REIMS

Release V2.1

Authors: D. Furfaro, J. Kosek

Sep 11, 2026

CONTENTS:

CHAPTER

ONE

USER MANUAL

1.1 Input file description

REIMS input file is used to describe the model which will be calculated by REIMS software.

Small example of input file is shown below:

```yaml
# yaml-language-server: $schema=https://iterorganization.github.io/REIMS/
↪tools/reims_schema.json

simulation:
  simulation_end: 1000         # Simulate 1000s
  implicit_tolerance: 0.0005   # Tolerance for implicit solver

write_results:
  file: reims_output.h5        # Write results to file: reims_output.h5

# Main model description is a list of components and their connections
components:          # One "state" component and one "link" component
  - type: channel    # Type of the component - In this case channel
    id: pipe         # Name of the component has to be unique
    nodes: 200       # Number of computation cells
    length: 140.0    # Total length in m
    diameter: 10e-3 # Channel diameter 12mm
    initial: {p: 5.0e5, t: 4.3} # initial conditions P = 5bar and T = 4.3K
  - type: pump    # Type of the component - In this case pump
    m0: 2.0e-3    # Mass flow rate: 2 g/s
    link:         # 2 links: link 1 - pump inlet, link 2 - pump outlet
     - id: pipe   # inlet of the pump connected to outlet of the 'pipe'
       node: out # outlet pipe
     - id: pipe   # outlet of the pump connected to inlet of the 'pipe'
       node: in   # inlet pipe
```

It describes 2 components:

1. `channel` which is state component, and name (id) `pipe`
2. `pump` which is link component which links 2 ends of `pipe`

connected in a closed loop.

1.2 YAML config file structure

1.2.1 Top level configuration

The top level configuration consists of 3 required elements:

- simulation

- write_results
- components

and optional elements:

- external_libs
- friction_correlations
- nusselt_correlations
- materials

Description of each element is provided below.

type	*object*	
properties		
• **simulation**	*Simulation parameters*	
	type	*object*
	properties	
	• **simulation_end**	*End of simulation (s)*
		numeric_value
	• R_correction	*Mechanical equilibrium correction - Introduction of an extra fluid equation*
		Value if nothing is specified by the user: `yes` • `yes`: Mechanical equilibrium correction is activated - Useful in case of density discontinuity + coarse mesh close to the critical region • `no`: No mechanical equilibrium correction - Less accurate in case of density discontinuity + coarse mesh close to the critical region - More accurate and robust far from critical region
		Boolean value
	• explicit_ tolerance	*Criteria for switching from explicit to implicit*
		Value if nothing is specified by the user: `9e-4` Reducing this value leads to prioritizing the explicit scheme over the implicit scheme. We therefore expect further flow stabilization before the implicit scheme is activated.
		numeric_value
	• **implicit_ tolerance**	*Tolerance of implicit solver*
		Reducing this value leads to longer computation times due to the use of smaller time steps, but potentially increased robustness.
		0D_signal
	• step_controller_ gain	*Gain of PI controller*
		Value if nothing is specified by the user: `4.0` This value is related to automatic time step management - It is advised to modify implicit_tolerance and keep the default value of the gain.
		numeric_value
	• max_time_step	*Maximum time step (s)*
		Value if nothing is specified by the user: `50.0`
		0D_signal

continues on next page

Table 1 – continued from previous page

• **write_results**	type	*object*
	properties	
	• active	*Writing results to hdf5 files activation*
		Value if nothing is specified by the user: `true`
		Boolean value
	• **file**	*File name to store output data*
		type *string*
	• minimum_time	*Minimum space between saved points (s)*
		Value if nothing is specified by the user: `0.0`
		numeric_value
	• active2D	*Writing to results to hdf5 files - Mesh2D components*
		Value if nothing is specified by the user: `true`
		Boolean value
	• time_between_2D_writes	*Time interval between two consecutive 2D result writes (s)*
		Value if nothing is specified by the user: `3600.0`
		numeric_value
• external_libs	*External dynamic libraries*	
	Optional list of dynamic libraries (DLL) to load at startup. Each entry is the library name without the `.dll` extension.	
	`` external_libs: - name_of_my_dll_1 - name_of_my_dll_2 ```	
	1D_string	

continues on next page

Table 1 – continued from previous page

• friction_ correlations	*Fanning friction factor correlations*		
	Optional list of named friction correlations available to channel components. Three use cases are supported: built-in with defaults, built-in with custom parameters, or user-defined via external DLL. See *Friction correlations details* for details and examples.		
	type	*array*	
	items	type	*object*
		properties	
		• **friction**	*Correlation name*
			Unique name used to reference this correlation in the `friction` field of channel components.
			type *string*
		• base	*Built-in correlation to use as base*
			Required only for Case 2: built-in correlation with custom parameters. Identifies which built-in initialization to use for this entry.
			type *string*
			enum blasius, katheder, central_spiral
		• alpha	*Friction coefficient alpha*
			numeric_value
		• beta	*Friction coefficient beta*
			numeric_value
		• VoidFr	*Void fraction (katheder only)*
			numeric_value
		• Re_min	*Minimum Reynolds number for Katheder activation (katheder only, default 1000)*
			Reynolds number threshold below which the Katheder correlation returns zero and only laminar friction applies (default: 1000). Set to 0 to activate Katheder for all Re > 0.
			numeric_value

continues on next page

Table 1 – continued from previous page

• nusselt_ correlations	*Nusselt number correlations*		
	Optional list of named Nusselt number correlations available to channel, fluidlink and thermalink components. Three use cases are supported: built-in with defaults, built-in with custom parameters, or user-defined via external DLL. See *Nusselt correlations details* for details and examples.		
	type	*array*	
	items	type	*object*
		properties	
		• **nusselt**	*Correlation name*
			Unique name used to reference this correlation in the `nusselt` field of channel, fluidlink and thermalink components.
			type *string*
		• base	*Built-in correlation to use as base*
			Required only for Case 2: built-in correlation with custom parameters. Identifies which built-in initialization to use for this entry.
			type *string*
			enum pipe, DBG
		• a	*Coefficient a*
			numeric_value
		• b	*Coefficient b*
			numeric_value
		• c	*Coefficient c*
			numeric_value
		• d	*Temperature ratio exponent (DBG only)*
			numeric_value
		• NuL	*Laminar Nusselt number (DBG only)*
			Minimum value applied in the DBG correlation: Nu = max(NuL, NuT).
			numeric_value

continues on next page

Table 1 – continued from previous page

• materials	*Material database*		
	Optional list of named materials available to `strand`, `solid`, `solidlink`, and `mesh2D` components. Three use cases are supported: built-in with defaults, built-in with custom parameters, or user-defined via external DLL. See *Materials details* for details and examples. Optional parameters: `density`, `RRR`, `E0`, `nPow`, `Bc20m`, `Tc0m`, `nu`, `Ca1`, `Ca2`, `e0a`, `emax`, `C0`, `p`, `q`, `str1`, `str2`, `Bc20`, `Tc0_p`, `CC0`, `n`.		
	type	*array*	
	items	type	*object*
		properties	
		• **material**	type	*string*
		• base	type	*string*
			enum	copper, nb3sn, nbti, stainless_steel, glass_epoxy, glass_kapton_glass
		• density	*numeric_value*	
		• RRR	*numeric_value*	
		• E0	*numeric_value*	
		• nPow	*numeric_value*	
		• Bc20m	*numeric_value*	
		• Tc0m	*numeric_value*	
		• nu	*numeric_value*	
		• Ca1	*numeric_value*	
		• Ca2	*numeric_value*	
		• e0a	*numeric_value*	
		• emax	*numeric_value*	
		• C0	*numeric_value*	
		• p	*numeric_value*	
		• q	*numeric_value*	
		• str1	*numeric_value*	
		• str2	*numeric_value*	
		• Bc20	*numeric_value*	
		• Tc0_p	*numeric_value*	
		• CC0	*numeric_value*	
		• n	*numeric_value*	
• **components**	*List of components to describe the model*		
	type	*array*	
	items	type	*object*
		oneOf	*channel*
			strand
			solid
			mesh2D
			junction
			fluidlink
			thermalink
			solidlink
			pump
			compressor
			boundary_PT
			boundary_mT

1.2.2 State components

channel

1D pipe filled with compressible helium

```
Example:
  - type: channel            # Component type
    id: pipe_3L_P1_Hol       # Component Id
    nodes: 193               # Number of nodes
    length: 147.66           # Total length of the pipe (m)
    diameter: 0.007          # Diameter of the pipe (m)
    initial: {p: 5.0e5, t: 4.3, u: 0}   # Initial conditions
    thermal: flux            # Heat transfer with a wall expected
    flux: 0                  # Zero flux in this specific case
    channel_link: yes        # Expected link with another channel component
    friction: blasius        # Friction factor correlation name
```

type	*object*	
properties		
• type	const	channel
• **id**	*"id" has to be unique and it is used by the links components*	
	type	*string*
• nodes	*Number of nodes equally distanced along the channel length - Only used in case of uniform mesh*	
	type	*integer*
• **length**	*length_value*	
• diameter	*Hydraulic diameter (m) - Constant value along the channel*	
	numeric_value	
• area	*Cross-section area (m^2) - Constant value along the channel*	
	To be used in case of Bundle Gap only - Otherwise, the area is automatically calculated based on the diameter	
	numeric_value	
• heat_transfert_coef	*Imposed heat transfer coefficient (W/m2/K)*	
	Required when `thermal: temp` and no `nusselt` correlation is specified.	
	numeric_value	
• **initial**	type	*object*
	properties	
	• **p**	*Initial pressure (Pa)*
		numeric_value
	• **t**	*Initial temperature (K)*
		numeric_value
	• **u**	*Initial speed (m/s)*
		numeric_value
• thermal	*Thermal boundary type of the channel*	
	Defining a wall temperature (`temp`), a thermal flux (`flux`) or specifying a link (`link`) with another component are the options available to the user - Value if nothing is specified by the user: `flux`	
	type	*string*
	enum	temp, flux, link
• temp	*Wall temperature (K) - To be used if temp is assigned to thermal*	
	1D_signal	
• flux	*Thermal flux (W/m) - To be used if flux is assigned to thermal*	
	1D_signal	

continues on next page

Table 2 – continued from previous page

• channel_link	*Hydraulic link with other channels - Mass, momentum and energy exchanges*
	Value if nothing is specified by the user: `false`
	Boolean value
• **friction**	*Friction factor correlation name*
	Built-in names: `blasius`, `katheder`, `central_spiral`. Custom correlations can be defined in `friction_correlations` (see *Top level configuration*).
	type *string*
• nusselt	*Nusselt number correlation name*
	Used only when `thermal: temp`. Built-in names: `pipe`, `DBG`. Custom correlations can be defined in `nusselt_correlations` (see *Top level configuration*).
	type *string*

strand

1D composite strand that contains a superconducting material in parallel with a stabilizer shunt

```yaml
Example:
  - type: strand                # Component type
    id: strand_3L_P1            # Component Id
    nodes: 193                  # Number of nodes
    length: 147.66              # Strand length
    initial: {t: 4.3}           # Initial conditions
    CICC_inner_radius: 0.005    # CICC inner radius (m) for effective field
                                # computation - Useless in this specific case
    CICC_outer_radius: 0.0168   # CICC outer radius (m) for effective field
                                # computation - Useless in this specific case
    stabilizer:
      material: Copper          # Stabilizer name
      area: 308.66e-6           # Stabilizer cross-section area (m2)
    superconductor:
      material: Nb3Sn_JAS       # Superconductor name
      area: 154.33e-6           # Superconductor cross-section area (m2)
    channel_link: yes           # Thermal contact expected with a channel␣
↪component
    flux:                       # Thermal flux imposed from h5 data
      time:  {h5: Q_CS3L_1.h5, data: t} # Time (s) scenario
      x:     {h5: Q_CS3L_1.h5, data: x} # Space (m) scenario
      value: {h5: Q_CS3L_1.h5, data: d} # Flux in W/m (structure of the data
                                # consistent with imposed time/space scenario)
      order:  1                 # 1st order interpolation in time
      repeat: 0                 # No scenario repeat
      event:  no                # No intervention on the time step management
    field: 0.0                  # No magnetic field (T)
    field_gradient: 0.0         # No magnetic field gradient (T/m)
    current: 0.0                # No current (A)
```

type	*object*	
properties		
• type	const	strand
• **id**	*'id' has to be unique and it is used for connections*	
	type	*string*

continues on next page

Table 3 – continued from previous page

• nodes	*Number of nodes equally distanced along the channel length - Only used in case of uniform mesh*	
	type	*integer*
• **length**	*length_value*	
• **initial**	type	*object*
	properties	
	• **t**	*Initial temperature (K)*
		numeric_value
• **CICC_inner_radius**	*CICC inner radius (m)*	
	numeric_value	
• **CICC_outer_radius**	*CICC outer radius (m)*	
	numeric_value	
• **stabilizer**	type	*object*
	properties	
	• **material**	*Material name*
		Built-in name for stabilizer: `copper` Custom materials can be defined in `materials` (see *Top level configuration*).
		type *string*
	• **area**	*Cross-section area (m^2)*
		numeric_value
• **superconductor**	type	*object*
	properties	
	• **material**	*Material name*
		Built-in names for superconductor: `nb3sn`, `nbti`. Custom materials can be defined in `materials` (see *Top level configuration*).
		type *string*
	• **area**	*Cross-section area (m^2)*
		numeric_value
• **channel_link**	*Thermal link with hydraulic channel*	
	Boolean value	
• flux	*Heat load applied to strand component (W/m) - Can be a single value OR based on time interpolation OR time/space one*	
	1D_signal	
• **field**	*Magnetic field map (T) of the specific strand - Can be a single value OR based on time interpolation OR time/space one*	
	1D_signal	
• field_gradient	*Magnetic field gradient map (T/m) of the specific strand - Can be a single value OR based on time interpolation OR time/space one*	
	1D_signal	
• current	*Electric current (A) - Can be a single value OR based on time interpolation*	
	0D_signal	

solid

The lumped mass approach is used to define the part of the structure in contact with CICCs - A solid component is an array of solid chunks, each carrying its own temperature

```
Example:
  - type: solid                          # Component type
    id: chunk_3L_P1                      # Component Id
    length: [1.02,1.03, 1.03, 1.04]      # Length of each solid chunk along␣
↪the
                                         # CICC trajectory
    volume: [1.48,1.49, 1.49, 1.50]      # Volume of each solid chunk
```

(continues on next page)

(continued from previous page)

```
    initial: {t: 4.3}                    # Initial temperature
    material: stainless_steel            # Solid material
    channel_link: [yes, yes, yes, yes]  # Thermal link with a channel
                                         # component expected for each solid chunk
    solid_link: yes                      # Thermal link with other solid
                                         # components expected
    flux: 100.0                          # Uniform heat load of 100 W/m (optional)
```

type	*object*	
properties		
• type	const	solid
• **id**	*'id' has to be unique and it is used for connections*	
	type	*string*
• nodes	*Number of nodes equally distanced along the solid length - Only used in case of uniform mesh*	
	type	*integer*
• **length**	*Length (m) of each metal chunk along the CICC trajectory*	
	length_value	
• **volume**	*Volume (m^3) of each metal chunk*	
	A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.	
	scalar_or_D_numeric_value	
• **initial**	type	*object*
	properties	
	• **t**	*Initial temperature (K)*
		numeric_value
• **conduction_between_nodes**	*Thermal conduction between metal chunks along the CICC trajectory*	
	Boolean value	
• **material**	*Material name*	
	Built-in names: `copper`, `nb3sn`, `nbti`, `stainless_steel`, `glass_epoxy`, `glass_kapton_glass`. Custom materials can be defined in `materials` (see *Top level configuration*).	
	type	*string*
• **channel_link**	*Thermal link with hydraulic channel per metal chunk*	
	1D Boolean array	
• **solid_link**	*Thermal link with other solid*	
	Boolean value	
• flux	*External heat load (W/m) applied to solid component - Can be a single value OR based on time interpolation OR time/space one*	
	1D_signal	

mesh2D

2D surface representing a slice of a 3D solid structure - Available for heat diffusion treatment

Note

Gmsh mesh ordering requirements (current code limitation)

In the `.geo` file, all geometric entities (points, lines, surfaces) must be defined **before** any physical group. Among physical groups, **physical lines must appear before physical surfaces**. Failing to respect this order will cause element labels to be misassigned at runtime. This constraint is due to the current implementation of the mesh reader and may be removed in a future version.

```yaml
Example:
  - type: mesh2D                    # Component type
    id: slice00                     # Component Id
    mesh: meshing/sliceXY_00.msh    # Reference mesh file
    initial: {t: 4.3}               # Initial temperature imposed in all the geometry
    regions:                        # Introduces the list of labels that identify␣
↪specific 2D
                                    # regions of the 2D geometry
      - label: SS                   # Label name
        material: stainless_steel   # Material name used in this specific region
        source:                     # Heat load scenario to be specified
          time:  {h5: Qcase_00.h5, data: t}  # Time scenario (s)
          value: {h5: Qcase_00.h5, data: d}  # Heat load scenario consistent with the
                                    # time array (W/m)
    edges:                          # Introduces the list of labels that identify␣
↪specific 1D
                                    # boundaries of the 2D geometry
      - {label: BC_bound,   type: flux,  value: 0.0}  # No heat flux to the 1D␣
↪boundary
                                    # identified by the label BC_bound
    channel_link:
      - OP_Sl00_Hole02              # label that is expected to be thermally connected
                                    # to a channel component
      - OP_Sl00_Hole01              # label that is expected to be thermally connected
                                    # to a channel component
    solid_link:
      - P00_Tur02_Sl00              # label that is expected to be thermally connected
                                    # to a solid component
```

type	*object*	
properties		
• type	const	mesh2D
• **id**	*'id' has to be unique and it is used for connections*	
	type	*string*
• **mesh**	*Mesh file specification with name and path defining the geometry location*	
	type	*string*
• extrusion_length	*Extrusion length of the 2D slice (m)*	
	Value if nothing is specified by the user: `1.062`	
	numeric_value	
• **initial**	type	*object*
	properties	
	• **t**	*Initial temperature (K)*
		numeric_value
• **regions**	type	*array*
	items	*mesh2D_region*
• **edges**	type	*array*
	items	*mesh2D_edge*
• channel_link	*List of labels dedicated to thermal contact with hydraulic channel - If they exist*	
	1D_string	
• solid_link	*List of labels dedicated to thermal contact with solid components - If they exist*	
	1D_string	

mesh2D_region

type	*object*	
properties		
• **label**	*Name of the label that identifies a specific 2D region of the 2D geometry*	
	This label must exist in the .msh file generated by Gmsh software from a .geo file that describes the 2D geometry and introduces such labels	
	type	*string*
• **material**	*Material name*	
	Built-in names: `copper`, `nb3sn`, `nbti`, `stainless_steel`, `glass_epoxy`, `glass_kapton_glass`. Custom materials can be defined in `materials` (see *Top level configuration*).	
	type	*string*
• **source**	*Heat load applied (W) to the region identified by the label*	
	0D_signal	

mesh2D_edge

type	*object*	
properties		
• **label**	*Name of the label that identifies a specific 1D boundary of the 2D geometry*	
	This label must exist in the .msh file generated by Gmsh software from a .geo file that describes the 2D geometry and introduces such labels	
	type	*string*
• **type**	*The user can specify either a temperature or a thermal flux at the level of the considered 1D slice boundary*	
	type	*string*
	enum	temp, flux
• **value**	*Imposed temperature (K) of thermal flux (W) depending on the type of the thermal boundary condition*	
	0D_signal	

1.2.3 Link components

junction

A junction connects 1D helium channels together

```
Example:
  - type: junction  # Component type
    link:           # Several branches expected - In this specific configuration,
                    # outlet of pipe1 is connected to inlets of both pipe2 and pipe3
    - id: pipe1     # Channel id
      node: out     # Involved channel boundary
      kappa: 1      # Kappa=1 means no pressure loss increase at the junction
    - id: pipe2
      node: in
      angle:  90    # Angle between the reference pipe1 and the present branch
    - {id: pipe3,   node: in,   angle:  90}
```

type	*object*	
properties		
• type	const	junction
• **link**	A junction can link several branches together (not limited number)	
	type	*array*
	items	*junction_link_type*

junction_link_type

type	*object*	
properties		
• **id**	*Identification of one of the branches involved in the junction* - 'id' of the corresponding channel component is required	
	type	*string*
• **node**	*Specification of the channel boundary connected to the junction*	
	It can be `in` for inlet or `out` for outlet	
	type	*string*
• kappa	*Correction of the pressure loss coefficient via the introduction of the factor ''kappa"*	
	Value if nothing is specified by the user: `1.0` (no pressure loss increase)	
	If ''kappa" is specified by the user, it must only be defined on the first item (the 1st branch is automatically referred as the reference one for this specific junction)	
	numeric_value	
• angle	*Angle (degrees) between the reference branch (1st item) and the branch associated to the current item*	
	Only define it from the second item	
	Value if nothing is specified by the user: `0.0` degree	
	numeric_value	

fluidlink

Fluid-FLuid link that connects one helium channel to another - Transverse mass, momentum and energy exchanges are allowed

```
Example:
  - type: fluidlink                  # Component type
    wetted_perim_channels: 31.42e-3  # Wetted perimeter delimiting the␣
↪boundary
```

(continues on next page)

(continued from previous page)

```
                                    # between the channels (m)
    link:  # Two channels must be identified - In this specific␣
↪configuration,
           # the cells 1 to 5 from pipeTF_Pan1_BG are connected to cells 1␣
↪to 5
           # from pipeTF_Pan1_Hol
    - id: pipeTF_Pan1_BG    # Channel Id
      node: [1, 2, 3, 4, 5] # Channel involved nodes
    - {id: pipeTF_Pan1_Hol, node: [1, 2, 3, 4, 5]}  # Id + involved nodes are
                                                    # specified for this␣
↪channel
```

type	*object*	
properties		
• type	const	fluidlink
• **wetted_perim_channels**	*Wetted perimeter delimiting the boundary between the channels (m)*	
	numeric_value	
• **nusselt**	*Nusselt number correlation name*	
	Built-in names: `pipe`, `DBG`. Custom correlations can be defined in `nusselt_correlations` (see *Top level configuration*).	
	type	*string*
• **link**	Two channels must be identified	
	type	*array*
	items	*fluidlink_link_type*
	maxItems	2
	minItems	2

fluidlink_link_type

type	*object*	
properties		
• **id**	*Identification of one of the channels involved*	
	'id' of the corresponding channel component is required	
	type	*string*
• **node**	*List of cell indexes that are connected to cells of the other channel component*	
	The lists of the two items must be the same size - The i-th element of the item 1 list will be connected to the i-th element of the item 2 list	
	1D_integer	

thermalink

Fluid-Solid link that thermally connects one helium channel to one of the following components (strand, solid or mesh2D)

```
Example:
  - type: thermalink              # Component type
    contact_surface: [0.067, 0.067, 0.067, 0.067, 0.068, 0.068]  # Contact surfaces
                                  # in (m2) for each of the 6 contacts
    link:                         # One channel and one solid component are specified
                                  # in this specific configuration
    - id: pipeTF_Pan1_BG          # Id + involved nodes are specified for this channel
      node: [1, 2, 3, 4, 5, 6]
    - id: MC_TF_Pan1_C            # Id + involved nodes are specified for this solid
      node: [1, 2, 3, 4, 5, 6]
```

(continues on next page)

(continued from previous page)

```
        distance: [0.01, 0.01, 0.01, 0.01, 0.01, 0.01] # distances per chunk
                                  # used for the computation of the temperature
                                  # gradient within the considered chunk
```

type	*object*	
properties		
• type	const	thermalink
• wetted_perimeter	*Wetted perimeter of the channel in contact with the strand (m)*	
	Link channel/strand only - Required for this specific configuration	
	numeric_value	
• contact_surface	*List of contact surfaces (m2)*	
	Link channel/solid only - Required for this specific configuration - Each element represents a contact between a given channel cell and its associated solid chunk. A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.	
	scalar_or_D_numeric_value	
• heat_coefficient	*Imposed heat transfer coefficient (W/m2/K)*	
	Link channel/mesh2D only - Required for this specific configuration	
	numeric_value	
• nusselt	*Nusselt number correlation name*	
	Required for channel/strand and channel/solid configurations. Built-in names: `pipe`, `DBG`. Custom correlations can be defined in `nusselt_correlations` (see *Top level configuration*).	
	type	*string*
• **link**	One channel and one of the following components (strand, solid or mesh2D) must be identified - The channel component is always the first item	
	type	*array*
	items	*thermalink_link_type*
	maxItems	2
	minItems	2

thermalink_link_type

type	*object*
properties	
• **id**	*Identification of one of the components involved*
	'id' of the corresponding component is required
	type *string*
• node	*List of cell indexes that are connected to cells of the other component*
	To be used if the item corresponds to one of the following components (channel, strand or solid) - The lists of the two items must be the same size - The i-th element of the item 1 list will be connected to the i-th element of the item 2 list
	1D_integer
• label	*List of labels that are connected to cells of the channel component*
	Must be used instead of ''node" if the item corresponds to the mesh2D component. The lists of the two items must be the same size - The i-th element of the item 1 (channel) list will be connected to the i-th element of the item 2 (mesh2D) list
	1D_string
• distance	*List of distances (m) used for the computation of the temperature gradient within the solid chunks defining the solid component*
	Link channel/solid only - Required for this specific configuration - Property only defined in the item dedicated to the solid component - Each element of the list represents the distance involved in the computation of the temperature gradient within the associated solid chunk - The list of distances and the list of nodes must be the same size. A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.
	1D_numeric_value

solidlink

Solid-Solid link that thermally connects a solid component with another, or a solid component with a mesh2D slice.

```
Example:
  - type: solidlink          # Component type
    material: glass_epoxy    # Insulation material name
    thickness: [0.0]         # Zero thickness in this example,
                             # meaning that the current thermal contact is␣
↪full
    link:
    - id: MC_TF_Pan4_R       # Id + involved node are specified for this solid
      node: [5]
      distance: [2.430e-03] # distance used for the computation of the␣
↪temperature gradient
                             # within the considered chunk
    - id: slice00
      label: [P00_Tur02_Sl00] # Id + label involved for this mesh2D component
```

type	*object*
properties	
• type	const solidlink
• contact_surface	*List of contact surfaces (m2)*
	Link solid/solid only - Required for this specific configuration - Each element represents a contact between a given solid chunk and its associated one. A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.
	1D_numeric_value
• **material**	*Insulation material name*
	Name of the material used in case of thermal resistance consideration for this specific Solid-Solid link. Built-in names: `glass_epoxy`, `glass_kapton_glass`. Custom materials can be defined in `materials` (see *Top level configuration*).
	type *string*
• **thickness**	*List of insulation thicknesses (m)*
	Each element represents a contact between a given solid chunk and its associated solid chunk or label (in case of a mesh2D component) - A non-zero thickness of insulation for a given contact is required in case of thermal resistance consideration - Put 0.0 for a given contact if full thermal contact is expected. A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.
	scalar_or_D_numeric_value
• **link**	One solid component and one of the following components (solid or mesh2D) must be identified - The mesh2D component (if present) is always the second item
	type *array*
	items *solidlink_link_type*
	maxItems 2
	minItems 2

solidlink_link_type

type	*object*
properties	
• id	*Identification of one of the components involved*
	'id' of the corresponding component is required
	type *string*
• node	*List of solid chunk indexes that are connected to solid chunks or labels of the other component, depending on its type*
	To be used if the item corresponds to a solid component - The lists of the two items must be the same size - The i-th element of the item 1 list will be connected to the i-th element of the item 2 list
	1D_integer
• label	*List of labels that are connected to solid chunks of the solid component*
	Must be used instead of ''node" if the item corresponds to the mesh2D component. The lists of the two items must be the same size - The i-th element of the item 1 (solid) list will be connected to the i-th element of the item 2 (mesh2D) list
	1D_string
• distance	*List of distances (m) used for the computation of the temperature gradient within the solid chunks defining the solid component*
	Property only defined for the item dedicated to a solid component - Each element of the list represents the distance involved in the computation of the temperature gradient within the associated solid chunk - The list of distances and the list of nodes must be the same size. A single scalar can be provided instead of a list: REIMS will fill the array with that value repeated for every node.
	scalar_or_D_numeric_value

pump

A pump connects two 1D helium channels together

```
Example:
  - type: pump     # Component type
    m0: 1e-3       # Mass flow rate: 1 g/s
    link:          # 2 links: link 1 - pump inlet, link 2 - pump outlet
     - id: pipe    # inlet of the pump connected to outlet of 'pipe'
       node: out   # outlet pipe
     - id: pipe    # outlet of the pump connected to inlet of 'pipe'
       node: in    # inlet pipe
```

type	*object*	
properties		
• type	const	pump
• **m0**	*Pump-imposed mass flow rate (kg/s)*	
	numeric_value	
• **link**	Two channels must be identified - The outlet of one channel component (1st link) must be connected to the inlet of the other	
	type	*array*
	items	*circulator_link_type*
	maxItems	2
	minItems	2

compressor

A compressor connects two 1D helium channels together

```
Example:
  - type: compressor     # Component type
    m0:  0.002           # Mass flow rate parameter (kg/s)
    dp0: 3.3e5           # Pressure parameter (Pa)
    link:                # 2 links: link 1 - pump inlet, link 2 - pump outlet
     - id: pipe          # inlet of the pump connected to outlet of 'pipe'
       node: out         # outlet pipe
     - id: pipe          # outlet of the pump connected to inlet of 'pipe'
       node: in          # inlet pipe
```

type	*object*	
properties		
• type	const	compressor
• **m0**	*Mass flow rate parameter (kg/s) involved in the compressor characteristic definition*	
	numeric_value	
• **dp0**	*Pressure parameter (Pa) involved in the compressor characteristic definition*	
	numeric_value	
• **link**	Two channels must be identified - The outlet of one channel component (1st link) must be connected to the inlet of the other	
	type	*array*
	items	*circulator_link_type*
	maxItems	2
	minItems	2

circulator_link_type

type	*object*
properties	
• **id**	*Identification of one of the branches connected to the circulator*
	'id' of the corresponding channel component is required
	type *string*
• **node**	*Specification of the channel boundary connected to the circulator*
	It can be `in` for inlet or `out` for outlet - However, the node of the 1st link must be `out`
	type *string*

boundary_PT

Helium tank with imposed pressure and temperature acting as an inflow boundary condition - This inflow may become an outflow under certain circumstances

```
Example:
  - type: boundary_PT             # Component type
    p: 2.0e5                      # Tank pressure (Pa)
    t: 293.0                      # Tank temperature (K)
    link:  {id: pipe, node: in}   # Id + involved boundary are specified for this␣
↪channel
```

type	*object*	
properties		
• type	const	boundary_PT
• **p**	*Imposed pressure (Pa)*	
	0D_signal	
• **t**	*Imposed temperature (K)*	
	0D_signal	
• **link**	type	*object*
	properties	
	• **id**	*Identification of the channel component for which the boundary condition is defined*
		'id' of the corresponding channel component is required
		type *string*
	• **node**	*Specification of the involved channel boundary*
		It can be `in` for inlet or `out` for outlet
		type *string*
• hugoniot_boundary	*Exact Rankine-Hugoniot relations at this boundary*	
	Value if nothing is specified by the user: `no` • `yes`: Enforces conservation of mass, momentum and energy across the pressure wave at the boundary (exact Rankine-Hugoniot relations). Recommended for high-amplitude pressure transients (shocks, sudden pressure ramps). • `no`: Acoustic (linearized) approximation. Valid for low-amplitude pressure transients.	
	Boolean value	

boundary_mT

Helium inflow boundary condition with imposed mass flow rate and temperature

```
Example:
  - type: boundary_mT             # Component type
    mdot: 2.0e-3                  # Imposed mass flow rate (kg/s)
    t: 293.0                      # Imposed temperature (K)
    link:  {id: pipe, node: in}   # Id + involved boundary are specified for this␣
↪channel
```

type	*object*	
properties		
• type	const	boundary_mT
• **mdot**	*Imposed mass flow rate (kg/s)*	
	0D_signal	
• **t**	*Imposed temperature (K)*	
	0D_signal	
• **link**	type	*object*
	properties	
	• **id**	*Identification of the channel component for which the boundary condition is defined*
		'id' of the corresponding channel component is required
		type *string*
	• **node**	*Specification of the involved channel boundary*
		It can be `in` for inlet or `out` for outlet
		type *string*

1.2.4 Common definitions

0D_signal

```
Example:
  p:                          # Pressure
    time:   [     0,     100] # Time scenario (s)
    value:  [500000,  600000] # Value (Pa)
    order:  1                 # 1st order in time activated
    repeat: 0                 # No scenario repeat
    event:  no                # No intervention on the time step management

Example (in case of an external DLL):
  p:
    external: my_signal_func  # Function name exported by the DLL
    my_param: 1.0             # Any extra parameters passed to the DLL at init
```

oneOf	type	*number*
	type	*object*
	properties	
	• external	*Function name exported by a DLL*
		When `external` is present (the DLL must be declared in `external_libs`), `time` and `value` are not required. The additional parameters defined in the same block as `external` are passed to the DLL at startup. See the DLL developer guide for the required interface.
		type *string*
	• **time**	*Time (s)*
		1D_numeric_value
	• **value**	*Corresponding value*
		1D_numeric_value
	• order	*Time interpolation order (0 or 1)*
		Value if nothing is specified by the user: `0`
		type *integer*
	• repeat	*Index from which the signal will be repeated*
		Value if nothing is specified by the user: `0`
		type *integer*
	• event	*Event consideration based on Time data.*
		It can be: • `'no'` (value if nothing is specified by the user) if the user does not want to take care of specific events based on the 'time' data (the time step automatically evolves without intervention), • `'explicit'`: 3 consecutive explicit steps are enforced each time the simulation physical time reaches one of the values specified in 'time', • `'implicit'`: 1 implicit step is enforced each time the simulation physical time reaches one of the values specified in 'time'.
		type *string*

1D_signal

```yaml
Example:
  flux:                          # Heat load
    time:  [   0,  100]          # Time scenario (s)     - 2 instants
    x:     [ 0.0,  5.0,  10.0]   # Spatial positions (m) - 3 points
    value: [[0.1,  0.2],         # x=0.0m : value at t=0, t=100
            [0.3,  0.5],         # x=5.0m : value at t=0, t=100
            [0.2,  0.3]]         # x=10.0m: value at t=0, t=100
                                 # value dimension is [n_x=3 x n_time=2]
    order:  0                    # 0 order in time
    repeat: 0                    # No scenario repeat
    event:  implicit             # 3 consecutive explicit steps are enforced each␣
↪time the simulation
                                 # physical time reaches one of the values specified␣
↪in 'time'
```

(continues on next page)

(continued from previous page)

```
Example (in case of an external DLL):
  flux:
    external: my_signal_func  # Function name exported by the DLL
    my_param: 1.0             # Any extra parameters passed to the DLL at init
```

oneOf	type	*number*
	type	*object*
	properties	
	• external	*Function name exported by a DLL*
		When `external` is present (the DLL must be declared in `external_libs`), `time`, `x` and `value` are not required. The additional parameters defined in the same block as `external` are passed to the DLL at startup. See the DLL developer guide for the required interface.
		type *string*
	• time	*Time (s)*
		1D_numeric_value
	• x	*Position (m)*
		1D_numeric_value
	• value	*Corresponding value*
		2D_numeric_value
	• order	*Time interpolation order (0 or 1)*
		Value if nothing is specified by the user: `0`
		type *integer*
	• repeat	*Index from which the signal will be repeated*
		Value if nothing is specified by the user: `0`
		type *integer*
	• event	*Event consideration based on Time data.*
		It can be • `'no'` (value if nothing is specified by the user) if the user does not want to take care of specific events based on the 'time' data (the time step automatically evolves without intervention), • `'explicit'`: 3 consecutive explicit steps are enforced each time the simulation physical time reaches one of the values specified in 'time', • `'implicit'`: 1 implicit step is enforced each time the simulation physical time reaches one of the values specified in 'time'.
		type *string*

length_value

Total length, per-node lengths, or an HDF5 dataset reference.

oneOf	*h5*		
	Total length (m) of helium channel - In case of uniform mesh		
	type	*number*	
	Length of each node (m) - In case of non-uniform mesh		
	type	*array*	
	items	type	*number*

h5

type	*object*	
properties		
• **data**	*Dataset with table*	
	type	*string*
• **h5**	*Input hdf5 file*	
	type	*string*

2D_numeric_value

examples	[[0.0,0.0],[0.0,0.0]]		
oneOf	data	group/dataset	
	h5	file.h5	
	type	*array*	
	items	type	*array*
		items	type *number*
	h5		

1D_numeric_value

examples	[0.0,0.0]		
oneOf	data	group/dataset	
	h5	file.h5	
	type	*array*	
	items	type	*number*
	h5		

scalar_or_D_numeric_value

examples	0.0		
	[0.0,0.0]		
oneOf	data	group/dataset	
	h5	file.h5	
	type	*number*	
	type	*array*	
	items	type	*number*
	h5		

numeric_value

examples	0.0	
	4	
	data	group/dataset
	h5	file.h5
oneOf	type	*number*
	h5	

1D_string

type	*array*	
examples	[‘name1’,’name2’]	
items	type	*string*

1D_integer

examples	[0,0]		
oneOf	type	*array*	
	items	type	*integer*
	Range for 1D integer array		

Range for 1D integer array

Range for 1D integer array, specifying the start (from), end (to), and step size (step). Step can be negative to allow for decreasing sequences.

type	*object*	
examples	to	10
	from	5
	to	-5
	step	3
properties		
• **to**	type	*integer*
• from	type	*integer*
• step	type	*integer*

1D Boolean array

type	*array*
examples	[no, y, ‘yes’]
	[n,n,n]
items	*Boolean value*

Boolean value

This value can be boolean or string which is interpreted as a boolean.

enum	True, False, true, True, TRUE, on, On, ON, y, Y, yes, Yes, YES, false, False, FALSE, off, Off, OFF, n, N, no, No, NO

1.2.5 Friction correlations details

Three use cases are supported:

Case 1 — Built-in correlation with default parameters

No `friction_correlations` section is needed. Reference the correlation directly in the channel component:

```
components:
  - type: channel
    friction: blasius
```

Built-in names: `blasius`, `katheder`, `central_spiral`.

`blasius`: f = alpha * Re^{-beta} — Defaults: alpha=0.079, beta=0.25.

`katheder`: f = 0.25 * (19.5 / Re^{beta} + alpha) / $VoidFr^{0.742}$ — Defaults: alpha=0.0231, beta=0.7953, VoidFr=0.297, Re_min=1000. Returns zero for Re $\leq$ `Re_min`; only laminar friction applies below this threshold. Set `Re_min: 0` to activate Katheder for all Re > 0.

`central_spiral`: f = 0.25 * alpha / Re^{beta} — Defaults: alpha=0.36, beta=0.038.

where Re is the Reynolds number.

Case 2 — Built-in correlation with custom parameters

Define a named entry in `friction_correlations` using `base` to select the built-in, then override the parameters:

```
friction_correlations:
  - friction: my_blasius
    base: blasius
    alpha: 0.04
    beta: 0.22

components:
  - type: channel
    friction: my_blasius
```

Case 3 — User-defined correlation via external DLL

Provide a DLL that exports `init_friction_ext` and the correlation function. Declare the library in `external_libs` and define the correlation name and its parameters in `friction_correlations`:

```
external_libs:
- my_lib_friction

friction_correlations:
  - friction: friction_test1
    alpha_test: 0.75

components:
  - type: channel
    friction: friction_test1
```

The parameters defined under `friction_correlations` are passed to the DLL at startup. See the DLL developer guide for the required interface.

1.2.6 Nusselt correlations details

Three use cases are supported:

Case 1 — Built-in correlation with default parameters

No `nusselt_correlations` section is needed. Reference the correlation directly in the component:

```
components:
  - type: fluidlink
    nusselt: pipe
```

Built-in names: `pipe`, `DBG`.

`pipe`: Nu = a * Re^b * Pra^c — Defaults: a=0.023, b=0.8, c=0.4.

`DBG`: NuT = a * Re^b * Pra^c * $(T2/T1)^d$ → Nu = max(NuL, NuT) — Defaults: a=0.0259, b=0.8, c=0.4, d=-0.716, NuL=8.235.

Re is the Reynolds number, Pra the Prandtl number, T1/T2 the temperatures (K) of the two components. For `channel` (`thermal: temp`), T2 is the imposed wall temperature.

Case 2 — Built-in correlation with custom parameters

Define a named entry in `nusselt_correlations` using `base`, then override the parameters:

```
nusselt_correlations:
  - nusselt: my_pipe
    base: pipe
    a: 0.02
    b: 0.85

components:
  - type: fluidlink
    nusselt: my_pipe
```

Case 3 — User-defined correlation via external DLL

Declare the library in `external_libs`, add a named entry with any custom parameters:

```
external_libs:
- my_lib_nusselt

nusselt_correlations:
  - nusselt: nusselt_test1
    param_test: 1.5

components:
  - type: fluidlink
    nusselt: nusselt_test1
```

The parameters are passed to `init_nusselt_ext` at startup. See the DLL developer guide for the required interface.

1.2.7 Materials details

Optional list of named materials available to `strand`, `solid`, `solidlink`, and `mesh2D` components. Three use cases are supported:

Case 1 — Built-in material with default parameters

No `materials` section is needed. Reference the built-in name directly in the component:

```
components:
  - type: solid
    material: stainless_steel
```

Available built-in materials and their implemented properties:

- `copper`:

  - density: 8960 kg/m^3
  - resistivity

    J. Simon, E.S. Drexler, R.P. Reed, Properties of Copper and Copper Alloys at Cryogenic Temperatures, NIST Monograph 177, Washington DC, 1992 (draft 1987 version including B dependence)

  - thermal conductivity

    J. Simon, E.S. Drexler, R.P. Reed, Properties of Copper and Copper Alloys at Cryogenic Temperatures, NIST Monograph 177, Washington DC, 1992. B dependence added via a magnetoresistive term alpha*B/T/L0 (alpha~5e-11 Ohm.m/T, L0=2.44e-8 V2/K2) complementing the NIST dataset.

  - heat capacity

    L. Dresner, Stability of Superconductors, Plenum Press, NY, 1995

- `nb3sn`:
  - density: 8040 kg/m^3
  - thermal conductivity

    Fit of MATPRO data: L. Rossi, M. Sorbi, MATPRO: A Computer Library of Material Property at Cryogenic Temperature, INFN/TC-06/02, CARE-Note-2005-018-HHH. Original data from: H. Brechna, Superconducting Magnet Systems, Springer, 1973, p. 434.

  - heat capacity

    ITER DRG1 Annex, Superconducting Material Database, Article 5, N 11 FDR 42 01-07-05 R 0.1, Thermal, Electrical and Mechanical Properties of Materials at Cryogenic Temperatures (internal ITER report). Original data from: V.D. Arp, Stability and Thermal Quenches in Force-Cooled Superconducting Cables, Superconducting MHD Magnet Design Conf., MIT, pp 142-157, 1980; G.S. Knapp, S.D. Bader, Z. Fisk, Phonon properties of A-15 superconductors obtained from heat capacity measurements, Phys. Rev. B, 13(9), pp 3783-3789, 1976.

  - strain

    Linear model with a constant term and an electromagnetic contribution proportional to I x B.

  - critical temperature

    L. Bottura, B. Bordini, Jc(B,T,epsilon) Parameterization for the ITER Nb3Sn Production, IEEE Trans. Appl. Sup., 19(2), 1477-1480, 2009

  - critical field

    L. Bottura, B. Bordini, Jc(B,T,epsilon) Parameterization for the ITER Nb3Sn Production, IEEE Trans. Appl. Sup., 19(2), 1477-1480, 2009

  - critical current density

    L. Bottura, B. Bordini, Jc(B,T,epsilon) Parameterization for the ITER Nb3Sn Production, IEEE Trans. Appl. Sup., 19(2), 1477-1480, 2009

  - current sharing temperature

- `nbti`:
  - density: 6000 kg/m^3
  - thermal conductivity

    6th-order polynomial fit from MATPRO dataset: L. Rossi, M. Sorbi, MATPRO: A Computer Library of Material Property at Cryogenic Temperature, INFN/TC-06/02, CARE-Note-2005-018-HHH. Original data from: H. Brechna, Superconducting Magnet Systems, Springer, 1973, p. 424.

  - heat capacity

    Elrod S.A., Miller J.R., Dresner L., The specific heat of NbTi from 0-7T between 4.2 and 20K, Advances in Cryogenic Engineering Materials, Vol. 28, 1981. Extended to the full temperature range by smooth transition to a high-temperature asymptote of 400 J/kg/K.

  - strain

    Linear model with a constant term and an electromagnetic contribution proportional to I x B.

  - critical temperature

    M.S. Lubell, Scaling formulas for critical current and critical field for commercial NbTi, IEEE Trans. Mag., 19, 1983

  - critical field

    M.S. Lubell, Scaling formulas for critical current and critical field for commercial NbTi, IEEE Trans. Mag., 19, 1983

  - critical current density

    M.A. Green, Calculating the Jc, B, T Surface for Niobium Titanium Using a Reduced State Model, IEEE Trans. Mag., 25(2), 1989; G. Morgan, A Comparison of Two Analytic Forms for the Jc(B,T) surface, SSC-MD-218, 1989; L. Bottura, B. Bordini, Jc(B,T,epsilon) Parameterization for the ITER Nb3Sn Production, IEEE Trans. Appl. Sup., 19(2), 1477-1480, 2009; L. Zani, J.P. Serries, H. Cloez, M. Tena, E. Mossang, Jc(B,T) characterization of NbTi strands used in the ITER PF (Poloidal Field Coil)-relevant Insert and Full-scale sample, INIS-FR–2832, 2004

  - current sharing temperature

- `stainless_steel`:

  - density: 7900 kg/m^3

  - thermal conductivity

    J.M. Poncet, CEA-Grenoble, EFDA CRYOLA task.

  - heat capacity

    ITER DRG1 Annex, Superconducting Material Database, Article 5, N 11 FDR 42 01-07-05 R 0.1, Thermal, Electrical and Mechanical Properties of Materials at Cryogenic Temperatures (internal ITER report). Debye fit; for the model formula see for instance: L. Dresner, Stability of Superconductors, Plenum Press, NY, 1995. Original data from: J.M. Corsan and N.I. Mitchem, The Specific Heat of fifteen stainless steels in the temperature range 4K-30K, Cryogenics 19, p11-p16; J.M. Corsan and N.I. Mitchem, The Specific Heat of stainless steels between 4K and 300K, Proc. of the 6th ICEC, Grenoble, 1976; Aerospace Structural Metals Handbook, Metals and Ceramics Information Center, Battelle's Columbus Laboratories, Columbus, OH.

- `glass_epoxy`:

  - density: 1948 kg/m^3

  - thermal conductivity

    Cubic polynomial fit of data from: M.B. Kasen, G.R. MacDonald, D.H. Beekman Jr and R.E. Schrmm, Mechanical, Electrical and Thermal Characterisation of G-10CR and G-11CR Glass-Cloth/Epoxy Laminates Between Room Temperature and 4K, National Bureau of Standards, Boulder, Colorado.

  - heat capacity

    Power-law fit of data from: G. Hartwig, Low Temperature Properties of Resins and Their Correlations, Adv. Cryog. Eng., Vol. 24, 1978.

- `glass_kapton_glass`:

  - density: 1800 kg/m^3

  - thermal conductivity

    J.M. Poncet, J.P. Arnaud, P. Saint Bonnet, Thermal conductivity of materials or sandwiches used for magnet insulation of ITER project, SBT report CT 12-42, February 2013.

  - heat capacity

    Power-law fit of data from: G. Hartwig, Low Temperature Properties of Resins and Their Correlations, Adv. Cryog. Eng., Vol. 24, 1978.

Built-in material parameters

`density`	Material density (kg/m^3). Overrides the built-in default for this entry.
`RRR`	Residual Resistivity Ratio. Magnetoresistance correction factor in the copper resistivity model. Default: 100. *copper only*
`E0`	Electric field criterion (V/m). Reference field defining critical current in the power law. Default: 1.0e-5 V/m. *superconductors only*
`nPow`	Power law exponent. Exponent n in E = E0*(J/Jc)^n. Default: 5. *superconductors only*
`Bc20m`	Upper critical field at 0 K and zero intrinsic strain (T). Default: 29.39 T. *nb3sn only*
`Tc0m`	Critical temperature at zero field and zero intrinsic strain (K). Default: 16.48 K. *nb3sn only*
`nu`	Shape exponent for the temperature dependence of Bc2. nb3sn: 1.52, nbti: 1.7.
`Ca1`	First strain fitting constant. Default: 45.74. *nb3sn only*
`Ca2`	Second strain fitting constant. Default: 4.431. *nb3sn only*
`e0a`	Residual strain component. Default: 0.00232. *nb3sn only*
`emax`	Applied strain at which critical properties reach their maximum. Default: 0.0. *nb3sn only*
`C0`	Overall Jc scaling constant (A.T/m^2). Default: 8.0771e10. *nb3sn only*
`p`	Low-field flux pinning exponent. nb3sn: 0.556, nbti: 0.98.
`q`	High-field flux pinning exponent. nb3sn: 1.698, nbti: 0.98.
`str1`	Strain constant term. nb3sn: 0.0060942, nbti: 0.00742.
`str2`	Strain electromagnetic coefficient (1/(A.T)). nb3sn: 1.0777e-9, nbti: 1.301e-9.
`Bc20`	Upper critical field at 0 K (T). Default: 13.72 T. *nbti only*
`Tc0_p`	Critical temperature at zero field (K). Default: 8.79 K. *nbti only*
`CC0`	Overall Jc scaling constant (A.T/m^2). Default: 8.92534e11. *nbti only*
`n`	Exponent of the temperature-dependent factor (1-t^nu) in the Jc parameterization. Default: 1.96. *nbti only*

Case 2 — Built-in material with custom parameters

Define a named entry in `materials` using `base` to select the built-in, then override only the parameters that differ from the defaults:

```yaml
materials:
  - material: nb3sn_custom
    base: nb3sn
    Bc20m: 30.23
    Tc0m:  16.73
    E0:    1.0e-5
    nPow:  5

  - material: copper_rrr110
    base: copper
    RRR: 110.0

components:
  - type: strand
    stabilizer:
      material: copper_rrr110
      area: 5.0e-7
    superconductor:
      material: nb3sn_custom
      area: 2.5e-7
```

Case 3 — User-defined material via external DLL

Provide a DLL that exports `init_material_ext`. Declare the library in `external_libs` and define the material name and its parameters in `materials`:

```yaml
external_libs:
- my_lib_material

materials:
  - material: my_sc
    my_param: 1.0

components:
  - type: strand
    superconductor:
      material: my_sc
      area: 2.5e-7
```

The parameters defined under `materials` are passed to the DLL at startup. Only the properties actually implemented in the DLL are overridden — the remaining properties fall back to the built-in values if `base` is also provided, otherwise they remain unimplemented and trigger a runtime error if called. See the DLL developer guide for the required interface.

1.3 YAML standard

To describe models YAML file is used. Please see documentation of file format here:

https://yaml.org/

Supported standards are:

- YAML 1.2.2
- YAML 1.1
- YAML 1.0

Extended with merge dictionary feature: `<<:` Which was not included in the standard however it is very useful. Description in the draft for YAML 1.1:

https://yaml.org/type/merge.html

Fortunately many other system support it. So it became "semi" standard.

1.4 Include other YAML or JSON files

Instead of putting any value in YAML file we replace it with dictionary contain:

```yaml
include: file_name.json
```

The file will be included in this specific place. Files can be `json` or `yaml`.

For example:

```yaml
components:
- type: channel
   id: pipe
   mesh: variable
   nodes: {include: nodes.yaml}
```

Please be aware that 3rd party software like language server or `schema` validator doesn't understand this feature and might issue an error however REIMS will work correctly.

1.5 Execution command lines

The following command lines allow the user to customize the configuration for parallel computing:

```
set OMP_DISPLAY_ENV=TRUE        # Shows OpenMP environment settings when program starts
set OMP_PROC_BIND=close         # Binds threads near each other (to nearby cores)
set OMP_PLACES=threads          # Places each OpenMP thread on a separate hardware␣
↪thread
set OMP_NUM_THREADS=96          # Uses 96 threads for OpenMP parallel regions - Number␣
↪of threads to be adapted
set MKL_NUM_THREADS=30          # Uses 30 threads for Intel MKL (Math Kernel Library) -
↪ Number of threads to be adapted
set MKL_DEBUG_CPU_TYPE=5        # Simulates a specific CPU type (for debugging MKL)
set MKL_ENABLE_INSTRUCTIONS=5   # Forces MKL to use a specific instruction set (like␣
↪AVX-512)
set MKL_DISPLAY_ENV=TRUE        # Shows MKL environment info at runtime
```

Here is the execution command line to apply in a command prompt, considering the executable reims.exe file as well as a given input.yaml file:

```
path_exe\reims.exe path_inp\input.yaml
```

The keywords `path_exe` and `path_inp` refer to the paths of the executable file and the input file, respectively.

1.6 How to cite REIMS

D. Furfaro, J. Kosek, A. Ovcharov, T. Schioler, R. Rotella, T. Luce, A new fast and robust thermo-hydraulic code for ITER superconducting magnet simulation, Cryogenics, Volume 144, 2024, 103978