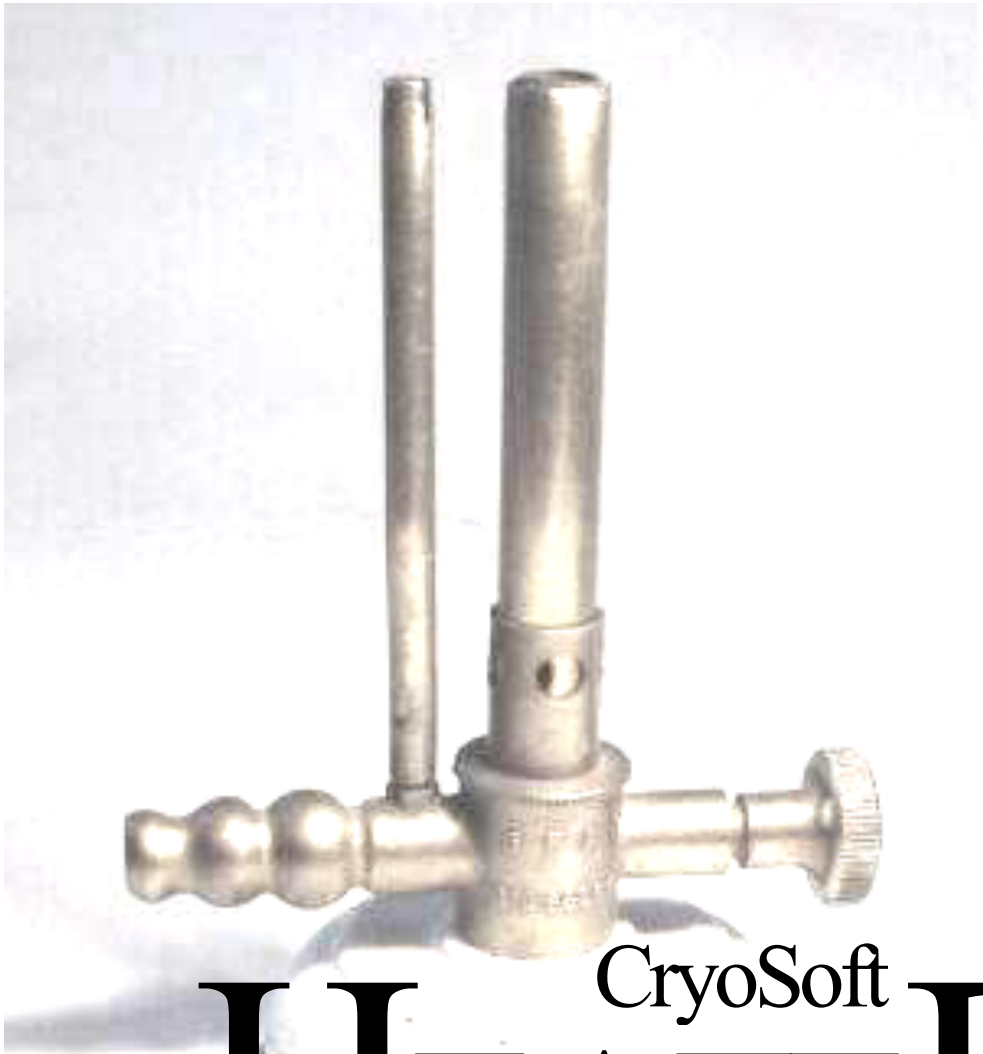
User's Guide

Version 2.1a
November 2021



CryoSoft

HEATER

Simulation of Heat Conduction

DISCLAIMER

Even though CryoSoft has carefully reviewed this manual, CRYOSOFT MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS PROVIDED "AS IS", AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

Contents

Roadmap

Before you start

This manual is the reference user's guide for HEATER and its post-processor, HEATERPOST. Throughout this manual we assume that the reader is familiar with the physics and engineering issues that are associated with solid heat transfer at cryogenic temperature conditions. We strongly suggest that the reader consults the relevant references before using this manual.

How to use this manual

This manual is structured as follows:

- Chapter 1 contains a brief and general introduction on the modeling principle and solution methods available.
- Chapter 2 gives basic information on the installation, explains how to start a HEATER run and launch the post-processor HEATERPOST on a UNIX workstation.
- Chapter 3 contains case studies that the reader should use to familiarize himself with the operation and features of the program.
- Chapter 4 contains additional information on the preparation of the input and the meaning of the input variables
- Chapter 5 describes the details of the post-processing command language.
- Chapter 6 describes the User Routines that can be used for advanced use. These routines can be linked to the standard code to provide powerful customization.
- Chapter 7 deals with troubleshooting and error messages;
- Chapter 8 gives the references and a general bibliography for documentation.

Beginners to HEATER should read chapters 1, 2 and 3 in sequence. They will make occasional cross reference to chapters 4 and 5 for detailed information. Experienced users will use chapters 4, 5 and 6 for daily operation. Chapter 7 is designed to be consulted as an indexed glossary for error messages and associated actions.

CHAPTER 1

Introduction

What is HEATER

HEATER is a program for the analysis of transient and steady-state heat transfer by heat conduction in three-dimensional solids. HEATER computes the evolution of an initial distribution of temperature in solid materials subjected to volumetric heating/cooling, with prescribed boundary conditions. It is based on a general 3-D finite element solver of Partial Differential Equations (PDE). The use of finite elements allows optimal flexibility on the modeled geometry, the heat sources and the boundary conditions considered.

A HEATER model

A HEATER model consists primarily of a mesh of solid finite elements in 3-D, assembled according to the common criteria of element congruency. The elements implemented range from 1-D line elements, which have only one significant dimension of heat flow, through 2-D shells of triangular and quadrilateral shape, to 3-D solid tetrahedra, pyramids and hexagons. The complete list of allowed elements and their properties is given below. A valid mesh can contain any combination of the above elements. A schematic example of such a model is shown in Fig. 1.

The transient heat conduction equation solved by the PDE routine is the following:

$$\rho pc\frac{\partial T}{\partial t} - \nabla\left(pk\nabla T\right) = p\dot{q}''' \tag{1}$$

where ρ is the material density, c its specific heat, k the thermal conductivity, $\dot{q}'''$ is the heating power per unit volume and, finally, p is an element *property* that represents either the cross section transverse to the heat flow in the case of 1-D elements, or the thickness in the plane normal to the heat flow in the case of 2-D elements. The element *property* is set to unity in case of 3-D elements, and hence disappears from the above equation.

The volumetric source term, $\dot{q}'''$, is defined in volumes within the solid domain, as a function of space and time if requested. The volumes, in turn, are defined as sets of elements, grouped and indexed. This mechanism allows to define heat sources with different intensity, distribution and waveforms across the solid domain analyzed.

For the solution of Eq. (1) requires a set of initial and boundary conditions. Initial conditions are specified in the mesh at all nodes, i.e.

$$T(x,0) = T_0(x) \qquad (2).$$

As for boundary conditions, HEATER allows to use different types of boundary conditions to be used in the model, i.e.

- adiabatic (no heat leak outwards of an external surface)
- isothermal (constant temperature)
- external source (prescribed heat flux at an external surface)
- convective (heat exchange through a heat resistance to a constant temperature ambient)
- *radiative (heat exchange through radiation to the outer surfaces)* [1]

The boundary conditions can be defined on different parts of the solid model, either at points, lines or surfaces. Points are single nodes that are marked and indexed. Lines are ordered lists of nodes, which typically (but not necessarily) lie on element edges. Surfaces are sets of single faces that (must) match faces of 3-D elements. This mechanism allows to efficiently define boundary conditions of different type, intensity and waveform in the domain analyzed.

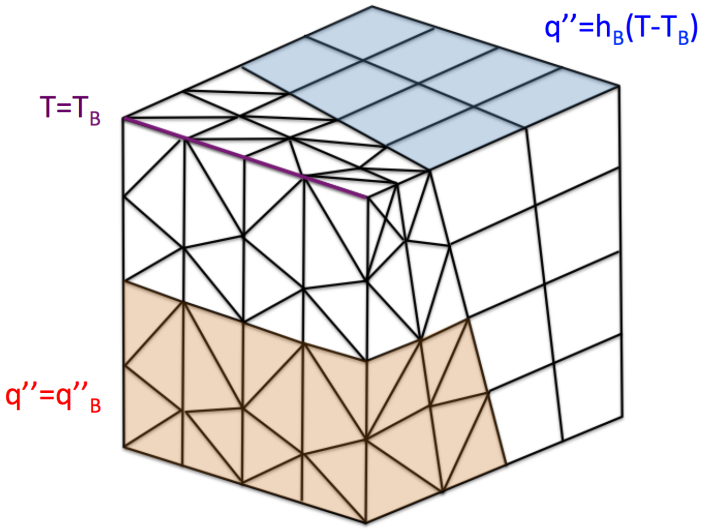


Figure 1 Schematic representation of the HEATER model. The 3-D solid mesh is composed of finite elements that can be of different type (e.g. hexahedra and tetrahedral in the example above). Different heat sources are defined in volume portions of the solid mesh. Boundary conditions are defined in specific points, along lines, or on faces bounding the exterior domain (e.g. heat flux and convection over a surface, and temperature along a line).

PDE Solution

The core solver of HEATER is a general-purpose PDE solver that we have developed in house. The PDE solver applies a finite element algorithm on the user mesh for the discretization of Eq. (1) in space. The PDE allows the use of Lagrangian finite elements with linear or parabolic shape functions. The elements programmed in the present version of HEATER are:

[1] Radiative heat transfer is included in the solver capability, but not accessible to the end-user in the present version.

- 1-D line elements (LINE), with 2 nodes and first order shape functions (linear), or 3 nodes and second order shape functions (parabolic);
- 2-D triangle elements (TRIA), with 3 nodes and first order shape functions (linear), or 6 nodes and second order shape functions (parabolic)
- 2-D quadrilaterial elements (QUAD), with 4 nodes and first order shape functions (linear), 8 nodes and second order shape functions (serendipity parabolic), or 9 nodes and second order shape functions (lagrangian parabolic);
- 3-D tetrahedron elements (TETR), with 4 nodes and first order shape functions (linear), or 10 nodes and second order shape functions (parabolic);
- 3-D pyramid elements (PYRA), with 6 nodes and first order shape functions (linear), or 15 nodes and second order shape functions (parabolic);
- 3-D hexahedron elements (HEXA), with 8 nodes and first order shape functions (linear), or 20 nodes and second order shape functions (parabolic).

The elements implemented are shown schematically in Fig. 2a (1-D elements), 2b (2-D elements) and 2c (3-D elements)

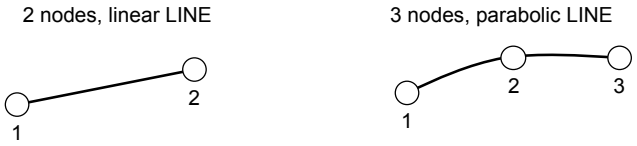


Figure 2a Schematic representation of the 1-D finite element types implemented in HEATER.

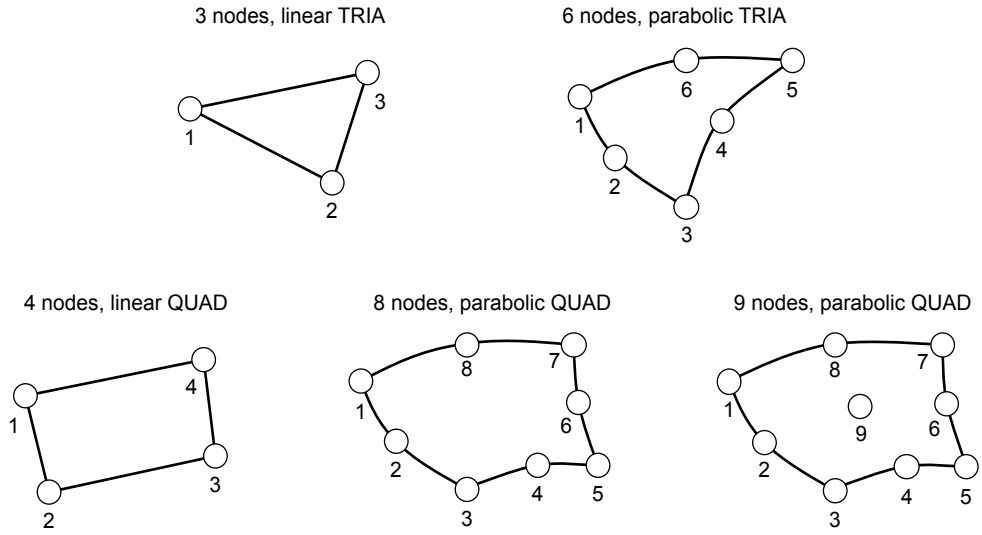


Figure 2b Schematic representation of the 2-D finite element types implemented in HEATER.

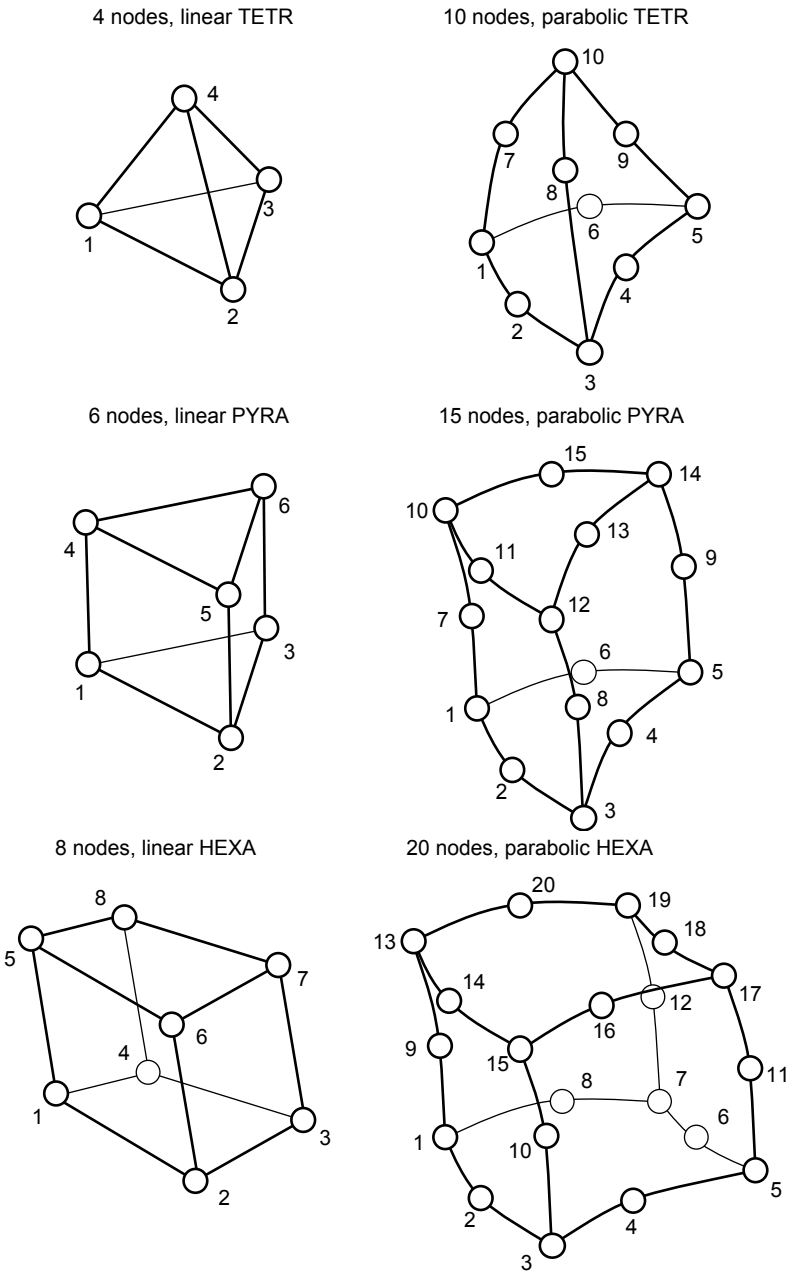


Figure 2c Schematic representation of the 3-D finite element types implemented in HEATER.

The time discretization is based on a multi-step finite difference algorithm of the Beam and Warming family with at most third order accuracy. The time step is adapted automatically to achieve a user-defined error, either using a predictive or an a-posteriori error estimate. The user has control on the time integration accuracy through the choice of algorithm, while the time adaptivity is controlled specifying the error estimator and the desired accuracy.

Results are produced at the times specified by the user, and output to binary storage for later post-processing

Post-processing

The results produced by HEATER are integrally stored and can be analysed to produce plots and reports by the post-processor HEATERPOST. HEATERPOST responds to a user-friendly command language and allows selection of results in time or space, plot and print-out of results vs. time or space, parametric plot of results at given time or space coordinate. See the case studies in Chapter 3 for examples of post-processing sessions, and Chapter 5 for the details on the syntax of the command language.

User Flexibility and Further Extensions

HEATER has several features that allow customizing its modeling capability beyond the allowable parameterization of the configuration that can be achieved using the standard input file. Specifically, the user can:

- modify the dependence of geometry, waveforms and material properties on space, time and solution variables, beyond the standard models implemented, using *User Routines* that can be statically linked to the program segments through a compilation step that produces a customized version of the code. See Chapter 6 for documentation on *User Routines*;
- change parametrically the behavior of the *User Routines* by making use of *Variables* that are read by the code input parser, and can be accessed at run-time using the *Variables* library. See Chapter 4 for details on the syntax to be adopted for the *Variables* input block;
- couple to other programs of the CryoSoft suite through the multi-tasking code manager SUPERMAGNET. This allows to augment the physics span of the simulation domain to include thermal networks (e.g. heat exchange in a coil), hydraulic networks (e.g. proximity cryogenics) or electrical circuits (e.g. magnet protection).

CHAPTER 2

Installing and Running HEATER

Platforms

HEATER and its post-processor HEATERPOST are provided as a package developed for running under UNIX or UNIX-like (e.g. Linux) operating system. The reason is that they require computer intensive calculations, orderly file management and little interactivity. At the time when this manual is written, the platform where HEATER is developed is

- Macintosh running MacOS-X (10.10.5 and higher) under XQuartz,(2.7.8) gcc (5.1) with gfortran.

At different time of the development and production, the code has been installed and tested on the following platforms:

- Mac-OS X (10.2 and higher) operating system;
- GNU/Linux operating system (most distributions).
- INTEL PC's running RedHat Linux OS;
- IBM-RISC workstations running the AIX-V4 operating system and later;
- SUN-SPARC workstation running the Solaris OS operating system;
- DEC-ALPHA workstation running the OSF-1 operating system;
- HP workstations running HP-UX OS;
- Windows-2000 and Windows-XP operating system, with an installed CYGWIN environment (the reference version tested is CYGWIN 1.5.24-2).

Although UNIX obeys strict standards, the architecture of the operating and file system may vary from vendor to vendor. It is therefore possible that porting may require minor adaption of code and libraries. Contact us for advice.

In the following sections we assume here that you are running under a UNIX or UNIX-like operating system, and that you are familiar with UNIX commands, directory and file handling. Contact your system administrator for matters regarding UNIX commands and file system.

Although versions of HEATER and HEATERPOST have been ported to PC's running the Windows OS, at the time when this manual is written this is not a platform directly supported and part of the instructions provided below (i.e. how to run and post-process a case) may not be directly applicable.

Installation

HEATER is one of the CryoSoft family of programs. You will have therefore received the CryoSoft package containing HEATER either as a tar-ball or in pre-installed form. Verify in the CryoSoft installation manual [1] the procedure to be followed for the proper installation of the complete package. The executable codes, `heater` and `heaterpost` are in the directory `~/CryoSoft/bin/`. You will find the example inputs and post-processing command files in the directory `~/CryoSoft/xample/heater/code_x.x/` (the symbol `~/` stands for your home directory, `x.x` for the version you received)

How to run HEATER

Start-up To run HEATER you will need to launch the executable code. In the standard installation on a UNIX system described above HEATER is launched typing the command:

```
~/CryoSoft/bin/heater [-i InputFile] [-v/-s] [-h]
```

Note that command line options are not mandatory (enclosed in brackets, following UNIX documentation standard). The meaning of the options is the following:

`-i, --input`	use `InputFile` to parse the input for the run
`-v, --verbose`	print simulation progress on stdout (default)
`-s, --silent`	no output to stdout
`-h, --help`	print a help message

Once launched, the program decodes the options, if any are given, and checks for the specific operation mode requested. If no input file is provided as an option, then the program prompts the user for the input file name. HEATER reads the problem definition from an ASCII file whose structure and content are described in detail in Chapter 4 of this manual. Examples of input files are given in Chapter 3. At this time you will enter the name of a file containing the input for the case to be run (e.g. `file.input`):

```
HEATER Enter input file name
file.input
```

HEATER then parses the input file, performs checks on consistency, configures the case and starts the simulation. A simulation starts from an initial condition at the starting time and advances in time using the time step selected. At each time step HEATER emits a message with the real time reached in the simulation (in s) the time step taken (in s) and the ratio of real time to the total time to be simulated:

```
....
Time : 4.949E-03   Step :  3.235E-05   Time/Tend :   0.98987
Time : 4.998E-03   Step :  4.852E-05   Time/Tend :   0.99957
....
```

until the end of the simulation. When the end time of the simulation is reached HEATER prints a message reporting the total CPU time used in the run:

```
Total Cpu [s]:   244.059998
```

Each run of HEATER produces:

- a binary storage file containing all results stored at user's specified times. The user can control the name of this file, the default file name is `heater.store`;

- a log file containing a report on the case run, run statistics and error messages. The user can control the name of this file; the default file name is `heater.log`.

Restart After a succesfull completion of a run it is possible to restart the simulation at the last time stored in the binary storage file and proceed with the time integration. A restart procedure is triggered if the input file read by HEATER contains the `Restart` command (see Chapter 3 and 4 for details). Assuming that this is the case for the input file `file.restart`, and the program is launched with no command line options, a restart in our example is obtained launching again HEATER:

```
~/CryoSoft/bin/heater
HEATER Enter input file name
file.restart
```

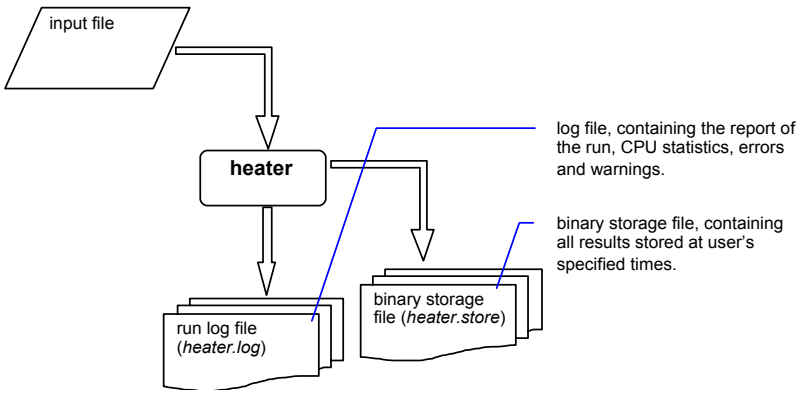
in which case HEATER reads the binary storage file and starts the simulation at the last time stored:

```
Time : 5.000E-03   Step :  1.000E-05   Time/Tend :   0.00000
```

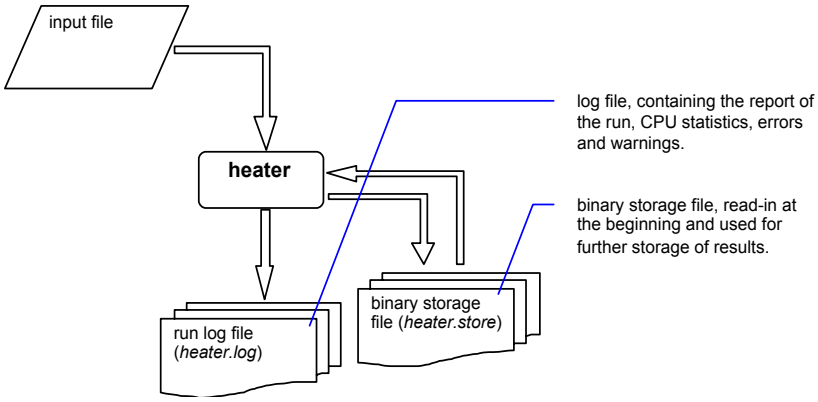
Until the final time specified in the input file `file.restart` is reached.

Note You can use an arbitrary sequence of restarts to simulate different time spans with varying resolution and accuracy. There is no limit to the number of restarts that can be executed for a single simulation.

We show below schematically the flow-diagram of a HEATER run:



as compared to the flow-diagram of a HEATER restart reported below. Data is read at the beginning of the restart from the binary storage file, and is appended to the same file while the simulation proceeds:



How to run HEATERPOST

To produce any detailed result, both in the form of printed tables or plotted curves in PostScript® format, it is necessary to run the HEATER post-processor HEATERPOST. HEATERPOST is launched under UNIX with the command:

```
~/CryoSoft/bin/heaterpost [-i InputFile] [-v/-s] [-h]
```

Also in this case command line options are not mandatory (enclosed in brackets, following UNIX documentation standard). The meaning of the options is the following:

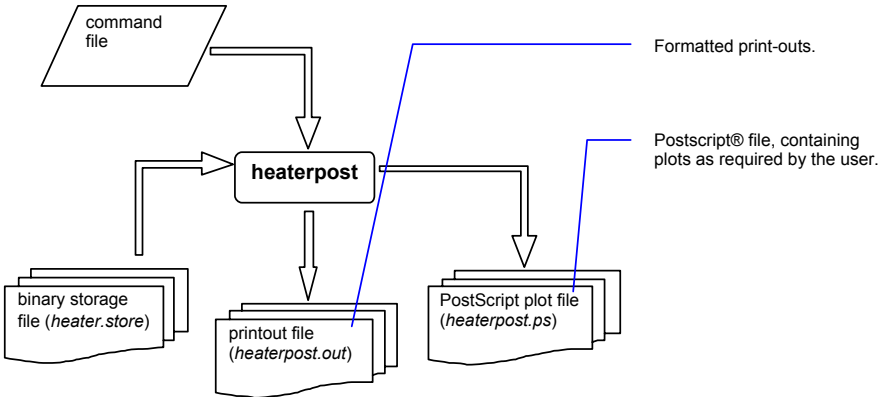
`-i, --input`	use `InputFile` to parse the input for the post-processor
`-v, --verbose`	print post-processing progress on stdout (default)
`-s, --silent`	no output to stdout
`-h, --help`	print a help message

Once launched, the program decodes the options, if any are given, and checks for the specific operation mode requested. If no input file is provided as an option, then the program prompts the user for the name of an ASCII file containing the series of commands that control the generation of the printouts and plots. The structure and content of this file is described in detail in Chapter 5 of this manual. Examples of command files are given in Chapter 3. At this time you will enter the name of the file containing the commands (e.g. `file.post`):

```
Enter command file name
file.post
```

HEATERPOST then parses, echoes and interprets the commands from the command file. The commands cause retrieval of the results of a run from the binary storage file generated by HEATER (by default from the file `heater.store`). As a result HEATERPOST generates:

- a file containing the formatted printouts of the results (by default `heaterpost.out`), and
- a file containing the plots requested in PostScript® format (by default `heaterpost.ps`).



Customization

The method described earlier provides the standard manner to run a HEATER simulation, and post-process the results. HEATER, however, as most other CryoSoft codes, gives the possibility to customize the physical models by using User Routines, as described in Chapter 6 (see later for details). The user has the possibility to adapt and extend the physics contained in the standard solver, at the additional complexity of writing FORTRAN routines that must obey to the language syntax, and parameter call specification. The customized User Routines need to be compiled and linked the program segments to generate the customized version of the code. Template for the User Routines are given in the directory `~/CryoSoft/usr/heater/code_x.x`. Compilation and link-editing can be done using the standard installation script `CSmake`, but we discourage users to modify the standard codes provided, as this will replace the reference installation. As a safer alternative, we strongly recommend copying the User Routines templates in a work directory, and generating in this location the customized version of the code by using an adapted compilation script, or a makefile. Consult the examples below, and contact us for guidelines on how to set-up one such customized structure.

CHAPTER 3

Case Studies

As discussed in Chapter 2, HEATER requires an input file with all definitions necessary to specify the assembly of components in the model structure, the characteristics of each component, the initial conditions, and the solution controls. We refer to this file as the *input file*. The input file is needed both for a start-up run and a restart run.

Similarly, post-processing of HEATER results using the post-processor HEATERPOST requires an input file with a sequence of commands that select results, print and plot them. We refer to this file as the *post-processing command file*.

In this Chapter we give examples of input files and post-processing command files to deal with practical modeling situations. The case studies given here are intended to guide the user from the formulation of a problem to its modeling, the creation of the input file for the case, running the case, and finally the generation of the results. For obvious reasons, they are of limited complexity and are intended as examples to illustrate minimum capability of the program. More complex situations can obviously be modeled, taking the following case studies as starting points and evolving or combining them. Refer to Chapter 2 on how to run the examples described here with HEATER and how to generate results and plots with HEATERPOST.

Note All input files and post-processing command files for the case studies discussed in this manual are provided with the standard installation. They are located in the directory:

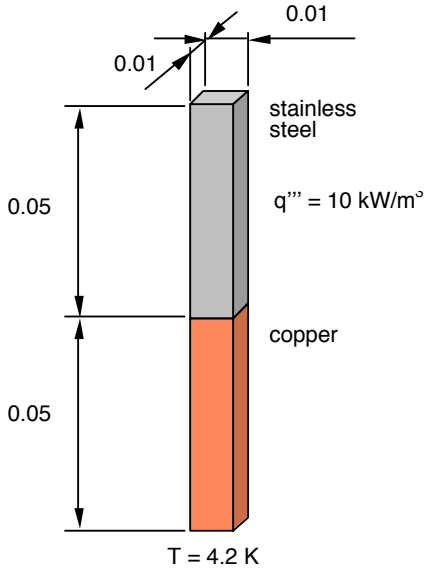
`~/CryoSoft/xample/heater/code_x.x`

where `x.x` stands for the version you received. In the following sections we use the `Courier` font to reproduce the content of those input files, while text in *italic* indicates our comments to the input.

Heat flow in a composite bar

Physical definition of the problem This test shows a calculation of the transient heat flow along a composite bar, made of two equal parts of copper (lower end) and stainless steel (upper end). The total length of the bar is 10 cm, and the cross section is 1 cm^2 The steel portion is subjected to a uniform volume heating source of 10 kW/m^2, and has an adiabatic surface. The lower surface of the copper part is kept at a fixed temperature of 4.2 K The solution is computed from the initial state of uniform temperature, i.e. 4.2 K in the bar, to steady state conditions.

The picture below shows schematically the geometry of the problem, with the main dimensions. Material properties are non-linear with temperature, following the standard definition ion the CryoSoft material properties database [2].



The solution of this problem requiires the definition of a mesh, containing the geometry, initial and boundary condition. For this problem it is possible the use of several type of elements. In this case we will show how to solve the same problem using 1-D LINE elements. Similar input for 2-D QUAD elements and 3-D HEXA elements can be found among the sample files provided in the standard installation. To ease post-processing, we will define a LINE on one of the sides of the bar, which will allow easy plotting of the results.

Input file for the run with LINE elements The step-by-step definition of the input file for the HEATER run corresponding to this problem is shown below.

`bar_line.input`

The General block contains the name of the model, used as a title of the problem

```
Begin General

   Name 'Composite Copper-Steel bar - LINE elements'

End
```

The mesh (nodes and elements) is defined in the Mesh block

```
Begin Mesh
```

```
Nodes 101     ; number of nodes in the mesh. The definition of the nodes follows
       index     x       y       z
Node   1         0       0       0
Node   2         0.001   0       0
Node   3         0.002   0       0
Node   4         0.003   0       0
Node   5         0.004   0       0

..... (lines omitted)

Node   95        0.094   0       0
Node   96        0.095   0       0
Node   97        0.096   0       0
Node   98        0.097   0       0
Node   99        0.098   0       0
Node   100       0.099   0       0
Node   101       0.1     0       0

Elements      100 ; number of elements in the mesh. The definition of the elements follows

           index     type    nodes  order  material          properties        n1   n2
Element    1         LINE    2      1      Cu                Area 1.0e-4       1    2
Element    2         LINE    2      1      Cu                Area 1.0e-4       2    3
Element    3         LINE    2      1      Cu                Area 1.0e-4       3    4
Element    4         LINE    2      1      Cu                Area 1.0e-4       4    5
Element    5         LINE    2      1      Cu                Area 1.0e-4       5    6

..... (lines omitted)

Element    95        LINE    2      1      AISI-304          Area 1.0e-4       95   96
Element    96        LINE    2      1      AISI-304          Area 1.0e-4       96   97
Element    97        LINE    2      1      AISI-304          Area 1.0e-4       97   98
Element    98        LINE    2      1      AISI-304          Area 1.0e-4       98   99
Element    99        LINE    2      1      AISI-304          Area 1.0e-4       99   100
Element    100       LINE    2      1      AISI-304          Area 1.0e-4       100  101

End

Begin Sets

Points can  be used to define boundary conditions in the mesh. The two points below
correspond to the lowest (point 1) and highest (point 2) in the mesh (see nodal coordinates)
   Point 1  Node  1
   Point 2  Node  101

The line below is defined along the bar, and used later for post-processing the results
   Line  1  Nodes 101    1   2   3   4   5   6   7   8   9   10
                        11  12  13  14  15  16  17  18  19  20
                        21  22  23  24  25  26  27  28  29  30
                        31  32  33  34  35  36  37  38  39  40
                        41  42  43  44  45  46  47  48  49  50
                        51  52  53  54  55  56  57  58  59  60
                        61  62  63  64  65  66  67  68  69  70
                        71  72  73  74  75  76  77  78  79  80
                        81  82  83  84  85  86  87  88  89  90
                        91  92  93  94  95  96  97  98  99 100
                       101

We define a volume as an arbitrary assembly of elements. In this case the elements are all
those of AISI304_Steel (see mesh befinition). The volume will be used later to define a heating
source
   Volume  1  Elements 50    51 52 53 54 55 56 57 58 59  60
```

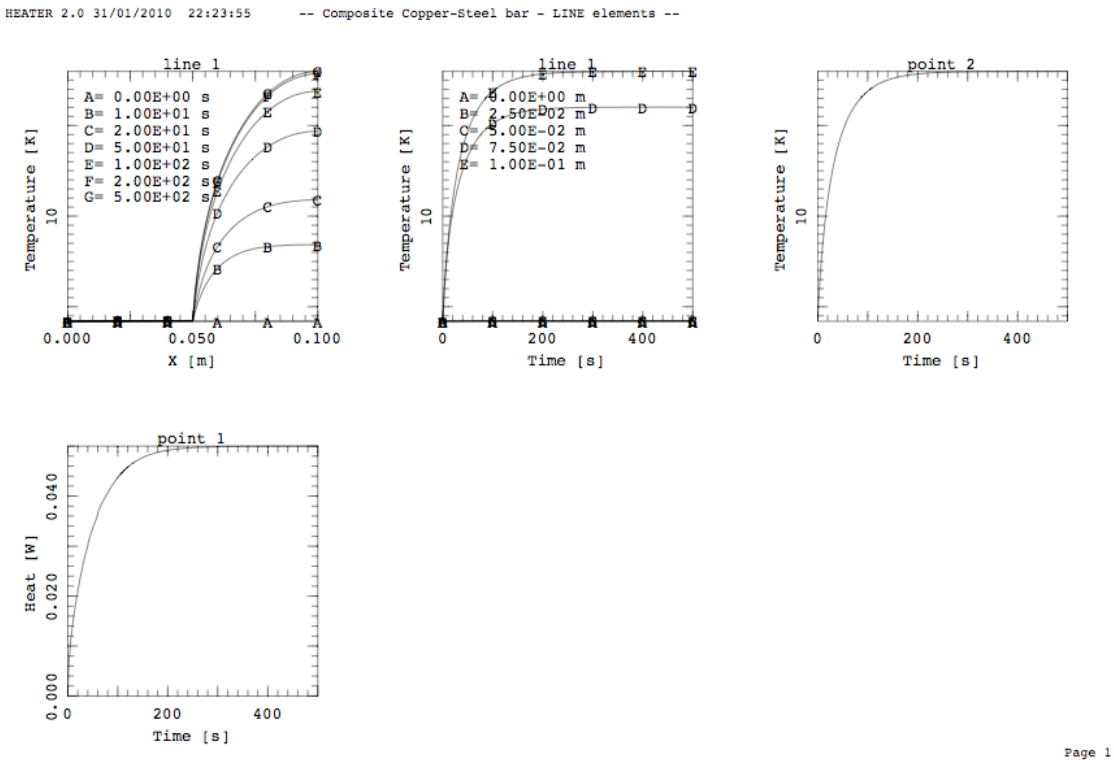
```
                                61 62 63 64 65 66 67 68 69  70
                                71 72 73 74 75 76 77 78 79  80
                                81 82 83 84 85 86 87 88 89  90
                                91 92 93 94 95 96 97 98 99 100

End

Begin Initial

Initial conditions are of constant temperature, 4.2 K, throughout the mesh
   InitialCondition Temperature constant 4.2

End

Begin Boundary

Define boundary conditions, of prescribed temperature of 4.2 K at the first point, located at
the bottom of the mesh...
   Point 1 Temperature Constant 4.2

... and adiabatic at the second point, top of the mesh
   Point 2 Adiabatic

End

Begin Source

All elements belonging to the volume defined earlier have a constant volume source of 10
kW/m**3
   Volume  1  constant  q0 10000.0

End

Begin Simulation

The simulation starts at t=0 s and proceeds till t=500 s, with a storage every 1 s
   StartTime             0.0
   EndTime             500.0
   OutputStep            1.0

The method selected for time integration is an implicit Euler-Backward (1st order in time)
   TimeMethod            EulerBackward

The time step is automatically adapted in the range 10**-3 s to 1 s to achieve a step-by-step
tolerance of 0.03. No time step iteration is performed
   MinimumStep           1.0e-3
   MaximumStep           1.0
   StepEstimate          smooth
   ErrorEstimate         change
   ErrorControl          none
   Tolerance             1.0e-2

Results are stored in the file bar.store for later post-processing
   LogFile               bar.log
   StorageFile           bar.store

End
```

Post-processing command file The following is an example of the sequence of commands necessary to generate of print-outs using the post-processor HEATERPOST.

bar.post

```
*Read data from the binary storage file*
StorageFile     bar.store

*Produce output and PostScript files with the names defined below*
OutputFile      bar.output
PostScriptFile  bar.ps

*Select specific times in the evolution*
select time 0.0 10.0 20.0 50.0 100.0 200.0 500.0

*Plot the temperature profile along the line defined in input*
plot temperature line 1

*Select specific locations in space*
select X 0.0 0.025 0.05 0.075 0.1

*Plot the evolution of the temperature at the selected locations along the line defined in input*
plot temperature line 1

*Plot the temperature evolution at point 2 (adiabatic boundary)*
plot temperature point 2

*Plot the heat flowing at point 1 (prescribed temperature)*
plot heat point 1

*Select the final time of the evolution*
select time 500.0

*Print the complete solution in the mesh, producing an output containing the node index, the
nodal coordinates (x,y,z), the nodal temperature and the net heat flow at the node*
print X Y Z temperature heat mesh 1
print temperature line 1

stop
```

Results One PostScript file, `bar.ps`, and one ASCII output file, `bar.output`, are generated running the post-processor HEATERPOST with the commands described above in the file `bar.post`. The first file contains the plots requested, shown below.



Page 1

The plots are in the order: the temperature profiles along the line defined (that spans the whole bar), the evolution of temperature at selected locations along the same line, the temperature of the adiabatic boundary, and the heat flow at the constant temperature boundary. Note how the temperature gradient develops in the upper part of the bar, made of stainless steel, heated and with much smaller thermal conductivity than copper. The temperature at the adiabatic end reaches approximately 18 K, while the total heat flowing at the constant temperature node is 0.05 W. This corresponds to the total heat entering the bar.

The second file, `bar.output`, contains the output requested, to be used for inspecting detailed numerical data or further post-processing. In our case the output requested are the nodal coordinates, the temperature and heat flow at each node at the end of the simulation.

`bar.output`

The following is the output of the results. The requested output are the coordinates, temperature and heat flow at all nodes. The header of the output file, containing an echo of the input, has been removed.

```
HEATER Version 2.0
file created at   31/01/2010  22:23:55
Storage file:   bar.store

Model
=====
Name.........................  Composite Copper-Steel bar - LINE elements

Mesh
====
Nodes.........................        101
     Node        X           Y           Z           T
        1   0.0000E+00   0.0000E+00   0.0000E+00   0.4200E+01
        2   0.1000E-02   0.0000E+00   0.0000E+00   0.4201E+01
        3   0.2000E-02   0.0000E+00   0.0000E+00   0.4202E+01
```

..... *(lines omitted)*

```
Simulation
 ==========
 TimeMethod.....................  EulerBackward
 Tolerance......................  3.000E-02
 ErrorEstimate..................  change
 ErrorControl...................  none
 StepEstimate...................  smooth
 Start Time [s].................  0.000E+00
 End Time [s]...................  5.000E+02
 Minimum Step [s]...............  1.000E-03
 Maximum Step [s]...............  1.000E+01
 Output Step [s]................  1.000E+00
```

X [m]	mesh 1 X [m] 5.00E+02 s	mesh 1 Y [m] 5.00E+02 s	mesh 1 Z [m] 5.00E+02 s	mesh 1 Temperature [K] 5.00E+02 s	mesh 1 Heat [W] 5.00E+02 s
1.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	4.2000E+00	5.0000E-02
2.0000E+00	1.0000E-03	0.0000E+00	0.0000E+00	4.2008E+00	0.0000E+00
3.0000E+00	2.0000E-03	0.0000E+00	0.0000E+00	4.2015E+00	0.0000E+00
4.0000E+00	3.0000E-03	0.0000E+00	0.0000E+00	4.2023E+00	0.0000E+00

..... *(lines omitted)*

9.7000E+01	9.6000E-02	0.0000E+00	0.0000E+00	1.7933E+01	0.0000E+00
9.8000E+01	9.7000E-02	0.0000E+00	0.0000E+00	1.7954E+01	0.0000E+00
9.9000E+01	9.8000E-02	0.0000E+00	0.0000E+00	1.7968E+01	0.0000E+00
1.0000E+02	9.9000E-02	0.0000E+00	0.0000E+00	1.7977E+01	0.0000E+00
1.0100E+02	1.0000E-01	0.0000E+00	0.0000E+00	1.7980E+01	0.0000E+00

CHAPTER 4

Input Reference

Structure and syntax

The input file is read by the input interpreter that parses and analyzes the syntax and the grammar of the various entries. In general the file contains a series of blocks that are structured as follows:

```
Begin BlockName
        VariableName    value(s)
        VariableName    value(s)
        ……..

        VariableName    value(s)
End
```

where *BlockName* is a keyword indicating the block type, and must be one of the following valid choices:

`General`	define the general properties of the model
`Mesh`	define the mesh of nodes, solid elements and their faces
`Sets`	define sets of geometric entities (points, lines, surfaces, volumes)
`Initial`	define the initial conditions of the simulation
`Boundary`	define the boundary conditions of the simulation
`Source`	define the internal heat sources
`Simulation`	define the simulation parameters
`Variables`	define user variables for use in routines and functions

The content of a block is a series of assignations of a set of values to a generic variable *VariableName*. *VariableName* must be chosen among the set of keywords described in the following sections.

The structure and content of the input file must comply with the following rules and conventions:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;

- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored;
- the variables in the block are read sequentially and are checked at read-in time. For this reason the order of precedence of the variables must be respected whenever a value is needed to proceed with the interpretation of a block. The same BlockName can appear more than once in a file;
- repeated variable assignation overrides previous values and is not checked at read-in time;
- the blocks in the file are read sequentially and are checked at read-in time. The same BlockName can appear more than once in a file

Parsing of the input file is finished as soon as an end-of-file is found. At this point the execution control is passed to the main program that executes checks on data consistency, configures the run and launches the simulation. For sample input files see Chapter 3.

Input variables reference

The following table contains, in alphabetical order, the keywords defining the input variables, their physical dimensions and meanings for each block type. Predefined possible values are reported in `Courier`. The default value is indicated in the table and underlined.

Note In the tables below we use the following convention for the type of variables:

C	character (a string delimited by blanks, tabs or apices)
R	real (a number in floating point or engineering notation)
I	integer (an integer number)

Typing must be respect in the input file to avoid errors or mis-interpretation by the parser.

General

The *general* block describes general quantities that apply to the model being prepared. A title can be defined to identify the case. The title appears in plots and print-outs generated by HEATERPOST.

Variable	Type	Units	Meaning
`ModelName`	C	(-)	Model name, used for labeling plots and print-outs.

Mesh

The *mesh* block contains the definition of the mesh for the case analysed, and more specifically, the number of nodes and nodal coordinates, the number of elements and element connectivity, the number of element faces and face connectivity. The mesh block is structured such that it can be easily generated from a general-purpose mesh generator, or obtained formatting the output of such a program.

Variable	Type	Units	Meaning
`Elements`	I	(-)	Number of elements in the mesh. Limited to the maximum memory allocation `MaxElements.`

List:
```
Element  E  Type  Nodes  Order  Material  Property  n1 ... nn
```

Composed of:

`E`	I	(-)	Element index, with allowable values between 1 and `Elements`.
`Type`	C	(-)	Element type, with possible values: `line` 1-D line; `tria` 2-D triangle; `quad` 2-D quadrilaterial; `tetr` 3-D tetrahedron; `pyra` 3-D pyramid; `hexa` 3-D hexagon.
`Nodes`	I	(-)	Number of nodes in the element, see Chapter 1 for the nodes allowed for a given element type.
`Order`	I	(-)	Order of shape functions, see Chapter 1 for the nodes allowed for a given element type.
`Material`	C	(-)	Element material, either selected from the list of standard materials supported by the SOLIDS package of CryoSoft (refer to the relative manual) In the case of a user's specified name the material properties and types are computed by the functions `UserConductivity`, `UserDensity`, `UserSpecificHeat,` that must be provided by the user (see Chapter 6).
`Property`	C	(?)	Property associated with the element, giving either the transverse cross section of a 1-D element, or the thickness of a 2-D element: `Area` `property` in (m^2); `Thickness` `property` in (m).
`n1 ... nn`	I	(-)	connectivity array, reporting the number of the nodes at the vertices of the element. The nodes must be defined, and the numbering must be given in the positive direction of the parent element.
`Faces`	I	(-)	Number of faces in the mesh. Faces are only necessary if they need to be composed in a set forming a surface (see later the description in the Block `Sets`). Each face defined must correspond to an element face. Limited to the maximum memory allocation `MaxFaces`.

List:
`Face F Type Nodes Order n1 ... nn`

Composed of:

`Face`	I	(-)	Face index, with allowable values between 1 and `Faces`.
`Type`	I	(-)	Face type, with possible values: `tria` 2-D triangle; `quad` 2-D quadrilaterial.
`Nodes`	I	(-)	Number of nodes in the face, see Chapter 1 for the nodes allowed for a given face element type.
`Order`	I	(-)	Order of shape functions, see Chapter 1 for the nodes allowed for a given face element type.
`n1 ... nn`	I	(-)	Connectivity array, reporting the number of the nodes at the vertices of the face. The nodes must be defined, and the numbering must be given in the positive direction of the parent face element.

`Nodes`	I	(-)	Number of nodes in the mesh. Limited to the maximum memory allocation `MaxNodes`.

List:
`Node N X Y Z`

Composed of:

`N`	I	(-)	node index, with allowable values between 1 and `Nodes`.
`X Y Z`	I	(-)	nodal coordinates.

Sets

The *sets* block describes groups of mesh objects that form either single points, lines of nodes, surface of faces, or volumes of elements. Sets are used to imposed boundary conditions, and to assign volume sources.

Variable	Type	Units	Meaning

List:
`Line L Nodes N n1 ... .nn`

Composed of:

`L`	I	(-)	Line index. Limited to the maximum memory allocation `MaxLines`.
`Nodes`			Keyword for the nodes list.
`N`	I	(-)	Number of nodes along the line, defined by the following connectivity array. Limited to the maximum memory allocation `MaxNodesPerLine`.
`n1 ... nn`	I	(-)	Connectivity array, reporting the index of the nodes forming the line. The nodes must be defined. The line is oriented from the first node to the last one.

List:
`Point P Node N`

Composed of:

`P`	I	(-)	Point index. Limited to the maximum memory allocation `MaxPoints`.
`Node`			Keyword for the node index.
`N`	I	(-)	Index of the node corresponding to the point.

List:
`Surface S Faces F f1 ... .fn`

Composed of:

`S`	I	(-)	Surface index. Limited to the maximum memory allocation `MaxSurfaces`.
`Faces`			Keyword for the face list.
`F`	I	(-)	Number of faces forming the surface, defined by the following connectivity array. Limited to the maximum memory allocation `MaxFacesPerSurface`.
`f1 ... fn`	I	(-)	Connectivity array, reporting the index of the faces forming the surface. The faces must be defined.

List:
`Volume V Elements E e1 ... .en`

Composed of:

`V`	I	(-)	Volume index. Limited to the maximum memory allocation `MaxVolumes`.
`Elements`			Keyword for the element list.
`E`	I	(-)	Number of elements forming the volume, defined by the following connectivity array. Limited to the maximum memory allocation `MaxElementsPerVolume`.
`e1 ... en`	I	(-)	Connectivity array, reporting the index of the elements forming the volume. The elements must be defined.

Initial

The *initial* block is used to define the initial conditions of the simulation.

Variable	Type	Units	Meaning

List:
`InitialCondition ICvariable ICtype ICdata`

Composed of:

`ICvariable`	C	(-)	keyword indicating the variable to which the initial condition applies. Possible values: `temperature` the initial condition applies to the temperature of all nodes.
`ICtype`	C	(-)	keyword defining the type of initial condition. Possible values: `constant` the initial condition is constant throughout the domain. `user` user defined through the function `UserTInitial` (see Chapter 6).
`ICdata`	R	(?)	value to be used as initial condition, only parsed if `ICtype` is `constant`, and depending on `ICvariable`: `Temperature` in (K).

Boundary

The *boundary* block is used to define the boundary conditions of the simulation. Boundary conditions can be defined in a similar manner at points, lines and surfaces, using the syntax detailed below. The only exception to this general rule is the use of `external` boundary conditions, used in the case of points and lines to couple HEATER to other codes of the SUPERMAGNET suite. This is so far not allowed for surfaces. Boundary conditions have the same physical units, independent on the type of geometric support. For this reason in case of heat flux or heat convection boundary condition at a point, a reference surface needs to be provided. In the case of a heat flux or heat convection boundary condition at a line, a reference perimeter is requested. This is obviously not necessary in case the boundary condition is of prescribed temperature.

Variable	Type	Units	Meaning

List:
`Set S BCvariable BCtype BCdata`

Composed of:

`Set`	C	(-)	keyword indicating the set to which the boundary condition applies. Possible values: `line` the boundary condition applies to a line of nodes. `point` the boundary condition applies to a nodal point. `surface` the boundary condition applies to a surface of element faces.
`S`	I	(-)	index of the set to which the boundary condition applies. Must be a defined set.
`BCvariable`	C	(-)	keyword indicating the variable assigned at the boundary. Possible values: `adiabatic` the boundary condition is of zero heat flux. `convection` the boundary condition is of heat convection to a surface at a given temperature. `heat` the boundary condition is of prescribed heat inflow/outflow. `temperature` the boundary condition is of prescribed temperature.
`BCtype`	C	(-)	keyword indicating the type of boundary condition, depending on the value of `BCvariable`. Possible values for `convection`, `temperature`, and `heat`: `constant` the boundary condition is constant throughout the domain. `user` user defined through the functions `UserBoundaryT`, `UserBoundaryQ`, or `UserBoundaryTh`, depending on the variable at the boundary `BCvariable` (see Chapter 6). `external` the boundary condition is obtained from one of the other CryoSoft simulators, through explicit coupling at each time step. This coupling requires execution under the SUPERMAGNET environment, and leads to an error in case it is used in stand-alone mode. See the SUPERMAGNET manual for more details. Only allowed for points and lines, not allowed for surfaces.
`BCdata`			set of keywords and data defined depending on the boundary variable `BCvariable` and the boundary type `BCtype` (see above), used to set numerical values for the boundary conditions. The syntax depends on the combination of `Set`, `BCvariable` and `BCtype`. The possible combinations of values are reported in the tables below. Note that the meaning of the quantities and keywords in the set of entries BCData is the following: `h` heat transfer coefficient (W/m^2K) `T` temperature in (K) `q` heat flux in (W/m^2) `Area` keyword for boundary surface

`A`	boundary surface in (m^2)
`Perimeter`	keyword for boundary perimeter
`A`	boundary perimeter in (m)

Set	BCvariable	BCtype	BCData
point	convection heat temperature	constant	h T Area A q Area A T
	convection heat temperature	user	h T Area A q Area A T
	convection heat temperature	external	

Set	BCvariable	BCtype	BCData
line	convection heat temperature	constant	h T Perimeter P q Perimeter P T
	convection heat temperature	user	h T Perimeter P q Perimeter P T
	convection heat temperature	external	

Set	BCvariable	BCtype	BCData
surface	convection heat temperature	constant	h T q T
	convection heat temperature	user	h T q T
	convection heat temperature	external	

Source

The *source* block is used to define the volumetric heat source in the elements. Volumetric heat sources are given in W/m^3, and are defined over sets of elements grouped in a given volume, using the syntax detailed below.

Variable	Type	Units	Meaning

List:
`Volume V Stype Sdata`

Composed of:

`Volume`	C	(-)	keyword indicating that the heat source is in a volume.
`V`	I	(-)	index of the volume to which the heat source applies. Must be a defined volume.
`Stype`	C	(-)	keyword indicating the type of heat source waveform in time. Possible values: `constant` constant heating power density in the volume, equal to `q`.

`exponential` the heating power density is defined as exponentially decreasing from an initial value, equal to `q`, with a time constant `Tauq`.
`linear` the heating power density is defined as linearly decreasing from an initial value, equal to `q`, to zero over a time `Tauq`, and remain zero after this time.
`none` no heating (default).
`user` user defined heating power density through the function `userQSource` (see Chapter 6).
`window` heating power density is defined as constant, equal to `q`, in an interval $0 < t < $ `Tauq` and zero outside this interval.

`Sdata` set of keywords and data defined depending on the type of heat source `Stype` (see above), used to set numerical values for the volumetric heat density. The syntax depends on the selected type `Stype`. The possible combinations of values are reported in the tables below. Note that the meaning of the quantities and keywords in the set of entries BCData is the following:
`q` heating power density in (W/m^3)
`Q0` keyword for heating power density
`Tauq` keyword for characteristic time
`tq` heating characteristic time in (s)

`Stype`	`Sdata`
`constant`	`Q0 q`

`exponential`
`linear`
`window`
	`Q0 q Tauq tq`

`user`	

Simulation

The *simulation* block describes the numerical parameters for time integration, logging and storage of results.

Variable	Type	Units	Meaning
`EndTime`	R	(s)	End time to be reached with the simulation.
`ErrorControl`	C	(-)	Switch for iterative error control during time integration. Possible values: `none` the time step is not iterated. `on` at each time step a check is performed to verify that the integration error is below the specified `Tolerance`. If this is not the case the time step is changed and the integration is tried again, iterating until the tolerance error is reached (default). `ErrorControl on` requires that an `ErrorEstimate` method is provided (`change` or `halving`) and that a `StepEstimate` is allowed (`smooth` or `power`). The iteration can significantly increase CPU time.

`ErrorEstimate`	C	(-)	Flag for the method used to estimate the time integration error control during a time step. Possible values: `none` no error estimate is provided `change` the error is estimated based on the change of the system solution during a time step (default). `halving` the error is estimated comparing the result obtained with a time step with the result obtained using two subsequent time steps of halved magnitude. This method can significantly increase CPU time.
`H0Extrapolate`	C	(-)	Switch for higher-order extrapolation of the results of a time step. The order of accuracy of the time stepping method chosen is used to extrapolate the solution to a higher order. Possible values: `none` no higher-order extrapolation applied (default). `on` at each time step the solution is extrapolated using the result of a time step and of two subsequent time steps of halved magnitude. The higher-order extrapolation can significantly increase CPU time and in pathological situations it leads to numerical instabilities.
`LogFile`	C	(-)	Log file name. This file contains the echo of the input and the log of the run, including error messages. If not given the default log file name is `heater.log`.
`MaximumStep`	R	(s)	Maximum time step allowed during adaptive time integration.
`MinimumStep`	R	(s)	Minimum time step allowed during adaptive time integration.
`OutputStep`	R	(s)	Time step for storage of the results. The results are written to the output binary file every `OutputStep` seconds of simulation.
`Restart`			Flag triggering a restart. If this key is present in this block HEATER reads the content of the specified `StorageFile` until the last stored time is found. The simulation begins then from this time. Storage of results continues on `StorageFile` (appended). All input will be ignored, except for `EndTime`, `ErrorControl`, `ErrorEstimate`, `LogFile`, `MaximumStep`, `MinimumStep`, `OutputStep`, `StepEstimate`, `TimeMethod` and `Tolerance`.
`StartTime`	R	(s)	Start time for the begin of the simulation.
`StepEstimate`	I	(-)	Flag for the method used to estimate the time step based on the time integration error and the requested `Tolerance`. Possible values:

			`none` no estimate of the time step is performed. The time step taken is equal to the `MinimumStep` specified. `smooth` the time step is increased/decreased smoothly by means of fixed percentage change (default). A `StepEstimate smooth` requires that an `ErrorEstimate` method is provided (`change` or `halving`). `power` the time step is increased/decreased scaling the ratio of the time integration error to the required `Tolerance` using the order of accuracy of the time integration method. A `StepEstimate power` requires that an `ErrorEstimate` method is provided (`change` or `halving`).
`StorageFile`	C	(-)	Binary storage file name. This file contains the results stored at the user's specified times, and is used for restarts or post-processing. If not given the default file name is `heater.store`.
`TimeMethod`	I	(-)	Flag for the selection of the time integration method. Possible values: `EulerBackward` Euler-backward, or full implicit, or θ=1 method. 1st order accurate (default). `Galerkin` Galerkin, or θ=2/3 method, 1st order accurate. `CrankNicolson` Crank-Nicolson, or trapezoidal, or θ=1/2 method, 2nd order accurate. `BackwardDifference` Two-stage backward differences method, 2nd order accurate. `ImplicitDifference` Two-stage, implicit third order differencing method, 3rd order accurate (mildly unstable). `AdamsMoulton` Adams-Moulton method, 3rd order accurate (mildly unstable) `Milne` Milne method, 4th order accurate (strongly unstable).
`Tolerance`	R	(-)	Relative error to be achieved at each time step during time integration, used to control the time step.

Variables

The *variables* block is used to define user variables, with given name and type, stored internally and shared among routines and procedures. The value of these user-defined variables is accessible through a simple calling protocol in FORTRAN, which greatly simplifies the preparation and parameterization of User Routines. Variables can be seen as an extension of the standard input parameters, i.e. a facility for easy customization.

Variables are defined with the following syntax:

VariableType *VariableName* Value

where *VariableType* is one of the types defined in the table below, *VariableName* is the name assigned to the variable, and used later to retrieve its value, and Value is the value, of the appropriate type, assigned to the variable.

Note We report below a short form of the variables syntax. For further reference, and for explanations on how to access variables from customized User Routines, consult the Variables manual [3]

VariableType	Meaning
`Character`	*VariableName* is a string, whose Value is read as a text, delimited by apexes if the text contains a blank (not recommended)
`Integer`	*VariableName* is an integer, whose Value is read according to FORTAN READ conventions
`Real`	*VariableName* is a real, whose Value is read according to FORTAN READ conventions (floating point or scientific notation)

The variables defined in the *variables* block are accessed from the User Routines (and elsewhere in subroutines and functions linked at run time) through calls to the function **`getXVariable`**`(VariableName,Value)`, where **`X`** stands for the variable type (i.e. **`C`**, **`I`** or **`R`**) as described in [3].

CHAPTER 5

Post-processing Language Reference

Structure and syntax

The post-processing command file is read by the post-processor interpreter of HEATERPOST. This parses and analyzes the syntax and the grammar of the various entries. In general the file contains a series of commands that are executed in sequence during a post-processing session.

The structure and content of the post-processing command file is similar to that of the input file already described in Chapter 4. In particular the following rules and conventions apply:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;
- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored.

Parsing of the input file is finished as soon as an end-of-file or the `stop` command are found. At this point the post-processor completes all pending print-outs and plots and closes the session. For sample input files see Chapter 3.

Commands reference

Post-processing commands In this section we report the list of the postprocessing commands and their meaning in alphabetical order. The keywords identifying commands and options are given in `Courier`. Parameters and values for the commands are given in *italic*.

Note The selection of the items to plot or to print is done identifying first the *target*, i.e. quantity to be plotted/printed, and then the *support*, i.e. the component over which the quantity is defined. Each support must be followed by its identification number, coherent with the input simulation file (e.g. `Line 2` for the second line component defined in the input for the simulation with HEATER).

`NewPage`

Force a new plot page to be generated

`OutputFile` *name*

Set the name of the file for printed output (generated with the command `Print`). The default file name for printed output is `heaterpost.out`. The file name can be changed only before the first printed output is generated. The command is ignored if a printed output has already been generated on another file or on the default file.

`Plot` *target* $support_1$ $support_2$ … $support_n$

Generate n plot frames of *target* for the specified *support(s)* as a function of time or space according to the selection done (see the `Select` command).
Example: `plot temperature mesh 1`

`Plot` $target_1$ $support_1$ `vs` $target_2$ $support_2$

Plot $target_1$ of $support_1$ versus $target_2$ of $support_2$ at all times or space positions selected (see the `Select` command).
Example: `plot heat mesh 1 vs temperature mesh 1`

`PostScriptFile` *name*

Set the name of the file containing Postscript® output. The default file name for printed output is `heaterpost.ps`. The file name can be changed only before the first plot is generated. The command is ignored if a PostScript® output has already been generated on another file or on the default file.

`Print` $target_1$ $target_2$ … $target_n$ $support_1$ $support_2$ … $support_m$

Generate a table of n x m columns of the *target(s)* in the *support(s)* for every time or space coordinate selected (see the `Select` command). Note that several targets and supports can be printed simultaneously.
Example: `print heat temperature mesh 1`

`Query` *query option*

List to standard output the input setting of *query option*, this can be one of the *BlockName* identifiers as for the input simulation file (`Model`, `Mesh`, `Sets`, `Initial`, `Boundary`, `Source`, `Simulation`) or `All` to list the complete input set.

`Reset EndTime`

Reset the end time for plots and listings to the last simulation time stored in the binary storage file.

`Reset EndX`

Reset the final space coordinate to be used for plots and listings. The effect is different depending on the support. In the case of a support of type `Mesh`, `EndX` is set to the last node (index `NrNodes`) as specified in the simulation input. In the case of a support of type `Line`, `EndX` is set to the length of the line. This option has no effect on a `Point`.

`Reset StartTime`

Reset the start time for plots and listings to the first simulation time stored in the binary storage file.

`Reset StartX`

Reset the initial space coordinate to be used for plots and listings. The effect is different depending on the support. In the case of a support of type `Mesh`, `StartX` is set to the first node (index `1`). In the case of a support of type `Line`, `StartX` is set to zero. This option has no effect on a `Point`.

`Select Time` t_1 t_2 … t_n

Select from the binary storage file the results at times closest to the specified times. The following `Plot` and `Print` commands will report the results as function of the spatial coordinate at the n requested times. The selection is overridden by a following `Select` command.

`Select X` x_1 x_2 … x_n

Select from the binary storage file the results at the specified locations. The following `Plot` and `Print` commands will report the results as function of the time at the the requested locations. The effect is different depending on the support. In the case of a support of type `Mesh`, the locations are node indices. In the case of a support of type `Line`, the location is intended as the coordinate along the length of the line. This option has no effect on a `Point`.The selection is overridden by a following `Select` command.

`Set Color on/off`

Switch among color coding and dashed-line coding (B/W) for curves plotted for different supports in the same plot frame, default is `off` (i.e. dashed-line coding).

`Set EndTime` t

Set the end time for plots and listings, default is the last time stored in the binary storage file.

`Set EndX` x

Set the final space coordinate to be used for plots and listings. The effect is different depending on the support. In the case of a support of type `Mesh`, the `EndX` is set to a specific node. In the case of a support of type `Line`, `EndX` is set to the coordinate along the line. This option has no effect on a `Point`.

`Set PlotsPerPage` n

Set the number of plots per page. The number n must be an integer equal to 1, 2, 3, 4 or 6, 6 being the default. Changing the number of plots per page will automatically generate the plots to a new page

`Set StartTime` t

Set the start time for plots and listings, default is the first time stored in the binary storage file.

`Set StartX` x

Set the initial space coordinate to be used for plots and listings. The effect is different depending on the support. In the case of a support of type `Mesh`, the `StartX` is set to a

specific node. In the case of a support of type `Line`, `StartX` is set to the coordinate along the line. This option has no effect on a `Point`.

`Stop`

Stop execution and close the session. An end-of-file during parsing of the command file results in the same effect.

`StorageFile` *name*

Set the name of the file containing the binary stored results from HEATER. The default file name for printed output is `heater.store`. Opening and reading of the binary storage file is automatic after parsing the first command. Therefore this command, if present, must be the first in the post-processing command file.

Supports and targets All plotting and print-out actions of the post-processor HEATERPOST need the selection of a target to be plotted/printed and the relative support. A target is a variables or an auxiliary quantity computed in the simulation (e.g. temperature). A support is the component on which the quantity is defined (e.g. line number 2). Target and support must be selected from a valid combination (e.g. temperature of line number 2). In the following table we report the keys for the valid combinations of targets and supports. Any invalid selection or combination of target and support results in a syntax error during parsing.

Support	Target	Units	Meaning
`Mesh`	`X`	(m)	X-coordinate of a node in the mesh
	`Y`	(m)	Y-coordinate of a node in the mesh
	`Z`	(m)	Z-coordinate of a node in the mesh
	`Temperature`	(K)	Temperature at a node in the mesh
	`Heat`	(W)	Heat flux at a node in the mesh
`Point`	`X`	(m)	X-coordinate of a point
	`Y`	(m)	Y-coordinate of a point
	`Z`	(m)	Z-coordinate of a point
	`Temperature`	(K)	Temperature at a point
	`Heat`	(W)	Point heat flux at the point
	`IntegratedHeat`	(W)	Integral heat flux at the point (identical to `Heat` at the given point)
`Line`	`X`	(m)	X-coordinate of a location along a line
	`Y`	(m)	Y-coordinate of a location along a line
	`Z`	(m)	Z-coordinate of a location along a line
	`Temperature`	(K)	Temperature at a a location along a line t
	`Heat`	(W/m)	Linear heat flux at a location along a line
	`IntegratedHeat`	(W)	Integral heat flux along the line
`Surface`	`IntegratedHeat`	(W)	Integral heat flux over the surface
`Volume`	`IntegratedHeat`	(W)	Integral heat flux over the volume

Chapter 6

User Routines

Warning User Routines give unlimited access to the data structure used by the main program. Improper programming of User Routines can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Linking user routines

The User Routines for HEATER are FORTRAN functions packaged in a series of files contained in the directory:

```
~/CryoSoft/usr/heater/code_x.x
```

(where `x.x` stands for the version you received) which you will have received with the standard installation. In order to customize the code you will need to write modified version of these files. We strongly suggest to create your own directory tree within the above directory, and to modify only copies of the User Routines in order to be able to safely retrieve the standard version at your wish. Once the modified routines are ready, you will need to compile them and link them to the standard part of the code, to produce a customized version of the executable of HEATER. For this purpose you can use the standard makefile

```
~/CryoSoft/etc/heater.make
```

that can be copied and modified. Once more we strongly suggest that you modify only a copy of the standard makefile. Refer to the installation guide [1] for more details on the use of the makefiles, compilation and link-editing of the program.

Calling protocol

The following sections describe the calling protocol for the User Routines. For clarity we have subdivided the description in sections that are either associated with the type of function or with the type of component involved. The convention followed for the definition of the FORTRAN type of variables is the same as described in Chapter 4.

The User Routines for HEATER are defined as FORTRAN `functions`. The `function` returns a single `real` or `integer` value that must be computed by the user within the routine. All parameters passed to the `function` must be regarded as input parameters and cannot be modified.

Note FORTRAN unit numbers above 50 are reserved by the CryoSoft library for internal use, and should not be allocated for read/write operations. Any allocation or use of units above 50 can result in I/O errors or malfunctions.

Properties of solid materials

The material properties can be defined by the user. The properties needed for the simulation are the material density (Kg/m^3), the thermal conductivity (W/m K) and the heat capacity (J/Kg K).

The routines and functions contained in file `userMaterials.f`.

```
real function UserDensity (MaterialName, XYZ, Temperature)
```

Returns the density (Kg/m^3) of the material. Called for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`XYZ`	R	(m)	array of nodal coordinates (x,y,z)
`Temperature`	R	(K)	temperature

```
real function UserConductivity (MaterialName, XYZ, Temperature)
```

Returns the thermal conductivity (W/m K) of the material. Called for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`XYZ`	R	(m)	array of nodal coordinates (x,y,z)
`Temperature`	R	(K)	temperature

```
real function UserSpecificHeat (MaterialName, XYZ, Temperature)
```

Returns the specific heat (J/Kg K) of the material. Called for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`XYZ`	R	(m)	array of nodal coordinates (x,y,z)
`Temperature`	R	(K)	temperature

Heating waveform

The heating of the volume sets can be defined using the routine described in this section. The heating power density (in W/m^3) is defined as a function of space and time for all volumes components independently.

The routines and functions contained in file `userSource.f`.

```
real function UserQSource (Time, Volume, Q, Q_Tau)
```

Returns the heat flux (W/m^3). Called if `QModel=user`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Volume`	I	(-)	volume
`Q`	R	(W/m^3)	volumetric heat flux density as from input
`Q_Tau`	R	(s)	end heating time, as from input

Initial conditions

The initial conditions in the mesh can be defined based on the node number and its coordinates.

The routines and functions contained in file `userInitial.f`.

```
real function UserTInitial (Node, XYZCoord, TInitial)
```

Returns the initial temperature (K). Called if `InitialCondition=user`.

Parameter	Type	Units	Meaning
`Node`	I	(-)	node index
`XYZCoord`	R	(m)	array of nodal coordinates (x,y,z)
`TInitial`	R	(K)	initial temperature, as from input

Boundary conditions

The boundary conditions can be defined in mesh points, along lines or on surfaces. The routines return values of heat flux, temperature or heat transfer coefficient and bulk temperature as defined from the boundary condition input. The same routines are used for boundaries on different geometric objects: point, line, and surface. The geometric boundary object is identified by its type and its index. The `Object` can only have the following values:

`Object = 'point'`	boundary condition at a point
`Object = 'line'`	boundary condition along a line
`Object = 'surface'`	boundary condition over a surface

The routines and functions contained in file `userBoundary.f`.

```
subroutine userBoundaryQ (Time, Object, Index, BCQ, Boundary,
                          Nodes, Q)
```

Returns the heat flux (W/m^2) at a boundary. Called if `BCvariable=heat` and `BCtype=user`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	simulation time
`Object`	C	(-)	type of boundary object (`point`, `line` or `surface`)
`Index`	I	(-)	index of the boundary object
`BCQ`	R	(W/m^2)	boundary heat flux, as from input
`Boundary`	I	(-)	index of the boundary condition
`Nodes`	I	(-)	number of nodes on the boundary
`Q`	R	(W/m^2)	array of `Nodes` entries containing boundary heat flux

```
subroutine userBoundaryT (Time, Object, Index, BCT, Boundary,
                          Nodes, T)
```

Returns the temperature (K) at a boundary. Called if `BCvariable=temperature` and `BCtype=user`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	simulation time
`Object`	C	(-)	type of boundary object (`point`, `line` or `surface`)
`Index`	I	(-)	index of the boundary object
`BCT`	R	(T)	boundary temperature, as from input
`Boundary`	I	(-)	index of the boundary condition
`Nodes`	I	(-)	number of nodes on the boundary
`T`	R	(K)	array of `Nodes` entries containing boundary temperature

```
subroutine userBoundaryTh (Time, Object, Index, BCT, BCh, Boundary,
                           Nodes, T)
```

Returns the bulk temperature (K) and the heat transfer coefficient (W/m^2 K) at a boundary. Called if `BCvariable=convection` and `BCtype=user`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	simulation time
`Object`	C	(-)	type of boundary object (`point`, `line` or `surface`)
`Index`	I	(-)	index of the boundary object
`BCT`	R	(T)	boundary bulk temperature, as from input
`BCh`	R	(W/m^{2}K)	boundary heat transfer coefficient, as from input
`Boundary`	I	(-)	index of the boundary condition
`Nodes`	I	(-)	number of nodes on the boundary
`T`	R	(K)	array of `Nodes` entries containing boundary bulk temperature
`BCh`	R	(W/m^{2}K)	array of `Nodes` entries containing boundary heat transfer coefficient

CHAPTER 7

Troubleshooting and Errors

Error messages are reported to the output ASCII log file and to the standard output. The form of a typical error report is the following

```
ERROR in procedure <procedure name>: <error message>
called by <calling procure> at position <n>
called by <calling procure> at position <m>
......
```

where <*procedure name*> is the name of the routine where the error occurred and <*error message*> reports a short description of the error situation. This line is followed by the trace of the <*calling procedure*> up to the main program. In case of queries about error conditions, please take care to report error messages completely, including the calling trace.

Errors can be generated at four different levels in the code:

- input parsing and syntax errors;
- data consistency errors;
- runtime errors;
- internal consistency errors.

Input parsing errors

Input parsing and syntax errors are detected during the interpretation of the input file. They indicate that the variable naming, the command syntax or the type and number of numerical data in the input file are incorrect. Verify syntax in the input file in this case.

Data consistency errors

Data consistency errors are detected when input data are not coherent among themselves and would result in a model that cannot be analyzed. Typical cases are selection of incompatible options, or input data out-of-range. Verify the consistency of the input data in this case.

Runtime errors

Runtime errors are detected either when the solver enters a physical or numerical instability, or when the size of the problem exceeds the maximum allowed. Physical instabilities can be triggered by improper setting of physical conditions (e.g. initial conditions or boundary

conditions), excessive transient conditions (e.g. very large heating powers), or because of incorrect values from solid properties. Verify input conditions in this case.

Numerical instabilities can be triggered by the use of very large time steps, coarse mesh, and algorithms with little to no damping. In case of numerical instability, attempt at reducing the maximum time step (value of `MaximumStep` in input), reducing the allowed integrator tolerance (value of `Tolerance` in input), or choosing a time integration method that is more robust (choose `EulerBackward` as `TimeMethod`).

The maximum size of the network that can be solved is determined by the requested memory allocation in the FORTRAN include file:

```
~/CryoSoft/src/heater/code_x.x/includes/parameters.inc
```

where a number of parameters are set statically. The main parameters affecting memory allocation are the following, with the associated meaning:

Parameter	Meaning
`MaxElements`	maximum number of elements
`MaxFaces`	maximum number of mesh faces (used to constitute a surface)
`MaxEdges`	maximum number of mesh edges (used to constitute a line)
`MaxPoints`	maximum number of points
`MaxLines`	maximum number of lines
`MaxEdgesPerLine`	maximum number of edges forming a single line
`MaxSurfaces`	maximum number of surfaces
`MaxFacesPerSurface`	maximum number of faces forming a single surface
`MaxVolumes`	maximum number of volumes
`MaxElementsPerVolume`	maximum number of elements forming a single volume
`MaxBCs`	maximum number of boundary conditions
`MaxBoundaries`	maximum number of boundary elements

The additional parameters `MaxWork4` and `MaxWork8` are set to accommodate the sparse system matrix in the equation solver, and need adjustment depending on the problem dimension. Contact us should the solver require more workspace memory.

The version of the code you received can be modified by adjusting these parameters as desired. The code then needs to be compiled and link-edited as explained in the installation manual you received [3].

Warning Modification of dimensioning parameters affects memory allocation. Improper programming of parameters can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Internal consistency errors

Internal consistency errors indicate corruption of the internal data structure of the program. An internal consistency error cannot be generated using the standard program and reading data from input only. However, they can be detected in case that customized User Routines with improper data handling are used. They diagnose a severe fault within the code. If you are

using User Routines, verify their consistency with the calling protocol. In case you are not using User Routines, report internal consistency errors to us.

CHAPTER 8

References

[1] CryoSoft Installation Manual, Version 8.2, 2021.

[2] CryoSoft, Solids - Properties of solid materials, Version 4.0, January 2021.

[3] CryoSoft Variables Manual, Version 1.0, 2016.