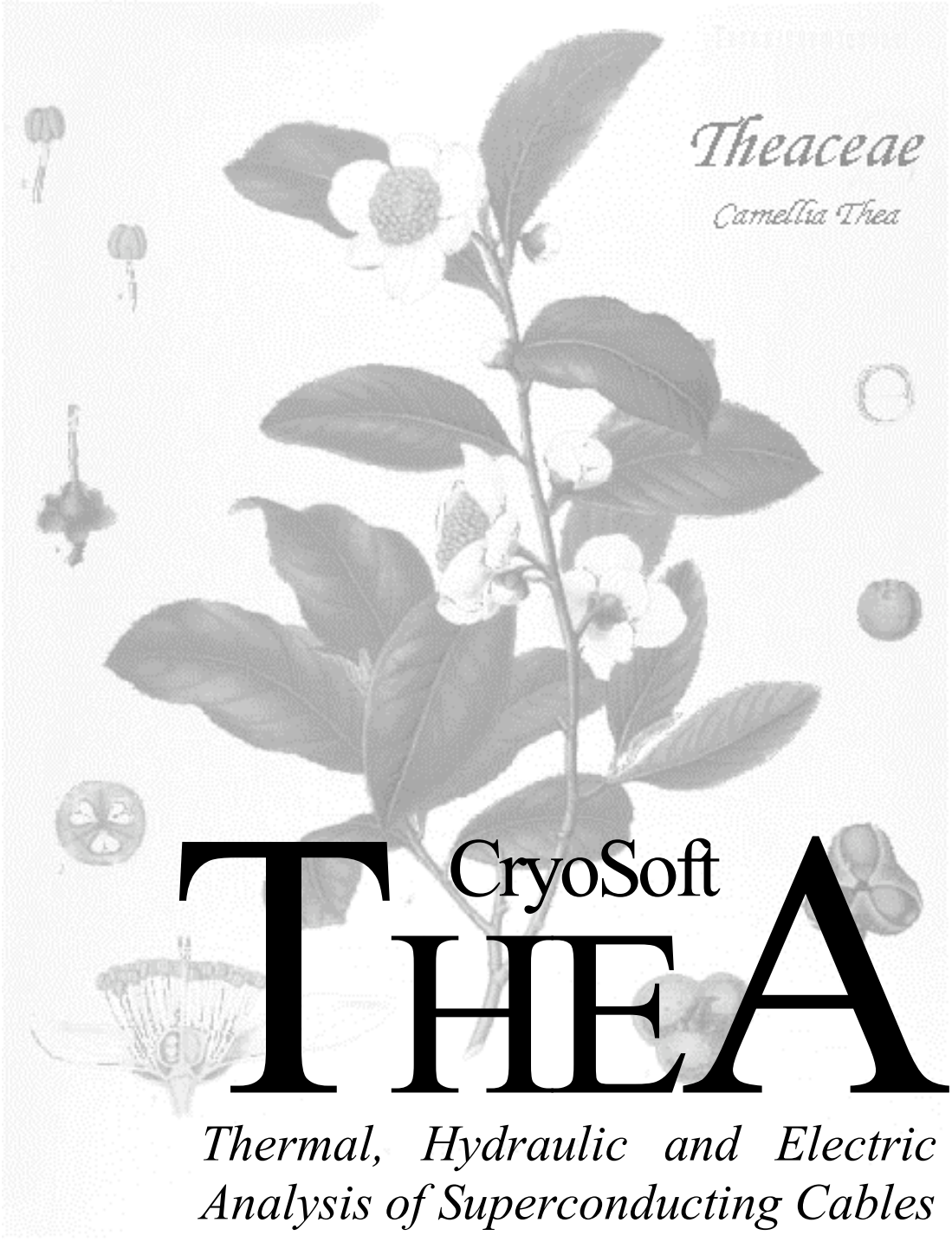
User's Guide

Version 2.4
November 2021



CryoSoft

THEA

Thermal, Hydraulic and Electric Analysis of Superconducting Cables

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 THEA and its post-processor, THEAPOST. Throughout this manual we assume that the reader is familiar with the physics and engineering issues that are associated with the design and analysis of a superconducting cable. Details on the physics modeling that is at the basis of the program are given in [1], [2] and [3]. We strongly suggest that the reader consults these 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 THEA run and launch the post-processor THEAPOST on a UNIX workstation.
- Chapter 3 contains case studies that the reader should use to familiarize 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 External 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 THEA 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 can be consulted as an indexed glossary for error messages and associated actions.

CHAPTER 1

Introduction

What is THEA

THEA is a computer program for the Thermal, Hydraulic and Electric Analysis of superconducting cables. THEA computes the evolution of the temperature, coolant flow and current distribution in a cable during fast transients such as stability perturbations and the following quench evolution, as well as slow transients such as normal operation ramps to steady state, or cool-down. In order to respond to the changing needs and evolving designs, we have designed THEA for maximum flexibility. As in other codes for the thermal and hydraulic analysis of superconductors, we have made in THEA the hypothesis that the conductor length is much larger than its transverse dimension, so that all phenomena can be dealt with in a 1-D approximation of the cable along its length. However, as compared to other similar codes, the main new features of THEA is that it allows:

- consistent, implicitly coupled analysis of thermal, hydraulic and electric transients in conductors;
- an arbitrary, user controlled configuration for the superconducting cable, additional structural components, cooling channels;
- variable geometry and properties along the conductor.

A THEA model

To achieve the modeling capability the superconductor cross section is subdivided by the user in components that can be of one of the following three types:

- thermal components, that model 1-D heat diffusion in solids, external heat sources, Joule heat and heat exchange with other solids or with coolants. The state of thermal components is identified by the instantaneous temperature of the solid;
- hydraulic components, that model 1-D compressible flow in a channel exchanging heat with the channel wall, and exchanging mass, momentum and heat with other adjacent channels. The state of a hydraulic component is identified by the instantaneous pressure, temperature and velocity of the fluid in the channel;
- electric components, that model 1-D current diffusion among resistive and inductive current carrying materials. The state of an electric component is identified by the instantaneous value of the current.

Components of the same type identify physically distinct units in the conductor, e.g. different sub-cable units or different channels. Components of different type can describe different phenomena in physically overlapping units. This is the case for thermal and electric

components that model the thermal conduction and current distribution in the same set of strands in a cable. The user couples components of different type, e.g. thermal and hydraulic components to simulate heat convection at the surface of a cooling channel, or thermal and electric components to achieve consistent treatment of current distribution and heat transfer in a cable. See the case studies in Chapter 3 for more details on the process of subdivision and coupling.

PDE Solution

THEA solves for each component defined by the user a set of partial differential equations (PDEs), coupled among components whenever chosen by the user, and obtains at any time required the distribution in space, along the conductor length, for the state variable(s). The solution satisfies the initial conditions chosen and the boundary conditions set by the user.

To solve the system of PDEs, THEA uses independent space and time discretization. The space discretization is based on the finite element method, and uses 1-D lagrangian elements with at most fifth order shape functions. The initial mesh is automatically adapted in time to achieve the following objectives:

- track discontinuities such as quench propagating fronts, or lambda transitions in the case of superfluid helium hydrodynamics;
- achieve a user-defined interpolation error on any state variable;
- maintain the element size between maximum and minimum user-defined values.

The user can control the meshing process through the choice of element order and of parameters that affect adaptivity. 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.

Structure

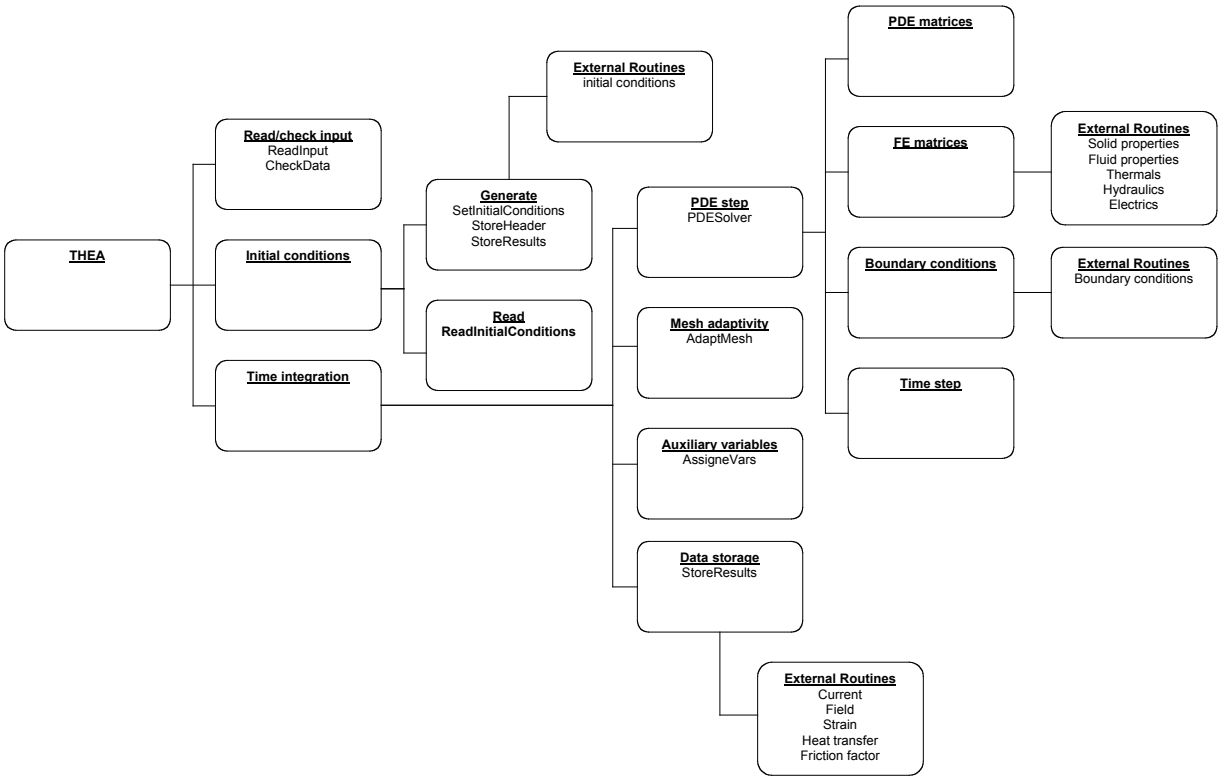
The overall structure of THEA is schematically shown below. THEA starts reading the data necessary to configure the run from an input file. It then checks the data for consistency. Depending on the type of run, it either initializes the state variables of the model (for a start-up run), or reads the state variables from a storage file (for a restart run). This stage is needed to determine the initial conditions for the time integration. When requested, the External Routines for customization of initial conditions are called once at this stage.

The time integration can then start, continuing until the end time is reached. The time integration consists of a loop that calls a solver routine at each step. The solver routine advances the solution by a single step in time over the complete space domain. This routine builds the matrix of the PDE to be solved. The External Routines for customization of material properties and characteristics of components are called at this stage, when user's defined materials are requested. The PDE solver then computes the finite element matrices and imposes the boundary conditions to the system. Here the External Routines are called in the case that the user has specified customized boundary conditions.

The calls to External Routines during the PDE solution are at the lowest level in the program tree. This implies that the calls are repeated several times during a run (typically millions to tens of millions of times for a practical problem). It is therefore very important that the user provides an efficient implementation for these routines.

After the solution has been advanced, the mesh is adapted according to the criteria set by the user.

Auxiliary variables (e.g. magnetic field, cable current, strain, fluid properties, transport properties such as friction factor, etc.) necessary for the solution of the system are computed after the solution of the system, at each step. The External Routines for the customization of the calculation of auxiliary variables are called at this point.



Post-processing

The results produced by THEA are integrally stored and can be analyzed to produce plots and reports by the post-processor THEAPOST. THEAPOST 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

THEA has several features that allow to customize its modeling capability beyond the allowable parameterization of the thermal/hydraulic/electric 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 *External 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 *External Routines*;

- change parametrically the behavior of the *External 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 THEA

Platforms

THEA and its post-processor THEAPOST 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 THEA is developed is:

- Macintosh running MacOS-X (10.10.5 and higher) under XQuartz,(2.7.4) 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 THEA and THEAPOST 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

THEA is one of the CryoSoft family of programs. You will have therefore received the CryoSoft package containing THEA either as a tar-ball or in pre-installed form. Verify in the CryoSoft installation manual [4] the procedure to be followed for the proper installation of the complete package. The executable codes, `thea` and `theapost` are in the directory `~/CryoSoft/bin/`. You will find the example inputs and post-processing command files in the directory `~/CryoSoft/xample/thea/code_x.x/` (the symbol `~/` stands for your home directory, `x.x` for the version you received)

How to run THEA

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

```
~/CryoSoft/bin/thea [-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. THEA 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`):

```
THEA Enter input file name
file.input
```

THEA 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 stepping algorithm selected. At each time step THEA 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 THEA prints a message reporting the total CPU time used in the run:

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

Each run of THEA 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 `thea.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 `thea.log`.

Restart After a successful 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 THEA 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 THEA:

```
~/CryoSoft/bin/thea
THEA Enter input file name
file.restart
```

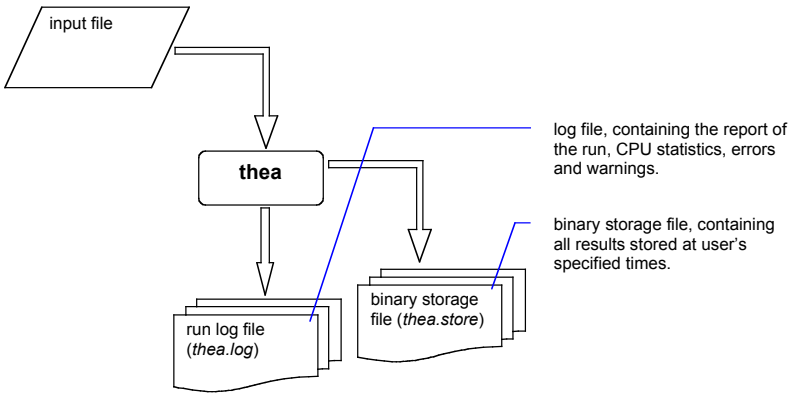
in which case THEA reads the binary storage file and starts the simulation at the last time stored:

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

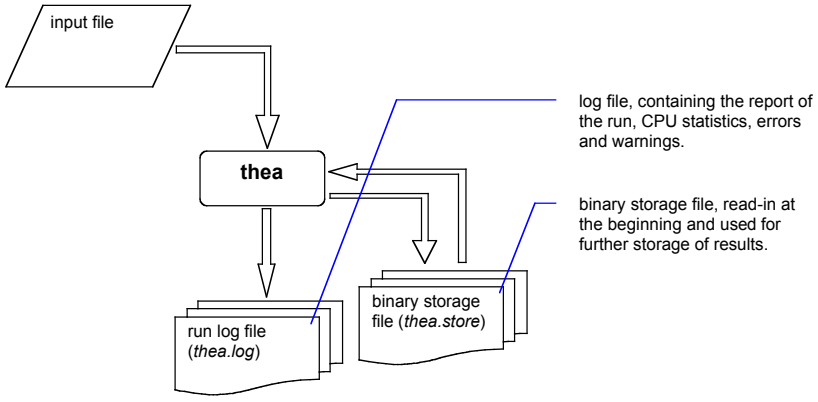
Until the final time specified in the input file `file.restart` is reached.

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 THEA run:



as compared to the flow-diagram of a THEA 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 THEAPOST

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

```
~/CryoSoft/bin/theapost [-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:

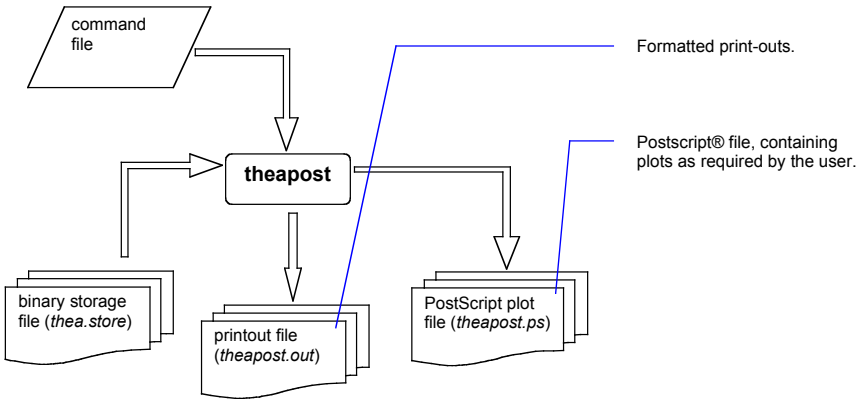
`-i, --input`	use `InputFile` to parse the input for the post-processor
`-v, --verbose`	print post-processing progress on stdout (default)
`-s, --silent`	no output to stdout
`-h, --help`	print a help message

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
```

THEAPOST 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 THEA (by default from the file `thea.store`). As a result THEAPOST generates:

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



Customization

The method described earlier provides the standard manner to run a THEA simulation, and post-process the results. THEA, however, as most other CryoSoft codes, gives the possibility to customize the physical models by using External 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 External Routines need to be compiled and linked the program segments to generate the customized version of the code. Template for the External Routines are given in the directory `~/CryoSoft/usr/thea/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 External 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, THEA requires an input file with all definitions necessary to specify the model structure, its characteristics, the operating conditions, initial and boundary 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 THEA results using the post-processor THEAPOST 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. In the last example reported we show how o use Eternal Routines. Using External Routines is the most advanced way to customize the operation of THEA.

Refer to Chapter 2 on how to run the examples described here with THEA and how to generate results and plots with THEAPOST.

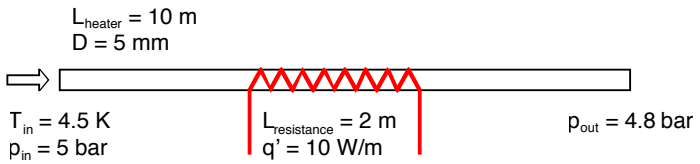
Note All input files and post-processing command files, the makefile and user routine files for the case studies discussed in this manual are provided with the standard installation. They are located in the directory:

`~/CryoSoft/xample/thea/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.

A heater for supercritical helium

Physical definition of the problem We consider the case of a heater pipe of 5 mm diameter and 10 m length supplied with a flow of supercritical helium at 4.5 K, 5 bar inlet pressure and 4.8 bar outlet pressure. A resistance deposits a power of 20 W over a length of 2 m in the middle of the pipe. The purpose of the analysis is to determine the helium flow and the evolution of the helium outlet temperature during the heating pulse, assuming that the heat pulse lasts a long time compared to the residence time of the helium. In this simple example we assume that the pipe has a negligible heat capacity, so that it can be neglected and the heat can be modeled as a direct input into the helium flow. The helium flow has a friction factor as determined by the turbulent correlation of Blasius.



Input file for the start-up run The problem requires the definition of a single hydraulic component, with given inlet and outlet conditions and heated at the location of the resistive heater. The step-by-step definition of the input file for `THEA` start-up run is shown below.

`heater.input`

Define the global model characteristics and parameters: a title used for labeling output and plots, total heater length (10 m), no electric current, magnetic field and strain.

```
Begin Model
```

Note the use of apex to delimit a text containing special characters or blanks.

```
  ModelName             'Supercritical helium heater'

  Length                10.0
  CurrentModel          none
  MagneticFieldModel    none
  StrainModel           none

end
```

Define the details of the hydraulic components.

```
Begin Hydraulics
```

A single component is defined, with cross section Area, hydraulic diameter Dh and hydraulic properties constant along the length. The Blasius correlation is used for the friction factor, and the Dittus-Boelter is used for the heat transfer coefficient.

```
  Components            1
  Fluid                 helium
  Model                 constant
  Area                  19.6e-6
  Dh                    5.0e-3

  fModel                Blasius
  hModel                DB
```

The helium has 4.5 K initial temperature, 5 bar initial pressure and zeroflow. These conditions are constant along the length.

```
InitialCondition     constant
TInitial             4.5
pInitial             5.0e5
mdotInitial          0.0
```

A heating power Q of 10 W/m is present in space between Q_XBegin and Q_XEnd and is constant in time.

```
QModel               constant
Q                    10.0
Q_XBegin             4.0
Q_XEnd               6.0
```

Boundary conditions are of prescribed pressure and temperature at both left and right side of the length analysed (infinite reservoir). The value of pressure and temperature in the two reservoirs is constant in time. On the left boundary the temperature is 4.5 K and the pressure 5 bar, on the right boundary the temperature is 4.5 K and the pressure 4.75 bar.

```
BoundaryType         reservoir reservoir
BoundaryConditions   constant  constant
TBoundary            4.5       4.5
pBoundary            5.0e5     4.75e5

end
```

Define the simulation parameters (numerics), storage and output.

```
Begin Simulation
```

The mesh is automatically generated using the given number of elements NrElements uniformly distributed in space. The element type is determined by the number of nodes ElementNodes and the interpolation order ElementOrder.

```
MeshType             uniform
NrElements           100
ElementOrder         2
ElementNodes         3
```

The time integration starts at StartTime and ends at EndTime, with output of the results every OutputStep

```
StartTime            0.0
EndTime              1.0
OutputStep           0.05
```

The time integration uses backward-differences (2nd order accurate in time). The time step is adapted automatically between the lower limit MinimumStep and the upper limit MaximumStep. The step adaption method StepEstimate is based on smooth decrease/increase, performed depending on the comparison of the estimated integration error and the desired tolerance Tolerance. The time integration error is estimated based on the change in the solution during a time step. As the error control ErrorControl is on, each time step is iterated changing the time step until the error is smaller than the desired tolerance.

```
TimeMethod           BackwardDifference
MinimumStep          1.0e-6
MaximumStep          1.0
StepEstimate         smooth
ErrorEstimate        change
```

```
  ErrorControl          on
  Tolerance             3.0e-2
```

Log output is directed to the file heater.log, while results are stored in the file heater.store for later restart and reporting

```
  LogFile               heater.log
  StorageFile           heater.store

end
```

Input file for the restart run To proceed with the simulation for a longer time than 1 s (the `EndTime` specified in the start-up run) we use the restart feature of THEA. Below we give the step-by-step definition of the input file for the restart of the simulation with a reduced time resolution in the storage of results and changing the time integration method.

`heater.restart`

In case of restart only the simulation parameters are needed. All other parameters are taken from the storage file generated during the previous run.

```
Begin Simulation
```

The presence of the Restart keyword is necessary to trigger a restart run.

```
  Restart
```

The time integration starts at the last time stored on file heater.store (as specified below) and proceeds to the new EndTime, with the prescribed OutputStep.

```
  EndTime                   5.0
  OutputStep                0.1
```

The time integration method is changed to Crank-Nicolson (2nd order accurate) while the other method options are left unchanged.

```
  TimeMethod                CrankNicolson
```

Log output and results are appended to the existing files during the restart.

```
  LogFile                   heater.log
  StorageFile               heater.store

end
```

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

`heater.post`

Define the file where results are stored.

```
StorageFile heater.store
```

Define the file for Postscript® output.

```
PostScriptFile heater.ps
```

Define the file for printed output.

```
OutputFile heater.out
```

The number of plots per page can be set to 1, 2, 3, 4 or 6.

```
set plotsperpage 4
```

Select the results of the simulation at the times closest to those below. All following plots are as f(x), the selected times are parameters.

```
select time 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5
```

Plot various quantities as f(x) selecting the quantity first, the component next.

```
plot pressure hydraulic 1
plot temperature hydraulic 1
plot velocity hydraulic 1
plot massflow hydraulic 1
```

Reselect times, this replaces previous selection.

```
select time 0.5 1.0 1.5 2.0 2.5 3.0
```

Plot quantities as f(x).

```
plot temperature hydraulic 1
plot velocity hydraulic 1
```

Plot now parametrically one variable versus a second variable

```
plot pressure hydraulic 1 vs temperature hydraulic 1
plot htc hydraulic 1 vs reynoldsnr hydraulic 1
```

Select the results of the simulation at the points with coordinate x closest to those below. All following plots are as a f(t), the selected x are parameters.

```
select x 1 3 5 7 10
```

Plot various quantities as f(t) selecting the quantity first, the component next.

```
set plotsperpage 6

plot pressure hydraulic 1
plot temperature hydraulic 1
plot velocity hydraulic 1
plot htc hydraulic 1
plot massflow hydraulic 1
```

Produce a printout of the heater outlet temperature as a function of time.

```
select x 10
print temperature hydraulic 1
```

The stop command terminates parsing, the post-processing session is finished.

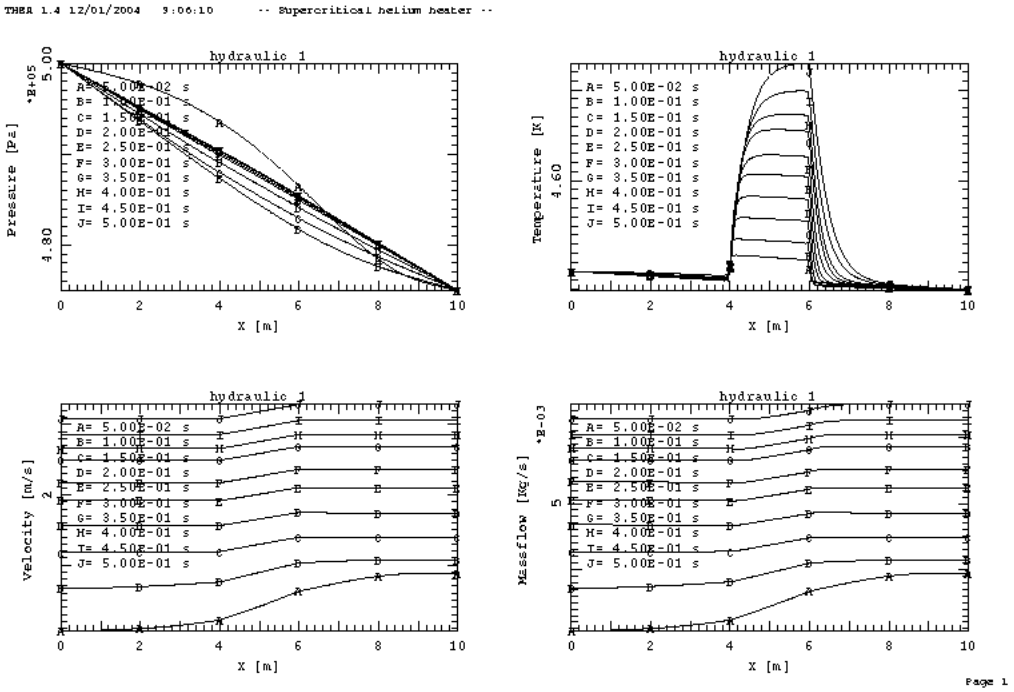
```
stop
```

Results Two files are generated running the post-processor THEAPOST with the commands described abovein the file `heater.post`: the PostScript output `heater.ps` and the ASCII output `heater.out`.

Note You will need a PostScript viewer to look at the plots in the PostScript file. The standard viewer, usually installed on UNIX systems, is `gs`. Try to launch the viewer with the command:

```
gs heater.ps
```

The plots below show the first page in the PostScript output `heater.ps`. As requested in the commands file, the first four plots are the pressure, temperature, velocity and massflow distributions along the heater at selected times. Note the establishment of a steady state pressure gradient along the pipe and the transient temperature increase under the heated region.



The file `heater.out` contains the output requested. In our case the only output requested is the temperature of the helium at the heater outlet (x=10 m). We report here only an abridged version of the full file.

`heater.out`

The following is the output of the results. In our case the temperature at x=10 m, at the outlet of the heater, as a function of time for all times stored in the binary storage file.

```
hydraulic 1
Time        temperature
[s]         [K]
             1.00E+01 m
--------------------------
 0.00E+00    4.50E+00
 5.00E-02    4.49E+00
 1.00E-01    4.48E+00
 1.50E-01    4.48E+00
```

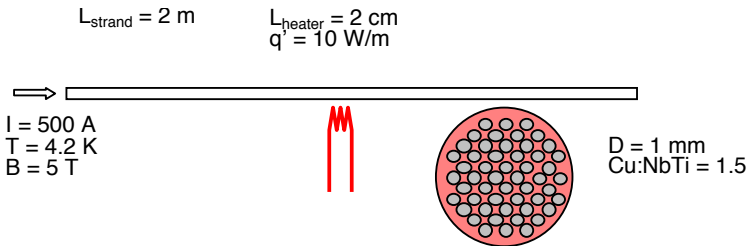
```
 2.00E-01    4.48E+00
```

..... *(lines omitted)*

```
 4.90E+00    4.94E+00
 5.00E+00    4.95E+00
```

Adiabatic strand quench

Physical definition of the problem This case study deals with the calculation of the evolution of temperature in a quenching NbTi strand assumed to be perfectly adiabatic. The strand is 2 m long (e.g. a critical current measurement sample) and is operated at 4.2 K, 500 A in a background field of 5 T. The strand itself has a diameter of 1 mm, is composed of Copper, with RRR of 100, and standard NbTi in a Cu:NbTi ratio of 1.5:1. The V-I resistive transition at the critical current is modeled using the power law approximation with a reference electric field of 1 μV/cm (10^{-4} V/m) and an exponent n of 20. The quench is initiated by a short heating pulse of 10 W/m with 1 ms duration and deposited over 2 cm length centered in the strand length corresponding to a total energy deposited of 0.2 mJ.



Input file for the run For this case we define a single thermal component, consisting of the two materials of the strand: Cu and NbTi. In absence of electric components, the total current in the cable is assumed to flow in the thermal model. The step-by-step definition of the input file is shown below.

`strand.input`

Define the global model characteristics and parameters: a title used for labeling output and plots, total strand length 2 m, total current 500 A, magnetic field 5 T and no strain.

```
Begin Model

  ModelName                 'single strand quench'

  Length                    2.0
  CurrentModel              constant
  InitialCurrent            500.0
  StrainModel               none
```

A left and right values are defined for the magnetic field. The field is interpolated linearly in x using the values below.

```
  MagneticFieldModel        constant
  MagneticFieldSS           5.0  5.0

end
```

Define the details of the thermal components.

```
Begin Thermals
```

Only one thermal component is defined: the composite NbTi strand, with constant cross section and properties along the length. The strand itself is made up of two materials: NbTi and Cu. The temperature is the same for all materials within a component.

```
  Components                1
```

```
Model                    constant

NrMaterials              2
Materials                Cu            Nb-Ti
```

Cross sections are defined for the two materials. For Cu the value of the RRR (residual resistivity ratio) is needed. Note that RRR must be given for all materials although it may not be necessary for the physical description. A dummy value (0) can be used. For the strand it is finally necessary to define the parameters of the power fit to the V-I curve for the critical current transition: E0 (electric field) and nPower (power law exponent).

```
Area                     0.4712e-6     0.3142e-6
RRR                      100.0         0.0
E0                       1.0e-4
nPower                   20
```

A heating power pulse with strength Q of 10 W/m is applied in space between Q_XBegin and Q_XEnd (2 cm between 0.99 and 1.01 m) and in time from 0 to Q_Tau (1 ms pulse).

```
QModel                   window
Q                        10.0
Q_Tau                    1.0e-3
Q_XBegin                 0.99
Q_XEnd                   1.01
```

The initial conditions are of constant temperature.

```
InitialCondition         constant
TInitial                 4.2
```

The right and left boundary conditions are of prescribed heat flux at the boundary. The heat flux is constant in time, and it is equal to zero. This corresponds to adiabatic boundary conditions.

```
BoundaryType             heat          heat
BoundaryConditions       constant      constant
qBoundary                0.0           0.0

end
```

Define the simulation parameters (numerics), storage and output.

```
Begin Simulation
```

The mesh is automatically generated using the given number of elements NrElements uniformly distributed in space. The element type is determined by the number of nodes ElementNodes and the interpolation order ElementOrder.

```
MeshType                 uniform
NrElements               200
ElementOrder             2
ElementNodes             3
```

The time integration starts at StartTime and ends at EndTime, with output of the results every OutputStep.

```
StartTime                0.0
EndTime                  5.0e-3
OutputStep               0.1e-3
```

The time integration uses the Crank-Nicolson method (2nd order accurate in time). The time step is adapted automatically between the lower limit MinimumStep and the upper limit MaximumStep. The step adaption method StepEstimate is based on smooth decrease/increase, performed depending on the comparison of the estimated integration error and the desired tolerance Tolerance. The time integration error is estimated based on the change in the solution during a time step. As the error control ErrorControl is on, the time step is iterated to achieve the desired tolerance.

```
  TimeMethod                CrankNicolson
  MinimumStep               1.0e-6
  MaximumStep               100.0e-3
  StepEstimate              smooth
  ErrorEstimate             change
  ErrorControl              on
  Tolerance                 1.0e-2
```

Log output is directed to the file strand.log, while results are stored in the file strand.store for later reporting and plots.

```
  LogFile                   strand.log
  StorageFile               strand.store

end
```

At this point the input definition is complete and execution starts.

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

`strand.post`

Define the file where results are stored.

```
StorageFile strand.store
```

Define the file for Postscript® output.

```
PostScriptFile strand.ps
```

The number of plots per page can be set to 1, 2, 3, 4 or 6.

```
set plotsperpage 2
```

Select the results of the simulation at the times closest to those below. All following plots are as f(x), the selected times are parameters.

```
select time 0.0 0.1e-3 0.2e-3 0.5e-3 1.0e-3 1.5e-3 2.0e-3 5.0e-3
```

Plot various quantities as f(x) selecting the quantity first, the component next.

```
plot temperature thermal 1
plot QJoule      thermal 1
```

Select the results of the simulation at the points with coordinate x closest to those below. All following plots are as a f(t), the selected x are parameters.

```
select x 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.9 1.0
```

Plot various quantities as f(t) selecting the quantity first, the component next.

```
plot temperature thermal 1
plot Resistance  thermal 1
```

Plot now parametrically one variable versus a second variable in the middle of the domain analysed.

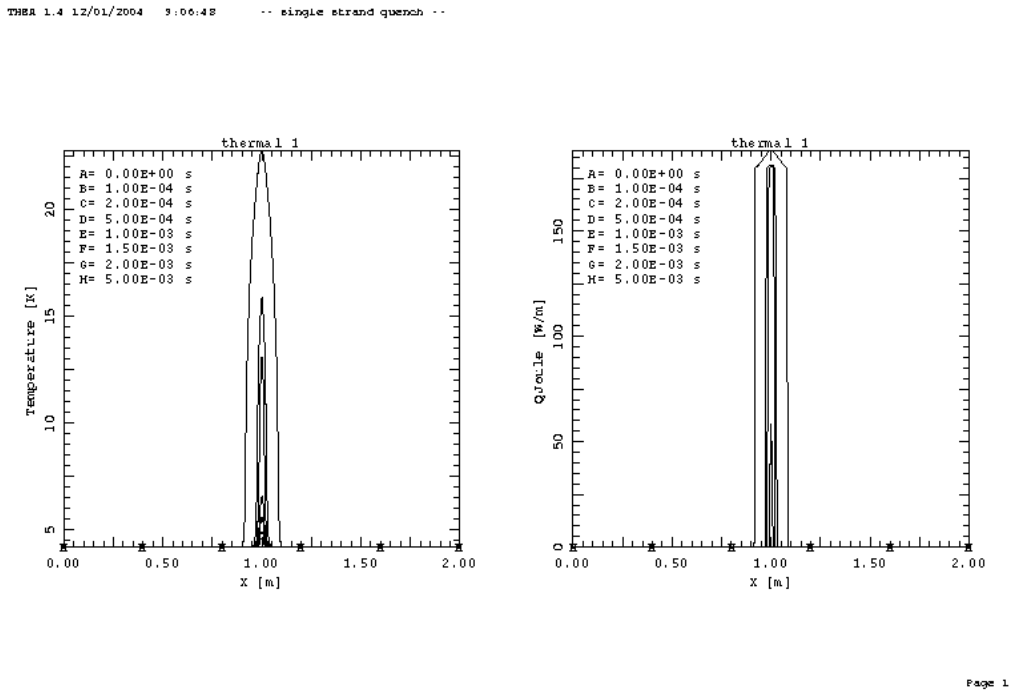
```
select x 1.0

plot resistance thermal 1 vs temperature thermal 1
plot QJoule thermal 1 vs temperature thermal 1
```

The stop command terminates parsing, the post-processing session is finished.

```
stop
```

Results In this case the output of the post-processor THEAPOST is the PostScript file `strand.ps`. The plots below show the first page in the PostScript output, and in particular they show the evolution of the temperature of the strand and of the Joule heat power density. Note the propagation of the normal zone from the center (the heated region) towards the ends.



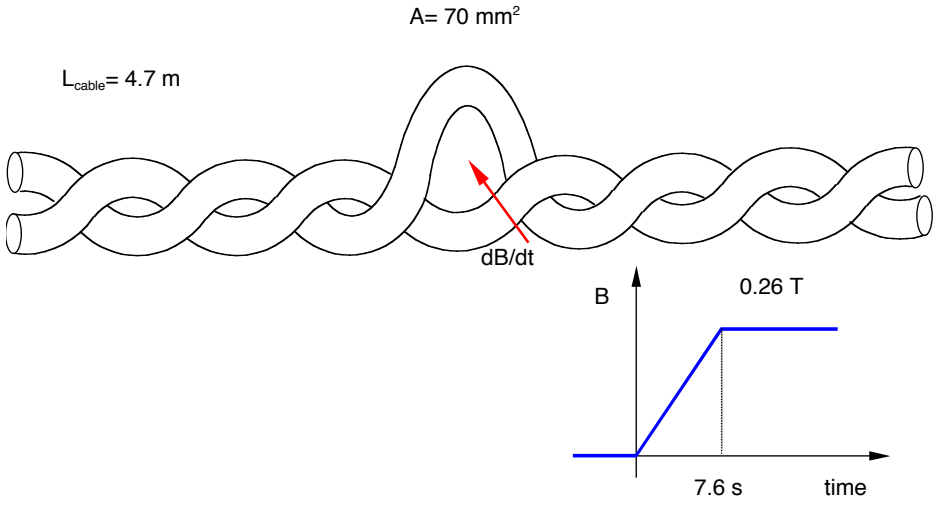
Current distribution in a two-strand cable

Physical definition of the problem In this case we model the current diffusion in a cable formed by two strands with a transposition error localised over a short length and subjected to an external field change. The conditions selected here are close to the experimental conditions used by Krempaski and Schmidt, and reported in [5].

The cable has a total length of 4.7 m, and the transposition error is an additional loop placed in the centre of the cable length. The loop has a cross section of 70 mm^2, and it is assumed to be smeared over a length of 10 mm. Over this length the change of the external magnetic field causes a voltage difference between the two strands proportional to the cross section of the loop and to the field ramp-rate. The effect of the field variation on the other regions of the cable is neglected. For the loop we take a field ramp-rate of 0.26 T in 7.6 s, thus resulting in a loop voltage of 2.4 μV. This voltage is applied over a length of 10 mm, so that the voltage per strand unit length is then 240 μV/m. This voltage is applied to the first strand, taking as reference (zero voltage) the second one.

The cable is soldered along its length, and the interstrand conductance is 52 MSiemens. The self inductance of a strand is 0.836 μH/m, and the mutual inductance between the two strands is 0.557 μH/m.

The problem is symmetric, and therefore only one half of the domain need to be analysed. The symmetry axis is place at the middle of the central loop. The symmetry boundary condition (left boundary) is expressed by a constant and zero voltage difference between the two strands (the center of the cable is the electrical axis). The boundary condition at the cable end (right boundary) models the fact that the strands are cut open and therefore no current can circulate at the boundary.



Input file for the run Two electric components are needed in this case, modelling the two strands. In the assumption of constant temperature, the two strands are superconducting. This results in zero longitudinal resistance, which is the default for electric components not linked to thermal components. Only the electric properties (transverse conductance and inductances) need then to be defined. The total cable current is set to zero (no transport current), and this condition is insured throughout the simulation. The input file is defined below.

`twostrand.input`

Define the global model characteristics and parameters. These are title, length, total cable current, field and strain.

```
Begin Model

  ModelName             'Current distribution'
```

The cable has a total length of 4.7 m. Here we assume symmetry in the center (see boundary conditions for electrics) and we analyse only half of the length.

```
  Length                2.35
```

The total cable current is constant and equal to zero. Neither magnetic field nor strain are defined. The effect of the magnetic field ramp is modeled as a voltage applied to the length of cable with the transposition error (see electric model).

```
  CurrentModel          constant
  InitialCurrent        0.0
  MagneticFieldModel    none
  StrainModel           none

end
```

Define details of electric model. These are electric parameters, voltage source, initial and boundary conditions for the two components defined

```
Begin Electrics

  Components            2
```

The electric (link) parameters are constant in space. In this case the inductance matrix of the 2 components is built as follows:

Self Mutual
Mutual Self

while the transverse conductance among the couple of electric components is constant and equal to Conductance, i.e. the matrix is built as follows:

0.0 Conductance
Conductance 0.0

The user should take care that the parameters are such that the resulting matrices are physically consistent.

```
  Links_Model           constant
  Self                  8.36e-7
  Mutual                5.57e-7
  Conductance           5.20e+7
```

The voltage in the electric 1 has a given value Voltage between V_XBegin and V_XEnd in space and between 0 and V_Tau in time. No voltage is applied to electric 2. Values for all parameters are needed for BOTH components, although they are not used for electric 2.

```
  VModel                window      none

  Voltage               2.4E-4      0.0
  V_XBegin              0.0         0.0
  V_XEnd                0.005       0.0
  V_Tau                 7.6         0.0
```

Both strands have initial current uniform in space, equal to zero.

```
  InitialCondition      constant  constant
```

```
    IInitial              0.0       0.0
```

The type of boundary conditions to be imposed is defined for both sides of both electric components - the order in the definition matters ! The left boundary of both components is of imposed voltage type, while the right boundary is of imposed current type. Types, flags and values are given in the following order:

left boundary, electric 1 right boundary, electric 1
left boundary, electric 2 right boundary, electric 2
left boundary, electric 3 right boundary, electric 3
left boundary, electric 4 right boundary, electric 4

and so on. The number of boundary conditions that can be imposed is equal to the total number of electric components minus one. This is a must to guarantee that current is conserved at the boundary as well.

```
    BoundaryType          voltage    current
```

The boundary conditions are constant in time. The values of the boundary currents and voltage differences are all needed although only some values are used (e.g. voltage on left boundary, current on right boundary). Note that repetition can be simplified using the keyword Nx where N stands for the number of entries to be taken equal. The entry below:

```
    BoundaryConditions    2x constant
```

means that the interpreter expands it during reading to the following equivalent:

```
    BoundaryConditions    constant   constant
```

note also that repetition of assignment is not an error. Useful for testing.

```
    IBoundary             0.0        0.0
    VBoundary             0.0        0.0

end
```

Define the simulation parameters (numerics), storage and output.

```
Begin Simulation
```

The mesh is automatically generated using the given number of elements NrElements uniformly distributed in space. The element type is determined by the number of nodes ElementNodes and the interpolation order ElementOrder.

```
    MeshType              uniform
    NrElements            1000
    ElementNodes          3
    ElementOrder          2
```

The time integration starts at StartTime 0 s and ends at EndTime 15 s, with output of the results every OutputStep 0.5 s.

```
    StartTime             0.0
    EndTime               15.0
    OutputStep            0.5
```

The time integration uses the Crank-Nicolson method (2nd order accurate in time). The time step is not adapted, as StepEstimate is set to none. No error estimate is provided (ErrorEstimate set to none) and as a consequence no iterative error control is possible

(ErrorControl is set to none). The effect of this combination is to perform time integration with a constant time step, equal to the minimum MinimumStep

```
  TimeMethod            CrankNicolson
  MinimumStep           0.05
  MaximumStep           0.05
  StepEstimate          none
  ErrorEstimate         none
  ErrorControl          none
```

Log output is directed to the file twostrand.log, while results are stored in the file twostrand.store for later reporting

```
  LogFile               twostrand.log
  StorageFile           twostrand.store

end
```

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

twostrand.post

Define the file where results are stored.

```
StorageFile twostrand.store
```

Define the file for PostScript® output.

```
PostScriptFile twostrand.ps
```

The number of plots per page can be set to 1, 2, 3, 4 or 6.

```
set plotsperpage 2
```

Select the results of the simulation at the points with coordinate x closest to those below. All following plots are as a f(t), the selected x are parameters.

```
select x 0 0.1 0.2 0.5
```

Plot various quantities as f(t) selecting the quantity first, the component next. The same quantity can be plotted on different components on the same plot

```
plot current     electric 1    electric 2
plot VExternal   electric 1    electric 2
```

Select the results of the simulation at the times closest to those below. All following plots are as f(x), the selected times are parameters.

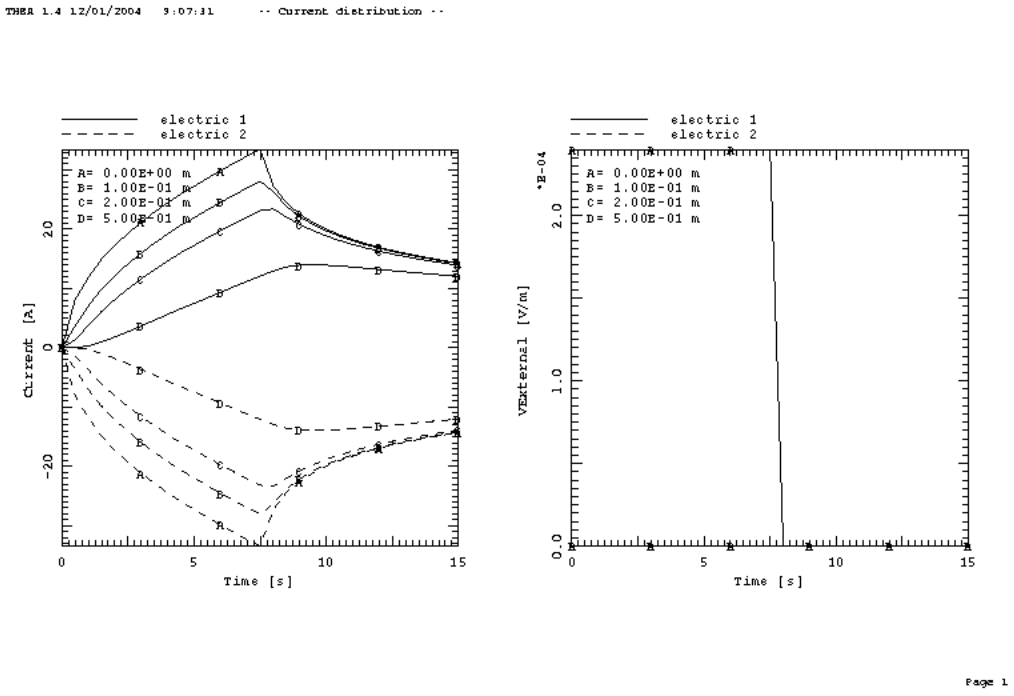
```
select time 1 2 5 10 15
```

Plot quantities as f(x).

```
plot current     electric 1
plot current     electric 2
```

The execution stops automatically at the end-of-file.

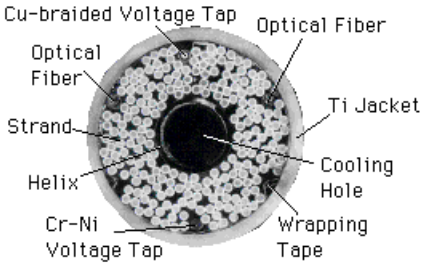
Results The results of the post-processor THEAPOST is the PostScript file `twostrand.ps`. The plots below show the first page in the PostScript output. The first plot contains the current in both the electric components plotted as a function of time at different locations x selected along the length of the cable. Note how the currents have been combined in a single plot. Similarly the second plot reports the external longitudinal voltage applied in the two components. According to the definition of this case a voltage is applied only to the first component during a time window of 7.6 s.



Page 1

Quench in a CICC with central cooling hole

Physical definition of the problem In this case we compute the evolution of the temperature in the QUELL conductor during a quench. The QUELL (Quench on Long Length) experiment was performed at EPFL/CRPP Villigen (CH) in the SULTAN test facility. This conductor has a central cooling hole and is a downsized model of an ITER-EDA CICC. Details on the experiment can be found in [6].



The QUELL cable can be modeled in first approximation as a bundle of strands forming the cable, cooled by intimate flow of helium. A large cooling channel is present in the middle of the conductor, separated by a loose spiral from the helium cooling the cable. The conductor has a Titanium alloy jacket which is externally insulated and epoxy impregnated. The sample length is approximately 100 m. The cable is supposed to operate at 8 kA current, in a 10 T magnetic field. The helium has inlet conditions at 4.5 K and 5 bar, outlet at 4.75 bar. The boundary are assumed to provide constant inlet and outlet helium state, while the cable, jacket and insulation are assumed to be adiabatic.

Input file for the start-up run For this case we assume that the current distribution within the cable is uniform, and we define three thermal components for the cable strands, the jacket and the insulation. The total current is distributed resistively and instantaneously among them. The thermal components are coupled among themselves through thermal resistance, and are cooled by heat convection with the helium flow in the bundle and in the cooling hole. These two flows are modelled as separate and coupled hydraulic components. The step-by-step definition of the input file is given below.

`quell.input`

Define the global model characteristics and parameters: a title used for labeling output and plots, total cable length 100 m, total current 8 kA, magnetic field 10 T and no strain.

```
Begin Model

  ModelName             QUELL

  Length                100.0
  CurrentModel          constant
  InitialCurrent        8000.0
  StrainModel           none
```

A left and right values are defined for the magnetic field. The field is interpolated linearly in x using the values below.

```
  MagneticFieldModel    constant
  MagneticfieldSS       10.0     10.0

end
```

Define the details of the thermal components.

```
Begin Thermals
```

Three thermal components are defined: the Cu/Nb3Sn superconducting strands that model the cable bundle, a Ti jacket and a composite glass-epoxy insulation. The components have constant cross section and properties along the length. The temperature evolution is computed for each component separately. The first component, the strands, have a composite structure made up of two materials, the Nb3Sn and the Cu stabilizer. The temperature is assumed to be the same for all materials within a component. Cross sections are defined for all materials. For Cu the value of the RRR (residual resistivity ratio) is needed (not necessary for other materials). For the strand it is finally necessary to define the parameters of the power fit to the V-I curve for the critical current transition: E0 (electric field) and nPower (power law exponent). Note that values and properties must be given for all materials (e.g. RRR) or all components (e.g. E0, nPower) although they may not be necessary for the physical description (and are not used).

```
Components          3

Model               constant            constant    constant

NrMaterials         2                   1           1
Materials           Cu        Nb3Sn     Ti          GE-warp
Area                60.8e-6   40.6e-6   73.5e-6     61.0e-6
RRR                 100.0     0.0       0.0         0.0
E0                  1.0e-4              0.0         0.0
nPower              20                  0           0
```

A heating power pulse with strength Q of 50 kW/m is applied in space between Q_XBegin and Q_XEnd (45 and 55 m) and in time from 0 to Q_Tau (10 ms).

```
QModel              window              none        none
Q                   5.0e+4              0.0         0.0
Q_Tau               10.0e-3             0.0         0.0
Q_XBegin            45.0                0.0         0.0
Q_XEnd              55.0                0.0         0.0
```

The initial conditions are of constant temperature in all components.

```
InitialCondition    constant            constant    constant
TInitial            4.5                 4.5         4.5
```

The boundary conditions are of prescribed heat flux at the boundary. The heat flux is constant in time, and it is equal to zero. This corresponds to adiabatic boundary conditions.

```
BoundaryType        heat      heat
                    heat      heat
                    heat      heat
BoundaryConditions  constant  constant
                    constant  constant
                    constant  constant
qBoundary           0.0       0.0
                    0.0       0.0
                    0.0       0.0
```

The thermal resistances define the thermal contact among the thermals and in this case they are given as a matrix. The order matters, the thermal resistances are in the following sequence:

Thermal 1 <---> Thermal 2 Thermal 1 <---> Thermal 3

Thermal 2 <---> Thermal 3

A value of 1 (K m / W) is taken between cable and jacket, and a value of 0.1 (K m / W) between between jacket and insulation. The thermal resistance among cable and insulation, on the other hand, is very high (ideally infinite)

```
Links_Model           matrix

ThermalResistanceMatrix    1.0    1.0e6
                           0.1


end
```

Define the details of the hydraulic components.

```
Begin Hydraulics
```

Two hydraulic components are defined, the first in intimate contact with the cable bundle, and the second contained in a cooling hole in the cable. The cross section Area, hydraulic diameter Dh are defined for both as constant along the length. The Katheder correlation is used for the friction factor of hydraulic 1, the Blasius correlation for hydraulic 2. The Dittus-Boelter correlation is used for the heat transfer coefficient of both hydraulics.

```
Components            2
Fluid                 helium
Model                 constant   constant
Area                  71.4e-6    19.6e-6
Dh                    0.865e-3   5.0e-3

fModel                Katheder   Blasius
hModel                DB         DB

Links_Model           constant
WettedPerimeter       15.7e-3
Perforation           1.0e-2
```

The helium has 4.5 K initial temperature, 5 bar initial pressure and 5 g/s initial flow. These conditions are constant along the length.

```
InitialCondition      constant   constant
TInitial              4.5        4.5
pInitial              5.0e5      5.0e5
mdotInitial           5.0e-3     5.0e-3
```

No heating power is input directly in the channels

```
QModel                none       none
```

Boundary conditions are of prescribed pressure and temperature at both left and right side of the length analysed (infinite reservoir). The value of pressure and temperature in the two reservoirs is constant in time. On the left boundary the temperature is 4.5 K and the pressure 5 bar, on the right boundary the temperature is 4.5 K and the pressure 4.75 bar.

```
BoundaryType          reservoir  reservoir
                      reservoir  reservoir
BoundaryConditions    constant   constant
                      constant   constant
TBoundary             4.5        4.5
                      4.5        4.5
```

```
  pBoundary            5.0e5       4.75e5
                       5.0e5       4.75e5

end
```

Define the details of the wetted perimeter among thermal and hydraulic components.

```
Begin Links
```

The S_H_Links_Model determines that the wetted perimeter is a constant along the length. The order matters, the links are in the following sequence:
Thermal 1 <---> Hydraulic 1 Thermal 1 <---> Hydraulic 2
Thermal 2 <---> Hydraulic 1 Thermal 2 <---> Hydraulic 2
Thermal 3 <---> Hydraulic 1 Thermal 3 <---> Hydraulic 2
The wetted perimeter is then defined for each link, in the same sequence

```
  S_H_Links_Model      constant    constant
                       constant    constant
                       constant    constant
  WettedPerimeter      0.33        15.7e-3
                       5.1e-2      0.0
                       0.0         0.0

end
```

Define the simulation parameters (numerics), storage and output.

```
Begin Simulation
```

The mesh is automatically generated using the given number of elements NrElements uniformly distributed in space. The element type is determined by the number of nodes ElementNodes and the interpolation order ElementOrder.

```
  MeshType             uniform
  NrElements           250
  ElementOrder         1
  ElementNodes         2
```

The time integration starts at StartTime and ends at EndTime, with output of the results every OutputStep.

```
  StartTime            0.0
  EndTime              5.0e-3
  OutputStep           5.0e-4
```

The time integration uses the Galerkin algorithm (1st order accurate in time). The time step is adapted automatically between the lower limit MinimumStep and the upper limit MaximumStep. The step adaption method StepEstimate is based on smooth decrease/increase, performed depending on the comparison of the estimated integration error and the desired tolerance Tolerance. The time integration error is estimated based on the change in the solution during a time step. As the error control ErrorControl is none, the time step is not iterated to achieve the desired tolerance, thus reducing execution time.

```
  TimeMethod           Galerkin
  MinimumStep          1.0e-5
  MaximumStep          1.0
  StepEstimate         smooth
  ErrorEstimate        change
  ErrorControl         none
  Tolerance            3.0e-2
```

Log output is directed to the file quell.log, while results are stored in the file quell.store for later reporting and plots.

```
  LogFile              quell.log
  StorageFile          quell.store

end
```

Input file for the restart run Below we give the step-by-step definition of the input file for a restart of the simulation performed using the input given above.

quell.restart

In case of restart only the simulation parameters are needed. All other parameters are taken from the storage file generated during the previous run.

```
Begin Simulation
```

The presence of the Restart keyword is necessary to trigger a restart run.

```
  Restart
```

The time integration starts at the last time stored on file quell.store (as specified below) and proceeds to the new EndTime, with the prescribed OutputStep.

```
  EndTime              500.0e-3
  OutputStep           2.0e-3
```

Log output is directed to the file quell.log, while results are stored in the file quell.store for later reporting and plots.

```
  LogFile              quell.log
  StorageFile          quell.store

end
```

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

quell.post

Define the file where results are stored.

```
StorageFile quell.store
```

Define the file for Postscript® output.

```
PostScriptFile quell.ps
```

Define the file for printed output.

```
OutputFile quell.out
```

The number of plots per page can be set to 1, 2, 3, 4 or 6.

```
set plotsperpage 2
```

Select the results of the simulation at the times closest to those below. All following plots are as f(x), the selected times are parameters.

```
select time 0.0 5.0e-4 1.0e-3 2.0e-3 5.0e-3
```

Plot various quantities as f(x) selecting the quantity first, the component next. Note that several components can be selected at the same time to overplot curves.

```
plot temperature thermal 1 thermal 2 thermal 3
plot specificresistance  thermal 1 thermal 2
```

Changing the number of plots per page will automatically generate a new page

```
set plotsperpage 3

plot pressure    hydraulic 1 hydraulic 2
plot temperature hydraulic 1 hydraulic 2
plot velocity    hydraulic 1 hydraulic 2
```

Cause the present plot page to be completed and a new page to be open. This usually happens automatically every PlotsPerPage plots, and can be done manually to separate plots.

```
newpage
```

The times can be re-selected to have a different sampling of results

```
select time 10.0e-3 50.0e-3 100.e-3
```

Plot various quantities as f(x) selecting the quantity first, the component next.

```
set plotsperpage 2
plot temperature thermal 1 thermal 2 thermal 3
plot resistance  thermal 1 thermal 2 thermal 3
set plotsperpage 3
plot pressure    hydraulic 1 hydraulic 2
plot temperature hydraulic 1 hydraulic 2
plot velocity    hydraulic 1 hydraulic 2
```

The above cycle is repeated, opening a new page, selecting results at different times and plotting results as f(x).

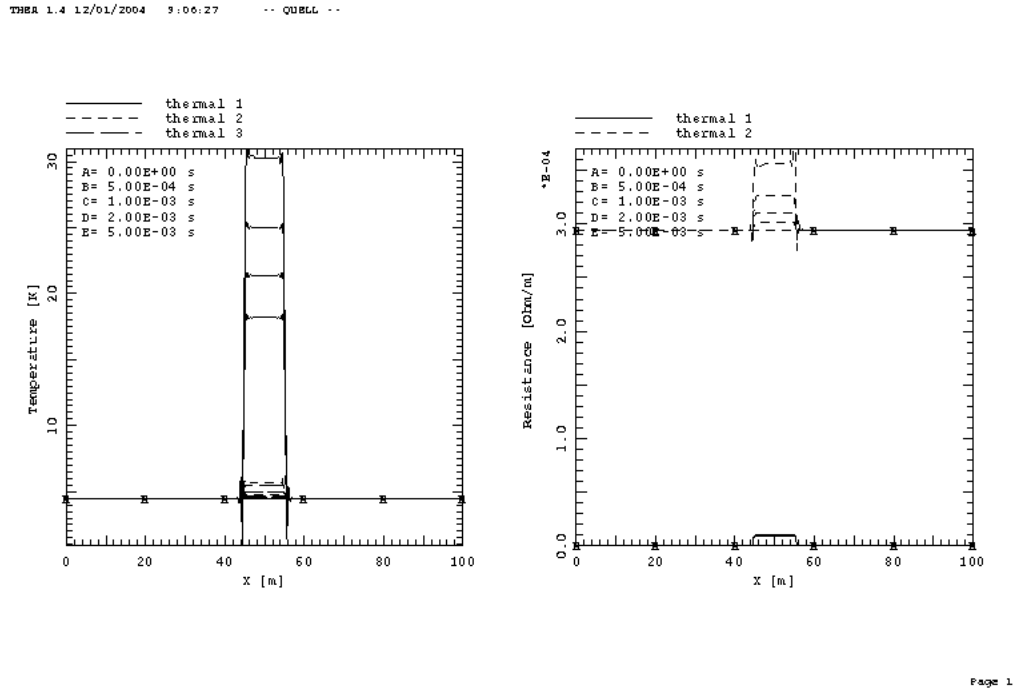
```
newpage
select time 200.0e-3 300.0e-3 400.0e-3 500.e-3
set plotsperpage 2
plot temperature thermal 1 thermal 2 thermal 3
plot resistance  thermal 1 thermal 2 thermal 3
set plotsperpage 3
plot pressure    hydraulic 1 hydraulic 2
plot temperature hydraulic 1 hydraulic 2
plot velocity    hydraulic 1 hydraulic 2
```

Select a location in the middle of the cable and produce a print-out of the temperatures of all components.

```
select x 50
print temperature thermal 1 thermal 2 thermal 3
print temperature hydraulic 1 hydraulic 2

stop
```

Results The results of the post-processor THEAPOST are in the PostScript file `quell.ps` and in the ASCII output file `quell.out`. The plots below show the first page in the PostScript output. The first plot contains the temperature evolution in the thermal components, plotted as a function of space at different locations times. The initiation of the quench is clear in the plot. The second plot reports the distribution of the resistance per unit length of the two conducting thermal components. Also there we can see clearly the normal zone with increased temperature, where the resistance grows.



The file `quell.out` contains the output requested. In our case the output requested is the temperature of the thermal and hydraulic components in the middle of the cable (x=50 m). We report here only an abridged version of the full file.

`quell.out`

The following is the output of the results. In our case first the temperatures of the thermal components at x=50 m, in the middle of the cable, as a function of time for all times stored in the binary storage file.

```
             thermal 1    thermal 2    thermal 3
Time         temperature  temperature  temperature
[s]          [K]          [K]          [K]
              5.00E+01 m   5.00E+01 m   5.00E+01 m
-------------------------------------------------
 0.00E+00     4.50E+00     4.50E+00     4.50E+00
 5.00E-04     1.82E+01     4.62E+00     4.58E+00
 1.00E-03     2.14E+01     4.76E+00     4.70E+00
 1.50E-03     2.34E+01     4.89E+00     4.82E+00

..... (lines omitted)

 4.97E-01     2.58E+01     1.95E+01     1.44E+01
 4.99E-01     2.58E+01     1.95E+01     1.44E+01
 5.00E-01     2.58E+01     1.95E+01     1.44E+01
```

Followed by temperatures of the hydraulic components at x=50 m.

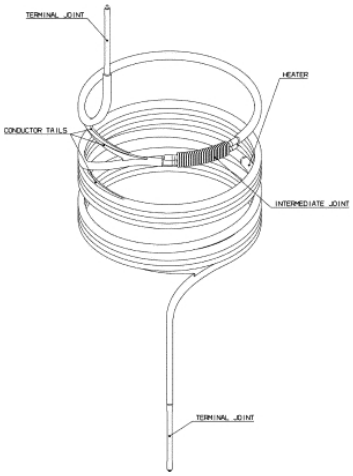
```
            hydraulic 1  hydraulic 2
Time        temperature  temperature
[s]         [K]          [K]
             5.00E+01 m   5.00E+01 m
-------------------------------------
 0.00E+00    4.50E+00     4.50E+00
 5.00E-04    4.58E+00     4.54E+00
 1.00E-03    4.69E+00     4.61E+00
 1.50E-03    4.82E+00     4.68E+00
 2.00E-03    4.97E+00     4.77E+00
 2.50E-03    5.12E+00     4.86E+00

..... (lines omitted)

 4.97E-01    1.88E+01     1.53E+01
 4.99E-01    1.89E+01     1.54E+01
 5.00E-01    1.89E+01     1.54E+01
```

Critical current measurement in a Nb-Ti CICC (External Routines)

Physical definition of the problem The example considered is the analysis of a critical current run in the PF Conductor Insert (PFCI) for ITER. This was a coil test that took place in 2008 in the CS Model Coil Test Facility at JAEA (Naka, Japan) to characterize the performance of one of the Nb-Ti CICC cables to be used in the ITER PF coils. A description of the test and its main results can be found in [12], which also contains reference dimensions for the insert coil. The figure below shows a cross section of the PFCI conductor, a dual-flow CICC with Cu/Nb-Ti strands in a steel jacket, and the 3-D rendering of the insert winding, which contains a joint not considered in this simple test case. The total cable length in the insert was 48.77 m (inlet to outlet).



The aim of this case study is to reproduce a critical current measurement, and more specifically run 27-2. During this run the temperature at inlet was set at 6.28 ± 0.05 K, the background field was 5.4 T, and the current was increased with a slow linear ramp till a quench was detected.

A complete simulation of the run, including all features of the PFCI, is beyond the scope of this simplified case study. This is why we focused here only on a few interesting customizations that are necessary to capture the main features of the experiment. Specifically:

- The conductor geometry has been reproduced based on the data reported in [12], and references therein;
- The critical current density of the NbTi material used was calibrated against an extensive set of measurements referenced in [12]. *This required defining a specific material property using External Routines*;
- Constant inlet and outlet conditions for the He flow were imposed, resulting in approximate flow conditions as measured in the experiment, using standard definitions for the friction factor and heat transfer coefficients (not calibrated to the conductor);
- The current was defined as a linear ramp lasting 300 s, attempting to reach 45 kA. A check is performed on the resistive voltage, and a quench dump with an exponential decay constant of 1 s is triggered if the resistive voltage exceeds 100 mV. *This required defining a specific material property using External Routines*;
- The magnetic field is taken as the sum of the background field imposed by the CS Model Coil (5.4 T) and the self field proportional to the PFCI current, with a proportionality constant of 15.7 mT/kA. The field profile along the length was ignored (in reality the field increases in the few meters of the insert), and the self field

quoted above corresponds to the peak field in the cable, which was found to match well the measured performance [12]. *This definition of the magnetic field required defining a specific material property using External Routines*.

Input files for the test case All files necessary to recreate the above conditions are contained in the directory:

```
~/CryoSoft/xample/thea/code_x.x/PFCI
```

which has been created to contain customized External Routines, a customized makefile, and (after compilation and link-editing) the customized executable code. This is the recommended way to organize specific work on parts of the External Routines, so to maintain a reference version of the code, and only modify local copies. The directory contains the following files:

`PFCI.input`	input file with standard format (described earlier in the manual), as well as reference to user definitions of operating current, magnetic field and a specific NbTi material;
`PFCI.make`	customized makefile, based on the standard THEA makefile, used to create a local version of THEA, including the desired customized features;
`PFCI.post`	post-processing command file, used to obtain plots after the simulation;
`obj`	a directory containing the object files of the External Routines, after compilation with the makefile script;
`usr`	a directory containing the source files of the External Routines, to be compiled using the makefile script. This directory contains the following files: `userCurrent.f` the External Routine for the current waveform; `userMagneticField.f` the External Routine for the field profile; `userSolids.f` External Routine for solids material properties.

We refer the reader to the specific files for comments on the actual inputs and programming solutions.

Running test case Before running this test case, please insure that a standard installation of THEA has been completed successfully. This is required because of the configuration settings (compilers, compiler options, libraries). All standard code segments should have been compiled (linked if available by the makefile script) and a standard code version available and tested (this a pre-requisite to verify that this THEA version can run in your installation). To run this test case the user will follow these steps:

Compile and link-edit the customized version of the code by using the makefile provided, using the command:

```
make –f  PFCI.make
```

The script compiles the External Routines, generates three object files in the `obj/` directory, and links the standard objects into a new executable file, `PFCI.thea`, that contains all custom features required. At this point the case can be run as a standard THEA run, using the new executable, as follows:

```
./PFCI.thea
```

the program will prompt for the input file, as in the case of a standard execution:

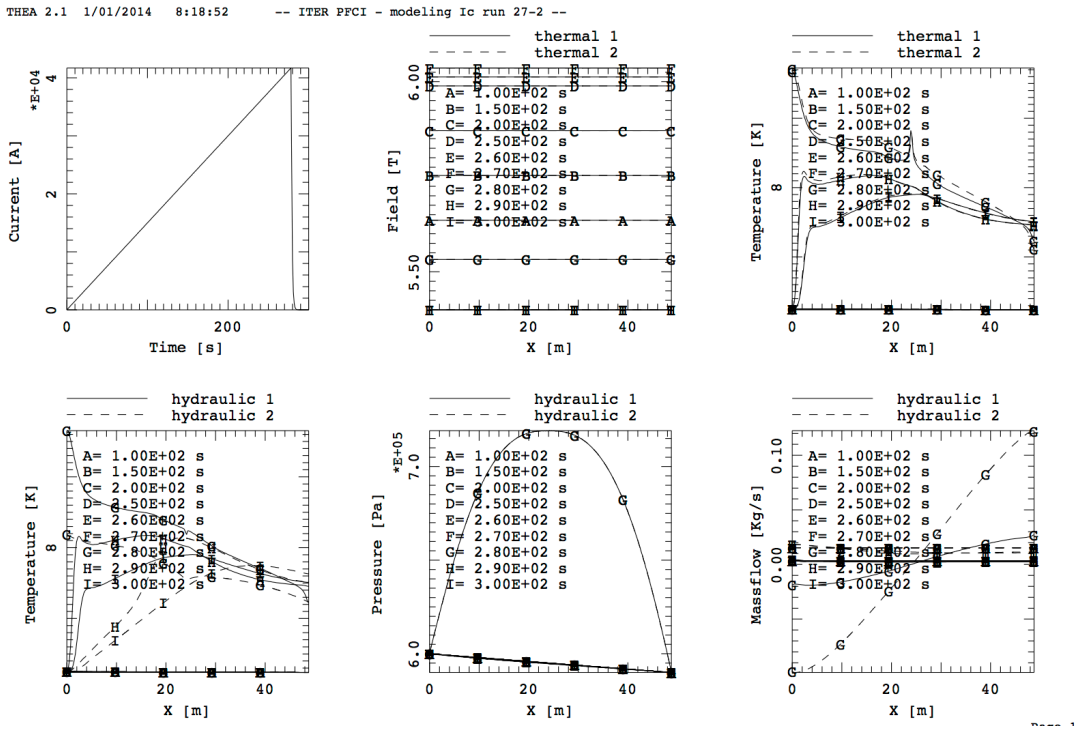
```
THEA Enter input file name
```

`PFCI.input`

Which can be followed by a standard post-processing run, using the command file `PFCI.post`.

Once again, as can be inferred from this brief explanation, and by examining the input and makefile, all customization, new executable, inputs and results are local to the directory of the test case, and in no ways they affect the standard code.

Results The results of the test case are shown below, where we report the first page of the PostScript® plot file generated by THEAPOST. The upper-left plot is the current waveforms, linear until a quench is triggered, at approximately 42 kA, and followed by the exponential fast dump.



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:

`Model`	define the general properties of the model
`Thermals`	define the number and properties of the thermal components
`Hydraulics`	define the number and properties of the hydraulic components
`Electrics`	define the number and properties of the electric components
`Links`	define the thermals-hydraulics and thermals-electrics links
`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;
- for an array of variables, the exact number of elements must follow the keyword. The expected number of elements is reported in the description of the variables below. If a keyword or a numeric entry *entry* is repeated N times within an array the alternative implicit syntax `Nx` *`entry`* can be used to shorten the input. In the case of matrices of entries, described below as arrays with 2 dimensions (e.g. boundary condition definitions) the implicit multiple definition applies to the first dimension of the matrix only;
- 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 (i.e. the total number of components must be available to read the physical characteristics of all components within a block);
- 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. This means that, if Electrics-Thermals and/or Hydraulics-Thermals links are requested, then the Electrics/Hydraulics and Thermals blocks must be assigned before the Links block. 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.

Model

The *model* block describes general quantities that apply to all components in the model being prepared. These are in particular total length and local elevation of the components (for the calculation of buoyancy effects), operating conditions such as total current, magnetic field and longitudinal strain. A title can be defined to identify the case. The title appears in plots and print-outs generated by THEAPOST.

Variable	Type	Units	Meaning
`CurrentModel`	C	(-)	Flag describing the behaviour of the current in time. Possible values: `user` user defined through the function `UserCurrent` (see Chapter 6). `none` no current defined (default). `constant` constant in time, equal to `InitialCurrent`

			`linear` combination of constant in time, equal to `InitialCurrent`, between 0 and `TauDetection`, followed by linear decay between `TauDetection` and `TauDetection` + `TauDump`. `exponential` combination of constant in time, equal to `InitialCurrent`, between 0 and `TauDetection`, followed by exponential decay after `TauDetection` with `TauDump` time constant. `External` the current 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.
`InitialCurrent`	R	(A)	Initial current, used for scaling the value of the current in time and also as a reference for scaling of the magnetic field and strain.
`Length`	R	(m)	Total length of the 1-D domain analysed (e.g. cable length)
`Height`	R	(m)	Array of 2 elements containing the elevation of the components at the left and right boundary.
`HeightModel`	C	(-)	Flag describing the definition of the local elevation z of the components along the length x. Possible values: `user` user defined through the function `UserHeight` (see Chapter 6). `none` no height defined (default to z=0) `linear` varying linearly in space. The value is obtained through linear interpolation along the total length between the left value `Height(1)` and the right value `Height(2)`.
`MagneticFieldAngle`	R	(T)	Array of 2 elements containing the value of the magnetic field angle at the left and right boundary, in degrees. The convention is an angle of 0 degrees for a field normal to the current direction, and an angle of 90 degrees for a field parallel to the current direction.
`MagneticFieldModel`	C	(-)	Flag describing the behaviour of the magnetic field in time and space. Possible values: `user` user defined through the function `UserMagneticField` and `UserMagneticFieldAngle` (see Chapter 6). `none` no magnetic field defined (default) `constant` constant in time and linear in space. The value is obtained through linear interpolation along the total length

between the left value `MagneticFieldSS(1)` and the right value `MagneticFieldSS(2)`. The field angle is obtained by linear interpolation along the total length between the left value `MagneticFieldAngle(1)` and the right value `MagneticFieldAngle(2)`.

`proportional` scaled proportionally to the cable current, linear in space. The magnetic field value is obtained as the sum of a component identical to the one described above, and a transient component obtained interpolating linearly in space between the left value `MagneticFieldTr(1)` and the right value `MagneticFieldTr(2)`. The transient component is scaled linearly with the ratio of instantaneous current to the initial current. The field angle remains always constant.

`MagneticFieldSS` R (T) Array of 2 elements containing the steady state value of the magnetic field at the left and right boundary.

`MagneticFieldTr` R (T) Array of 2 elements containing the transient value of the magnetic field at the left and right boundary.

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

`StrainModel` C (-) Flag describing the behaviour of the longitudinal strain in time and space. Possible values:

`user` user defined through the function `UserStrain` (see Chapter 6).

`none` no strain defined (default)

`constant` constant in time and linear in space. The value is obtained through linear interpolation along the total length between the left value `StrainSS(1)` and the right value `StrainSS(2)`.

`proportional` scaled proportionally to the square of the cable current, linear in space. The strain value is obtained as the sum of a component identical to the one described above, and a transient component obtained interpolating linearly in space between the left value `StrainTr(1)` and the right value `StrainTr(2)`. The transient component is scaled quadratically with the ratio of instantaneous current to the initial current.

`StrainSS`	R	(-)	Array of 2 elements containing the steady state value of the longitudinal strain at the left and right boundary.
`StrainTr`	R	(-)	Array of 2 elements containing the transient value of the longitudinal strain at the left and right boundary.
`TauDetection`	R	(s)	Delay time (detection) before the current dump.
`TauDump`	R	(s)	Dump time constant for the current decay.

Thermals

The *thermals* block describes the configuration and detailed properties for the thermal components. Thermal components are heat conducting solids, possibly superconducting, carrying a current and generating Joule heat. This block defines their number, material composition, cross sections and material properties. Heating can be set on a component-by-component basis. Thermal links within thermal components are defined through thermal resistances. In addition this block gives initial temperature and boundary conditions for the thermal components.

Note In the case of keywords associated with an array, all elements of the array must follow in the input file, even if these are not defined or are not used. Dummy values can be used.

Variable	Type	Units	Meaning
`Area`	R	(m^2)	Array of (`NrMaterials,Components`) elements containing the cross sections of all materials in each component. The entries must be in the following order: (m_i,T_j) (m_{i+1},T_j) (m_{i+2},T_j) … (m_i,T_{j+1}) (m_{i+1},T_{j+1}) (m_{i+2},T_{j+1}) … end so on, where m_i stands for the i-th material and T_j for the j-th thermal component.
`BoundaryConditions`	C	(-)	Array of (`2,Components`) elements containing the flag defining the time dependence of the boundary value. The entries must be in the following order: (b_1,T_j) (b_2,T_j) (b_1,T_{j+1}) (b_2,T_{j+1}) end so on, where b_i stands for the i-th boundary and T_j for the j-th thermal component. Possible values: `user` user defined through the functions `UserSTBoundary` and `UserQBoundary` (see Chapter 6). `constant` constant boundary value in time (default).
`BoundaryType`	C	(-)	Array of (`2,Components`) elements containing the flag defining the type of boundary. The entries must be in the following order: (b_1,T_j) (b_2,T_j) (b_1,T_{j+1}) (b_2,T_{j+1}) end so on, where b_i stands for the i-th boundary and T_j for the j-th thermal component. Possible values: `temperature` prescribed temperature at the boundary (default). `heat` prescribed heating power at the boundary.
`Components`	I	(-)	Number of thermal components defined.
`E0`	R	(V/m)	Array of `Components` elements containing the electric field used as reference for the definition of the critical

current Jc in the superconducting components. The longitudinal electric field (V-I curve) is computed using the power law fit with scaling field `E0` and power exponent `nPower`.

`InitialCondition` C (-) Array of `Components` elements containing the flags defining the initial conditions of temperature in each thermal component. Possible values:

- `user` user defined through the function `UserSTInitial` (see Chapter 6).
- `constant` constant in space, equal to `TInitial` (default).

`Links_Model` C (-) Flag defining the thermal links (thermal resistances) among thermal components. Possible values:

- `user` user defined for each couple of thermal components through the function `UserThermalResistance` (see Chapter 6).
- `none` no links among thermal components (default).
- `constant` all componentsd are thermally linked, and a single value `ThermalResistance` is used for all couples of thermal components, constant in space.
- `matrix` the components are thermally linked, and the thermal resistance for each couple of thermal components is given in `ThermalResistanceMatrix`.

`Materials` C (-) Array of (`NrMaterials,Components`) elements containing the material names. Details on the material properties can be found in the CryoSoft Solids Library [7]. The material name can be one of the predefined standard names or any user specified names.
In the case of a user's specified name the material properties and types are computed by the functions:

```
UserConductivity
UserDensity
UserResistivity
UserCriticalCurrentDensity
UserSpecificHeat
UserCriticalTemperature,
UserCurrentSharing
UserMaterialType
```

that must be provided by the user (see Chapter 6). At most one superconductor material per thermal component is allowed. The entries must be in the following order:

(m_i, T_j) (m_{i+1}, T_j) (m_{i+2}, T_j) …
(m_i, T_{j+1}) (m_{i+1}, T_{j+1}) (m_{i+2}, T_{j+1}) …

end so on, where m_i stands for the i-th material and T_j for the j-th thermal component.

`Model` C (-) Array of `Components` elements containing the flags defining the property variation in space. Possible values:

			`user` cross sections and material propertis are user defined through the functions `UserSArea`, `UserConductivity`, `UserDensity`, `UserRRR`, `UserE0`, `UsernPower`, `UserSpecificHeat`, `UserCriticalCurrent`, `UserCriticalTemperature`, `UserCurrentSharing`, `UserResistivity` (see Chapter 6). `constant` constant in time and space, as read-in from input (default).
`nPower`	I	(-)	Array of `Components` elements containing the exponent used for the power-law description of the longitudinal electric field in a superconductor (V-I curve). For a value of `nPower` larger than 250, a sharp transition is assumed.
`NrMaterials`	I	(-)	Array of `Components` elements defining the number of materials in the same component.
`RRR`	R	(-)	Array of (`NrMaterials`,`Components`) elements containing the Residual Resistivity Ratio of each material The entries must be in the following order: (m_i,T_j) (m_{i+1},T_j) (m_{i+2},T_j) … (m_i,T_{j+1}) (m_{i+1},T_{j+1}) (m_{i+2},T_{j+1}) … end so on, where m_i stands for the i-th material and T_j for the j-th thermal component.
`Q`	R	(W/m)	Array of `Components` elements defining the linear heat flux density, and used depending on the heating model `QModel` (see below).
`QBoundary`	R	(W)	Array of (`2`,`Components`) elements defining the heat flux in the left and right boundaries, used when the corresponding `BoundaryType = heat` The entries must be in the following order: (b_1,T_j) (b_2,T_j) (b_1,T_{j+1}) (b_2,T_{j+1}) end so on, where b_i stands for the i-th boundary and T_j for the j-th thermal component.
`QModel`	C	(-)	Array of `Components` elements containing the flag defining the model for heating along the length. Possible values: `user` user defined through the function `UserSHeating` (see Chapter 6). `none` no heating (default) `constant` linear power density equal to `Q` within the space frame between `Q_XBegin` and `Q_XEnd`, constant in time, zero otherwise. `window` linear power density equal to `Q` within the space frame between `Q_XBegin`

and Q_XEnd, from time 0 to Q_Tau, zero otherwise.

exponential — linear power density equal to Q within the space frame between Q_XBegin and Q_XEnd, exponential decay in time with time constant Q_Tau, zero otherwise.

external — the linear power density 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.

Q_Tau R (s) Array of Components elements containing the heating time constant.

Q_XBegin R (m) Array of Components elements containing the start coordinate of the heating window.

Q_XEnd R (m) Array of Components elements containing the end coordinate of the heating window.

TBoundary R (K) Array of (2,Components) elements defining the temperature at the left and right boundaries, used when the corresponding BoundaryType = temperature. The entries must be in the following order:
(b_1,T_j) (b_2,T_j)
(b_1,T_{j+1}) (b_2,T_{j+1})
end so on, where b_i stands for the i-th boundary and T_j for the j-th thermal component.

ThermalResistance R (K m/W) Thermal resistance among all couples of thermal components, used il Links_Model is constant. All thermal components are thermally linked through the value given.

ThermalResistanceMatrix R (K m/W) array of (Components,Components) elements defining the thermal resistance among all possible couples of thermal components (used if Links_Model is matrix). The thermal resistance for the thermal couple (i,j) is the same as the thermal resistance of the couple (j,i). For this reson only the upper triangle of the matrix is given in input, in the following order:

```
(1,2)  (1,3)   ...  (1,N-1)  (1,N)
       (2,3)   ...  (2,N-1)  (2,N)
               ...
               ...           (N-1,N)
```

for a total of Components*(Components-1)/2 entries.

`TInitial`	R	(K)	Array of `Components` elements containing the initial temperature in each thermal component (uniform in space).

Hydraulics

The *hydraulics* block describes the configuration and detailed properties for the hydraulic components. Hydraulic components are channels where the fluid flows. This block defines the fluid flowing in the channels, their number, cross sections, hydraulic properties and turbulent correlations. Heating can be set on a component-by-component basis. Links within hydraulic components are defined through heat and/or mass transfer among channels. In addition this block sets initial temperature, pressure, flow and boundary conditions for all hydraulic components.

Variable	Type	Units	Meaning
`Area`	R	(m^2)	Array of `Components` elements containing the cross sections of all hydraulic components.
`BoundaryConditions`	C	(-)	Array of `(2,Components)` elements containing the flag defining the time dependence of the boundary value. The entries must be in the following order: (b_1,H_j) (b_2,H_j) (b_1,H_{j+1}) (b_2,H_{j+1}) end so on, where b_i stands for the i-th boundary and H_j for the j-th hydraulic component. Possible values: `user` user defined through the functions `UsermdotBoundary` , `UserpBoundary` and `UserHTBoundary` (see Chapter 6). `constant` constant boundary value in time (default). `External` the boundary conditions are 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.
`BoundaryType`	C	(-)	Array of `(2,Components)` elements containing the flag defining the type of boundary. The entries must be in the following order: (b_1,H_j) (b_2,H_j) (b_1,H_{j+1}) (b_2,H_{j+1}) end so on, where b_i stands for the i-th boundary and H_j for the j-th hydraulic component. Possible values: `reservoir` prescribed temperature and pressure as provided by a large (infinite) volume reservoir connected at the end of the channel (default). `closed` closed pipe boundary (zero flow).
`Components`	I	(-)	Number of hydraulic components defined.

`CModel`	C	(-)	Array of `Components` elements containing the flag defining the model for convection along the length. Possible values: `user` user defined, through the heat transfer coefficient of the flow (see `hModel`), the wetted perimeter `T0_WP` and a wall temperature given by the function `UserHT0` (see Chapter 6). `none` no convection heating (default) `constant` convection heat transfer as computed from the heat transfer coefficient of the flow (see `hModel`), the wetted perimeter `T0_WP` and a constant wall temperature `T0`, within the space frame between `T0_XBegin` and `T0_XEnd`, zero otherwise. `external` the convection heat transfer 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.

Note Hydraulic heating either through surface convection (`CModel` different from `None`) or direct heating (`QModel` different from `None`) is mutually exclusive. Only one of the two input definitions is allowed.

`Dh`	R	(m)	Array of `Components` elements containing the hydraulic diameter of all hydraulic components.
`Fluid`	C	(-)	Name of the fluid flowing in all channels defined. Details on the material properties can be found in the CryoSoft Fluids Library [8]. The fluid name can be one of the following predefined standard names: `Helium` Single phase ^{4}He in any state, including superfluid `Nitrogen` Single phase N_2. Only one fluid can be defined for all channels defined, to avoid inconsistencies in case that flow mixing among channels is allowed through non-zero perforation of the channel walls.
`fModel`	C	(-)	Array of `Components` elements containing the flag defining the friction factor model. In general the friction factor is defined as the maximum value between the correlation chosen and the laminar flow limit. This limiting procedure is not used in the case of `fModel user` or `none`. Possible values: `user` user defined through the function `UserFrictionFactor` (see Chapter 6) `none` no friction factor (default)

			`constant` constant in time and space, equal to `FrictionFactor` as defined in input. `Blasius` Blasius correlation. `Katheder` Katheder correlation for CICC's with 40 % void fraction. `Nikuradse` Nikuradse-von Karman correlation. `Smooth` smooth tube correlation. `Westinghouse` Westinghouse correlation for CICC's. Details on the correlations can be found in [9].
`FrictionFactor`	R	(-)	Array of `Components` elements containing a constant value of the friction factor of the flow, used if `fModel` is `constant`.
`HTC`	R	(W/m^2 K)	Array of `Components` elements containing a constant value of the heat transfer coefficient of the flow, used if `hModel` is `constant`.
`hModel`	C	(-)	Array of `Components` elements containing the flag defining the heat transfer coefficient model. Possible values: `user` user defined through the function `UserHTC` (see Chapter 6). `none` no heat transfer coefficient (default). `constant` constant in time and space. `BLQ` Boundary layer filling with step in wall heat flux. `BLT` Boundary layer filling with step in wall temperature. `DB` Dittus-Bölter correlation. `DBG` Dittus-Bölter-Giarratano correlation for supercritical helium. `Kapitza` Kapitza thermal resistance. Details on the correlations can be found in [10].
`InitialCondition`	C	(-)	Array of `Components` elements containing the flags defining the initial conditions of temperature, pressure and flow in each hydraulic component. Possible values: `user` user defined through the function `UserHTInitial` `UserpInitial` and `UsermdotInitial` (see Chapter 6). `constant` constant in space, equal to `TInitial` , `pInitial` and `mdotInitial` (default).
`Links_Model`	C	(-)	Flag defining the nature of the thermal and mass links among hydraulic components. Possible values: `user` user defined for each couple of hydraulic components through the function `UserWettedPerimeter` and `UserPerforation` (see Chapter 6). `none` no links among hydraulic components (default). `constant` the hydraulic components are linked by heat and mass exchange through the wetted perimeter and perforation of the hydraulic

channels. A single value of `WettedPerimeter` and `Perforation` is used for all couples of hydraulic components, constant in space.

`matrix` the hydraulic components are linked by heat and mass exchange through the wetted perimeter and perforation of the hydraulic channels. The values of the wetted pertimeter and perforation for each couple of hydraulic channels are given in the `WettedPerimeterMatrix` and in the `PerforationMatrix`, constant in space.

`mdotBoundary` R (kg/s) Array of `(2,Components)` elements defining the massflow at the left and right boundaries, not used. The entries must be in the following order:
(b_1,H_j) (b_2,H_j)
(b_1,H_{j+1}) (b_2,H_{j+1})
end so on, where b_i stands for the i-th boundary and H_j for the j-th hydraulic component.

`mdotInitial` R (kg/s) Array of `Components` elements containing the initial massflow in each hydraulic component (uniform in space).

`Model` C (-) Array of `Components` elements containing the flags defining the cross section and properties variation in space. Possible values:

`user` cross sections and hydraulic diameter are user defined through the functions `UserHArea, UserDh,` (see Chapter 6).

`constant` constant in time and space, as read-in from input (default).

`pBoundary` R (Pa) Array of `(2,Components)` elements defining the pressure at the left and right boundaries, used when the corresponding `BoundaryType=reservoir`.
The entries must be in the following order:
(b_1,H_j) (b_2,H_j)
(b_1,H_{j+1}) (b_2,H_{j+1})
end so on, where b_i stands for the i-th boundary and H_j for the j-th hydraulic component.

`Perforation` R (-) Perforation factor of the wetted perimeter for each couple of hydraulic components, used if `Links_Model` is `constant`. The perforation factor is between 0 (no perforation) and 1 (full perforation) and governs mass transfer at the wetted perimeter of two channels.

`PerforationMatrix` R (-) Matrix of `(Components,Components)` elements containing the perforation factor of the wetted perimeter for each couple of hydraulic components, used if `Links_Model` is `matrix`. The perforation factor is between 0 (no perforation) and 1 (full perforation) and governs mass transfer at the wetted perimeter of two channels. The perforation factor for

the couple (i,j) of hydraulic i and hydraulic j is the same as the perforation factor of the couple (j,i). For this reason only the upper triangle of the matrix is given, in the following order:

```
(1,2)  (1,3)   ...  (1,N-1)  (1,N)
       (2,3)   ...  (2,N-1)  (2,N)
               ...
               ...           (N-1,N)
```

for a total of `Components*(Components-1)/2` entries.

`pInitial` R (Pa) Array of `Components` elements containing the initial pressure in each hydraulic component (uniform in space).

`Q` R (W/m) Array of `Components` elements defining the linear heat flux density, and used depending on the heating model `QModel` (see below).

`QModel` C (-) Array of `Components` elements containing the flag defining the model for heating along the length. Possible values:

`user`	user defined through the function `UserHHeating` (see Chapter 6).
`none`	no heating (default)
`constant`	linear power density equal to `Q` within the space frame between `Q_XBegin` and `Q_XEnd`, constant in time, zero otherwise.
`window`	linear power density equal to `Q` within the space frame between `Q_XBegin` and `Q_XEnd`, from time 0 to `Q_Tau`, zero otherwise.
`exponential`	linear power density equal to `Q` within the space frame between `Q_XBegin` and `Q_XEnd`, exponential decay in time with time constant `Q_Tau`, zero otherwise.
`external`	the linear power density 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.

Note Hydraulic heating either through surface convection (`CModel` different from `None`) or direct heating (`QModel` different from `None`) is mutually exclusive. Only one of the two input definitions is allowed.

`Q_Tau` R (s) Array of `Components` elements containing the heating time constant.

`Q_XBegin`	R	(m)	Array of `Components` elements containing the start coordinate of the heating window.
`Q_XEnd`	R	(m)	Array of `Components` elements containing the end coordinate of the heating window.
`T0`	R	(m)	Array of `Components` elements containing the wall temperature used for the calculation of the convection heat transfer, depending on the convection model `CModel`.
`T0_WP`	R	(m)	Array of `Components` elements containing the wetted perimeter used for the calculation of the convection heat transfer.
`T0_XBegin`	R	(m)	Array of `Components` elements containing the start coordinate of the convection heat transfer window, depending on the convection model `CModel`.
`T0_XEnd`	R	(m)	Array of `Components` elements containing the end coordinate of the convection heat transfer window, depending on the convection model `CModel`.
`TBoundary`	R	(K)	Array of `(2,Components)` elements defining the temperature at the left and right boundaries, used when the corresponding `BoundaryType=reservoir`. The entries must be in the following order: (b_1,H_j) (b_2,H_j) (b_1,H_{j+1}) (b_2,H_{j+1}) end so on, where b_i stands for the i-th boundary and H_j for the j-th hydraulic component.
`TInitial`	R	(K)	Array of `Components` elements containing the initial temperature in each hydraulic component (uniform in space).
`WettedPerimeter`	R	(m)	Wetted perimeter for each couple of hydraulic components, used if `Links_Model` is `constant`. The wetted perimeter governs heat transfer and mass transfer (through the relative `Perforation`) between two channels.
`WettedPerimeterMatrix`	R	(m)	Matrix of `(Components,Components)` elements containing the wetted perimeter for each couple of hydraulic components, used if `Links_Model` is `matrix`. The wetted perimeter governs heat transfer and mass transfer (through the relative `Perforation`) between two channels. The wetted perimeter for the couple (i,j) of hydraulic i and hydraulic j is the same as the wetted perimeter of the couple (j,i). For this reson only the upper triangle of the matrix is given, in the following order:

```
(1,2)  (1,3)   ...  (1,N-1)  (1,N)
       (2,3)   ...  (2,N-1)  (2,N)
               ...
               ...           (N-1,N)
```

for a total of `Components*(Components-1)/2` entries.

Electrics

The *electrics* block describes the configuration and detailed properties for the electric components. Electric component carry current. This block defines their number and electrical properties. Voltage sources can be chosen for each component. Finally this block is used to define initial current and boundary conditions for the electric components.

Variable	Type	Units	Meaning
`BoundaryConditions`	C	(-)	Array of (2,`Components-1`) elements containing the flag defining the time dependence of the boundary value for all the electric components up to the last one. No boundary condition can be prescribed for the last electric component as this equation is used to guarantee the total current conservation. The entries must be in the following order: (b_1,E_j) (b_2,E_j) (b_1,E_{j+1}) (b_2,E_{j+1}) end so on, where b_i stands for the i-th boundary and E_j for the j-th electric component. Possible values: `user` user defined through the functions `UserVBoundary` and `UserIBoundary` (see Chapter 6). `constant` constant boundary value in time (default).
`BoundaryType`	C	(-)	Array of (2,`Components-1`) elements containing the flag defining the type of boundary for all the electric components up to the last one. No boundary condition can be prescribed for the last electric component as this equation is used to guarantee the total current conservation. The entries must be in the following order: (b_1,E_j) (b_2,E_j) (b_1,E_{j+1}) (b_2,E_{j+1}) end so on, where b_i stands for the i-th boundary and E_j for the j-th electric component. Possible values: `current` prescribed current at the boundary. `voltage` prescribed voltage difference with respect to the last electric component at the boundary(default).
`Components`	I	(-)	Number of electric components defined.
`Conductance`	R	(1/Ωm)	Conductance per unit length among electric components (used if `Links_Model` is `constant`).
`ConductanceMatrix`	R	(1/Ωm)	Array of (`Components,Components`) elements defining the conductance per unit length among electric components (used if `Links_Model` is `matrix`). The conductance factor for the couple (i,j) of electric i and electric j is the same as the conductance of the couple (j,i). For this reason only the upper triangle of the matrix is given, in the following order: `(1,2) (1,3) ... (1,N-1) (1,N)`

```
        (2,3)   ...  (2,N-1)   (2,N)
                ...
                ...            (N-1,N)
```

for a total of `Components*(Components-1)/2` entries.

`IBoundary` R (A) Array of (2,`Components-1`) elements defining the current in the left and right boundaries, used when the corresponding `BoundaryType = current`. No boundary condition can be prescribed for the last electric component as this equation is used to guarantee the total current conservation.
The entries must be in the following order:
(b_1,E_j) (b_2,E_j)
(b_1,E_{j+1}) (b_2,E_{j+1})
end so on, where b_i stands for the i-th boundary and E_j for the j-th electric component.

`IInitial` R (A) Array of `Components` elements containing the initial current in each electric component (uniform in space).

`InductanceMatrix` R (H/m) Array of (`Components,Components`) elements defining the inductance per unit length among electric components (used if `Links_Model` is `matrix`). The inductance for the couple (i,j) of electric i and electric j is the same as the inductance of the couple (j,i). For this reason only the diagonal and the upper triangle of the matrix are given, in the following order:

```
(1,1) (1,2)   ...  (1,N-1)   (1,N)
      (2,2)   ...  (2,N-1)   (2,N)
                ...
                ...  (N-1,N-1) (N-1,N)
                ...            (N,N)
```

for a total of `Components*(Components+1)/2` entries.

`InitialCondition` C (-) Array of `Components` elements containing the flags defining the initial conditions of current for each electric component. Possible values:

`user`	user defined through the function `UserIInitial` (see Chapter 6).
`constant`	constant in space, equal to `IInitial` (default).

`Links_Model` C (-) Flag defining the electric links (transverse conductance and inductance) among electric components. Possible values:

`user`	user defined for each couple of thermal components through the functions `UserConductance` and `UserInductance` (see Chapter 6).
`constant`	the transverse conductance and inductance matrices are built using the input values of `Conductance`, `Self` and `Mutual`. The matrices are the same for all possible couples of components, and are constant in space (default).

matrix — the transverse conductance and inductance matrices are given in ConductanceMatrix and InductanceMatrix.

Mutual	R	(H/m)	Mutual inductance per unit length for any couple of electric components (used if Links_Model is constant).
Self	R	(H/m)	Self inductance per unit length for any electric component (used if Links_Model is constant).
RLongitudinal	R	(Ohm/m)	Array of Components elements containing the constant longitudinal resistance per unit length in each electric component (used if RModel is constant).
RModel	C	(-)	Array of Components elements containing the flag defining the model for longitudinal electric resistance of the electric component. Possible values: user — the longitudinal resistance is user defined through the function UserResistance (see Chapter 6). none — the longitudinal resistance is taken to be zero throughout the simulation. constant — the longitudinal resistance is constant in space and time, as defined by the value of the variable RLongitudinal. standard — the longitudinal resistance is computed consistently using the properties of the coupled thermal components. The default result when the electric component is not linked to a thermal component, or when no thermal components are present in the model, is zero longitudinal resistance, i.e. as if option none were chosen.
VBoundary	R	(V)	Array of (2,Components-1) elements defining the voltage difference among all components and the last component defined in the left and right boundaries, used when the corresponding BoundaryType = voltage. No boundary condition can be prescribed for the last electric component as this equation is used to guarantee the total current conservation. The entries must be in the following order: (b_1,E_j) (b_2,E_j) (b_1,E_{j+1}) (b_2,E_{j+1}) end so on, where b_i stands for the i-th boundary and E_j for the j-th electric component.
VModel	C	(-)	Array of Components elements containing the flag defining the model for the longitudinal voltage source along the length. Possible values: user — user defined through the function UserVoltage (see Chapter 6).

			none — no voltage (default) constant — linear voltage per unit length equal to Voltage within the space frame between V_XBegin and V_XEnd, constant in time, zero otherwise. window — linear voltage per unit length equal to Voltage within the space frame between V_XBegin and V_XEnd, from time 0 to V_Tau, zero otherwise. exponential — linear voltage per unit length equal to Voltage within the space frame between V_XBegin and V_XEnd, exponential decay in time with time constant V_Tau, zero otherwise.
Voltage	R	(V/m)	Array of Components elements defining the longitudinal voltage per unit length in the component, and used depending on the voltage model VModel (see above).
V_Tau	R	(s)	Array of Components elements containing the longitudinal voltage source time constant.
V_XBegin	R	(m)	Array of Components elements containing the start coordinate of the window of longitudinal voltage source.
V_XEnd	R	(m)	Array of Components elements containing the end coordinate of the window of longitudinal voltage source.

Links

The *links* block determines the coupling among different components. Such couplings are between thermal and hydraulic components (through heat transfer at the wetted perimeter) and between thermal and electric components (through Joule heat in the thermal components and resistance in the electric components). Coupling can be determined component by component in a structure of matrices containing either the type of coupling, the components coupled and the properties of the coupling.

Note All components must be defined before defining their mutual links. This means that the *links* block must come after the *thermal*, *hydraulic* and *electric* blocks in the input file. A parsing error is generated if this is not the case.

Variable	Type	Units	Meaning
`S_E_Links`	I	(-)	Matrix with size (`NrOfThermalsComponents`, `NrOfElectricComponents`) containing the entries for coupling thermal and electric components. The entry (i,j) for the thermal component i and the electric component j contains 0 for no coupling and 1 for coupling. The matrix is entered in the following order (T_1,E_1) (T_1,E_2) … (T_1,E_{NE}) (T_2,E_1) (T_2,E_2) … (T_2,E_{NE}) … (T_{NT},E_1) (T_{NT},E_2) … (T_{NT},E_{NE}) where (T_i,E_j) stands for the entry (0 or 1) of thermal component T_i and electric component E_j.
`S_H_Links_Model`	C	(-)	Matrix with size (`NrOfThermalsComponents`, `NrOfHydraulicComponents`) containing the flags determining the type of links among thermal and hydraulic components. The flags can be different for each couple. Possible values: `user` the thermal coupling happens on a wetted perimeter defined by the user for each couple of thermal and hydraulic component through the function `UserSHWettedPerimeter` (see Chapter 6). `none` no coupling (default). `constant` the thermal coupling takes place as defined by the `WettedPerimeter` and is constant in space. The matrix is entered in the following order (T_1,H_1) (T_1,H_2) … (T_1,H_{NH}) (T_2,H_1) (T_2,H_2) … (T_2,H_{NH}) … (T_{NT},H_1) (T_{NT},H_2) … (T_{NT},H_{NH}) where (T_i,H_j) stands for the flag entry of thermal component T_i and hydraulic component H_j.
`WettedPerimeter`	R	(m)	Matrix with size (`NrOfThermalsComponents`, `NrOfHydraulicComponents`) containing in the location (i,j) the wetted perimeter for each couple of a thermal component i and an hydraulic component j. The matrix is entered in the following order

(T_1,H_1)	(T_1,H_2)	…	(T_1,H_{NH})
(T_2,H_1)	(T_2,H_2)	…	(T_2,H_{NH})
		…	
(T_{NT},H_1)	(T_{NT},H_2)	…	(T_{NT},H_{NH})

where (T_i,H_j) stands for the wetted perimeter between thermal component T_i and hydraulic component H_j.

Simulation

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

Variable	Type	Units	Meaning

`AdaptivityMethod`	C	(-)	Switch for mesh adaptivity method. Possible values:
			`none` no adaptivity method selected
			`Threshold` the mesh is adapted based on a threshold condition on a value, i.e. the variable `AdptVariable` in the component `AdptIndex` of type `AdptComponent` crossing a pre-defined value `AdptValue`. The syntax of the command is:
			`Threshold` ***`AdptVariable AdptComponent AdptIndex`*** *AdptValue*
			The combination of the variable selection ***`AdptVariable`*** `and of component` ***`AdptComponent`*** can be one of the following:
			`Temperature Thermal`
			`Temperature Hydraulic`
			`Pressure Hydraulic`
			`Velocity Hydraulic`
			`Current Electric`
			`Front` the mesh is adapted based on a threshold condition on a derivative, i.e. the space derivative of the variable `AdptVariable` in the component `AdptIndex` of type `AdptComponent` crossing a pre-defined value `AdptValue`. The syntax of the command is:
			`Front` ***`AdptVariable AdptComponent AdptIndex`*** *AdptValue*
			The same combination of the variable selection ***`AdptVariable`*** `and of component` ***`AdptComponent`*** is possible as in the case of `Threshold` tracking (see above).

Note The `Front` tracking option is presently available for compatibility with future developments. The input is correctly parsed and checked, but no mesh adaptivity is performed at run time.

			`Lambda` the mesh is adapted to track the lambda transition (He-I to He-II) in the component `AdptIndex` of type hydraulic (i.e. in this case `AdptComponent` must be `hydraulic`). The syntax of the command is: `Lambda Hydraulic` ***`AdptIndex`***

`Quench` the mesh is adapted to track the quench front in the component `AdptIndex` of type thermal (i.e. in this case `AdptComponent` must be `thermal`). The syntax of the command is:

```
Quench Thermal AdptIndex
```

`ArtificialViscosity` C (-) Switch to control the amount of artificial viscosity added in the solution of the compressible fluid flow equations. Artificial viscosity is always needed to stabilize the solution of compressible flow at sharp discontinuities and moving fronts in the fluid. Possible values:

- `none` no artificial viscosity is applied. This choice can lead to large oscillation in flow quantities (velocity, pressure, temperature) especially at sharp fronts such as moving normal zones.
- `Lapidus` Second-order artificial viscosity as defined by Lapidus. The artificial viscosity is proportional to the velocity gradient and to the square of the element size through an empirical coefficient. This choice provides good smoothing for velocity and pressure at fronts, but is not effective for temperature fronts.
- `Upwind` First-order upwind. The artificial viscosity is proportional to the fluid speed and the the element size, resulting in optimal balancing of the hyperbolic transport term at high Peclet number. This choice is effective for velocity at fronts and large temperature gradients, but can lead to pressure oscillations during transients.

`ElementNodes` I (-) Number of nodes per element. This is in the range of 2 (linear element) to 6 (quintic element). The same number of nodes is used for all elements in an automatic mesh.

`ElementOrder` I (-) Interpolation order of the element. This parameter defines the order of the shape functions. At the moment only Lagrangian finite elements are implemented, meaning that the order of interpolation is equal to `ElementNodes-1`. Any other choice results in a run-time error.

`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 thea.log.
MaximumSize	R	(m)	Maximum element size allowed during automatic mesh adaptivity.
MaximumStep	R	(s)	Maximum time step allowed during adaptive time integration.
MeshAdaptivity	C	(-)	Switch for mesh adaptivity. Possible values: none the initial mesh is steady on adaptive mesh refinement, as defined by AdaptivityMethod
MeshType	C	(-)	Flag defining the initial mesh type. Possible values: uniform uniform initial mesh. The mesh consists of NrElements elements with ElementNodes nodes

			`refined` refined initial mesh. The mesh consists of a total of `NrElements` elements with `ElementNodes` nodes, of which `NrRefined` elements are placed in a region between `BeginRefined` and `EndRefined` `user` user's defined initial mesh. This option is active but not documented in the present version
`MinimumSize`	R	(m)	Minimum mesh size allowed during automatic mesh adaptivity.
`MinimumStep`	R	(s)	Minimum time step allowed during adaptive time integration.
`NrElements`	I	(-)	Total number of elements in the mesh.
`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 THEA 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 `thea.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 External 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 External Routines, consult the Variables manual [11]

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 External 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 [11].

CHAPTER 5

Post-processing Language Reference

Structure and syntax

The post-processing command file is read by the post-processor interpreter of THEAPOST. 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. `Thermal 2` for the second thermal component defined in the input for the simulation with THEA).

`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 `theapost.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 current electric 1 electric 2`

`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 temperature hydraulic 1 vs temperature hydraulic 2`

`PostScriptFile` *name*

Set the name of the file containing Postscript® output. The default file name for printed output is `theapost.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 temperature pressure hydraulic 1 hydraulic 2`

`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`, `Thermals`, `Hydraulics`, `Electrics`, `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 end spatial coordinate for plots and listings to the `Length` as specified in the simulation input.

`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 start spatial coordinate for plots and listings to 0.

`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 positions specified. Interpolation is performed if the specified positions fall between nodes. The following `Plot` and `Print` commands will report the results as function of the time at the n requested positions. 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 end spatial coordinate for plots and listings, default is the simulated `Length`.

`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 start spatial coordinate for plots and listings, default is the simulated 0.

`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 THEA. The default file name for printed output is `thea.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 THEAPOST 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. thermal component number 2). Target and support must be selected from a valid combination (e.g. temperature of thermal component number 2). In the following table we report the keys for the valid combinations of targets and supports. Note that a void support is allowed for variables that are overall cable quantities (e.g. cable current). Any invalid selection or combination of target and support results in a syntax error during parsing.

Support	Target	Units	Meaning
	`Current`	(A)	Total cable current
	`Height`	(m)	Local elevation of the cable/channel
	`Mesh`	(1/m)	Mesh density
	`Resistance`	(Ω)	Total cable resistance
	`TotalQExternal`	(W)	Total external heat
	`TotalQJoule`	(W)	Total Joule heat
	`TotalQTransverse`	(W)	Total Joule heat generated by the current transfer among thermal and electric components
`Electric`	`Current`	(A)	Current of the electric component
	`DeltaV`	(V)	Voltage difference between the electric component and the last electric component
	`ElectricField`	(V/m)	electric field of the electric component
	`QLongitudinal`	(W/m)	Joule heat due to current transfer among electric components and power due to external longitudinal voltage source
	`QTransverse`	(W/m)	Joule heat due to current transfer among electric components
	`SpecificResistance`	(Ω/m)	Resistance per unit length of the electric component
	`VExternal`	(V/m)	External longitudinal voltage per unit length on the electric component
	`Voltage`	(V)	Voltage of the electric component
`Hydraulic`	`Conductivity`	(W/m K)	Thermal conductivity of the fluid in the hydraulic component
	`Cp`	(J/kg K)	Specific heat of the fluid in the hydraulic component
	`Density`	(kg/m^3)	Density of the fluid in the hydraulic component
	`Friction`	(-)	Friction factor of the flow
	`HTC`	(W/m^2 K)	Heat transfer coefficient of the flow
	`IntegratedQexternal`	(W)	Total external heat in the hydraulic component
	`Massflow`	(kg/s)	Massflow in the hydraulic component
	`PrandtlNr`	(-)	Prandtl number of the flow
	`Pressure`	(Pa)	Pressure in the hydraulic component
	`QExternal`	(W/m)	External heat flux per unit length in the hydraulic component
	`CExternal`	(W/m)	Convection heat flux per unit length in the hydraulic component

	`T0External`	(K)	Convection wall temperature for the hydraulic component
	`ReynoldsNr`	(-)	Reynolds number of the flow
	`Temperature`	(K)	Temperature of the hydraulic component
	`Velocity`	(m/s)	Velocity in the hydraulic component
	`Viscosity`	(Pa/s)	Viscosity of the fluid in the hydraulic component
`Thermal`	`Current`	(A)	Current in the thermal component
	`Field`	(T)	Magnetic field
	`IntegratedQExternal`	(W)	Total external heat in the thermal component
	`IntegratedQJoule`	(W)	Total Joule heat in the thermal component
	`IntegratedQTransverse`	(W)	Total Joule heat in the thermal component generated by the current transfer among the electric components coupled
	`Ic`	(A)	Critical current of the superconducting material in the thermal component
	`Jc`	(A/m^2)	Critical current density of the superconducting material in the thermal component
	`QExternal`	(W/m)	External heat flux per unit length in the thermal component
	`NormalLength`	(m)	Total normal length in the thermal component
	`QJoule`	(W/m)	Joule heat flux per unit length in the thermal component
	`QVExternal`	(W/m)	Heat per unit length in the thermal component generated by the longitudinal external voltage source in the electric components coupled
	`QTransverse`	(W/m)	Joule heat per unit length in the thermal component generated by the current transfer among the electric components coupled
	`Resistance`	(Ω)	Total resistance of the thermal component
	`SpecificResistance`	(Ω/m)	Resistance per unit length of the thermal component
	`Strain`	(-)	Longitudinal strain
	`Tc`	(K)	Critical temperature of the superconducting material in the thermal component
	`Tcs`	(K)	Current sharing temperature of the superconducting material in the thermal component
	`Temperature`	(K)	Temperature of the thermal component
	`Tmargin`	(K)	Temperature margin in the thermal component (`Tcs-Temperature`)

CHAPTER 6

External Routines

Although the modeling power of THEA is above that of any previously developed computer code for the analysis of superconducting cables, situations may arise when you may like to customize the code to use special functions, correlations, material properties or to read measured quantities to provide initial or boundary data. To this purpose THEA provides a very powerful customization mechanism through the External Routines, a wide set of procedures that gives access to low level functionalities within the code. You should be well familiar with FORTRAN programming, the operation of the code and input data before you use the additional capability provided by External Routines.

Warning External Routines give unlimited access to the data structure used by the main program. Improper programming of External 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 external routines

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

```
~/CryoSoft/usr/thea/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 External 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 THEA. For this purpose you can use the standard makefile

```
~/CryoSoft/etc/thea.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 [4] 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 External 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 External Routines for THEA 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.

Boundary conditions

The following routines are used to set boundary conditions for thermal, hydraulic or electric components. All THEA simulations require specification of both left and right boundary conditions. In all routines below the two boundaries are identified as follows:

`Boundary = 1` left boundary
`Boundary = 2` right boundary

The routines described in this section are contained in the file `userBoundary.f`.

```
real function UserSTBoundary (Time, Boundary, Thermal, Tboundary)
```

Returns the boundary temperature (K) for the thermal component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=temperature`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Thermal`	I	(-)	index of the thermal component
`TBoundary`	R	(K)	boundary temperature as read-in from input

```
real function UserSQBoundary (Time, Boundary, Thermal, Qboundary)
```

Returns the boundary heating power (W) on the thermal component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=heat`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Thermal`	I	(-)	index of the thermal component
`QBoundary`	R	(W)	boundary heat flux as read-in from input

```
real function UserHTBoundary (Time, Boundary, Hydraulic, Tboundary)
```

Returns the boundary temperature (K) for the hydraulic component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=reservoir`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Hydraulic`	I	(-)	index of the hydraulic component
`TBoundary`	R	(K)	boundary temperature as read-in from input

```
real function UserpBoundary (Time, Boundary, Hydraulic, pBounbdary)
```

Returns the boundary pressure (Pa) for the hydraulic component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=reservoir`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Hydraulic`	I	(-)	index of the hydraulic component
`pBoundary`	R	(Pa)	boundary pressure as read-in from input

```
real function UsermdotBoundary (Time, Boundary, Hydraulic,
                                mdotBoundary)
```

Returns the boundary temperature (K) for the hydraulic component. At present not called.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Hydraulic`	I	(-)	index of the hydraulic component
`mdotBoundary`	R	(kg/s)	boundary massflow as read-in from input

```
real function UserIBoundary (Time, Current, Boundary, Electric,
                             IBoundary)
```

Returns the boundary current (A) for the electric component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=current`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Current`	R	(s)	total operating current
`Boundary`	I	(-)	boundary index (left/right boundary)
`Electric`	I	(-)	index of the electric component
`IBoundary`	R	(A)	boundary current as read-in from input

```
real function UserVBoundary (Time, Boundary, Electric, VBoundary)
```

Returns the boundary voltage (V) for the electric component. Called if `BoundaryConditions(Boundary)=user` and `BoundaryType=voltage`.

Parameter	Type	Units	Meaning
`Time`	R	(s)	time
`Boundary`	I	(-)	boundary index (left/right boundary)
`Electric`	I	(-)	index of the electric component
`VBoundary`	R	(V)	boundary voltage as read-in from input

Cable current

The following routine is used to set the total cable current as a function of time. The routine described in this section is contained in the file `userCurrent.f`.

```
real function UserCurrent (Time, Resistance, InitialCurrent,
                           TauDetection, TauDump)
```

Returns the total cable current (A). Called if `CurrentModel=user`.

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Resistance`	R	(Ω)	total cable resistance
`InitialCurrent`	R	(A)	initial cable current as from input
`TauDetection`	R	(s)	detection time constant as from input
`TauDump`	R	(s)	dump time constant as from input

Electric components

The electric characteristics of the electric components are customized through the External Routines described in this section. Two electrical characteristics are needed, namely the transverse conductance c_{ij} and the mutual inductance l_{ij} per unit length for any couple (i,j) of electric components. Both can be defined as a function of position. The values returned for the conductance (in 1/Ωm) and inductance (in H/m) are assembled in two matrices. Note that the two matrices of transverse conductance and inductance cannot to be singular. The user should take care that this is the case.

The routines described in this section are contained in the file `userElectrics.f`

`real function **UserConductance** (Electric1, Electric2, X, Conductance)`

Returns the conductance per unit length (1/Ωm) between any two components. Called if `Links_Model=user.`

Parameter	Type	Units	Meaning
`Electric1`	I	(-)	index of the first electric component
`Electric2`	I	(-)	index of the second electric component
`X`	R	(m)	nodal coordinate
`Conductance`	R	(1/Ωm)	conductance as from input

`real function **UserInductance** (Electric1, Electric2, X, Self, Mutual)`

Returns the inductance per unit length (H/m) between any two components. Called if `Links_Model=user`

Parameter	Type	Units	Meaning
`Electric1`	I	(-)	index of the first electric component
`Electric2`	I	(-)	index of the second electric component
`X`	R	(m)	nodal coordinate
`Self`	R	(H)	self inductance as from input
`Mutual`	R	(H)	mutual inductance as from input

`real function **UserResistance** (Electric, X, Resistance, I0, Current)`

Returns the longitudinal resistance per unit length (Ω/m). Called if `RModel=user.`

Parameter	Type	Units	Meaning
`Electric`	I	(-)	index of the electric component
`X`	R	(m)	nodal coordinate
`Resistance`	R	(Ω/m)	resistance per unit length as from input
`I0`	R	(A)	initial current in the electric component (`IInitial`)
`Current`	R	(A)	current

Properties of fluids

The thermophysical properties of the fluids can be customized using the routines described in this section. These routines are called if the fluid name used in the hydraulic block is not within the set of standard fluids. The properties computed are density ρ, specific heat at constant pressure Cp, specific heat at constant volume Cv, specific enthalpy h, specific entropy s, viscosity ν, thermal conductivity K, sound speed c, Gruneisen factor ϕ, superfluid effective conductivity function F. All properties are computed as a function of pressure p and temperature T

The routines described in this section are contained in the file `userFluids.f`

`real function` **`UserFluidDensity`** `(FluidName,p,T)`

Returns the density of the fluid (kg/m^3). Called if `Fluid=user`.

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidCp`** `(FluidName,p,T)`

Returns the specific heat at constant pressure of the fluid (J/kg K). Called if `Fluid=user`.

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidCv`** `(FluidName,p,T)`

Returns the specific heat at constant volume of the fluid (J/kg K). Called if `Fluid=user`.

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidEnthalpy`** `(FluidName,p,T)`

Returns the specific enthalpy of the fluid (J/kg). Called if `Fluid=user`.

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidEntropy`** `(FluidName,p,T)`

Returns the specific entropy of the fluid (J/kg). Called if `Fluid=user`.

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidViscosity`** `(FluidName,p,T)`

Returns the dynamic viscosity of the fluid (Poise). Called if `Fluid=user.`

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidConductivity`** `(FluidName,p,T)`

Returns the thermal conductivity of the fluid (W/m K). Called if `Fluid=user.`

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidSound`** `(FluidName,p,T)`

Returns the sound speed of the fluid (m/s). Called if `Fluid=user.`

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserFluidGruneisen`** `(FluidName,p,T)`

Returns the Gruneisen factor of the fluid (-), defined as $\rho/T\ (dT/d\rho)_s$. Called if `Fluid=user.`

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

`real function` **`UserSuperFluid`** `(FluidName,p,T)`

Returns the superfluid thermal conductance function of the fluid (W^3/m^5 K). Called if `Fluid=user.` Only applies to superfluid conditions (should be set to zero)

Parameter	Type	Units	Meaning
`FluidName`	C	(-)	name of the fluid
`p`	R	(Pa)	fluid pressure
`T`	R	(Pa)	fluid temperature

Friction factor

The routine described in this section allows the customization of the friction factor as a function of the Reynolds number, position and the hydraulic component. The friction factor f is defined following the US convention, so that the pressure drop along a channel is given by:

$$\frac{\partial p}{\partial x} = -2f\frac{\rho v^2}{D_h}$$

where symbol notation is conventional. The friction factor can be defined for all hydraulic components independently.

The routine is contained in the file `userFriction.f`.

```
real function UserFrictionFactor (Hydraulic, X, ReynoldsNr,
                                  FrictionFactor)
```

Returns the friction factor (-) of the component. Called if `fModel=user`.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`ReynoldsNr`	R	(-)	Reynolds number
`FrictionFactor`	R	(-)	friction factor as from input

Heating of hydraulic components

The heating of the hydraulic components can be defined using the routine described in this section. The heating power density (in W/m) or the wall temperature (in K) are defined as a function of space and time for all hydraulic components independently.

The routines are contained in the file `userHHeating.f`.

```
real function UserHHeating (Time, Hydraulic, X, Temperature, Q,
                            Q_XBegin, Q_XEnd, Q_Tau)
```

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

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature of the hydraulic component at the given coordinate
`Q`	R	(W/m)	heat flux as from input
`Q_XBegin`	R	(m)	start coordinate of the heating space, as from input
`Q_XEnd`	R	(m)	end coordinate of the heating space, as from input
`Q_Tau`	R	(s)	end heating time, as from input

```
real function UserHT0 (Time, Hydraulic, X, Temperature, T0, T0_WP,
                       T0_XBegin, T0_XEnd)
```

Returns the wall temperature (K) for the component Called if `CModel=user`.

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature of the hydraulic component at the given coordinate
`T0`	R	(K)	wall temperature, as from input
`T0_WP`	R	(m)	wall wetted perimeter, as from input
`T0_XBegin`	R	(m)	start coordinate of the convection space, as from input
`T0_XEnd`	R	(m)	end coordinate of the convection space, as from input

Heat transfer coefficient

The routine described in this section allows the customization of the heat transfer coefficient as a function of the Reynolds number, fluid state, average wall temperature, position and the hydraulic component. The friction factor can be defined for all hydraulic components independently.

The routine is contained in the file `userHTC.f`.

```
real function Userhtc (Hydraulic, X, Temperature, Pressure, Density,
                       Twall, Dh, ReynoldsNr, HTC)
```

Returns the heat transfer coefficient (W/m^2K) of the component. Called if `hModel=user`.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature
`Pressure`	R	(Pa)	pressure
`Density`	R	(kg/m^3)	density
`Twall`	R	(K)	average wall temperature
`Dh`	R	(m)	hydraulic diameter
`ReynoldsNr`	R	(-)	Reynolds number
`HTC`	R	(W/m^2K)	heat transfer coefficient as from input

Local elevation

The following routine is used to set the local elevation (height) of the components as a function of position. The height can be set for each hydraulic component in the cable. The routine described in this section is contained in the file `userHeight.f`.

```
real function UserHeight (X, Height)
```

Returns the height (m) of the hydraulic components. Called if `HeightModel=user`.

Parameter	Type	Units	Meaning
`X`	I	(m)	coordinate
`Height`	R	(m)	array containing the left and right value of the elevation as from input

Hydraulic components

The geometric characteristics of the hydraulic components can be customized through the External Routines described in this section. In particular the cross section of the channel and its hydraulic diameter can be set for each hydraulic component as a function of position.

The routines described in this section are contained in the file `userHydraulics.f`

```
real function UserHArea (Hydraulic, X, Area)
```

Returns the area (m^2) of the component. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`Area`	R	(m^2)	area as from input

```
real function UserDh (Hydraulic, X, Dh)
```

Returns the hydraulic diameter (m) of the component. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`Dh`	R	(m)	hydraulic diameter of the component, as from input

Initial conditions

The following routines are used to set initial conditions for the three type of components. The initial conditions are the starting point for a simulation. Care should be taken that they are physically consistent and that they follow the boundary conditions. Numerical instabilities can be generated should this not be the case.

Different variables must be set depending on the type of component. Thermal components require setting of the temperature, hydraulic components require the pressure, temperature and massflow, and electric components require the current. The setting of the variables is required at all locations `X` within the domain of analysis.

The routines described in this section are contained in the file `userInitial.f`.

`real function` **`UserSTInitial`** `(Thermal, X, TInitial)`

Returns the initial temperature (K) of the thermal component. Called if `InitialCondition=user` in the Thermals block.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`X`	R	(m)	nodal coordinate
`TInitial`	R	(K)	initial temperature as from input

`real function` **`UserHTInitial`** `(Hydraulic, X, TInitial)`

Returns the initial temperature (K) of the hydraulic component. Called if `InitialCondition=user` in the Hydraulics block.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`TInitial`	R	(K)	initial temperature as from input

`real function` **`UsermdotInitial`** `(Hydraulic, X, mdotInitial)`

Returns the initial mass flow (kg/s) of the hydraulic component. Called if `InitialCondition=user` in the Hydraulics block.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`mdotInitial`	R	(kg/s)	initial massflow as from input

`real function` **`UserpInitial`** `(Hydraulic, X, pInitial)`

Returns the initial pressure (Pa) of the hydraulic component. Called if `InitialCondition=user` in the Hydraulics block.

Parameter	Type	Units	Meaning
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`pInitial`	R	(Pa)	initial pressure as from input

```
real function UserIInitial (Electric, X, IInitial)
```

Returns the initial current (A) of the electric component. Called if `InitialCondition=user` in the Electrics block.

Parameter	Type	Units	Meaning
`Electric`	I	(-)	index of the electric component
`X`	R	(m)	nodal coordinate
`IInitial`	R	(A)	initial current as from input

Links

The characteristics of the thermal/hydraulic links can be defined through the External Routines described in this section. The links among thermal elements are defined by thermal resistances that can be function of position. The links among hydraulic components depend on the wetted perimeter of two channels (i.e. the common perimeter) and on the degree of perforation of the common wall. Both can be defined as functions of position. Links among thermal and hydraulic components depend on the wetted perimeter, which can be defined as a function of position.

The corresponding routines are contained in the file `userLinks.f`.

```
real function UserThermalResistance (Thermal1, Thermal2, X,
                                     ThermalResistance)
```

Returns the thermal resistance (K m/W) between two components. Called if `Links_Model=user` in the thermals block.

Parameter	Type	Units	Meaning
`Thermal1`	I	(-)	index of the first thermal component
`Thermal2`	I	(-)	index of the second thermal component
`X`	R	(m)	nodal coordinate
`ThermalResistance`	R	(K m/W)	thermal resistance as from input

```
real function UserPerforation (Hydraulic1, Hydraulic2, X, Perforation)
```

Returns the perforation (m) of two components. Called if `Links_Model=user` in the hydraulics block.

Parameter	Type	Units	Meaning
`Hydraulic1`	I	(-)	index of the first hydraulic component
`Hydraulic2`	I	(-)	index of the second hydraulic component
`X`	R	(m)	nodal coordinate
`Perforation`	R	(m)	perforation factor as from input

```
real function UserWettedPerimeter (Hydraulic1, Hydraulic2, X,
                                   WettedPerimeter)
```

Returns the wetted perimeter (m) between two components. Called if `Links_Model=user` in the hydraulics block.

Parameter	Type	Units	Meaning
`Hydraulic1`	I	(-)	index of the first hydraulic component
`Hydraulic2`	I	(-)	index of the second hydraulic component
`X`	R	(m)	nodal coordinate
`WettedPerimeter`	R	(m)	wetted perimeter as from input

```
real function UserSHWettedPerimeter (Thermal, Hydraulic, X,
                                     WettedPerimeter)
```

Returns the wetted perimeter (m) between thermal and hydraulic component. Called if `S_H_Links_Model=user`.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`Hydraulic`	I	(-)	index of the hydraulic component
`X`	R	(m)	nodal coordinate
`WettedPerimeter`	R	(m)	wetted perimeter as from input

Magnetic field

The following routine is used to set the magnetic field in the cable as a function of position, time and current. The magnetic field can be set for each thermal component in the cable.

The corresponding routines are contained in the file `userMagneticField.f`.

```
real function UserMagneticField (Time, Thermal, X, Current,
                                 InitialCurrent, MagneticFieldSS,
                                 MagneticFieldTr)
```

Returns the magnetic field (T) of the component. Called if `MagneticFieldModel=user`.

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Thermal`	I	(-)	index of the thermal component
`X`	I	(m)	coordinate
`Current`	R	(A)	cable current as from input
`InitialCurrent`	R	(A)	initial cable current as from input
`MagneticFieldSS`	R	(T)	array containing the left and right value of the steady-state magnetic field as from input
`MagneticFieldTr`	R	(T)	array containing the left and right value of the transient magnetic field as from input

Heating of thermal components

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

The routine is contained in the file `userSHeating.f`.

```
real function UserSHeating (Time, Thermal, X, Temperature, Q,
                            Q_XBegin, Q_XEnd, Q_Tau)
```

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

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Thermal`	I	(-)	index of the thermal component
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature of the thermal component at the given coordinate
`Q`	R	(W/m)	heat flux as from input
`Q_XBegin`	R	(m)	start coordinate of the heating space, as from input
`Q_XEnd`	R	(m)	end coordinate of the heating space, as from input
`Q_Tau`	R	(s)	end heating time, as from input

Properties of solid materials

The thermophysical and electrical properties of the solid materials can be customized using the routines described in this section. These routines are called if the material name used in a thermal component is not within the set of standard materials, or in the case that the thermal model is explicitly set to `user`. The properties computed are thermal conductivity K, heat capacity C, density ρ, electrical resistivity η, critical current density Jc, critical temperature Tc, current sharing temperature Tcs. The routine `UserMaterialType` in addition identifies the type of material. The types allowed are reported in the table below. The properties used for the simulation depend on the type of material, as also defined in the table below where a symbol ✓ indicates that the corresponding property is needed.

Material type	K	C	ρ	η	Jc	Tc	Tcs
`SuperConductor`	✓	✓	✓		✓	✓	✓
`Alloy`	✓	✓	✓	✓			
`Metal`	✓	✓	✓	✓			
`Insulator`	✓	✓	✓				
`Composite`	✓	✓	✓				

Note In any case for a user's defined material all routines below must be provided. Depending on the type of material (SuperConductor, Alloy, Metal, Insulator or Composite) some of the routines can return dummy values (e.g. zero critical current density if the material is not a superconductor).

The routines described in this section are contained in the file `userSolids.f`

```
character*72 function UserMaterialType (MaterialName)
```

Returns the type of the material: "SuperConductor", "Alloy", "Metal", "Insulator" or "Composite". Called for user specified materials.

Parameter	Type	Units	Meaning
MaterialName	C	(-)	name of the material

```
real function UserConductivity (MaterialName, X, Temperature, B, RRR)
```

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

Parameter	Type	Units	Meaning
MaterialName	C	(-)	name of the material
X	R	(m)	nodal coordinate
Temperature	R	(K)	temperature
B	R	(T)	magnetic field
RRR	R	(-)	residual resistivity ratio

```
real function UserCriticalCurrent (MaterialName, X, Temperature, B,
                                   Epslon)
```

Returns the critical current density (A/m^2) of the material. Called if the Model=user or for user specified materials.

Parameter	Type	Units	Meaning
MaterialName	C	(-)	name of the material
X	R	(m)	nodal coordinate
Temperature	R	(K)	temperature
B	R	(T)	magnetic field
Epslon	R	(-)	longitudinal strain

```
real function UserCriticalTemperature (MaterialName, X, B, Epslon)
```

Returns the critical temperature (K) of the material. Called if the Model=user or for user specified materials.

Parameter	Type	Units	Meaning
MaterialName	C	(-)	name of the material
X	R	(m)	nodal coordinate
B	R	(T)	magnetic field
Epslon	R	(-)	longitudinal strain

```
real function UserCurrentSharing (MaterialName, X, B, Jop, Epslon)
```

Returns the current sharing temperature (K) of the material. Called if the `Model=user` or for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`B`	R	(T)	magnetic field
`Jop`	R	(A/m^2)	operating current density
`Epslon`	R	(-)	longitudinal strain

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

Returns the density (kg/m^3) of the material. Called if the `Model=user` or for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature

```
real function UserResistivity (MaterialName, X, Temperature, B, RRR)
```

Returns the resistivity (Ω m) of the material. Called if `Model=user` or for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature
`B`	R	(T)	magnetic field
`RRR`	R	(-)	residual resistivity ratio

```
real function UserSpecificHeat (MaterialName, X, Temperature, B, Tcs,
                                Epslon)
```

Returns the specific heat (J/kg K) of the material. Called if `Model=user` or for user specified materials.

Parameter	Type	Units	Meaning
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`Temperature`	R	(K)	temperature
`B`	R	(T)	magnetic field
`Tcs`	R	(K)	current sharing temperature
`Epslon`	R	(-)	longitudinal strain

Longitudinal strain

The following routine is used to set the longitudinal strain on the cable as a function of position, time and current. The strain can be set for each thermal component in the cable.

The routine described in this section is contained in the file `userStrain.f`.

```
real function UserStrain (Time, Thermal, X, Current, InitialCurrent,
                          StrainSS ,StrainTr)
```

Returns the longitudinal strain (-) of the component. Called if `StrainModel=user`.

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Thermal`	I	(-)	index of the thermal component
`X`	I	(m)	coordinate
`Current`	R	(A)	cable current as from input
`InitialCurrent`	R	(A)	initial cable current as from input
`StrainSS`	R	(T)	array containing the left and right value of the steady-state longitudinal strain as from input
`StrainTr`	R	(T)	array containing the left and right value of the transient longitudinal strain as from input

Thermal components

The External Routines described in this section can be used to customize the characteristics of the thermal components. They are called when the thermal model is set to `user`. General characteristics of a component, such as its cross section, RRR or the parameters for the superconducting transition, can be varied as a function of position.

The routines described in this section are contained in the file `userThermals.f`:

```
real function UserE0 (Thermal, X, E0)
```

Returns E_0 (V/m) of the component. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`X`	R	(m)	nodal coordinate
`E0`	R	(V/m)	E0 as from input

```
integer function UsernPower (Thermal, X, nPower)
```

Returns n (-) of the component. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`X`	R	(m)	nodal coordinate
`nPower`	I	(-)	nPower as from input

```
real function UserRRR (Thermal, MaterialName, X, RRR)
```

Returns the residual resistivity ratio (-) of the material. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`RRR`	R	(-)	residual resistivity ratio as from input

```
real function UserSArea (Thermal, MaterialName, X, Area)
```

Returns the area (m^2) of the component. Called if `Model=user`.

Parameter	Type	Units	Meaning
`Thermal`	I	(-)	index of the thermal component
`MaterialName`	C	(-)	name of the material
`X`	R	(m)	nodal coordinate
`Area`	R	(m^2)	area as from input

Voltage source in electric components

The voltage source in the electric components can be defined using the routine described in this section. The longitudinal voltage density (electric field in V/m) is returned as a function of space and time for all electric components independently.

The routine is contained in the file `userVoltage.f`.

```
real function UserVoltage (Time, Electric, X, Voltage, V_XBegin,
                           V_XEnd, V_Tau)
```

Returns the voltage (V/m) of the component. Called if `Links_Model=user`

Parameter	Type	Units	Meaning
`Time`	I	(s)	time
`Electric`	I	(-)	index of the electric component
`X`	R	(m)	nodal coordinate
`Voltage`	R	(V/m)	voltage as from input
`V_XBegin`	R	(m)	start coordinate for setting the voltage as from input
`V_XEnd`	R	(m)	end coordinate for setting the voltage as from input
`V_Tau`	R	(s)	end time for setting the voltage as from input

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 pressure

differences), or because of incorrect values from fluid and 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 problem that can be solved is determined by the requested memory allocation in the FORTRAN include file:

```
~/CryoSoft/src/thea/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
`MaxSComp`	maximum number of thermal components
`MaxMatOfSComp`	maximum number of materials in a thermal component
`MaxHComp`	maximum number of hydraulic components
`MaxEComp`	maximum number of electric components
`MaxElements`	maximum number of finite elements in the mesh

The additional parameters `MaxWork4` and `MaxWork8` are set to accommodate the bandwidth system matrix in the equation solver, and may need adjustment in case the PDE solver needs more work space.

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 [4].

Warning Modifying the code dimensioning parameters requires understanding of the memory allocation for the system variables, and of the internal structure of the code. 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 External Routines with improper data handling are used. They diagnose a severe fault within the code. If you are using External Routines, verify their consistency with the calling protocol. In case you are not using External Routines, report internal consistency errors to us.

CHAPTER 8
References

[1] Bottura L., *Modelling Stability in Superconducting Cables*, Physica C, **310**, 316-326, 1998.

[2] Bottura L., Rosso C., Breschi M., *A General Model For Thermal, Hydraulic and Electric Analysis of Superconducting Cables*, Cryogenics, **40**, 617-626, 2000.

[3] Bottura L., Rosso C., *Thermal, Hydraulic and Electric Analysis of Superconducting Cables: Model Description*, CryoSoft Internal Note CRY0/00/017, 2000.

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

[5] Krempaski L., Schmidt C., *Experimental Verification of "Supercurrents" in Superconducting Cables Exposed to AC-Fields*, Cryogenics, **39**, 23-33, 1999.

[6] Anghel A., *QUELL Experiment: Analysis and Interpretation of the Quench Propagation Results*, Cryogenics, **38**, 459-466, 1998.

[7] CryoSoft Solids Manual, Version 4.0, 2021.

[8] CryoSoft Fluids Manual, Version 3.0, 2002.

[9] Bottura L., *Friction Factor Correlations*, CryoSoft Internal Note CRY0/98/009, 1998.

[10] Bottura L., *Heat Transfer Correlations*, CryoSoft Internal Note CRY0/98/010, 1998.

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

[12] D. Bessette, et al, *Test Results From the PF Conductor Insert Coil and Implications for the ITER PF System*, IEEE Trans. Appl. Sup., **19**(3), 1525-1531, 2009.