid Sphere, Solid Cylinder



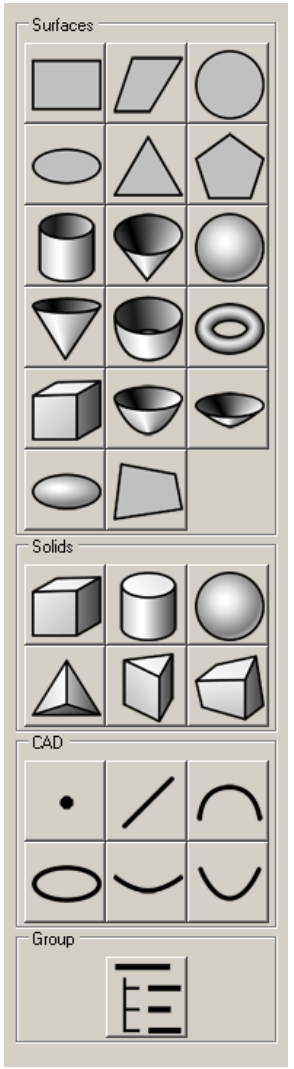
Define primitive shells

Surfaces in TSS .tssgm file:

Surfacetype surface*name*

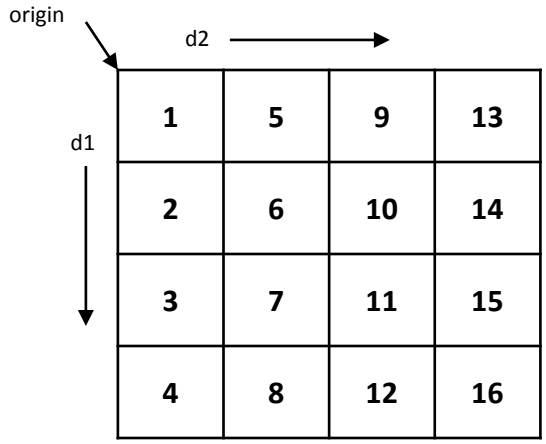
units =
param =
active = sides = submodel =
node_ids =
iconductor =
nbeta =
ngamma =
rot1 = rot2 = rot3 = tx = ty = tz =
optics =
optics_angles =
material = thickness =
color =
initial_temp =

Examples of surfaces supported in TSS: Rectangle, Disc, Cone, Cylinder, Polygon, Triangle, Quadrilateral, Sphere, Elliptic Cone, Brick, Solid Cylinder



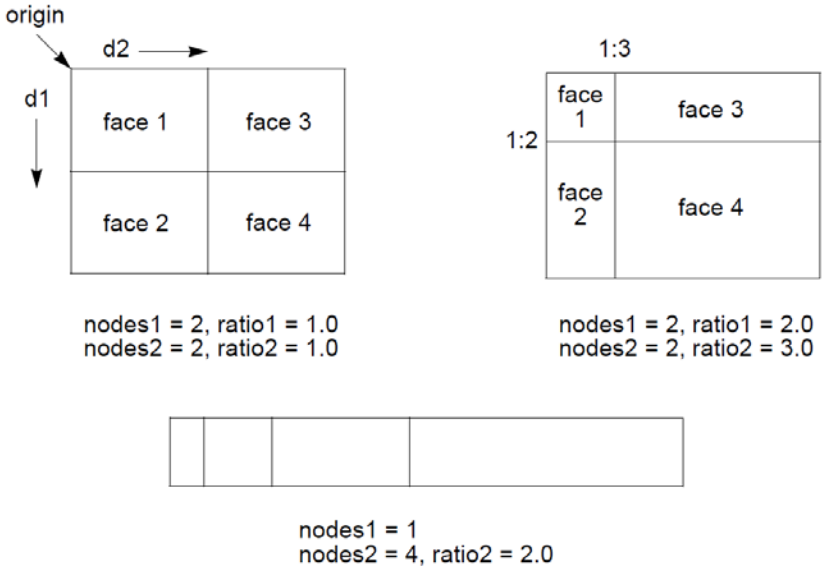
Create a Mesh

- Within the primitive shell definition, there are two different variables for specifying the nodal division on a surface:



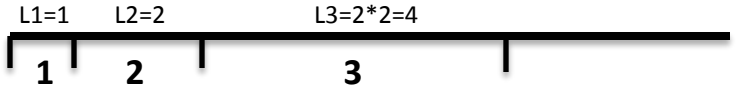
node1, node2: determines the number of nodes on each side of the surface

Note: nodes increment from the origin towards the first defined direction, then loop back and increments in direction 2



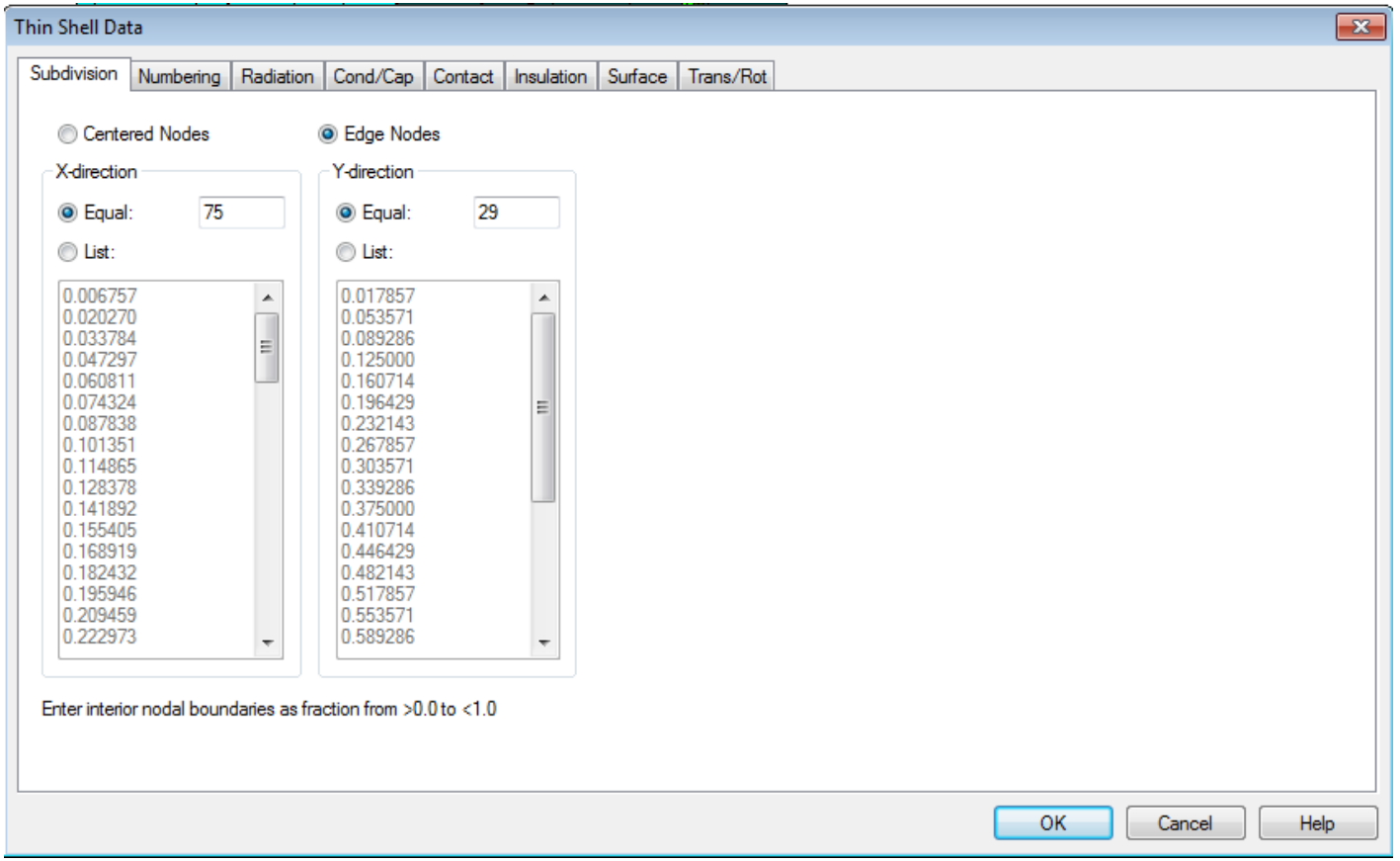
ratio1, ratio2: determines the change in spacing between the current node and the next node in sequence.

Example: for ratio2 = 2, on side 2…



Create a Mesh

Nodal subdivisions defined in Thermal Desktop using the “subdivision” tab when editing surface properties.



Create a Mesh

Nodal subdivisions can be defined in the Space3D GUI, and then incorporated into the surface definition in the TSS .tssgm file:

Surfacetype surface*name*

.

.

param =

.

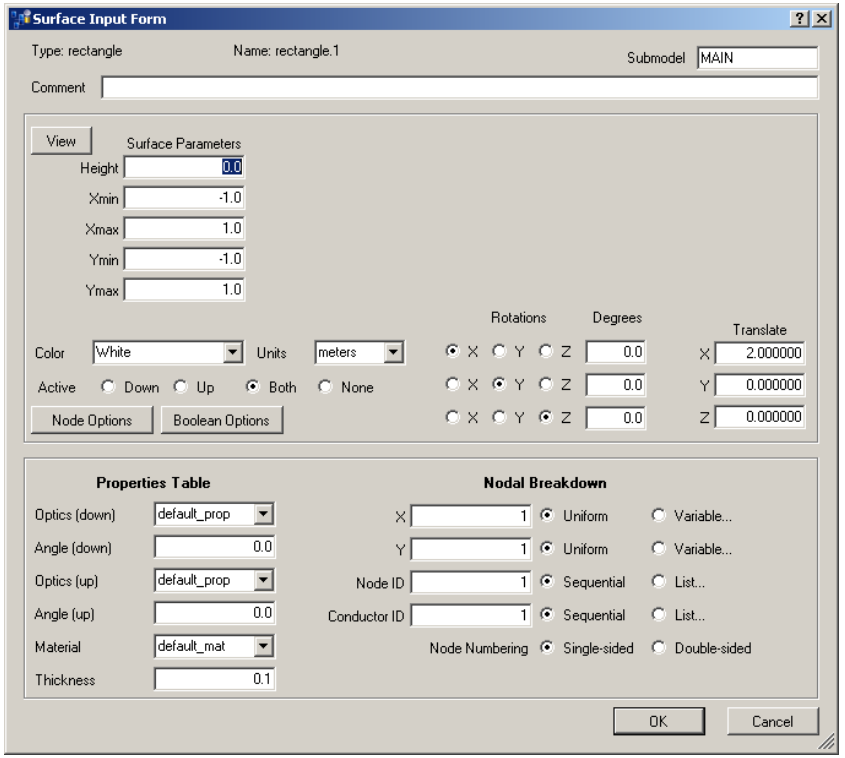
node_ids = or **initial_id =**

nbeta = division of nodes in surface's x-direction

ngamma = division of nodes in surface's y-direction

ntheta = division of nodes in surface's z-direction

.



Create a Mesh

Nodal subdivision methods supported by each program:

Subdivision Method	ESATAN	Thermal Desktop	TSS
Centroid Nodes	Supported	Supported	Supported
Edge Nodes	Not supported	Supported	Not supported
Finite Element Nodes	Supported**	Supported	Supported
Ratio Function	Supported	Not supported*	Not supported*

* Though there is no “one stop shop” ratio function in Thermal Desktop and TSS:
  - Thermal Desktop allows the user to specify nodal boundaries
  - TSS allows the user to specify node locations

  (neither of these are allowed in ESARAD)

** ESARAD does not allow the user to specify node numbering for Finite Element nodes

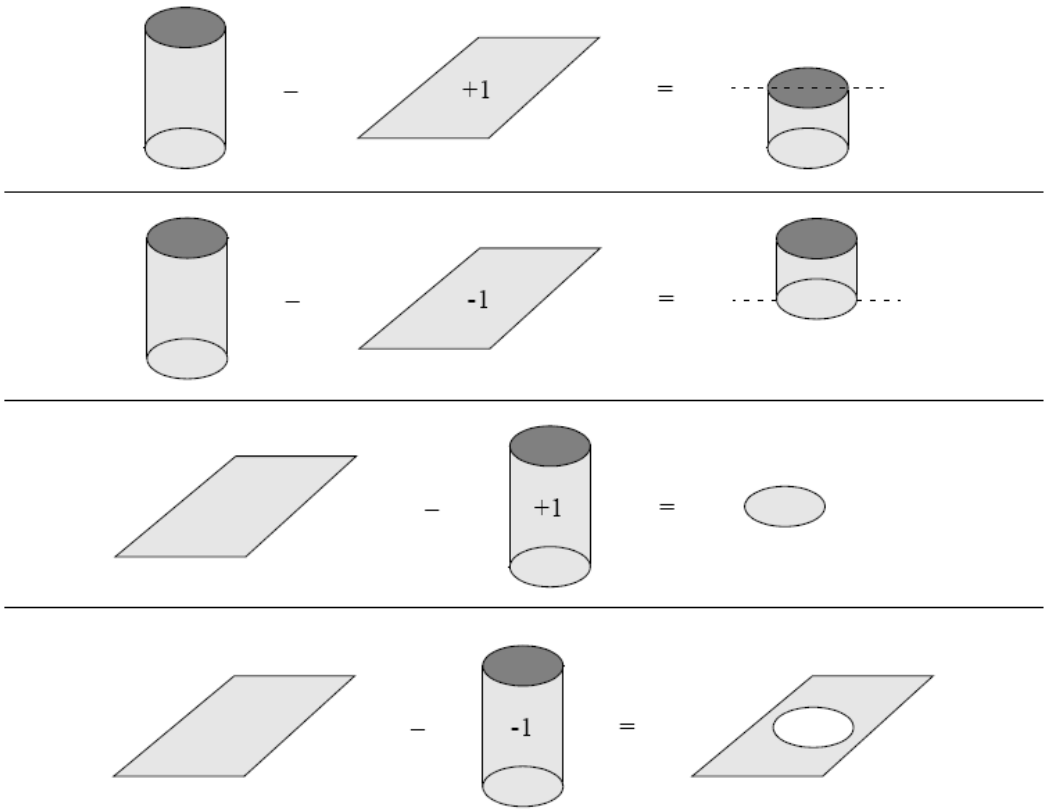
Cut primitive shells

- Sense is an ESATAN-specific “cutting tool” ➔ geometric shells can be used to “cut” into previously defined surfaces

Two values for Sense:

- +1 (INSIDE):
  Cutting tool will cut away any surfaces which are *not* enclosed by its volume

- -1 (OUTSIDE):
  Cutting tool will cut away any surfaces which are enclosed by its volume



Cut primitive shells

- ESARAD Syntax:
  - Sense variable is included in the surface definition

```
SHELL Cutshell1;
Cutshell1 = SHELL_CYLINDER(
point1 = [-5.0, -5.0, -1.0],
point2 = [-5.0, -5.0, 1.0],
point3 = [-3.0, -5.0, -1.0],
sense = -1,
```

To cut shells in ESATAN: TEST1 = Shell1 - Cutshell1;

Target Shell **Shell to be cut** **Cut geometry**

Thermal Desktop does not support cutting operations

TSS allows for Boolean geometry with radiative surfaces, not conductive surfaces

Define Assemblies

- Assemblies are groups of shell surfaces or groups of assemblies
- They are defined with the following construct:

  SHELL *assemblyname*;

  assemblyname = *shellname_1* + *shellname_2* + … + *shellname_N;*

- Examples:

  SHELL BATTERY;

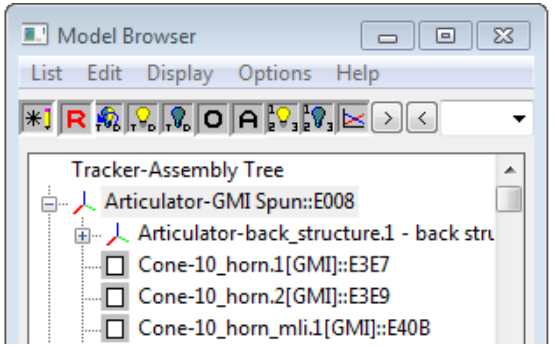
  BATTERY = BATTERYCELL_1 + BATTERYCELL_2 + BATTERYCELL_3;

  SHELL AVONICS;

  AVONICS = BATTERY + POWERSYS + CDH;

Define Assemblies

Thermal Desktop GUI allows for surfaces to be placed into assemblies and trackers (Tracker-Assembly Tree in Model Browser shown)



Assembly declarations are built into the TSS .tssgm file hierarchy (surfaces are defined under sets of assemblies)

Assembly *assemblyname*
units =
mirror =
rot1 = rot2 = rot3 = tx = ty = tz =
Surfacetype surface*name1*
(surface properties)
.
.
Surfacetype surface*name2*
(surface properties)
.
.

Define Boundary Conditions

- ESARAD allows you to define the following types of boundary conditions:

Type	Function
Initial Temperature	Initial temperature is set for the node
Temperature	Node is set to a boundary node with given temperature
Heat Load/Unit Area	Heat load value assigned to a surface is the given value multiplied by that surface's area
Heat Load/Face	Given heat load is assigned to a surface
Total Heat Load	Heat load applied is divided evenly among nodes on that surface

Define Boundary Conditions

- ESARAD syntax:

BOUNDARY_CONDITION *boundary condition name*;
boundary condition name = *boundary condition type*(
reference = "*shell or node name*",
value = *boundary condition value*);

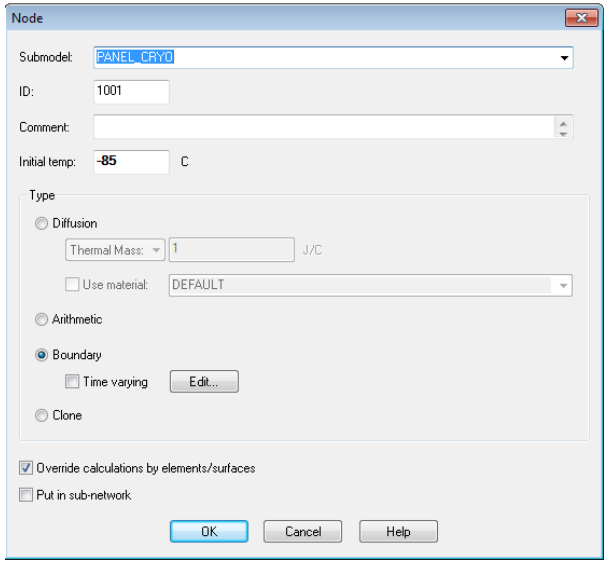
Example:

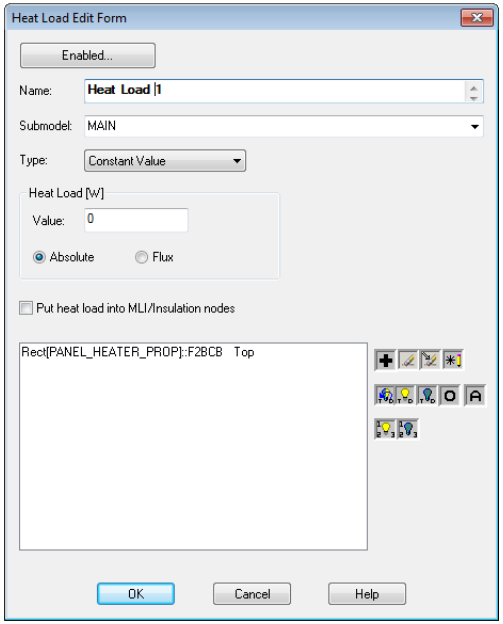
BOUNDARY_CONDITION BOUND1;
BOUND1 = TOTAL_HEAT_LOAD(
reference = "Shell1",
value = 15.0);

Define Boundary Conditions

In Thermal Desktop:

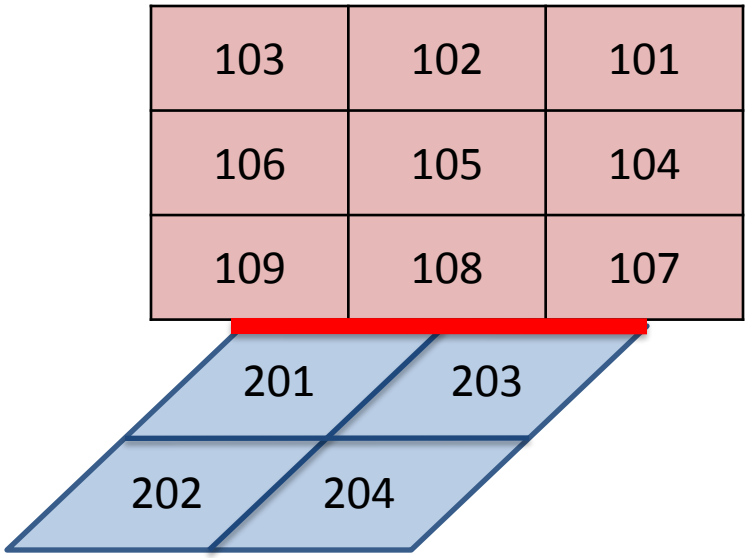
- Setting the initial temperature or boundary temperature on a surface sets all of the nodes that surface to the specified value
- Heat loads / unit area or total heat loads may be applied to surfaces or individual nodes





Boundary temperatures and heat loads for surfaces cannot be defined in TSS (as far as I know)

Generate Conductive interfaces



- For any two shell surfaces that share a common edge/overlap, ESARAD can generate linear conductors (using automatic conductor generation) for pairs of nodes at that interface
  Ex: (109,201), (108,201), (108,203), (107,203)

Generate Conductive interfaces

- Types of conductive interfaces:

Designation	Definition
FUSED	Two shells form a continuous surface (no thermal resistance across interface)
CONTACT	Two shells have physical interface with contact conductance (thermal resistance is non-zero)
NOT_PROCESSED	Automatically generated conductive interface, type is not yet specified
NOT_REQUIRED	Automatically detected conductive interface, but should not be used to generate shell conductances
NOT_CONNECTED	Defines that two shells are not connected on an interface

Generate Conductive interfaces

- ESARAD uses the following rules in allowing automatic conductor generation to:

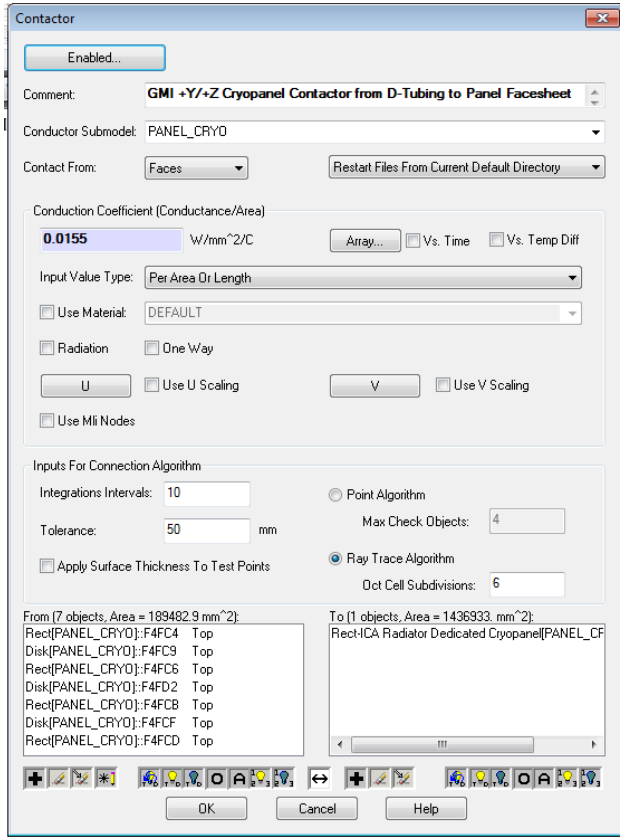
1. Calculate linear conductors within the same shell (intra-shell):
   - Conductors are generated based on the bulk properties and thickness specified for that side of the shell

2. Calculate linear conductors between two different shells (inter-shell):
   - When two shells are connected, the nodes on one side of the first shell are matched to the one side of the second shell based on orientation (even if the thicknesses do not match)
   - For a contact conductor, the thickness of the contact is the minimum of the thickness of the two nodes

3. Calculate linear conductors through the thickness of the same shell (if node pattern is different on either side of the shell):
   - Two sets of conductors calculated, one per side; through-thickness conductors can then be calculated

Generate Conductive interfaces

Similar to contactors
in Thermal Desktop



No equivalent in TSS
(as far as I know)

Final Model Build

ONLY on top level model:

MODEL *modelname* = *shell1*+ *shell2* + … + *shellN*

PURGE_MODEL (); **Get rid of unused surfaces**

END_MODEL

Summary of ESATAN-TMS equivalent capabilities vs. Thermal Desktop / TSS

Capability	ESATAN	Thermal Desktop	TSS
GMM generation	ESATAN-TMS GUI	AutoCAD .dwg	Space3D
Bulk property definition	supported	supported	supported (.tssma)
Optical property definition	supported	supported	supported (.tssop)
Shell surface definition	supported	supported	supported (.tssgm)
Generic meshing	supported	supported	supported
Mesh cutting operations	supported (sense)	Not supported	Supported (boolean)
Defining assemblies	supported	≈supported	supported
Surface boundary conditions/ heat loads	supported	supported	Not supported
Generating conductive interfaces between surf.	supported (automatic GL generation)	≈supported (contactors)	Not supported

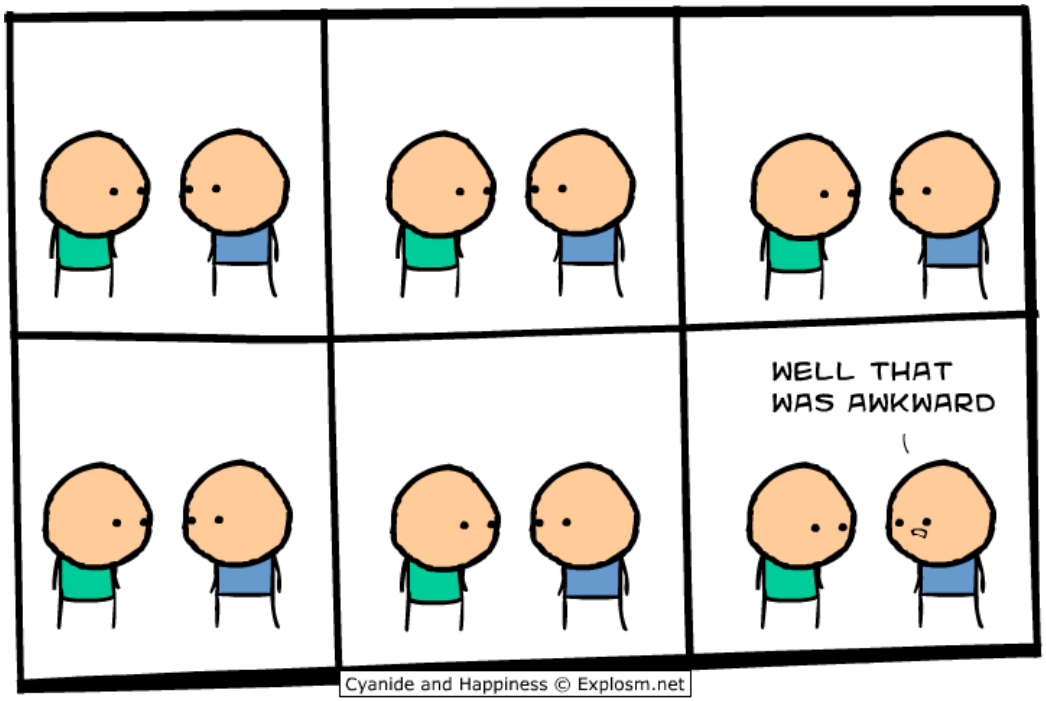
Once the model geometry is redefined in Thermal Desktop or TSS …

… RadCAD or TSS Radk can be used to calculate the radiation couplings and replace the radks generated by the ESARAD file

But what about the ESATAN thermal file?

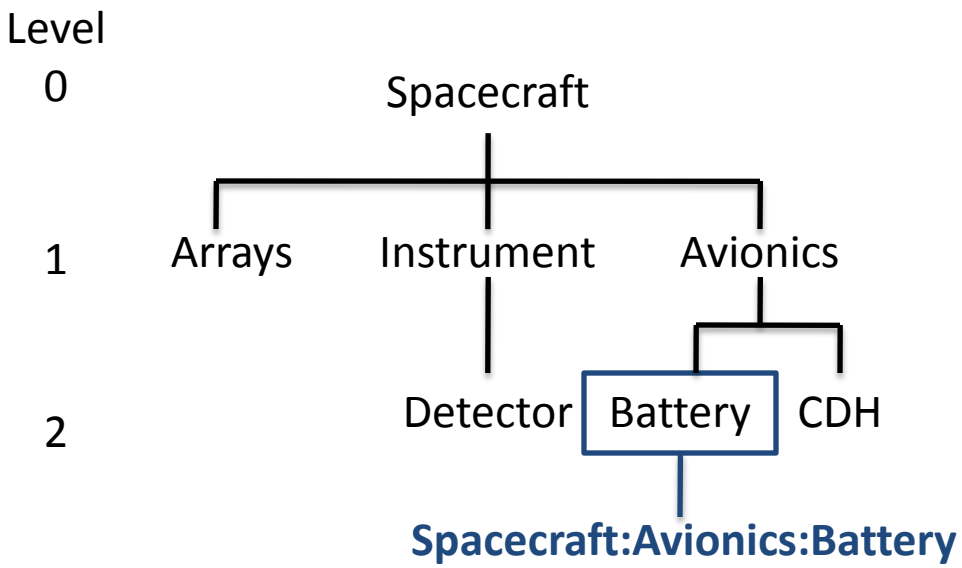
… This can be translated to SINDA syntax.

ESATAN Thermal File



ESATAN Model Structure

- Any ESATAN model may contain submodels; those submodels may contain submodels also (down to 99 submodel levels)
  - Each submodel may be separately analyzed as a standalone model




```
$MODEL SPACECRAFT
    $MODEL ARRAYS
    $ENDMODEL ARRAYS
    $MODEL INSTRUMENT
        $MODEL DETECTOR
        $ENDMODEL DETECTOR
    $ENDMODEL INSTRUMENT
    $MODEL AVIONICS
        $MODEL BATTERY
        $ENDMODEL BATTERY
        $MODEL CDH
        $ENDMODEL CDH
    $ENDMODEL AVIONICS
$ENDMODEL SPACECRAFT
```

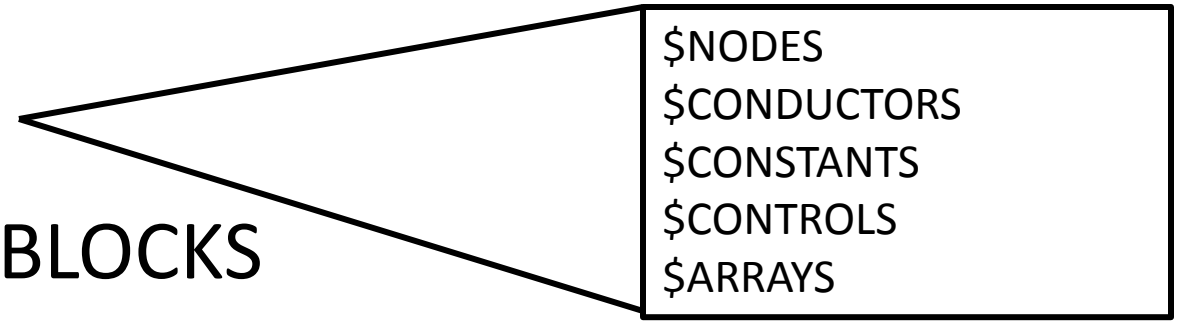
Basic ESATAN Thermal File Construct

$MODEL

DATA BLOCKS

OPERATIONS BLOCKS

$ENDMODEL



SINDA Equivalent:

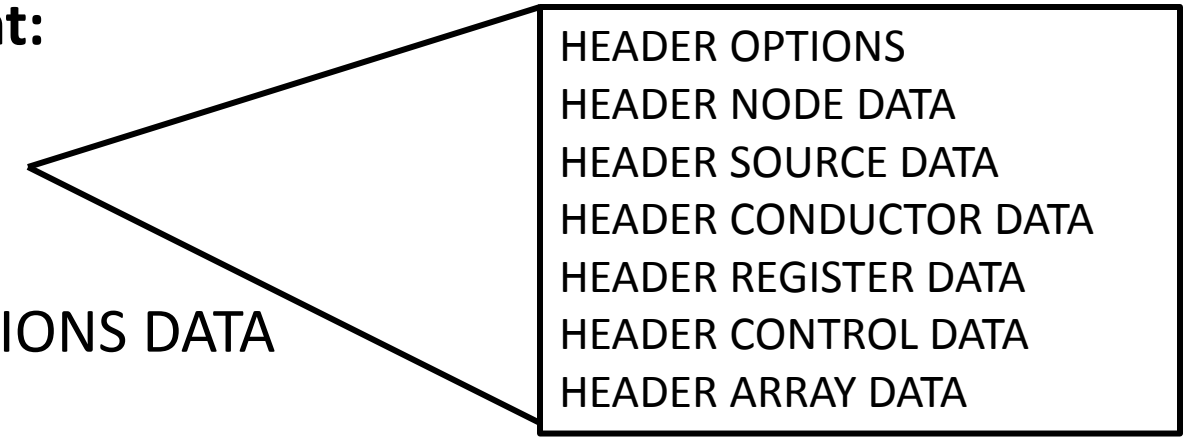
SINDA .cc file:

(DATA BLOCKS)

SINDA .inp file:

HEADER OPERATIONS DATA

(LOGIC BLOCKS)



ESATAN Syntax

Basic notation:

Symbol	Definition	Note	SINDA Equivalent
$	Defines a new block	i.e. declare a submodel, etc.	HEADER
#	Comment	Like Unix syntax	C, $
;	End of line	Like C/C++ syntax	(none)
:	Divider for entity definitions	Ex: T:SUBMODEL:10	.

Node Definitions

Node type *node #* = **'name', T** = temp, **C** = cap, **A** = area, **ALP** = α, **EPS** = ε, ….;

Node Types:

D: diffusion node

- an arithmetic node is defined as diffusion with capacitance = 0

(there is no separate arithmetic node definition in ESATAN)

B: boundary node

X: inactive node (all connected entities are ignored)

Node Definitions

Node type *node #* = **'name'**, **T** = temp, **C** = cap, **A** = area, **ALP** = α, **EPS** = ε, ….;

Node #: an integer

'name': a string describing the node, i.e. 'nozzle'

Node Definitions

Node type *node #* **= 'name', T = temp, C = cap, A = area, ALP = α, EPS = ε, ….;**

Designation	Definition	Type
T	Initial temperature of the node	Real
C	Capacitance of the node	Real/Expression*
A	Area of the node	Real
ALP	Absorptivity	Real
EPS	Emissivity	Real

NOTE: "Real" declarations in ESATAN prefer "d" syntax to declare double precision variables, e.g. 2.0D0 (double precision 2.0)

* An equation can be used to define capacitance based on temp. or other variable

Node Definitions

Node type *node #* **= 'name', T = temp, C = cap, A = area, ALP = α, EPS = ε, ….;**

Designation	Definition	Type
QE	Total heat to node from Earth IR	Real
QI	Total impressed heat to node (internal)	Real
QR	Total residual ("other") heat to node	Real
QA	Total heat to node from Albedo (Reflected solar)	Real
QS	Total heat to node from Solar	Real

NOTE: Q in SINDA = QA+QS+QI+QR+QE in ESATAN
THRMST can be used to incorporate heater logic into ESATAN

Node Definitions

Node type *node #* **= 'name',** **T** = temp, **C** = cap, **A** = area, **ALP** = α, **EPS** = ε, ….;

Designation	Definition	Type
QEI	Incident albedo heat source	Real
QAI	Incident Earth IR heat source	Real
QSI	Incident solar heat source	Real
FX	Node location in X Cartesian coordinate	Real
FY	Node location in Y Cartesian coordinate	Real
FZ	Node location in Z Cartesian coordinate	Real

Node Definitions

Node type *node #* **= 'name',** **T** = temp, **C** = cap, **A** = area, **ALP** = α, **EPS** = ε, ….;

Examples:
D5001 = 'Top radiator', T = 22.0, C = 250.1;
X9012 = 'MLI', T = 53.1, C = 0.0, A = 0.05, ALP = 0.4, EPS = 0.79;

SINDA Equivalent:
HEADER NODE DATA, *submodel name*
node #, initial temp, capacitance

NOTE: There are no equivalents in ESATAN to SINDA GEN, SIV, SIM, etc. statements

Conductors

Conductor type (*node1*, *node2*) = value;

Designation	Definition
GL	Linear Conductor (heat flows either way)
GR	Radiative Conductor

Node does not need node type designation in front

Value GL in heat/temp (e.g. W/K), for GR in heat/temp^4 (e.g. W/K^4)

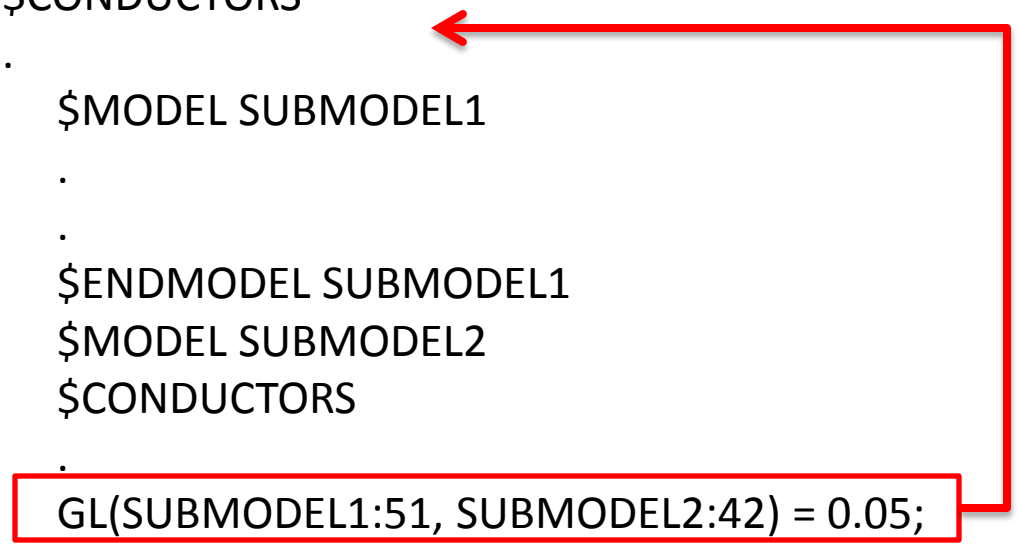
Example: GL(5001,5002) = 1.413E-02;

Conductors

Conductor type (*node1*, *node2*) = value;

Note: ESATAN will **NOT** allow definition of a conductor between two submodels within the current submodel's conductors block *if both submodels are on the same level*

```
$MODEL MAIN
$CONDUCTORS
.
    $MODEL SUBMODEL1
    .
    .
    $ENDMODEL SUBMODEL1
    $MODEL SUBMODEL2
    $CONDUCTORS
    .
    GL(SUBMODEL1:51, SUBMODEL2:42) = 0.05;
```



Conductors

- If more than one conductor between two nodes is desired, the following syntax can be used:

  Conductor type (*node1*, *node2*, *n3*) = value;

  where *n3* is the sequence number for the conductor

- Conductors can be defined with a function for conductance value:

  GL(3,4) = AL_CONDUCTIVITY*0.2/0.25;

  can be updated with temperature:

  GL(3,4) = T3*COND_VAL;

  or can be interpolated:

  GL(3,4) = 0.04* INTRP1((T2+T4)/2.0, Conductivity, 1);

NOTE: Radiative conductor to space MUST be placed in the top-level model

Conductors

SINDA Equivalent:

HEADER CONDUCTOR DATA, *submodel name*
conductor#, node 1, node 2, conductance

ESATAN	SINDA	Definition
GL	(*positive conductor number*)	Linear Conductor
GR	(*negative conductor number*)	Radiative Conductor

NOTE: There are no conductor numbers in ESATAN. To refer to a conductor, can use the syntax:

SUBMODELA:SUBMODELB:GL(3,4)

Constants

- $CONSTANTS defines constants block; multiple blocks may be used in one submodel
  - Can define type blocks $REAL, $INTEGER, or $CHARACTER
  - TYPE**name* = *value;* defines individual type

- Constant names up to 18 characters, first character must be alphabetic (SINDA allows 32 characters)

- User constants can be referenced and/or reassigned in all operations blocks, but only referenced in $NODES and $CONDUCTORS
  - In operations, value of user constant can be changed during solution run (e.g. change boundary temp. value of node)

Constants

- Examples of constant definitions:

```
$CONSTANTS
#
INTEGER*NODE_COUNT =  99; REAL*NODE_TEMP = 15.7;
CHARACETER*ZPROCESS = ‘HELLO’;
#
$REAL
AZ23 = 19.3E-06; UB_AST = 23.52; flux = INTRP1(TIMEM, Qarray, 1);
#
$CHARACTER
JTC = ‘TIME STEP’;
```

NOTE: SINDA allows global constants to be defined in the HEADER REGISTER DATA; in ESATAN, *all* constants only local to the submodel they are contained in (even constants in top model)

Constants need to be passed between submodels to allow reference

Control Constants

- Control constants can be defined in $CONTROL block
  - Global: top-level model definition only
  - Local: defined in submodel
- Examples of control constants:

ESATAN	PURPOSE	SINDA Equivalent
DTIMEI	Input time step	DTIMEI
DTIMEU	Time step used	DTIMEU
DTMAX	Maximum time step	DTIMEH
DTMIN	Minimum time step	DTIMEL
LOOPCT	Number of solution iterations	LOOPCT
NLOOP	Maximum allowable number of iterations	NLOOPS, NLOOPT
TIMEN	Time at end of time step (“current” step)	TIMEN
TIMEO	Time at start of time step	TIMEO
TIMEND	Time at end of solution	TIMEND
STEFAN	Stefan-Boltzmann Constant	SIGMA

Control Constants

- Examples of more control constants:

ESATAN	PURPOSE	SINDA Equivalent
CSGMIN	Smallest characteristic nodal time constant (bound on time step size to prevent instability in model)	CSGMIN
DAMPT	Temperature Damping	DAMPD
RELXCA	Temperature Convergence Criterion	DRLXCA
RELXCC	Calculated Temperature Convergence	DRLXCC
DTPMAX	Maximum allowable temperature change over a time step	DTMPCA
ENBALA	Absolute energy balance	ENGBAL
NCSGMN	Node of CSGMIN	NCSGM
OUTINT	Output interval	OUTPUT

Arrays

- $ARRAYS defines arrays block; multiple array blocks allowed per submodel, naming convention same as constants

- /SIZE = N/ defines the size of the array (placed after array name declaration; for example:

```
$REAL
        ANGLES(2, 19) /SIZE = 40/ =
                10.0, 3.3E-04, 20.0, 1.002E-03, ….;
```

- Arrays may be referenced in operations blocks or wherever expressions are allowed in data blocks

```
$CONDUCTORS
  GL(1,3) = MASS_FLOW(1) * 0.4;
```

Arrays

- Two ways to specify arrays:
  - $TABLE sub-block: values for single dependent variable V tabled against several independent variables X, Y, Z

```
V(X, Y, Z)
X = X1,    Y = Y1,    Z1, V, Z2, V, … ZN, V,
           Y = Y2,    Z1, V, Z2, V, … ZN, V,
X = X2,    Y = Y1,    Z1, V, Z2, V, … ZN, V,
.          .
.          .
X = XN,    Y = Y2,    Z1, V, Z2, V, … ZN, V;
```

Arrays

- Two ways to specify arrays:
  - $TABLE sub-block:

    This is a shortened form of the array on the last slide:

    V(X, Y, Z)
    Z = Z1, Z2, Z3 …, ZM,
    X = X1, Y = Y1, V1, V2, V3, … VM,
    X = X2, V1, V2, V3, … VM,
    X = X1, Y = Y2, V1, V2, V3, … VM,
    .
    .
    ;

    } Y values held constant while X being specified

  - Or, if just one independent variable:

    V(X)
    X1, V1, X2, V2 …;

Arrays

SINDA Equivalent:

HEADER ARRAY DATA, *submodel name*

array number = *value 1*, *value 2*, … *value N* (singlet)

array number = $x_1, y_1, x_2, y_2, \ldots x_N, y_N$ (doublet)

array number = N, x_1, x_2, … x_N (bivariate)

$y_1, Z_{11}, Z_{12}, \ldots Z_{1N}$

.

.

$y_m, Z_{m1}, Z_{m2}, \ldots Z_{mN}$

Arrays

NOTES:

- SINDA allows arrays of mixed types (e.g. an integer and a real can be in the same array); this is not allowed in ESATAN

- There are no array numbers in ESATAN. To refer to an array, can use the following syntax:

  SUBMODELA:SUBMODELB:V(X,Y,Z)

Basic ESATAN Thermal File Construct

$MODEL

DATA BLOCKS

OPERATIONS BLOCKS

$ENDMODEL

- $SUBROUTINES
- $INITIAL
- $VARIABLES1
- $VARIABLES2
- $EXECUTION
- $OUTPUTS

SINDA Equivalent:

SINDA .cc file:

(DATA BLOCKS)

SINDA .inp file:

HEADER OPERATIONS DATA

(LOGIC BLOCKS)

- HEADER OPERATIONS DATA
  (BUILD statements)
- HEADER SUBROUTINE
- HEADER VARIABLES 0/1/2
- HEADER CONTROL DATA, GLOBAL
- HEADER OUTPUT CALLS

Subroutines

SUBROUTINE *name* LANG = *language*

- Subroutines defined in $SUBROUTINES block
- To call subroutine (e.g. subroutine SUB1 in submodel ROCKET:MOTOR):

  CALL ROCKET:MOTOR:SUB1(…)

- *language* is either MORTRAN or FORTRAN
  - MORTRAN is a superset of FORTRAN77; in addition to all FORTRAN77 commands, some additional features include:
    - SELECT CASE (*expression); … CASE ELSE; … END SELECT*

      where CASE is identified by point values or ranges
    - *WHILE(expression); … ENDWHILE;*
    - *REPEAT; … UNTIL (expression);*
    - *Node/Conductor properties may be referenced (e.g. T1012, GL(1,5), etc.)*

Subroutines

SUBROUTINE *name* LANG = *language*

- Functions can also be defined in the subroutines block:

```
$SUBROUTINES
        DOUBLE PRECISION FUNCTION TAV(XX, YY)
        DOUBLE PRECISION XX, YY
        TAV = (XX + YY) / 2.0D0
        RETURN
        END
```

- Note: values defined inside a subroutine do NOT re-zero if subroutine is called multiple times!

  Safe option: set all local values inside the subroutine to zero at the beginning of the routine

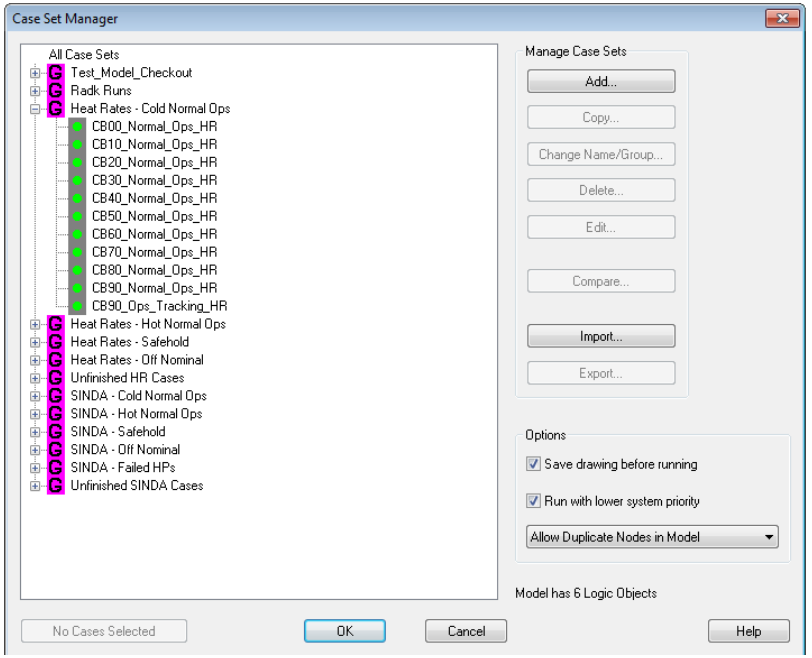
Initial

- $INITIAL block allows initialization of data prior to the program solution

  ➔First “subroutine”: program executes code in $INITIAL block only once

- This is useful for initializing temperatures

Rough Equivalent in Thermal Desktop:
“Case Set Manager”



Variables 1

- $VARIABLES1 defines block
- Instructions placed here are executed at the start of each iteration or time step in the solution
  - Useful for time- or temperature-dependent quantities
- Example:

```
$VARIABLES1
        CALL TCAPCA(C10,T10)      #CALCULATE TEMP DEPENDENT CAPACITANCE
        IF(T10 .LE. 10.0D0)THEN
                QI10 = 5.6D0
        ELSE
                QI10 = 0.0D0
        END IF
```

NOTES:

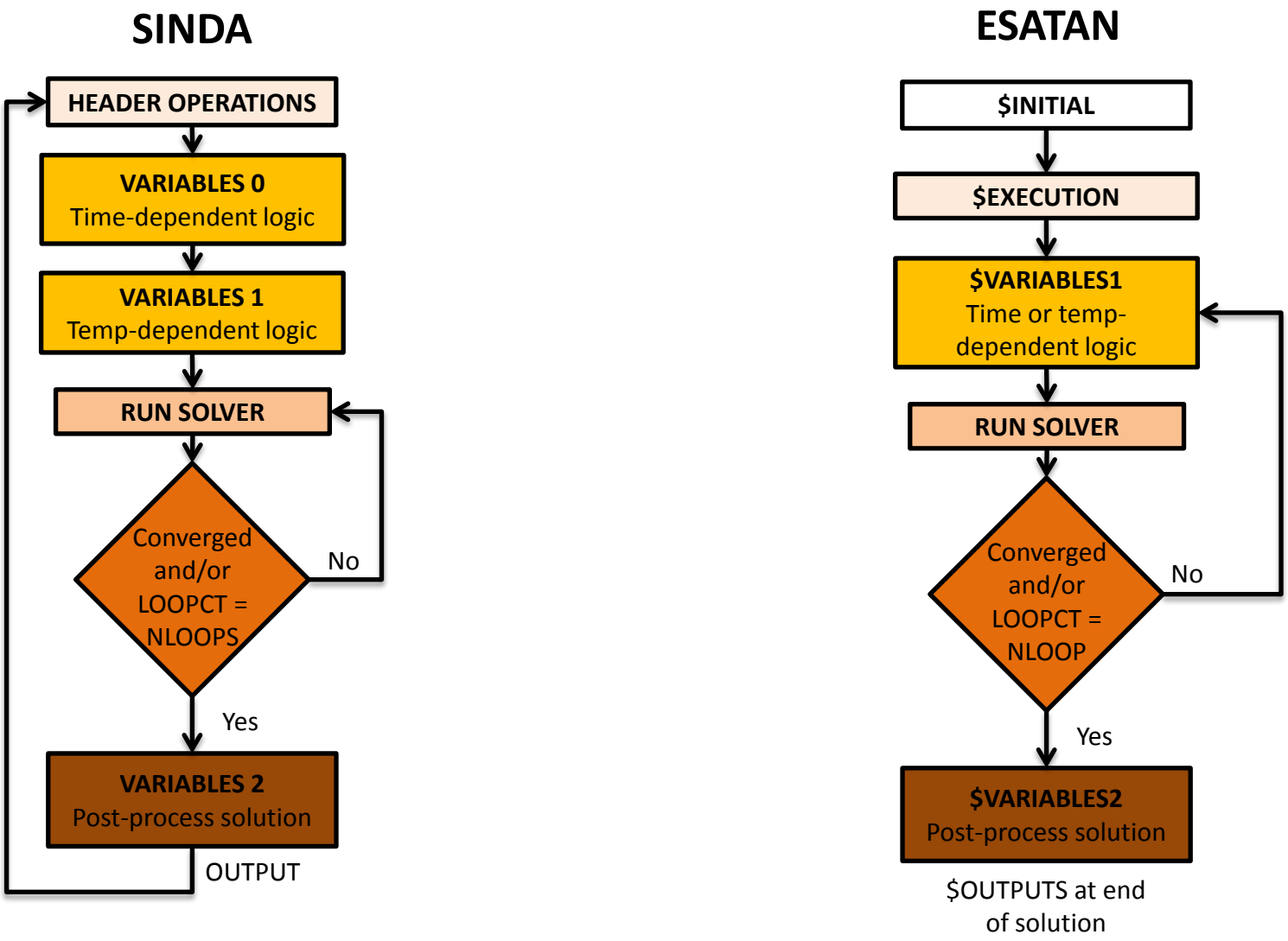
- There are no semicolons in VARIABLES1 since it uses Fortran-type language
- ESATAN does not have an equivalent of VARIABLES 0 (operations after timestep incremented, before heat transfer equations are integrated) ➔ use VARIABLES 1 instead

Variables 2

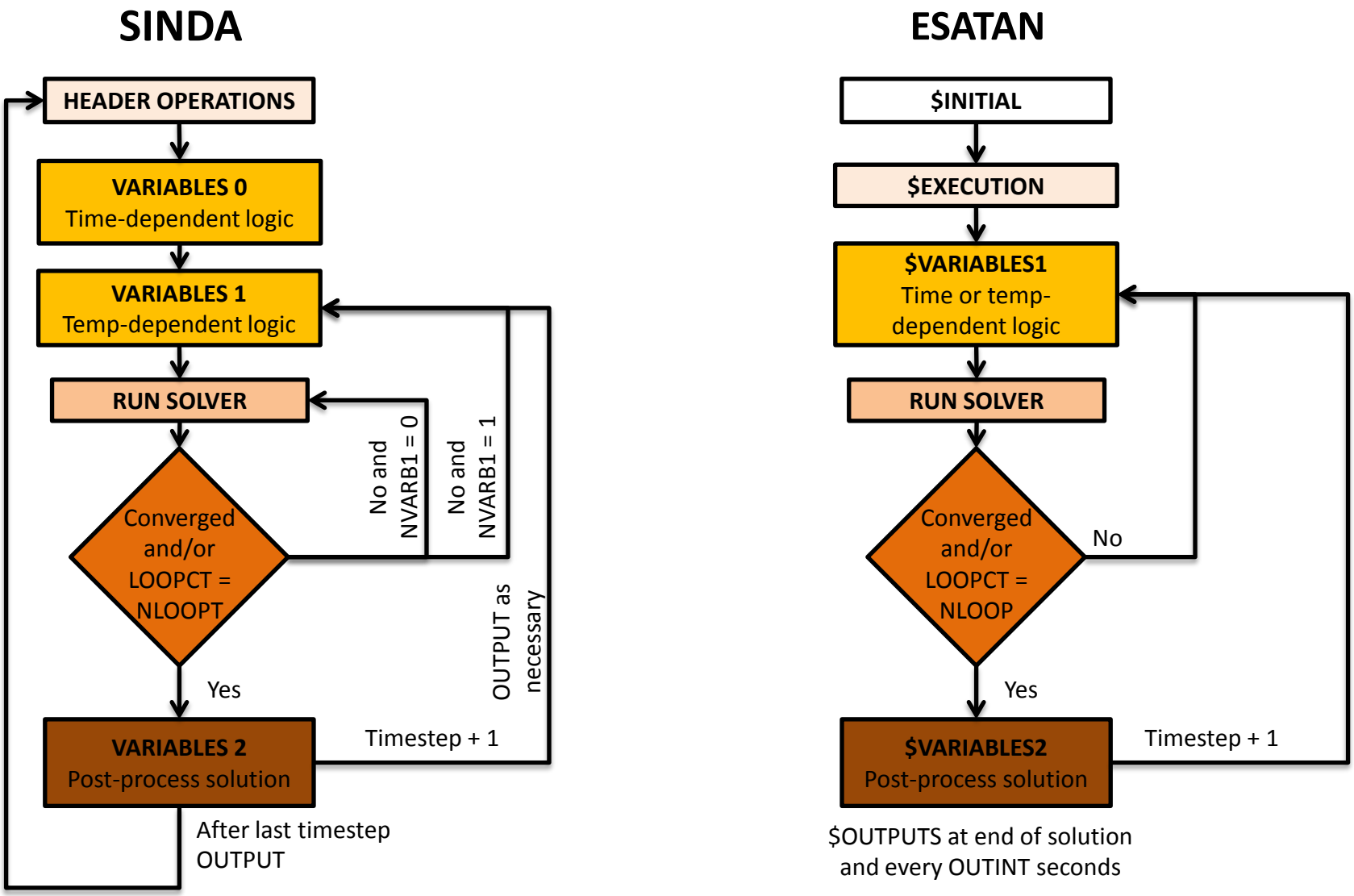
- $VARIABLES2 defines block
- Instructions placed here are executed at the end of each time step in the solution
  - Useful for integrating or metering quantities, rearranging the model based on a time-dependent event, or comparison of latest results with test data
- Example:

```
$VARIABLES2
        CALL ROTATE      # SET LATEST MODEL POSITION
        HX=FLUXML(CURRENT, PIPE)
                         # FLUXML IS A LIBRARY ROUTINE USED TO OBTAIN
                         # LINEAR HEAT FLOW BETWEEN TWO SUBMODELS
```

Steady State Execution



Transient Execution



Model Execution

- $EXECUTION block only called on top level model after $INITIAL
  - $EXECUTION on lower levels ignored

```
$EXECUTION
        INTEGER I
        I = 1
        WHILE (I .LE. 2)
                SELECT CASE I
                CASE 1
                        T999 = 0.0D0
                CASE 2
                        T999 = 100.0D0
                END SELECT
                CALL SOLVIT
                I =I+1
        ENDWHILE
```

Example code to select case to run

SOLVIT performs steady state solution

Model Execution

```
        CALL SAVET(CURRENT)
#
        CALL PRQNOD(‘ ‘, CURRENT)
#
        HEATER = ‘TRANSIENT’
#
        CALL STATST(’N:Thruster1:2’, ’D’)
#
        CALL SLFWBK
```

} Save steady state temps

} Print heat flows

} STATST changes the status of the node; in this case, it changes node Thruster1:2 to a diffusion node

} SLFWBK performs transient thermal analysis by implicit forward-backward differencing

Model Execution

- Other keywords typically used during execution are:

Keyword	Definition
MODULE	Current Solution Module
SOLVER	Solver Type
SOLVFM	Full matrix steady state solver
SLFRWD	Forward differencing transient solver
SOLVIT	Iterative steady state solver
SLCRNC	Crank-Nicholson forward-backward transient solver

ESATAN	SINDA
CALL SOLVIT	CALL STDSTL
CALL SLFRWD	CALL FORWRD
CALL SLFWBK	CALL FWDBCK

Outputs

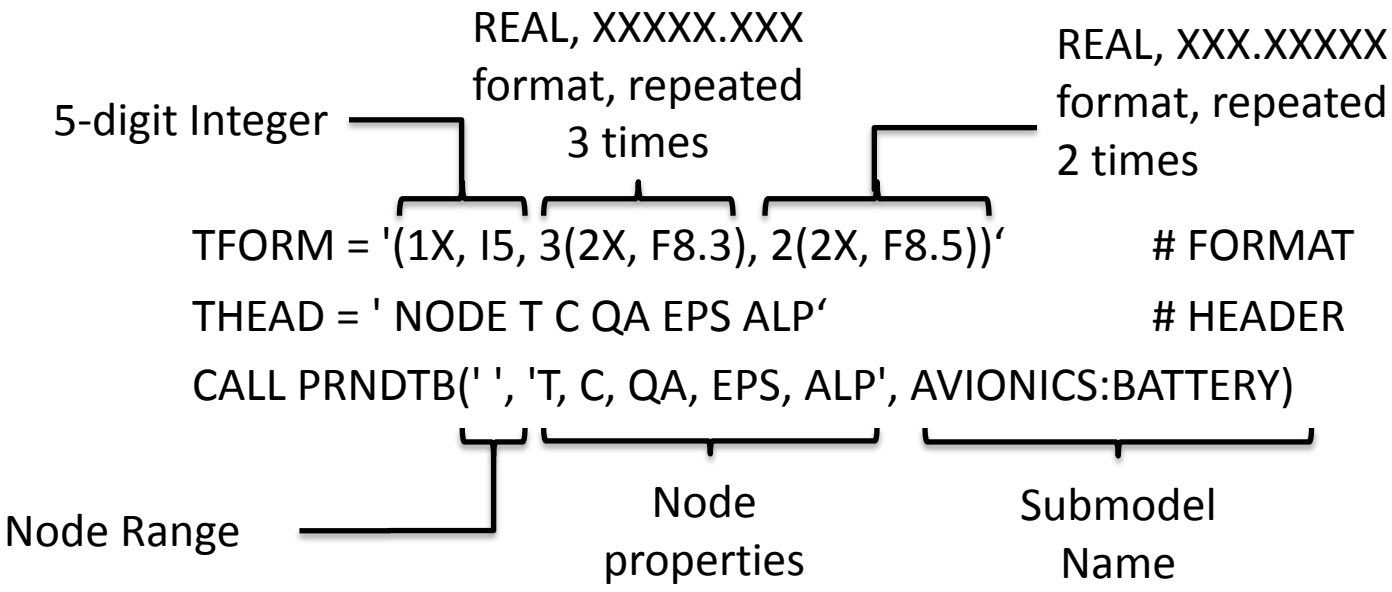
- $OUTPUTS block: only main model block is executed
- Instructions in $OUTPUTS block are executed at:
  - End of the steady state run
  - Before the first transient timestep
  - At each transient output interval
  - After the last transient timestep
- Control Constants:

OUTINT = (*value*)	# outputs every (*value*) seconds of simulation time
OUTIME = (‘ALL’/ ‘NONE’)	# ‘ALL’ (default): output block executed
	# ‘NONE’: code bypasses output block

Outputs

- Typical output subroutines:

  PRNDTB: Table output of entities (uses Fortran formatting)
  Example:



Outputs

- Typical output subroutines:

PRNCSV	Comma-separated list output of entities
PRNDBL	Block output of entities
PRQG	Print heat flow between two nodes
PRQLIN	Print heat flow over linear conductors
PRQNOD	Print heat flow to node
PRTARR	Print array elements
PRTNAV	Print average of entities
PRTSUM	Print sum of entities
PRTTMD	Print maximum temperature difference
PTSINK	Print sink temperatures

Outputs

ESATAN	SINDA
CALL PRNDBL (‘ ‘, ‘T’, CURRENT)	CALL TPRINT
CALL PRNDBL (‘ ‘, ‘C’, CURRENT)	CALL CPRINT
CALL PRNDBL (‘ ‘, ‘GF,GL,GR’, CURRENT)	CALL GPRINT
CALL PRNDBL(‘ ‘, ‘QS,QA, QE, QI,QR’, CURRENT)	CALL QPRINT

Summary of ESATAN Declarations: Data Blocks

ESATAN	SINDA
$MODEL	HEADER OPTIONS DATA
$NODES	HEADER NODE DATA
≈ QI (impressed heat) in $VARIABLES1	HEADER SOURCE DATA
$CONDUCTORS	HEADER CONDUCTOR DATA
$CONSTANTS	HEADER REGISTER DATA
$CONTROLS	HEADER CONTROL DATA
$ARRAYS	HEADER ARRAY DATA

Summary of ESATAN Declarations: Operations Blocks (Logic Blocks)

ESATAN	SINDA
$SUBROUTINES	HEADER SUBROUTINES
$INCLUDE	INCLUDE, INSERT
$INITIAL	≈Case Set Manager in Thermal Desktop
$VARIABLES1	HEADER VARIABLES 0, *submodel name* HEADER VARIABLES 1, *submodel name*
$VARIABLES2	HEADER VARIABLES 2, *submodel name*
$EXECUTION	HEADER OPERATIONS
$OUTPUTS	HEADER OUTPUT CALLS, *submodel name*

ESATAN-TMS Workbench GUI Demonstration

Questions?

Thank You!