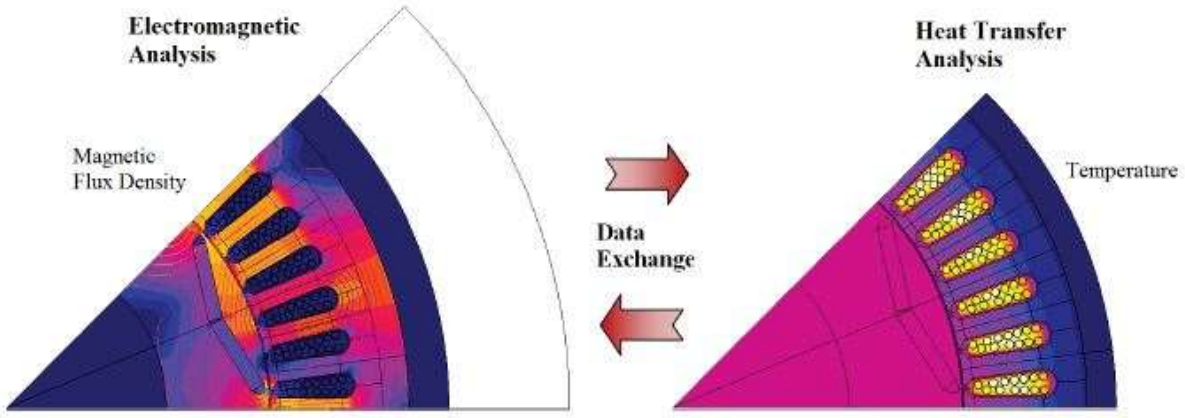
*CAE package for electromagnetic and thermal analysis using finite elements*

# Altair® Flux®

# *Electromagnetic and thermal analysis of a brushless IPM motor by multi-physics coupling*

## *2D technical example*




ALTAIR

This technical paper was designed and edited on 4 November 2022 with the participation of EPM_NM Lab, University POLITEHNICA of Bucharest, Romania

**Altair**
15 Chemin de Malacher - Inovallée
38246 Meylan Cedex
FRANCE
Phone: +33 (0)4 76 90 50 45
Fax: +33 (0)4 56 38 08 30

Web: http://www.altair.com

# Foreword

*(Please read before starting this document)

**Description of the example**

The goal of this technical example is to demonstrate the ability and advantage of Flux for the simulation of interior permanent magnet motor computation problems. This document contains the general steps and all the data needed to describe the different simulations.

**To begin**

This example is designed for the user who is already familiar with the basic functions of Flux software.
For beginner users, please report to the "Flux Starting Guide" opened automatically by the supervisor. (If not opened, please open it by clicking on the button "?" on the top right of the supervisor). The interface contains videos, which helps the beginners while using Flux for the first time

**Support files included...**

To view the completed phases of the example project, the user will find the .py files, including the geometry, physics and post-processing descriptions. The .py files corresponding to the different study cases in this example are available in the folder:

..\DocExamples\Examples2D\Tutorial_Technical\Brushless_IPM_Motor_Thermalanalysis\

Supplied files are command files written in Pyflux language. The user can launch them in order to automatically produce the Flux projects for each case.

**(*.py files are launched by accessing **Project/Command file** from the Flux drop down menu.)*

| **Supplied files** | | **Flux file obtained after launching the .py file** |
|---|---|---|
| CASE1 | CASE_1_MG_INI.FLU | |
| | CASE_1_MG1.FLU | |
| | CASE_1_MG2.FLU | |
| | MG_CIRC.xcir | |
| CASE2 | CASE_2_TH_INI.FLU | |
| | CASE_2_TH.FLU | |
| CASE3 | CASE_3_MG_INI.FLU | |
| | CASE_3_TH_INI.FLU | |
| | CASE_3_MG.FLU | |
| | CASE_3_TH.FLU | |
| | MAIN_MG.py; MAIN_EX_MG.py; CASE_3_MG.py | CASE_3_MG.FLU |
| | MAIN_TH.py; MAIN_EX_TH.py; CASE_3_TH.py | CASE_3_TH.FLU |

The content of the files in the previous table is described below:

CASE_1_MG_INI.FLU file contains the geometry, mesh, physical application and solving scenario for the transient / time dependent magnetic analysis of the studied motor (first study case);
CASE_1_MG1.FLU and CASE_1_MG2.FLU files contain the results of the first study case;
MG_CIRC.xcir file contains the electric circuit model used for the analysis of the studied motor in the first and the third study cases;

CASE_2_TH_INI.FLU file contains the geometry, mesh, physical application and solving scenario for the steady state thermal analysis of the studied motor (second study case);
CASE_2_TH.FLU file contain the results of the second study case;

CASE_3_MG_INI.FLU file contains the geometry, mesh, physical application and solving scenario for the magnetic problem of the multiphysics analysis (third study case);
CASE_3_TH_INI.FLU file contains the geometry, mesh and physical application for the thermal problem of the multiphysics analysis (third study case);
CASE_3_MG.FLU and CASE_3_TH.FLU files contain the results of the third study case;
MAIN_MG.py; MAIN_EX_MG.py; CASE_3_MG.py; MAIN_TH.py; MAIN_EX_TH.py; CASE_3_TH.py are files written in Pyflux language and they include the commands for the multiphysics analysis (third study case);

The user may modify the *_INI.FLU files to study other motor configurations. In case of an electromagnetic and thermal multiphysics analysis if the geometry is modified the modifications should be made in both the magnetic and thermal *_INI.FLU files.

# TABLE OF CONTENTS

# 1. Theoretical aspects related to the coupling between transient / time dependent magnetic and steady state thermal

**Introduction**

The coupling between **transient / time dependent magnetic** and **steady state thermal** takes into account the interaction between periodic electromagnetic field (e.g. periodic variation in time of magnetic field strength, currents, voltages) and steady state thermal field that occurs in electromagnetic devices.
The operation of a brushless Interior Permanent Magnet (IPM) motor from electromagnetic point of view is described by Maxwell's equations. The steady state heating as a result of the power losses in the machine (Joule losses in the stator windings, iron losses in the stator/rotor magnetic cores, eddy currents in the PMs) is described by the Fourier equation.
The electromagnetic properties of materials (e.g. resistivity of copper conductors) usually depend on the temperature. That is why the electromagnetic and the heat transfer phenomena in electromagnetic devices depend more or less upon each other.
An accurate analysis of a brushless IPM motor requires the coupling of specific electromagnetic and thermal phenomena. The electromagnetic analysis in time domain of the machine with relative rotor- stator motion is necessary to reach the periodic steady state variation of quantities that determine the mean values in time over an electric cycle of the Joule losses (in the stator windings and in the PMs) and of the iron losses using Bertotti model (in the stator/rotor magnetic cores). The proper evaluation of these losses is very important since they represent the heat sources for the steady state thermal analysis.
The temperature dependence of some material properties (other than the resistivity of stator windings and resistivity of magnets) is not considered in this document.

**Contents**

This chapter contains the followings topics:

# 1.1. Mathematical model of transient / time dependent magnetic problems

**Introduction**

The vector potential ***A*** model of the electromagnetic field is frequently used in transient / time dependent magnetic 2D problems.

**Equation**

The partial differential equation associated to the transient / time dependent magnetic field analysis of brushless IPM motors expressed in magnetic vector potential ***A*** is:

$$curl\ (\nu \cdot curl\ \boldsymbol{A} - \boldsymbol{H_c}) + \sigma \cdot \partial \boldsymbol{A} / \partial t = 0$$

where:

- $\sigma = 1/\rho$ is the electrical conductivity, inverse of the resistivity
- $\nu = 1/\mu$ is the inverse of magnetic permeability ($\mu$)
- $\boldsymbol{H}_c$ is the coercive magnetic field of PM regions

In the 2D applications, where the magnetic vector potential has the structure $\boldsymbol{A}[0, 0, A(x,y,t)]$, the Coulomb gauge condition $div\boldsymbol{A} = 0$ (which ensures the solution uniqueness) is automatically satisfied

**About materials … (temperature dependent properties)**

In the coupled electromagnetic and thermal problems there are material properties that depend on the temperature θ (e.g. resistivity $\rho$ of the copper wires).
Since the electrical machines typically operate at low temperatures (to avoid insulation damage) the variation with temperature of certain properties of materials can be neglected (e.g. electric and magnetic properties of laminations etc.).

The Joule losses in the brushless IPM motor can be split in two components: Joule losses in the stator copper wires and eddy current Joules losses in the PM regions due to eddy currents.
The current density in the copper wires can be expressed by the formula: $J = I/Scc$ where $I$ is the rms value of the stator current per phase and $Scc$ is the cross-section area of the wire. The expression of the volume density of Joule losses in the copper wires is: $\rho \cdot J^2$, where $\rho$ is the resistivity, locally dependent on temperature.
The expression of the eddy current density in the PM regions is $j = -\sigma \cdot \partial A/\partial t$ and the volume density of the induced power is $p = \rho |J|^2$, where $J$ is the local rms value of the current density.

**Iron losses**

The iron losses in the stator and rotor magnetic cores of the brushless IPM motor have three components (hysteresis losses, eddy current losses and excess losses). The evaluation of these losses by using Bertotti model requires the solution in time domain of the periodic electromagnetic field over an electric cycle.

# 1.2. Mathematical model of thermal steady state problems

**Introduction**

Thermal steady state analysis is useful to estimate the time independent temperature fields.

**Equations**

The evaluation of the thermal steady state field $\theta(\mathbf{r})$ in thermal conductive regions, characterized by thermal conductivity $k$, where the heat sources are expressed by the volume density $p$, supposes the integration of the **Fourier equation**:

$$div(k\ grad\ \theta) + p = 0$$

The associated boundary condition of the general form:

$$k\ \mathbf{n} grad\ \theta = h(\theta - \theta_a) + \varepsilon C_n[(T/100)^4 - (T_a/100)^4] + p_s,$$

where $T = \theta + 273.15\ [K]$ is the thermodynamic temperature, expresses the thermal exchange by convection (coefficient $h$) and by radiation (coefficient $\varepsilon$) on the surface/boundary of a thermal conductive region to/from the environment having the ambient temperature $\theta_a$. The quantity $p_s$ is the imposed surface density of the thermal flux on the related boundary. $C_n$ is the Stefan - Boltzmann constant.

# 1.3. Multiphysics coupling between transient / time dependent magnetic and steady state thermal: solving principle

**Solving principle**

The multiphysics coupling between a **transient /time dependent magnetic** problem **and steady state thermal** problem characterizes periodic time dependent steady-state electromagnetic phenomena and thermal phenomena in devices such as: electrical machines, transformers, coils. A periodic steady state of a time dependent quantity does not mean necessarily a harmonic variation in time of that quantity.
The coupling between time dependent magnetic and steady state thermal applications supposes first to solve the equations of a time dependent magnetic field until the periodic steady state is reached. Then, the thermal steady state equations are solved, in which the thermal sources coresponding to a full electric cycle are extracted from the time dependent electromagnetic field problem (i.e. Joule and iron losses).
After the update of the temperature dependent material properties, the equations of the time dependent magnetic field are solved again until periodic steady state is reached and the thermal steady state equations are solved again too. The procedure is continued until convergence is achieved.

The time derivatives in the transient / time dependent magnetic field analysis are approximated by finite differences. The step by step in time domain solution of the magnetic problem considers a series of discrete time step values ($t_0$, $t_1$, $t_2$, ... $t_i$ ..., $t_f$) in the study time domain [$t_0$, $t_f$]. The limit $t_f$ of this time domain is that for which the periodic steady state is reached and for which a full electric cycle is included. The time step value should be chosen so as to obtain a good compromise between accuracy and computational effort.

Before proceeding to the thermal problem, the results of the last electric cycle in the electromagnetic field analysis are used to compute the heat sources (i.e. Joule and iron losses).

The algorithm of the multiphisics coupling is summarized in the table and diagram below.

| Phase | Description |
|---|---|
| 1 | Step by step in time domain solving of Maxwell's equations taking into account the material properties for the imposed initial temperature field (step 1 thermal), until the periodic steady state of electromagnetic field quantities is reached. Then, compute the volume density $p$ or the surface density $p_s$ of losses (i.e. iron losses and Joule losses) over the last full electric cycle. |
| 2 | Solve the Fourier equation using the heat sources from the previous electromagnetic step. The new solution of the temperature field (step 2 thermal) is used to prepare the next transient / time dependent magnetic field calculation. |

*Continued on next page*

| | |
|---|---|
| 3 | Compute new fields of the electromagnetic quantities with thermal properties, which correspond to step thermal 2 of the temperature field. |
| 4 | Return to Phase 1 for the computations that represent the next step of the multiphysics coupling. |

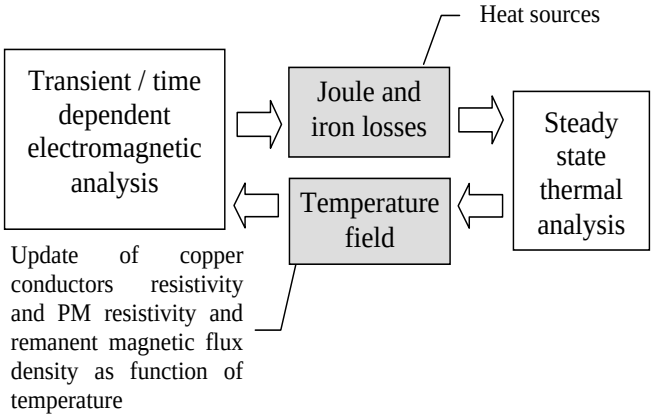



*Diagram of a transient / time dependent electromagnetic – steady state thermal multiphysics analysis*

**Computation time**

The solving of a multiphysics coupling usually requires a long computation time. In addition, if problems are magnetic nonlinear, this time can increase considerably.

# 2. Overview

**Introduction**

The main goal of this technical paper is to demonstrate the Flux capabilities in the 2D coupled electromagnetic and thermal analysis of a brushless IPM motor by multiphysics coupling.

This section presents an overview of the studied problem. It includes a brief description of the device and the studied cases.

**Contents**

This chapter contains the followings topics:

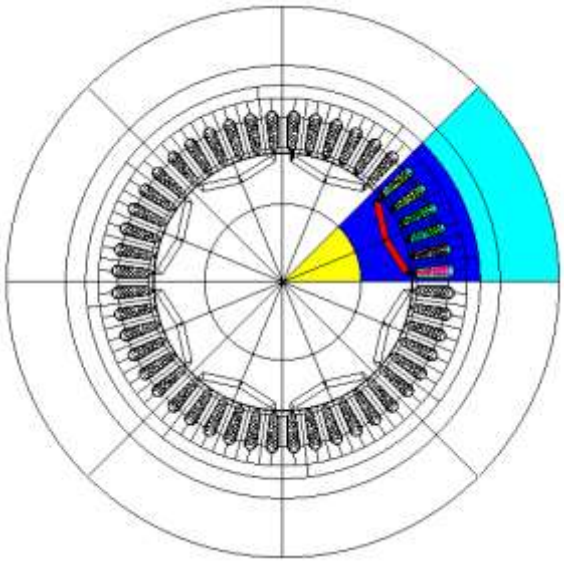
# 2.1. General presentation

**Studied device**

The studied device is a brushless Interior Permanent Magnet (IPM) motor and it includes the following parts:

- a fixed part (stator) including the stator core, the slots, and the winding
- a movable part (rotor) with embedded permanent magnets
- an air gap between the stator and the rotor

A cross-section of the studied device, perpendicular to the motor axis, is presented in the figure below.



**Motor ratings**

This motor with the following data:

- Max bus voltage: 500 V
- Peak torque: 400 Nm
- Max speed: 6000 rpm
- Peak power rating: 50 kW at 1200-1500 rpm

is designed for a hybrid electric vehicle.

**Motor main characteristics**

Other characteristics of the studied motor are presented below:

- 48 stator slots
- 3 phase wye connected stator winding
- 4 pole pairs of the magnetic field
- NdFeB magnets
- Lamination type M270-35A
- Outer diameter: 242 mm
- Stack length: 75 mm

## 2.2. Studied cases

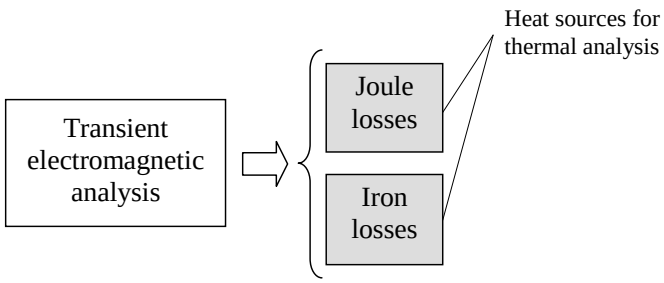
**Studied case**

Three cases are studied in this technical paper:

- **Case 1:** transient / time dependent magnetic analysis of the motor until periodic steady state is reached. The speed of the rotor is imposed;
- **Case 2:** steady state thermal analysis of the motor. The efficiency of the cooling system is analyzed by parameterizing the heat exchange coefficients;
- **Case 3:** multiphysics coupling periodic steady state electromagnetic - steady state thermal analysis of the motor for constant rotor speed. Material properties of copper conductors are temperature dependent.

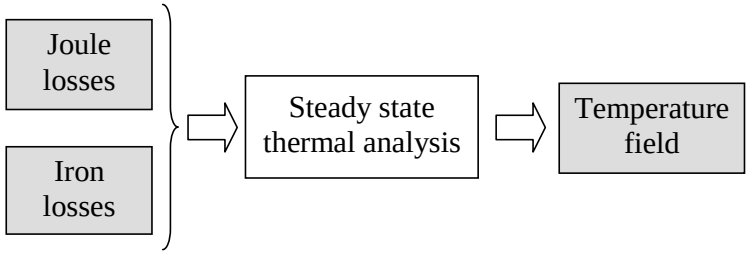
**Case 1**

The material properties are temperature independent.
The rotor speed is constant and the phase currents are harmonic.
The Joule losses and iron losses are evaluated after the periodic steady state of the time dependent electromagnetic field is reached. These losses are the heat sources for the thermal analysis.
The computations will be carried out for two different peak values of the stator currents.




**Case 2**

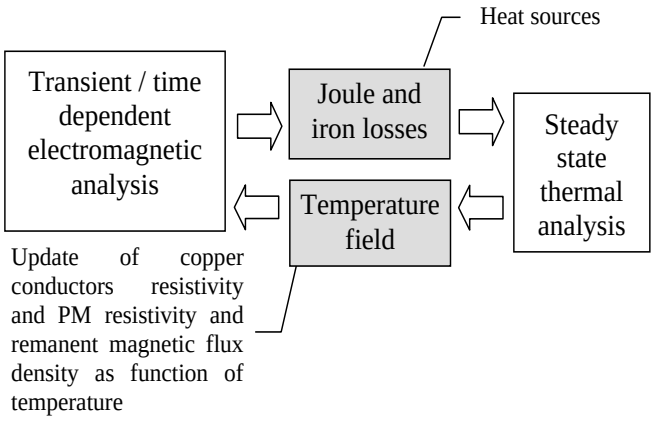
The material properties are temperature independent.
The Joule and iron losses computed previously represent the heat sources in the steady state thermal analysis of the machine.
The temperature field depends on the values of the heat exchange coefficients have; that is why the influence of these coefficients will be analyzed by a parametric study.




*Continued on next page*

**Case 3**

Electromagnetic and thermal phenomena are coupled since the resistivity of the copper conductors is temperature dependent as well as the PMs properties (resistivity and remanent magnetic flux density).



## 2.3. Electromagnetic phenomena: description of the device

**Application**

This part of the study referes to a **Transient Magnetic 2D application**.

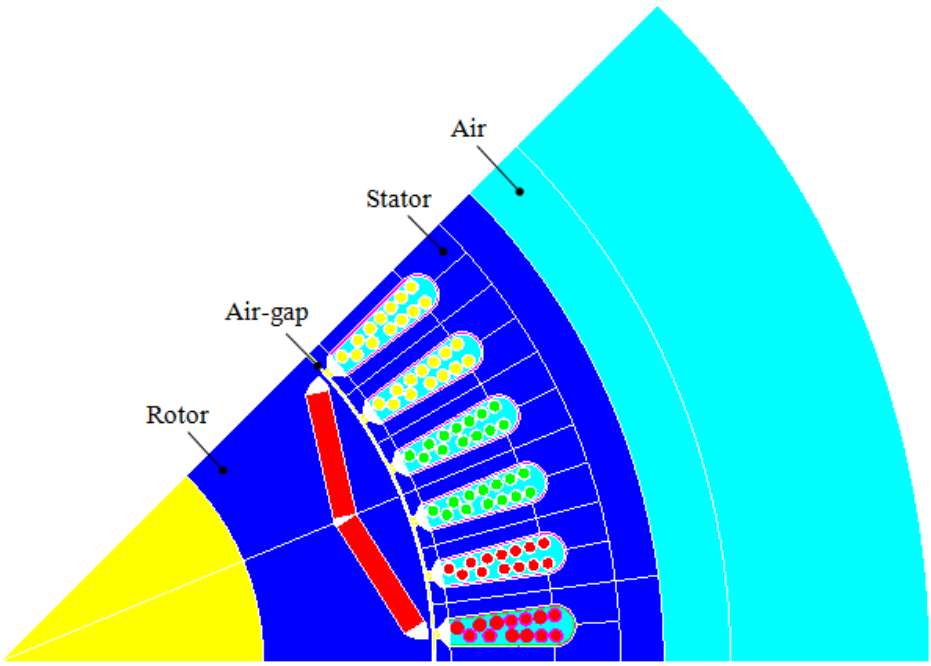
**Project name**

The magnetic application (without results) is saved under the name **CASE_1_MG_INI.FLU** and it is situated in the folder **CASE_1**.
The electric circuit is saved under the name **MG_CIR.xcir** and it is situated in the same folder.

**Geometry**

The geometry of the computation domain in the figure below is represented by 1/8 of the cross section through the studied brushless IPM motor surrounded by air.
The main regions of the 2D computation domain are:

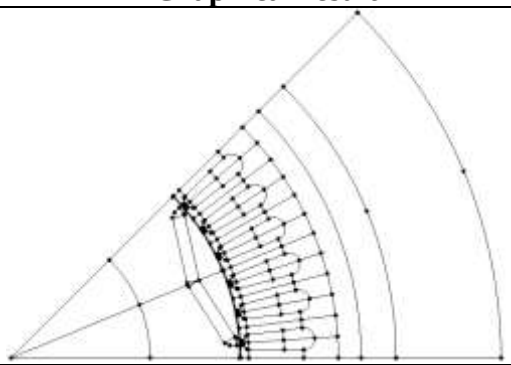
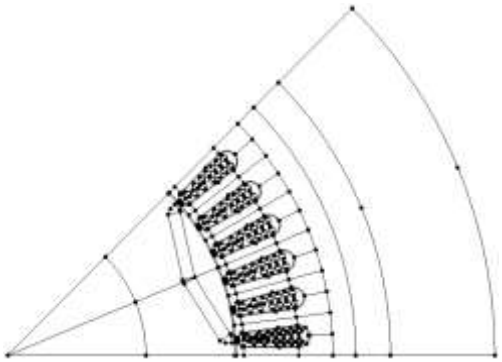
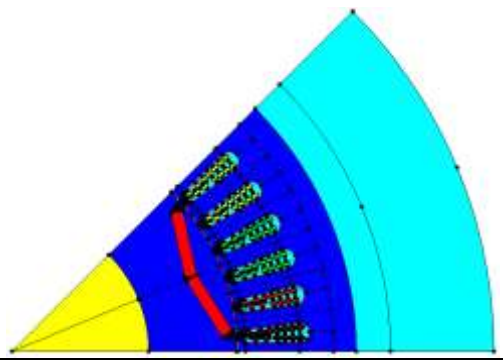
- *Stator* - immobile part of the machine, which includes the laminations made stator magnetic core and the stator slots that contain the copper conductors (13 conductors per slot), conductor insulation, impregnation, liner, preslot, wedge;
- *Air* - air surrounding the motor; it includes an *Infinite Box* region used in open boundary problems;
- *Rotor* - movable part of the machine that includes the rotor magnetic core made of laminations, the permanent magnets and small air regions near the magnets;
- *Rotating airgap* - a layer of air situated between the stator and the rotor that is a special air region used to take into account the relative rotor-stator motion.




*Computation domain for electromagnetic field analysis: one magnetic pole*

*Continued on next page*

An outline of the geometry building process of the brushless IPM motor is presented in the table below.

| No. | Stage description | Graphical result |
|---|---|---|
| 1 | **Creation of preliminary geometry by using Overlay facility** (See section 2.1.2. *Create a brushless permanent motor using the overlay* in *Brushless IPM motor tutorial - 2D technical example document*) |  |
| 2 | **Completion of geometry:**<br>- forced removal of the central points of the stator slots<br>- creation of points and lines of the copper conductors with 2.5 mm diameter (13 conductors per slot)<br>- creation of points and lines of the conductors insulation (thickness of insulation 0.15 mm)<br>- creation of points and lines of the slot liner (thickness of 0.5 mm)<br>- multiplication of the newly created points of the slot liner and copper conductors using the STATOR_ROT transformation |  |
| 3 | **Building faces** |  |

*Continued on next page*

The user may use the following **Geometry** menu options to create new points/lines or to force delete existing points and lines:

| | |
|---|---|
| Geometry → Point → New |  |
| Geometry → Line → New |  |
| Geometry → Point → Force Delete |  |
| Geometry → Line → Force Delete |  |

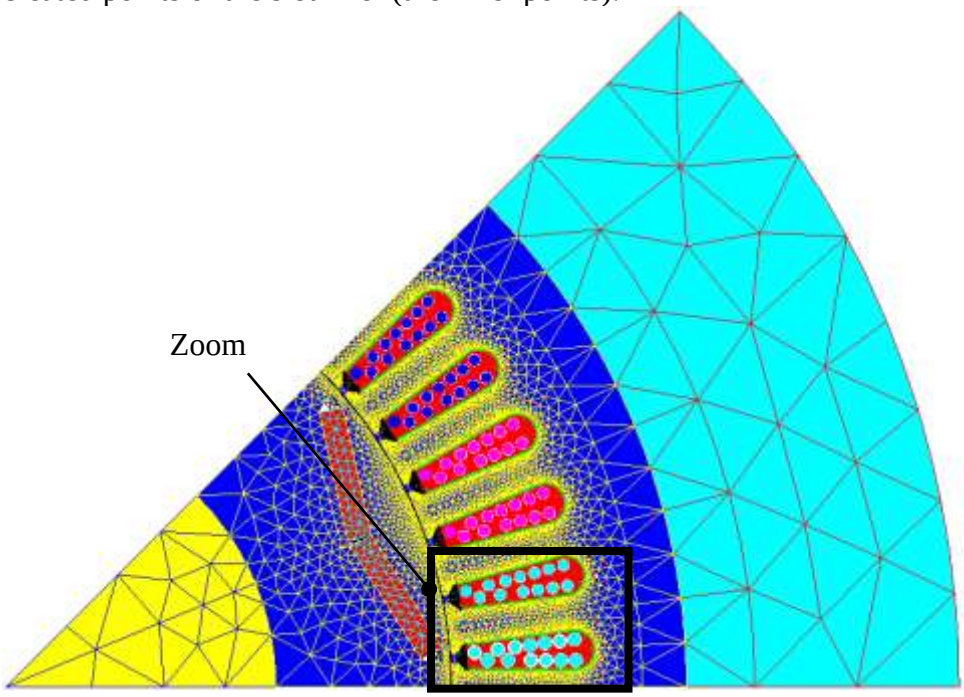
To build faces the following **Geometry** menu options should be used:

| | |
|---|---|
| Geometry → Build → Build Faces |  |

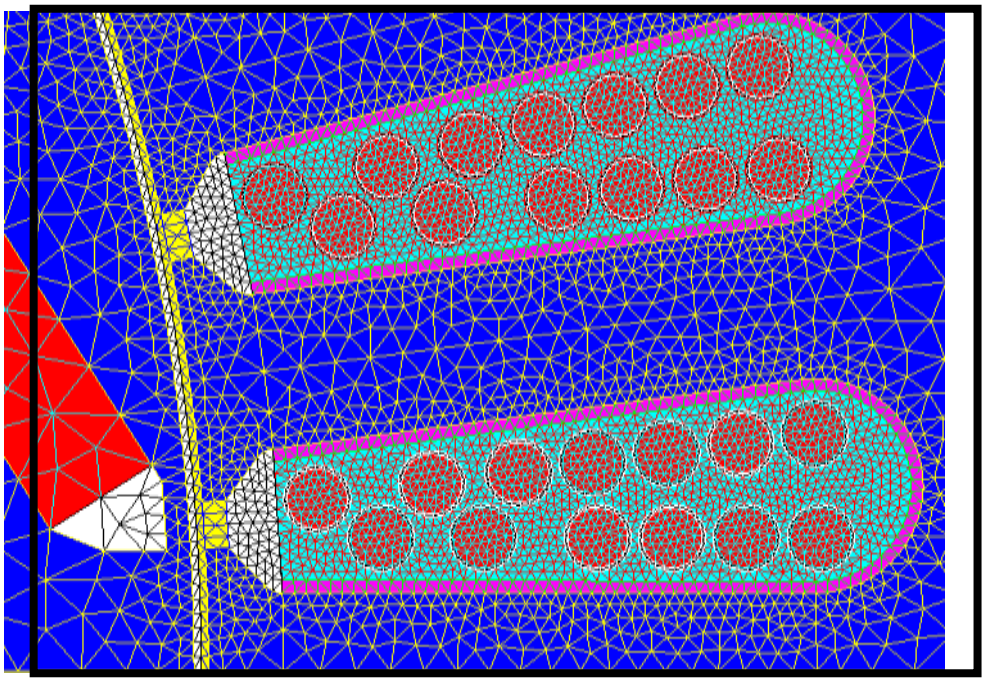
*Continued on next page*

**Mesh**

The computation domain is meshed as shown in the figure below.
Besides the mesh entities generated by the *Motor Template* an additional mesh point named COND (mesh size of 0.4 mm) is created and assigned to the points defining the copper conductors and their insulation layer.
The STATOR_INTERNAL_WEDGE mesh point is associated to the newly created points of the slot liner (the inner points).




*Mesh of the electromagnetic field computation domain*



*Zoom of the mesh including two stator slots*

To add a new mesh point the following **Mesh** menu options should be used:

| | |
|---|---|
| ☞ Mesh → Mesh point → New |  |

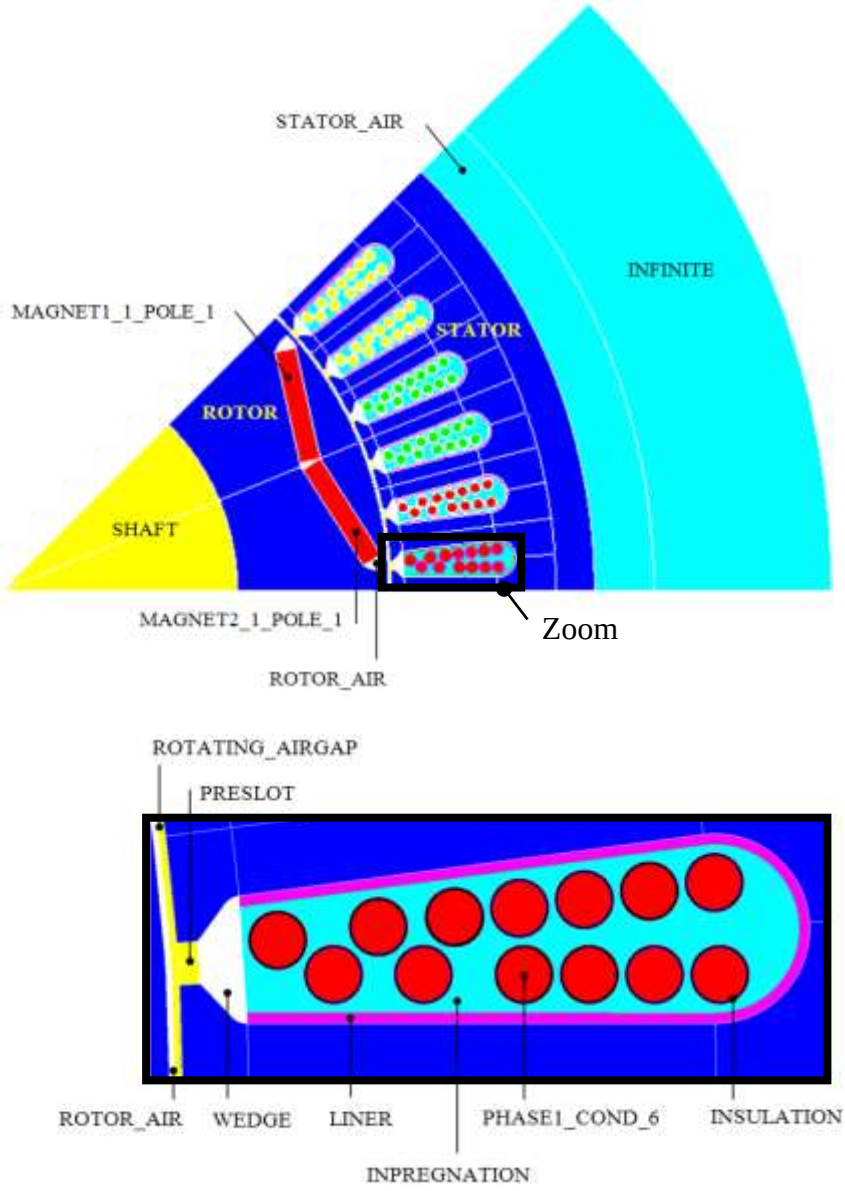
To build the mesh on the computation domain and to generate second order elements the user should follow the **Mesh** menu options below:

| | |
|---|---|
| ☞ Mesh → Mesh → Mesh faces |  |
| ☞ Mesh → Mesh → Generate second order elements |  |

*Continued on next page*

**Physics**

- The computation domain used for the transient / time dependent magnetic analysis of the brushless IPM motor is composed of 92 face regions named: IMPREGNATION, INFINITE, INSULATION, LINER, MAGNET1_1_POLE_1, MAGNET2_1_POLE_1, PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26, PRESLOT, ROTATING_AIRGAP, ROTOR, ROTOR_AIR, SHAFT, STATOR, STATOR_AIR, WEDGE




*Face regions of electromagnetic field computation domain*

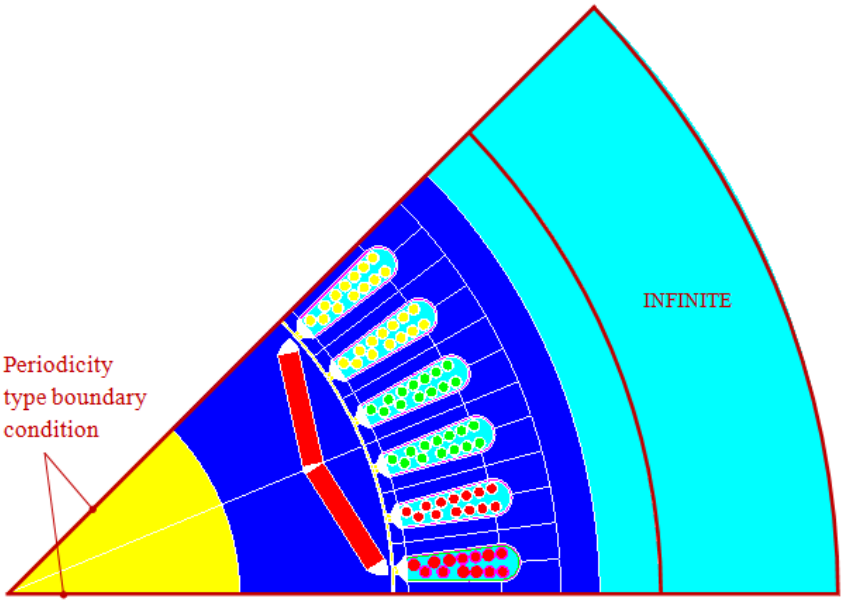
To create a new face region the user should follow the **Physics** menu options below:

| | |
|---|---|
| Physics → Face region → New |  |

To assign regions to faces the following **Physics** menu options should be used:

| | |
|---|---|
| Physics → Face region → Assign regions to faces (completion mode) |  |

- The computation domain is delimited by:
  - an INFINITE face region (special region called also INFINITE BOX used to model the open boundary problems) and
  - 2 radial boundaries on which a *periodicity type boundary condition* is imposed. This boundary condition allows the connection of the magnetic vector potential values in the nodes situated along the two radial boundaries of the computation domain. By applying such special boundary conditions - *odd or anticyclic boundary condition in this case*, the computation domain is reduced to only a magnetic pole, respectively 1/8$^{th}$ of the entire geometry.




*Infinite box region (INFINITE) and periodicity type boundary condition*

To create a periodicity the user should use the following **Physics** menu options:

| | |
|---|---|
| Physics → Periodicity → New |  |

To assign regions to lines the following **Physics** menu options could be used:

Physics → Line region → Assign regions to lines 

**Materials**

The following materials should be generated (created or imported) for this study:

| Material | | | |
|---|---|---|---|
| **Name** | **Comment** | **B(H)** | **J(E)** |
| COPPER | Material used for stator copper conductor regions | See following blocks | |
| COPPER_TEMP | Material used for stator copper conductor regions with temperature dependent resistivity | See following blocks | |
| FLU_M270_35A | Material used for stator and rotor magnetic core regions | See following blocks | |
| NDFEB | Material used for permanent magnet regions with temperature independent properties | See following blocks | |
| NDFEB_1TEMP and NDFEB_2TEMP | Materials used for permanent magnet regions with temperature dependent resistivity and remanent magnetic flux density | See following blocks | |

**B(H): COPPER**

The model used for the B(H) characteristic is the following:
*Linear isotropic.*
The magnetic property of this model is the relative permeability $\mu_r = 1$.

**J(E): COPPER**

The model used for the J(E) characteristic is the following:
*Isotropic constant resistivity.*
The corresponding value is: $\rho = 1.7E\text{-}8$ [Ωm]

**B(H): COPPER_TEMP**

The model used for the B(H) characteristic is the following:
*Linear isotropic.*
The magnetic property of this model is the relative permeability $\mu_r = 1$.

**J(E): COPPER_TEMP**

The model used for the J(E) characteristic is the following:
*Isotropic constant resistivity.*
The corresponding value is: $\rho$ = RESISTIVITY_CU
where:
RESISTIVITY_CU = 1.7*(1+0.00427*(TKelvin-293.15))*1E-8 [Ωm]

**B(H):**
**FLU_M270_35A**

The model used for the B(H) characteristic is the following:
*Isotropic spline saturation.*
The characteristics of the magnetic property are given as a table and included in the Flux Materials Database (imported material).

---

**B(H):**
**NDFEB**

The model used for the B(H) characteristic is the following:
*Linear magnet described by the Br module.*
The characteristics of the magnetic property are presented in the table below.

| **B(H)** magnetic property**: Linear magnet described by the Br module** | |
|---|---|
| **Remanent flux density [T]** | **Relative permeability $\mu_r$** |
| 1.2 | 1.05 |

---

**J(E):**
**NDFEB**

The model used for the J(E) characteristic is the following:
*Isotropic resistivity.*
The corresponding constant value is: $\rho$ = 1.4E-6 [Ωm]

---

**B(H):**
**NDFEB_1TEMP**

The model used for the B(H) characteristic is the following:
*Spatial linear magnet with temperature dependent properties.*
The characteristics of the magnetic property are presented in the table below.

| **B(H)** magnetic property**: Spatial linear magnet** | |
|---|---|
| **Spatial remanent flux density [T]** | **Relative permeability $\mu_r$** |
| *Vec3(BX_1, BY_1, 0) | 1.05 |

* where:
BR_NDFEB = 1.2*(1-0.0013*(TKelvin-293.15));
BX_1= BR_NDFEB*Cosd(22.5+10+IO(ANGPOS_ROTOR_DEG))
BY_1= BR_NDFEB*Sind(22.5+10+IO(ANGPOS_ROTOR_DEG)).

**J(E):**
**NDFEB_1TEMP**

The model used for the J(E) characteristic is the following:
*Spatial isotropic resistivity (temperature dependent).*
The corresponding value is: $\rho$ = RESISTIVITY_NDFEB [Ωm]
where:
RESISTIVITY_NDFEB = 1.4*(1+0.0014*(TKelvin-293.15))*1E-6 [Ωm]

**B(H):**
**NDFEB_2TEMP**

The model used for the B(H) characteristic is the following:
*Spatial linear magnet (with temperature dependent properties).*
The characteristics of the magnetic property are presented in the table below.

| **B(H)** magnetic property**: Spatial linear magnet** | |
|---|---|
| **Spatial remanent flux density [T]** | **Relative permeability $\mu_r$** |
| **Vec3(BX_2, BY_2, 0) | 1.05 |

** where:
BR_NDFEB = 1.2*(1-0.0013*(TKelvin-293.15));
BX_2 = BR_NDFEB*Cosd(22.5-10+IO(ANGPOS_ROTOR_DEG));
BY_2 = BR_NDFEB*Sind(22.5-10+IO(ANGPOS_ROTOR_DEG)).

**J(E):**
**NDFEB_2TEMP**

The model used for the J(E) characteristic is the following:
*Spatial isotropic resistivity (temperature dependent).*
The corresponding value is: $\rho$ = RESISTIVITY_NDFEB [Ωm]

**Materials and cases**

The materials used in this study are presented in the table below along with their associated face regions and study cases.

| Material | | |
|---|---|---|
| **Materials** | **Regions** | **Cases** |
| COPPER | PHASE1_COND_1, PHASE1_COND_2... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2... PHASE3_COND_26 | Case 1 |
| COPPER_TEMP | PHASE1_COND_1, PHASE1_COND_2... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2... PHASE3_COND_26 | Case3 |
| FLU_M270_35A | STATOR and ROTOR | Case 1, Case 3 |
| NDFEB | MAGNET1_1_POLE_1, MAGNET2_1_POLE_1 | Case 1 |
| NDFEB_1TEMP | MAGNET1_1_POLE_1 | Case 3 |
| NDFEB_2TEMP | MAGNET2_1_POLE_1 | Case 3 |
| VACUUM (default) | IMPREGNATION, INFINITE, INSULATION, LINER, PRESLOT, ROTATING_AIRGAP, ROTOR_AIR, SHAFT, STATOR_AIR, WEDGE | Case 1, Case 3 |

To create new materials the following **Physics** menu options could be used:

| | |
|---|---|
| Physics → Material → New |  |
| Physics → Face region → Assign regions to faces (completion mode) |  |

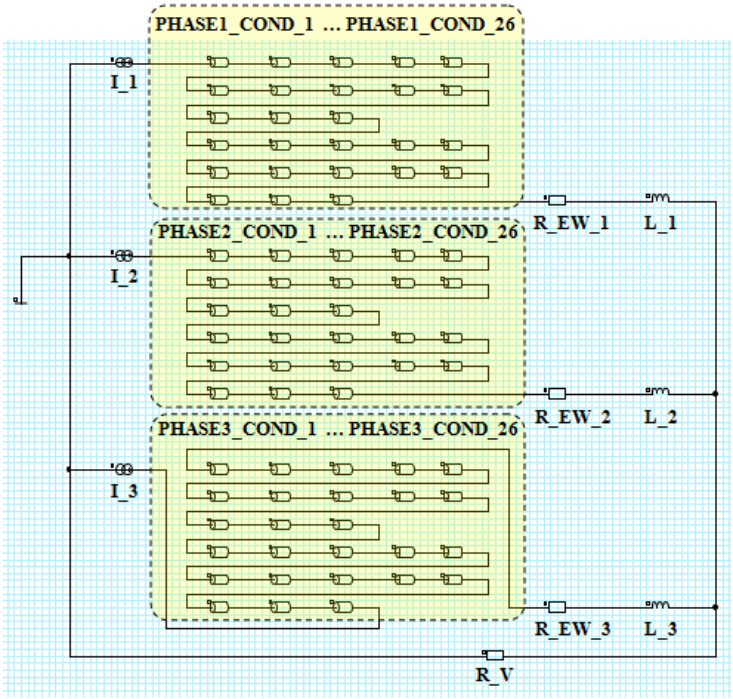
*Continued on next page*

**The circuit model**

The electric circuit in the figure below associated to the electromagnetic field model includes the following components:

- 3 x 26 = 78 solid conductors (26 series connected solid conductors per phase): PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26;
- 3 a.c. current sources I_1, I_2, I_3 (one a.c. current source per each phase) with the following expressions:
  I_1 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180);
  I_2 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180-2*Pi()/3);
  I_3 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180-4*Pi()/3);
  The constants used in the expressions above are the following:
  MAX_CURRENT = 100 A (or 80 A in case 1);
  OMEGA = 2*Pi()*FREQUENCY;
  FREQUENCY = SPEED/60*POLES/2 = 1200/60*8/2 = 80 Hz,
  GAMMA = 45°;
- 3 inductors of 159 μH (one per each phase): L_1, L_2, L_3;
- 3 resistors of 34 mΩ (one per each phase): R_EW_1, R_EW_2, R_EW_3 (in case 3 the resistance value will be temperature dependent);
- 1 resistor of 10 kΩ modeling a voltmeter: R_V.

Each conductor placed in a stator slot is considered as a distinct thermally insulated solid conductor in which Joule losses are dissipated. This assumption is needed for a proper heat transfer analysis of the machine.




*Electric circuit asociated to the studied brushless IPM motor*

*Continued on next page*

To create an electric circuit the following **Physics** menu options could be used:

| | |
|---|---|
| Physics → Circuit → Circuit editor context |  |

| | |
|---|---|
| Physics → Circuit → Circuit editor context |  |

To assign a coil conductor component to a face region the user could use the following **Physics** menu options:

| |
|---|
| Physics → Assign coil conductor components to regions → Assign coil conductor components to face regions |

## 2.4. Thermal phenomena: description of the device

**Application**

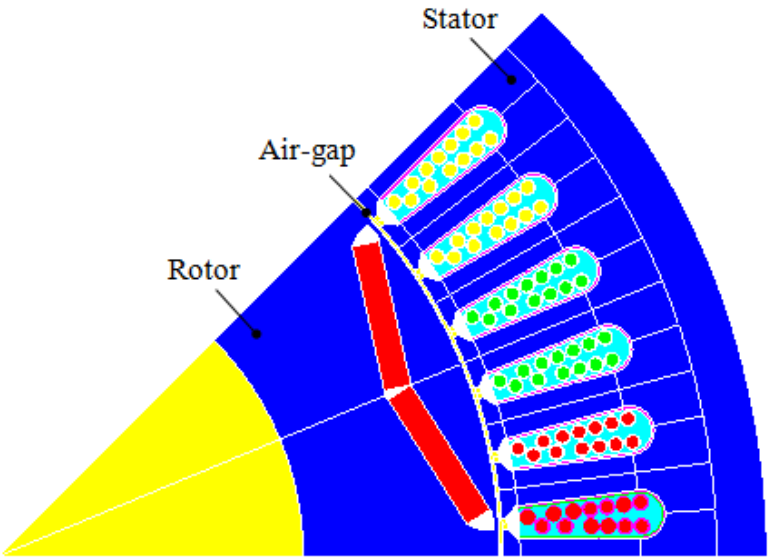
The application **Steady State Thermal 2D** for the thermal analysis of the device uses the temperature expressed in Kelvin degrees.

**Project name**

The thermal application (without results) is saved under the name **CASE_2_TH_INI.FLU** and it is situated in the folder **CASE_2**.

**Geometry**

The computation domain is similar to the one used for the **Transient Magnetic 2D** analysis of the machine but it does not contain the INFINITE and STATOR_AIR regions.

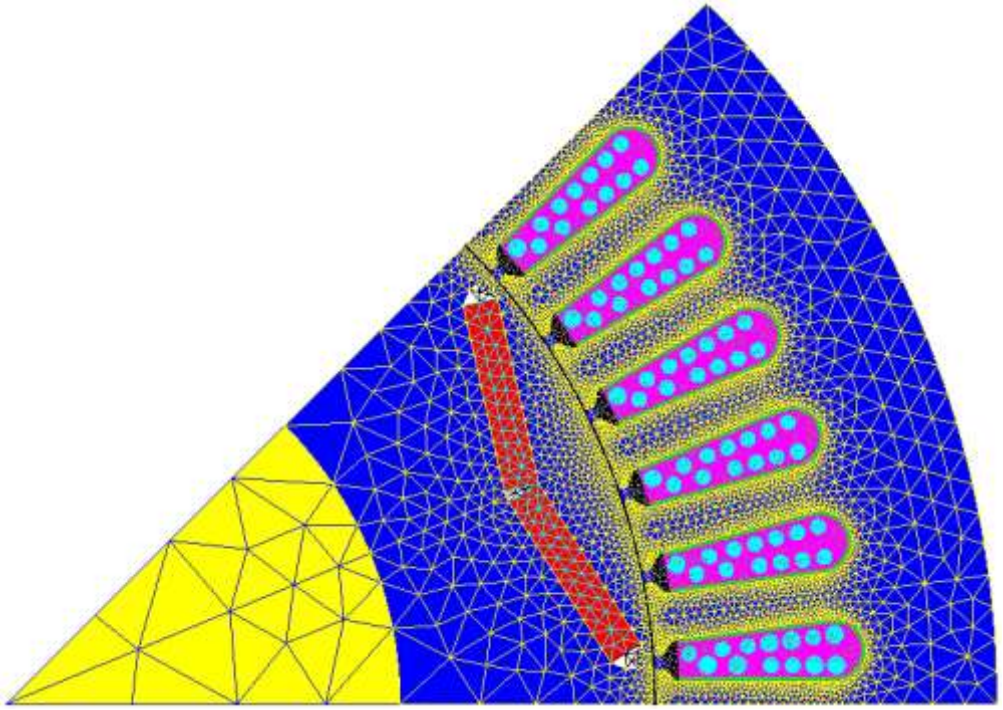



*Computation domain for heat transfer analysis: one magnetic pole*

*Continued on next page*

**Mesh**

The computation domain is meshed as presented in the figure below. The mesh similar to the one used for the **Transient Magnetic 2D** study.



*Mesh of the thermal field computation domain*

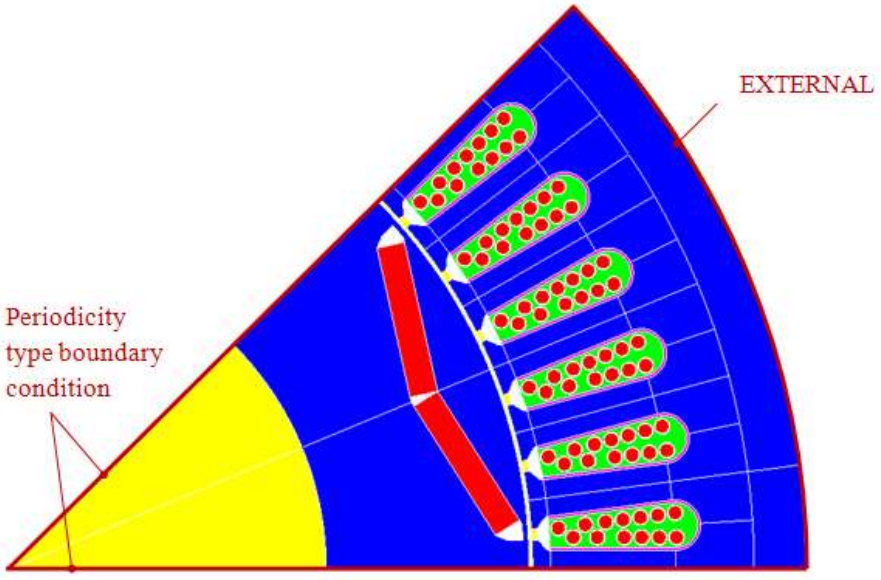
| | |
|---|---|
| Mesh → Mesh → Mesh faces |  |
| Mesh → Mesh → Generate second order elements |  |

*Continued on next page*

**Physics**

- The computation domain used for the steady state thermal analysis of the brushless IPM motor is composed of 15 face regions named: IMPREGNATION, INSULATION, LINER, MAGNET1_1_POLE_1, MAGNET2_1_POLE_1, STATOR_COND_ PHASE1, STATOR_COND_ PHASE2, STATOR_COND_ PHASE3, PRESLOT, ROTATING_AIRGAP, ROTOR, ROTOR_AIR, SHAFT, STATOR, WEDGE
  The face regions are practically the same as for the transient magnetic analysis with the following exceptions:
  - STATOR_AIR and INFINITE face regions are completely removed along with the points and lines that define them;
  - STATOR_COND_ PHASE1 is composed of the following 26 face regions used in the transient magnetic analysis: PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26;
  - STATOR_COND_ PHASE2 is composed of the following 26 face regions used in the transient magnetic analysis: PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26;
  - STATOR_COND_ PHASE3 is composed of the following 26 face regions used in the transient magnetic analysis: PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26.

- The computation domain is delimited by:
  - a *line region* named EXTERNAL situated at the exterior of the stator and
  - 2 radial boundaries on which a *periodicity type boundary condition* is imposed (*Cyclic boundary condition* should be imposed in this case).




*Line region EXTERNAL and priodicity type boundary condition*

| | |
|---|---|
| Physics → Face region → New |  |
| Physics → Face region → Assign regions to faces (completion mode) |  |

☞ Physics → Face region → Force delete 

To create new line regions the following **Physics** menu options should be used:

☞ Physics → Line region → New 

To assign regions to lines the following **Physics** menu options could be used:

☞ Physics → Line region → Assign regions to lines 

Simulating a forced circulation water cooling system, the following thermal exchange coefficients are used on the EXTERNAL line region:
*Convection coefficient:* $h = 1000$ [$Wm^{-2}°C^{-1}$] (in case 2 a parametric analysis is carried out this coefficient varying in the range {100 ÷ 1200} [$Wm^{-2}°C^{-1}$])
*Emissivity coefficient:* $\varepsilon = 0$
*The external water temperature:* $\theta_a = 70$ [°C]

**Materials**

The following materials are used in the thermal analysis of the motor:

| Material | | |
|---|---|---|
| **Name** | **Comment** | **k(T)** |
| AIR_AIR_GAP | Airgap region | See following blocks |
| AIR_CLASSICAL | Air regions | See following blocks |
| FLU_COPPER | Stator conductors | See following blocks |
| IMPREGNATION | Stator impregnation material | See following blocks |
| INSULATION | Insulation of stator conductors | See following blocks |
| IRON | Stator and rotor magnetic cores | See following blocks |
| LINER | Slot liner | See following blocks |
| NDFEB | Permanent magnets | See following blocks |

**k(T): AIR_AIR_GAP**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding constant value is: $k = 0.025+3\cdot10^{-3}$ [$Wm^{-1}°C^{-1}$]

**k(T): AIR_CLASSICAL**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 0.025$ [$Wm^{-1}°C^{-1}$]

**k(T): FLU_COPPER**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 394$ [$Wm^{-1}°C^{-1}$]

**k(T): IMPREGNATION**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 0.2$ [$Wm^{-1}°C^{-1}$]

**k(T): INSULATION**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 0.21$ [$Wm^{-1}°C^{-1}$]

**k(T): IRON**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 55$ [$Wm^{-1}°C^{-1}$]

**k(T): LINER**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding mathematical formula is: $k = 0.21$ [$Wm^{-1}°C^{-1}$]

**k(T): NDFEB**

The model used for the thermal conductivity is the following:
*Isotropic constant thermal conductivity*.
The corresponding constant value is: $k = 9$ [$Wm^{-1}°C^{-1}$]

**Materials and cases**

The materials used in this study are presented in the table below along with their associated face regions and study cases.

| Material | | |
|---|---|---|
| **Materials** | **Regions** | **Cases** |
| AIR_AIR_GAP | Airgap region | Case 2, Case 3 |
| AIR_CLASSICAL | Air regions | Case 2, Case 3 |
| FLU_COPPER | Stator conductors | Case 2, Case 3 |
| IMPREGNATION | Stator impregnation material | Case 2, Case 3 |
| INSULATION | Insulation of stator conductors | Case 2, Case 3 |
| IRON | Stator and rotor magnetic cores | Case 2, Case 3 |
| LINER | Slot liner | Case 2, Case 3 |
| NDFEB | Permanent magnets | Case 2, Case 3 |

Physics → Material → New

# 3. About multiphysics / data exchange and synchronization

**Definition**

From an operating perspective, the multiphysics coupling is defined as two processes that are executed **in parallel** with **exchanged data**.

The available software tools are:
- the tools for **handling** the **exchanged data**
- the tools for **synchronization** of processes and exchanges

**Contents**

This chapter contains the following topics:

# 3.1. Data exchange: multipoint support

**Exchanged data**

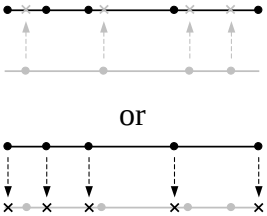
In our example, the exchanged data includes:
- the chart of volume density of power losses (M ⇒ T)
- the chart of temperature (T ⇒ M)

The data is spatial data, which requires **exchange supports**.

**Exchange support**

Various solutions are possible:
- in specific cases (if two problems have the same mesh) the nodes of the mesh can constitute the support of exchange.
- in general cases (if two problems have different mesh) the coordinates of the nodes are transferred from one problem to another via the Flux entity named **multipoint support**.

These two solutions are illustrated in the figure below.

| If identical mesh | If different mesh |
|---|---|
| Data exchange at the nodes of the mesh | Data exchange at a group of points |
|  |   |
| Support of exchange = group of nodes | Support of exchange = group of nodes |

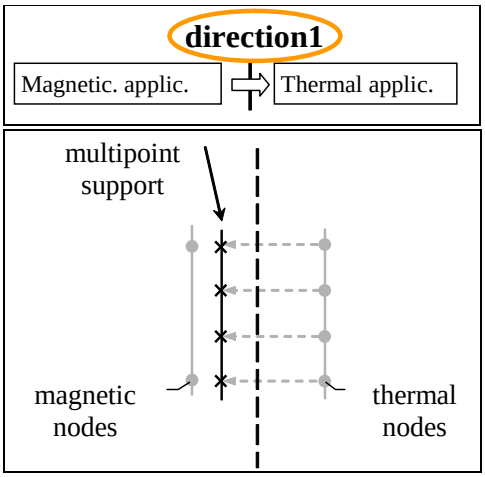
To each **node** (of one mesh) a **position** (of the other mesh) is associated

*Continued on next page*

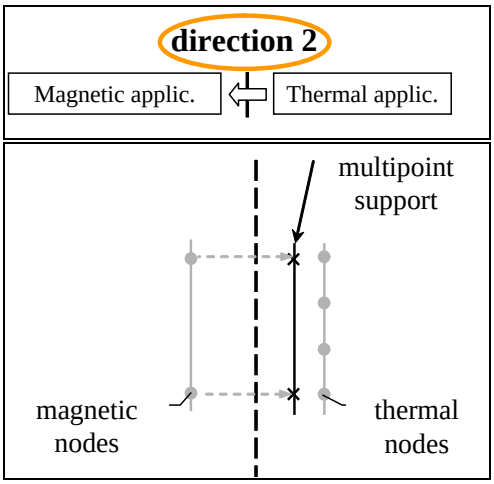
**… in our example**

In our example the mesh of the regions of data transfer (the regions where the heating is produced) may be different for the electromagnetic and thermal applications; this is why we use multipoint supports.
The multipoint supports necessary for data exchange are presented below.

To carry out the transfer of data in direction 1 (which means the transfer of power density from the electromagnetic application into the thermal application) several multipoint supports are created.
The **multipoint supports** in the electrical problem are **Flux objects** represented by groups of points corresponding to the thermal nodes (coordinates of the thermal nodes, imported into the electromagnetic problem).




To carry out the transfer of data in direction 2 (which means the transfer of temperature from the thermal application into the electromagnetic application) several multipoint supports are created.
The **multipoint supports** in the thermal problem are **Flux objects** represented by groups of points corresponding to the electromagnetic nodes (coordinates of the electromagnetic nodes, imported into the thermal problem).




**Interpolations**

During the transfer of the quantities (power losses, temperature...) there is an **interpolation** of the values on the points, which do not coincide with the nodes of the mesh.

# 3.2. Data exchange: multiphysics spatial quantity

**Exchanged data**

In our example, the exchanged data includes:
- the chart of volume density of power losses (M ⇒ T)
- the chart of temperature (T ⇒ M)

The data is spatial data, which requires the definition of a **storage structure**.

**Multiphysics spatial quantity**

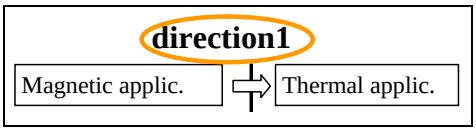
The exchanged data is stored via the Flux entity named **multiphysics spatial quantity**.

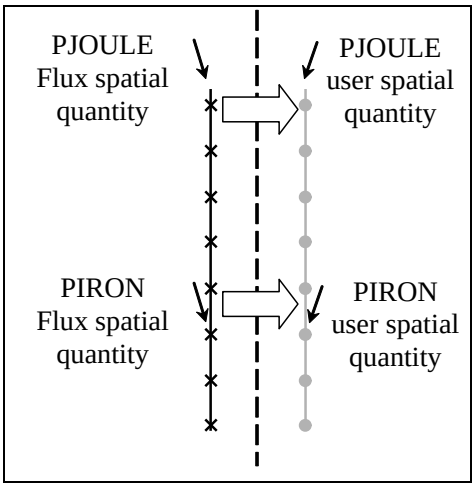
**… in our example**

In our example, we use several multiphysics spatial quantities.

To carry out the transfer of data in direction 1 (which means the transfer of volume density of Joule and iron losses from the electromagnetic application into the thermal application) two multiphysics spatial quantities are created: PJOULE and PIRON.

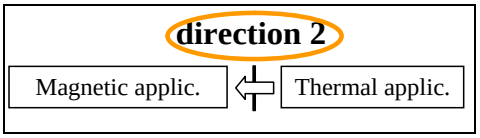
The PJOULE and PIRON multiphysics spatial quantities are storage structures of the power density used in the thermal problem.

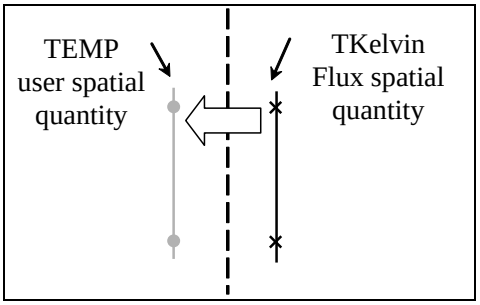



To carry out the transfer of data in direction 2 (which means the transfer of temperature from the thermal application into the electromagnetic application) a multiphysics spatial quantity is created TEMP.

The TEMP multiphysics spatial quantity is a storage structure of the temperature used in the electromagnetic problem.



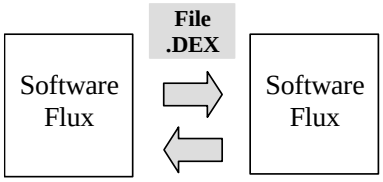



# 3.3. Multiphysics coupling: sequences

**Function**

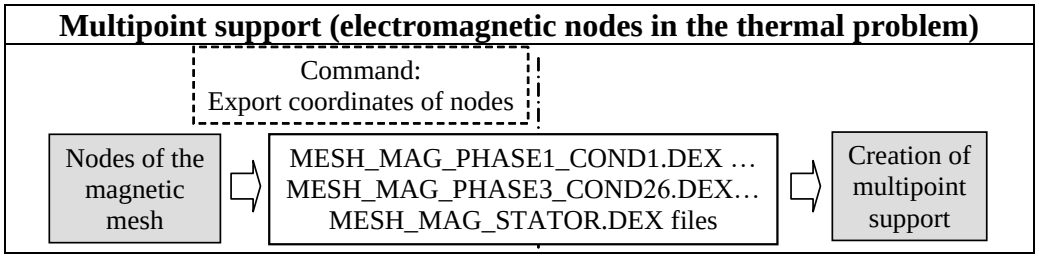
The principal function characteristics are as follows:

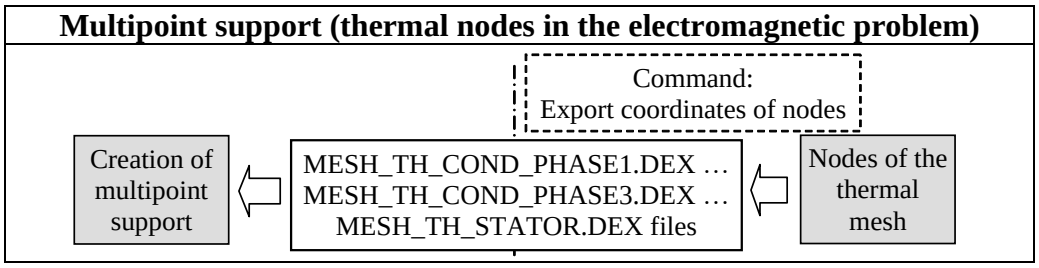
- the Flux application is executed in parallel with the other Flux application
- Data is exchanged via Flux exchange files (extension DEX: **D**ata **EX**change)
- The synchronization is carried out manually or via command files




**Exchange of node coordinates**

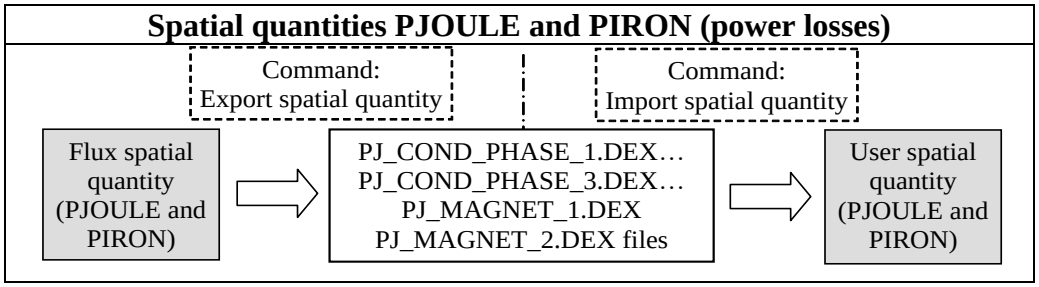
The 1st exchange concerns the transfer of the coordinates of the nodes (stored by means of multipoint support). The exchanges for both applications are presented in the tables below.

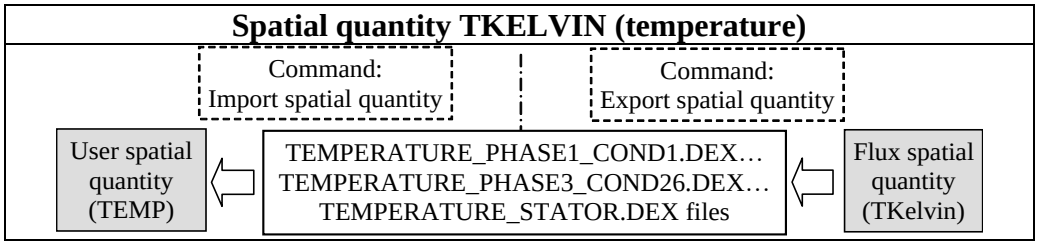



**Exchange of physical quantities**

The 2nd type of exchange concerns the transfer of the physical quantities (stored by means of multiphysics spatial quantity). The exchanges for both applications are represented in the tables below.




*Continued on next page*

**Precision**

In an iterative process with several exchanges between the magnetic and the thermal problems Flux evaluates a specific **precision** of the exchanged values at each iteration (i.e. for each exchange).
The precision calculated in Flux is the difference between the previous and the current value. The precision decreases with the convergence of the result values.

# 3.4. Data exchange: temperature unit and material models

**Problem**

The user wants to carry out a transfer of temperature from external software in Flux and use this temperature to define the physical characteristic of a material.

The user can ask the following questions:

- What unit is used for temperature within the transfer; a degree Celsius or a Kelvin?
- How are material models used?

To answer these questions, this paragraph describes:

- a reminder on **thermal models** provided by Flux for materials and a choice of units in these models
- **two operations** proposed in Flux to transfer the temperature according to the selected models

**Material models: reminder**

There is a certain number of models provided in Flux for physical characteristics of a material depending on temperature: J(E, T), $B_r$(T), etc.

It is possible to make out:

- **predefined models**:
  - a list of models available in the user's guide
- **non-predefined models**:
  - spatial model (characteristic described via spatial formula)
  - user model (characteristic described via user subroutine and user version)

**Predefined models**

With regard to the predefined models …

it is important to remember that in Flux the models are written in **degrees Celsius**, **for any temperature unit of the application**

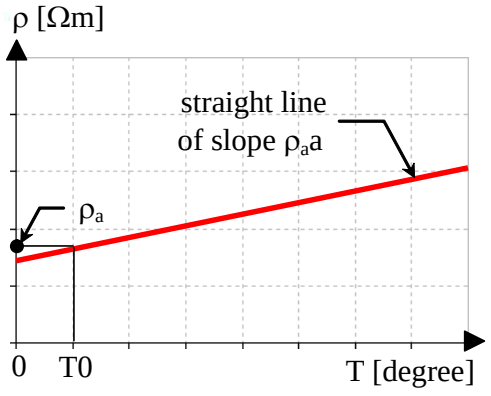
***Example***: isotropic resistivity ρ(T) / linear model

In the relation

$\rho = \rho_a(1 + a(T - T0))$

T is in **degrees Celsius for any temperature unit of the application**

- T0 is the reference temperature [°C]
- $\rho_a$ is the resistivity in **[**Ω.m] of the material at T0
- *a* is a constant [$°C^{-1}$]




Caution: If the unit of the application is Kelvin, it is not necessary to enter a value of $\rho_a$ extrapolated for T = 0 [K].

*Continued on next page*

**Non-predefined models**

In the non-predefined models and mostly in the spatial model (characteristic described via spatial formula), the user can choose the unit of temperature. The user can write the spatial formula in degrees Celsius or Kelvin.

**Temperature transfer and predefined model**

***Operation 1:***
Use of a predefined model for the physical characteristic of the material **and** transfer of the temperature in Kelvin…

| If the user wants ... | then ... |
|---|---|
| to use a material with a B(H), J(E), etc. characteristic defined using a **model dependent on temperature** | The user must apply the **TKelvin** predefined spatial quantity corresponding to the temperature in Kelvin |
| | the data transfer must be carried out in Kelvin |

***Example:***

- In the Flux project:
  - material defined with the J(E, T) model as follows:
    isotropic resistivity ρ(T) / linear model $\rho = \rho_a(1 + a(T - T0))$
  - **TKelvin** predefined spatial quantity
- Data transfer:
  - in the DEX file: temperature in Kelvin

Caution:
Even though in the model the formula is defined in degrees Celsius, it is necessary to use the TKelvin predefined spatial quantity and carry out an importation in Kelvin.

**Temperature transfer and spatial model**

***Operation 2:***
Use of a non-predefined model for the physical characteristic of the material and temperature transfer in degrees Celsius or Kelvin …

| If the user wants ... | then ... |
|---|---|
| to define a material with a B(H), J(E), ... characteristic defined using a **spatial formula** | he must use a spatial quantity defined by himself |
| | the data transfer must be carried out in the appropriate unit |

***Example:***

- In the Flux project:
  - material defined with the J(E, T) model as follows:
    isotropic resistivity ρ(T) / spatial model
    ρ described via spatial formula: $\rho = \rho_0(1 + \mathrm{aTemper} + \mathrm{bTemper}^2)$
  - **Temper** spatial quantity defined by the user
- Data transfer:
  - in the DEX file: temperature in degrees Celsius

# 4. Case 1: Transient / time dependent magnetic analysis

**Introduction**

In this chapter is described the case 1 whose objectives are the following:
- the solving of transient / time dependent magnetic field problem associated to the studied brushless IPM motor for *two different peak currents*,
- the evaluation of magnetic state of the motor and especially of PMs,
- the computation of *Joule losses* in the stator conductors and PMs and *iron losses* in the stator and rotor cores.

The stator and rotor magnetic core regions are magnetic non-linear. The simulation is performed using the transient / time dependent magnetic 2D application for the analysis of the brushless IPM motor.

**Flux projects**

The Flux projects are presented in the table below.

| Application | | Starting Flux project | Working Flux project |
|---|---|---|---|
| 1 | Transient Magnetic | CASE_1_MG_INI.FLU (rotor, stator, airgap and surrounding air geometry, mesh and physical description) | CASE_1_MG1.FLU and CASE_1_MG2.FLU (computation results) |

**Contents**

This chapter contains the following topics:

# 4.1. Case 1: Presentation and analysis

**Case 1**

For the case 1 we will solve the *transient / time dependent magnetic problem* associated to the studied brushless IPM motor and we will evaluate the active power losses in the stator windings (Joule losses), in the PMs (eddy current losses) and in the stator and rotor cores (iron losses) for *two different peak values* ($I_{1max}$ and $I_{2max}$) *of the current*. The computation time will be chosen so as for the motor to reach the steady state from electromagnetic point of view.
The evaluation of the active power losses (Joule losses and iron losses) for thermal computations requires their calculation over a complete electric cycle, after the steady state of the motor is reached.
The motor speed (synchronous speed) taken into account in this study is $n = 1200$ [rpm].
Since the motor has $p = 4$ pole pairs the synchronous speed $n$ should be correlated with the supply frequency $f$ by the relation: $n = 60f/p$.
Thus the supply frequency will be $f = p \cdot n/60 = 80$ [Hz], the corresponding period of the electric cycle will be $T = 1/f = 0.0125$ sec. and the time step will be $\Delta t = T/25 = 0.0005$ sec.
The three-phase currents injected in the stator windings are harmonic with the peak values $I_{1max} = 100$ [A] and $I_{2max} = 80$ [A].

**Contents**

This chapter contains the following topics:

## 4.1.1. Define the physical application

**Goal**

First, the physical application is defined. The required physical application is the **Transient Magnetic 2D** application.

**Data**

The characteristics of the application are presented in the table below.

| Transient Magnetic 2D application | | | | |
|---|---|---|---|---|
| Definition | | | Coils coefficient | Type of initialization |
| 2D Domain type | Length unit | Depth of the domain | | |
| 2D plane | MILLIMETER | 75 | Automatic coefficient | Initialized by static computation |

☞ Application → Define → Magnetic → Transient Magnetic 2D

## 4.1.2. Define physical aspects of the periodicity

**Goal**

The physical aspects of the periodicity created during the geometry description should be defined.

**Data**

The characteristics of the periodicity are presented in the table below.

| Periodicity - Rotation about Z axis with number of repetitions | | | |
|---|---|---|---|
| ***Name (automatic)*** | **Geometrical type** | | **Even or odd periodicity/number of modeled poles** |
| | **Repetition number of the periodicity about Z*** | **Offset angle with respect to the X line (ZOX) plane** | |
| *PeriodicityNumberZaxis* | 8 | 0 | **Odd (anticyclic boundary condition)** |

☞ Physics → Periodicity → New 

## 4.1.3. Create Input/Output parameters

**Goal**

Several I/O parameters are created for an easier description of the problem:
- the motor speed;
- the supply frequency;
- the angular speed;
- the maximum current;
- the shift angle used for defining the supply current.

**Data**

The characteristics of the I/O parameters are presented in the table below.

| I/O parameters | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type of I/O parameter** | **Reference value*** |
| SPEED | Motor speed | Parameter defined by a formula | 1200 rpm |
| FREQUENCY | Supply frequency | Parameter defined by a formula | SPEED/60*POLES/2 |
| OMEGA | Angular speed | Parameter defined by a formula | 2*Pi()*FREQUENCY |
| MAX_CURRENT | Peak value of supply current | Parameter defined by a formula | 100 A for project CASE_1_MG1.FLU |
| | | | 80 A for project CASE_1_MG2.FLU |
| GAMMA | Shift angle used for defining the supply current | Parameter defined by a formula | 45 |

*POLES is a geometrical parameter that represents the number of rotor poles (POLES = 8)

☞ Parameter/Quantity → I/O parameter → New 

## 4.1.4. Create the materials

**Materials**

The following materials are used in this case :

| Material | | | |
|---|---|---|---|
| **Name** | **Comment** | **B(H)** | **J(E)** |
| COPPER | Material used for the regions: PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26, | See following blocks | |
| FLU_M270_35A | Material used for the regions: ROTOR, STATOR | See following blocks | |
| NDFEB | Material used for the regions: MAGNET1_1_POLE_1, MAGNET2_1_POLE_1 | See following blocks | |

**B(H): COPPER**

The model used for the B(H) characteristic is the following:
*Linear isotropic.*
The magnetic property of this model is the relative permeability $\mu_r = 1$.

**J(E): COPPER**

The model used for the J(E) characteristic is the following:
*Isotropic constant resistivity.*
The corresponding constant value is: $\rho = 1.7E\text{-}8$ [Ωm]

**B(H): FLU_M270_35A**

The model used for the B(H) characteristic is the following:
*Isotropic spline saturation.*
The characteristics of the magnetic property are given as a table and included in the Flux Materials Database (imported material).

**B(H): NDFEB**

The model used for the B(H) characteristic is the following:
*Linear magnet described by the Br module.*
The characteristics of the magnetic property are presented in the table below.

| **B(H)** magnetic property**: Linear magnet described by the Br module** | |
|---|---|
| **Remanent flux density [T]** | **Relative permeability $\mu_r$** |
| 1.2 | 1.05 |

**J(E): NDFEB**

The model used for the J(E) characteristic is the following:
*Isotropic resistivity.*
The corresponding constant value is: $\rho$ = 1.4E-6 [Ωm]

☞ Physics → Material → New 

## 4.1.5. Import the electric circuit

**Introduction**

The stator windings and the rotor PMs are modeled by an electric circuit named **MG_CIR.xcir** that should be imported.

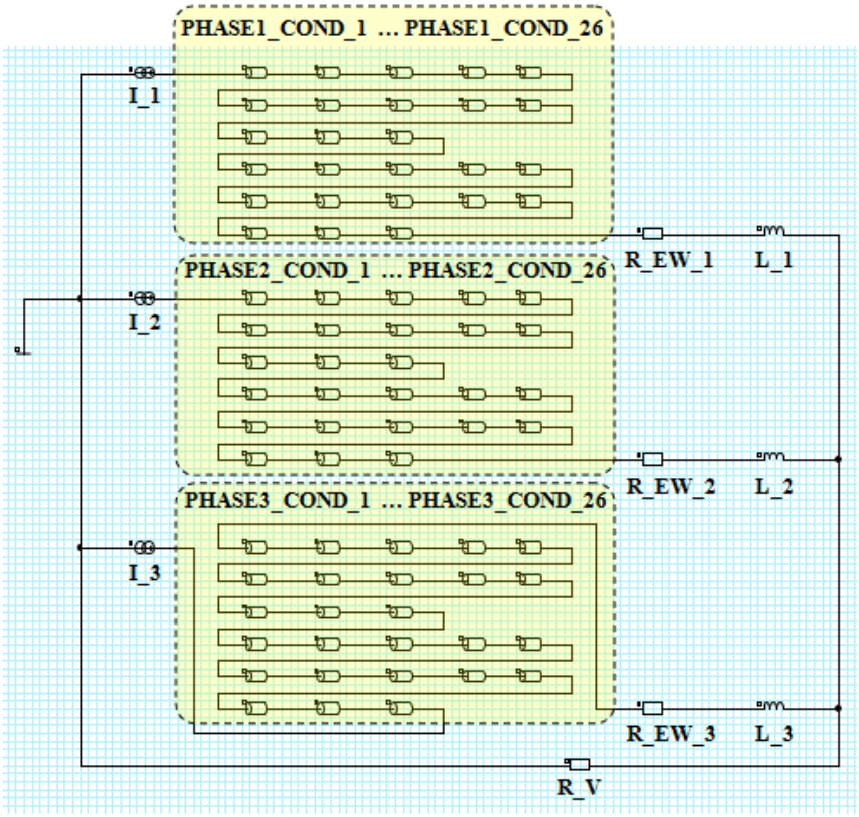
☞ Physics → Circuit → Import circuit from a xcir file 

**Data**

The initial electric circuit in the figure below associated to the electromagnetic field model includes the following components:

- 3 x 26 = 78 solid conductors (26 series connected solid conductors per phase): PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26, PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26, PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26;
- 3 a.c. current sources I_1, I_2, I_3 (one a.c. current source per each phase) with the following expressions:
  I_1 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180);
  I_2 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180-2*Pi()/3);
  I_3 = MAX_CURRENT*Sin(OMEGA*TIME+GAMMA*Pi()/180-4*Pi()/3);
  The constants used in the expressions above are the following:
  MAX_CURRENT = 100 A (or 80 A in case 1);
  OMEGA = 2*Pi()*FREQUENCY;
  FREQUENCY = SPEED/60*POLES/2 = 1200/60*8/2 = 80 Hz,
  GAMMA = 45°;
- 3 inductors of 159 [μH] (one per each phase): L_1, L_2, L_3;
- 3 resistors of 34 [mΩ] (one per each phase): R_EW_1, R_EW_2, R_EW_3;
- 1 resistor of 10 [kΩ] modeling a voltmeter: R_V.

*Continued on next page*




*Electric circuit asociated to the studied brushless IPM motor*

| | |
|---|---|
| ☞ | **Physics → Circuit → Circuit editor context** |
| ☞ | **Physics → Assign coil conductor components to regions → Assign coil conductor components to face regions** |

## 4.1.6. Create and assign face regions

**Goal**

Ninety two surface regions, necessary for the physical description, are created and assigned to surfaces.

**Data**

The characteristics of the surfaces regions are presented in the table below.

| Surface region | | |
|---|---|---|
| **Name** | **Comment** | **Color** |
| MAGNET1_1_POLE_1 | Permanent magnet surface | Red |
| MAGNET2_1_POLE_1 | Permanent magnet surface | Red |
| PHASE1_COND_1 | Copper conductor surface | Red |
| PHASE1_COND_2 | Copper conductor surface | Red |
| … | Copper conductor surfaces | Red |
| PHASE1_COND_26 | Copper conductor surface | Red |
| PHASE2_COND_1 | Copper conductor surface | Yellow |
| PHASE2_COND_2 | Copper conductor surface | Yellow |
| … | Copper conductor surfaces | Yellow |
| PHASE2_COND_26 | Copper conductor surface | Yellow |
| PHASE3_COND_1 | Copper conductor surface | Green |
| PHASE3_COND_2 | Copper conductor surface | Green |
| … | Copper conductor surfaces | Green |
| PHASE3_COND_26 | Copper conductor surface | Green |
| IMPREGNATION | Insulation surface | Turquoise |
| INFINITE | Infinite surface | Turquoise |
| INSULATION | Insulation surface | Black |
| LINER | Insulation surface | Magenta |
| PRESLOT | Insulation surface | Yellow |
| ROTATING_AIRGAP | Rotating airgap surface | Yellow |
| ROTOR_AIR | Air surface | White |
| SHAFT | Air surface | Yellow |
| STATOR_AIR | Air surface | Turquoise |
| WEDGE | Insulation surface | White |
| ROTOR | Magnetic core surface | Blue |
| STATOR | Magnetic core surface | Blue |

| Regions | Type | Type of the conductor | Associated solid conductor | Materials |
|---|---|---|---|---|
| MAGNET1_1_POLE_1 | Solid conductor region | No circuit (open- circuit conductor) | - | NDFEB |
| MAGNET2_1_POLE_1 | Solid conductor region | No circuit (open- circuit conductor) | - | NDFEB |
| PHASE1_COND_1 | Solid conductor region | Circuit | PHASE1_COND_1 | COPPER |
| PHASE1_COND_2 | Solid conductor region | Circuit | PHASE1_COND_2 | COPPER |
| … | Solid conductor regions | Circuit | … | COPPER |

*Continued on next page*

| | | | | |
|---|---|---|---|---|
| PHASE1_COND_26 | Solid conductor region | Circuit | PHASE1_COND_26 | COPPER |
| PHASE2_COND_1 | Solid conductor region | Circuit | PHASE2_COND_1 | COPPER |
| PHASE2_COND_2 | Solid conductor region | Circuit | PHASE2_COND_2 | COPPER |
| … | Solid conductor regions | Circuit | … | COPPER |
| PHASE2_COND_26 | Solid conductor region | Circuit | PHASE2_COND_26 | COPPER |
| PHASE3_COND_1 | Solid conductor region | Circuit | PHASE3_COND_1 | COPPER |
| PHASE3_COND_2 | Solid conductor region | Circuit | PHASE3_COND_2 | COPPER |
| … | Solid conductor regions | Circuit | … | COPPER |
| PHASE3_COND_26 | Solid conductor region | Circuit | PHASE3_COND_26 | COPPER |
| IMPREGNATION, INSULATION, LINER, PRESLOT, WEDGE, SHAFT, ROTOR_AIR, STATOR_AIR, ROTATING_AIRGAP, INFINITE | Air or vacuum region | - | - | - |
| ROTOR | Magnetic non conducting region | - | - | FLU_M270_35A |
| STATOR | Magnetic non conducting region | - | - | FLU_M270_35A |

☞ **Physics → Face region → New** 

☞ **Physics → Face region → Assign regions to faces (completion mode)** 

☞ **Physics → Face region → Force delete** 

## 4.1.7. Create the mechanical sets

**Mechanical sets** The following mechanical sets should be created for this study case :

| Mechanical set | | |
|---|---|---|
| **Name** | **Comment** | **Type** |
| STATOR | Mechanical set for the stator | Fixed |

| Mechanical set | | | | | | |
|---|---|---|---|---|---|---|
| **Name** | **Comment** | **Type** | **Rotation Axis** | **Coordinate system** | **Pivot point** | |
| | | | | | **First coordinate** | **Second coordinate** |
| ROTOR | Mechanical set for the rotor | Rotation around one axis | Rotation around one axis parallel to OZ | XY1 | 0 | 0 |

| **Type of kinematics** | **Velocity (rpm)** | **Position at time t = 0s (deg)** | **Internal characteristics** | | |
|---|---|---|---|---|---|
| | | | **Type of load** | **Moment of inertia (kg.m2)** | **Resistive torque (N.m)** |
| Imposed speed | SPEED | 7.5 | Inertia and resistive load | 0 | 0 |

| External characteristics | | |
|---|---|---|
| **Type of load** | **Moment of inertia (kg.m2)** | **Resistive torque (N.m)** |
| Inertia and resistive load | 0 | 0 |

☞ Physics → Mechanical set → New 

## 4.1.8. Orient the materials for face regions

**Materials**

The magnetization orientation of NDFEB material should be defined for the permanent magnets face regions:

| Orient material for face region | | | | |
|---|---|---|---|---|
| **Material** | **Orient region's materials** | **Oriented type** | **Coordinate system** | **Angle** |
| | UNIDIRECTIONAL MAGNETIZATION | | | |
| NDFEB | MAGNET1_1_POLE_1 | Direction | ROTOR_COORD | 10 |
| | MAGNET2_1_POLE_1 | Direction | ROTOR_COORD | -10 |

☞ Physics → Material → Orient material for face region 

## 4.2. Case 1: Physical description and solving process

**Goal**

Two different electromagnetic analysis projects are defined for two different values of the stator peak current. The first project CASE_1_MG1.FLU corresponds to a peak current value $I_{1max}$ = 100 [A] and the second project CASE_1_MG2.FLU corresponds to a peak current value $I_{2max}$ = 80 [A]. Both projects derive form the initial problem CASE_1_MG_INI.FLU.

**Data**

The values of the I/O parameter MAX_CURRENT and the characteristics of the solving scenario are presented in the tables below.

| I/O parameter | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type of I/O parameter** | **Reference value*** |
| MAX_CURRENT | Peak value of supply current | Parameter defined by a formula | 100 A for project CASE_1_MG1.FLU |
| | | | 80 A for project CASE_1_MG2.FLU |

☞ Parameter/Quantity → I/O Parameter → New 

| Solving scenario | | |
|---|---|---|
| **Name** | **Comment** | **Control of transient state** |
| SCENARIO_MG | Motor speed: 1200 rpm | Control by time |

| Solving scenario | | | | |
|---|---|---|---|---|
| **Parameter control of SCENARIO_MG** | | | | |
| **Controlled parameter** | **Method** | **Lower limit** | **Higher limit** | **Step value** |
| TIME | Step value | 0.0 [sec.] | 0.0135 [sec.] | 5.0E-4 [sec.] |

| Solving options for nonlinear system solvers | | |
|---|---|---|
| **Precision for Newton-Raphson** | **Maximum number of iterations for Newton-Raphson** | **Method for relaxation factor computation** |
| 1.0E-4 | 100 | Fujiwara method (computation) |

**Action**

Starting from the initial problem CASE_1_MG_INI.FLU the results of the solving process will be saved under the names CASE_1_MG1.FLU (for MAX_CURRENT = 100 [A]) and CASE_1_MG2.FLU (for MAX_CURRENT = 80 [A]).

☞ **Solving → Solving scenario → New** 

## 4.3. Case 1: Results post-processing

**Introduction**

This section explains how to analyze the principal results of **case 1** stored in the Flux projects named **CASE_1_MG1.FLU** and **CASE_1_MG2.FLU**.

**Principal results**

The principal results that can be analyzed from the two Flux projects are the following:
Magnetic state of the studied motor,
Current density and Joule losses in the stator windings,
Current density and Joule losses in the PMs regions,
Iron losses in the stator and rotor magnetic cores.

**Contents**

This section contains the following topics:

- Charts of the magnetic flux density and equi-flux lines
- Evaluation of current density and Joule losses
- Evaluation of iron losses

## 4.3.1. Charts of the magnetic flux density and equi-flux lines

**Magnetic flux density and equi-flux lines**

The color-shaded of the magnetic flux density allows us to visualize the magnetic state of the studied brushless IPM motor and to evaluate the saturation level of the machine (e.g. the demagnetization state of PMs).
The equi-flux lines are useful to evaluate the orientation of the magnetic field lines.

**Results**

The charts of the magnetic flux density on the 2D computation domain are presented in the figures below along with a detail on PMs regions for the two values of the stator peak currents. The equi-flux lines are also presented. All these results correspond to the time instant t = 0.001 sec.
The most saturated regions are the rotor magnetic bridges situated towards the motor airgap. In those regions the magnetic flux density values are larger than 2.5 [T]. We can notice that these values are larger higher for the larger stator peak current value.

The minimum values of the magnetic flux density in the PM regions are about 0.4 [T] for the peak current value $I_{1max}$ = 100 [A] and about 0.49 [T] in case of peak current value $I_{2max}$ = 80 [A].

**Results for $I_{1max}$ = 100 [A] (CASE_1_MG1.FLU)**

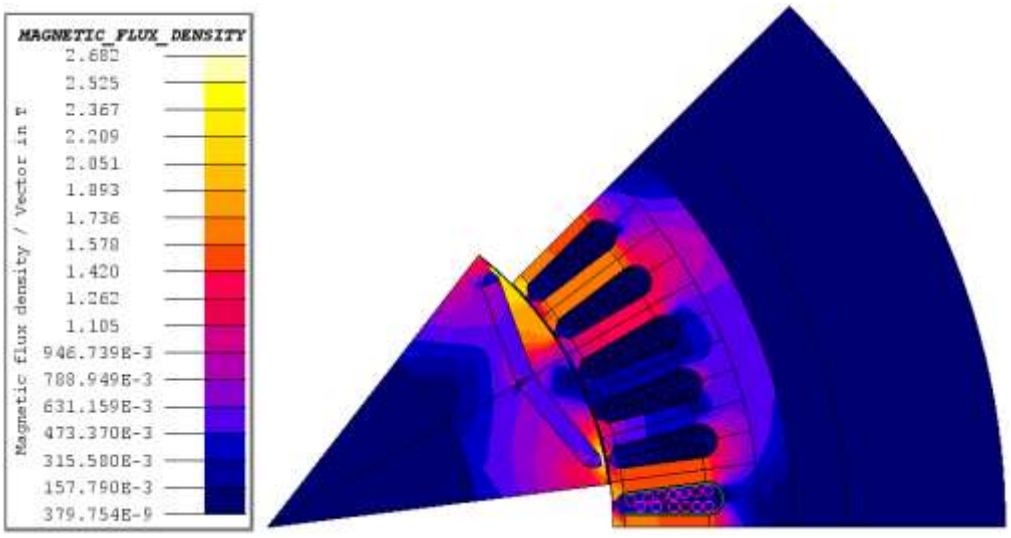



Chart of the magnetic flux density on the 2D computation domain (100 [A])

☞ Graphic → Isovalues → New

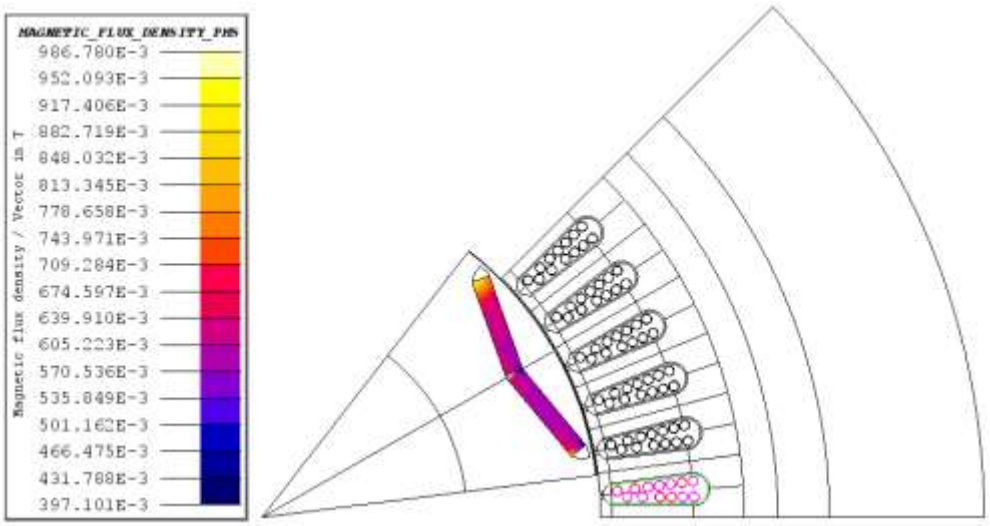
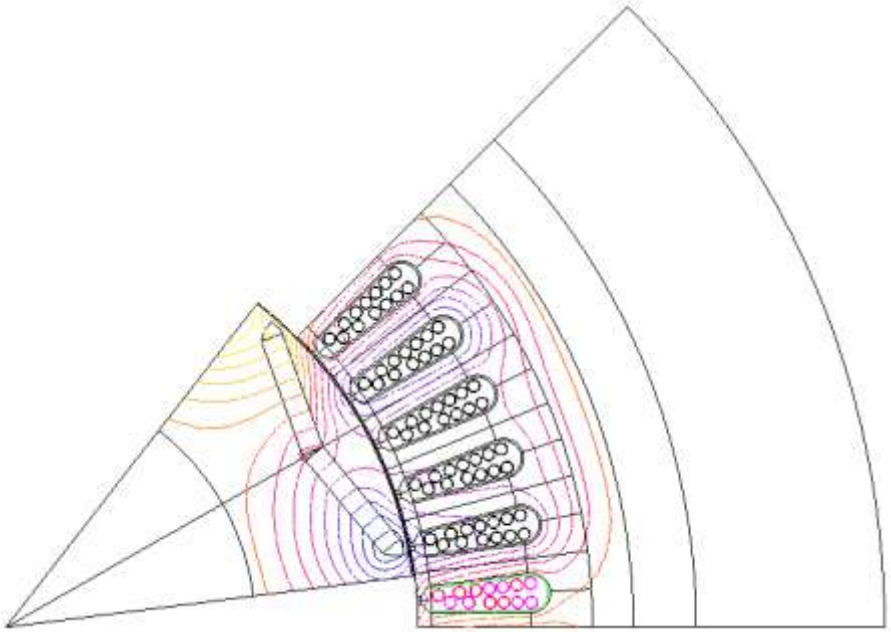



Chart of the magnetic flux density on the PMs regions (100 [A])



Equi-flux lines on the 2D computation domain (100 [A])

☞ Graphic → Isolines → New

**Results for $I_{2max}$ = 80 [A] (CASE_1_MG2.FLU)**

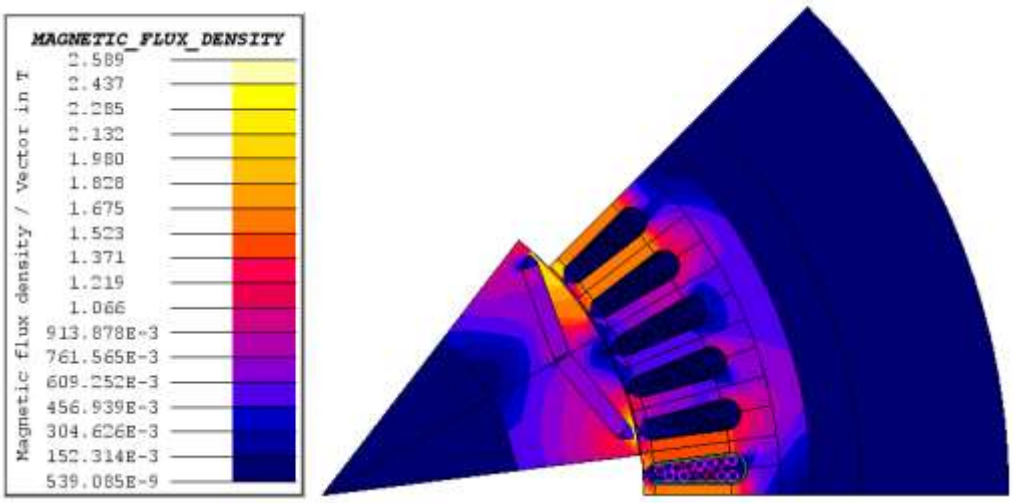



Chart of the magnetic flux density on the 2D computation domain (80 [A])

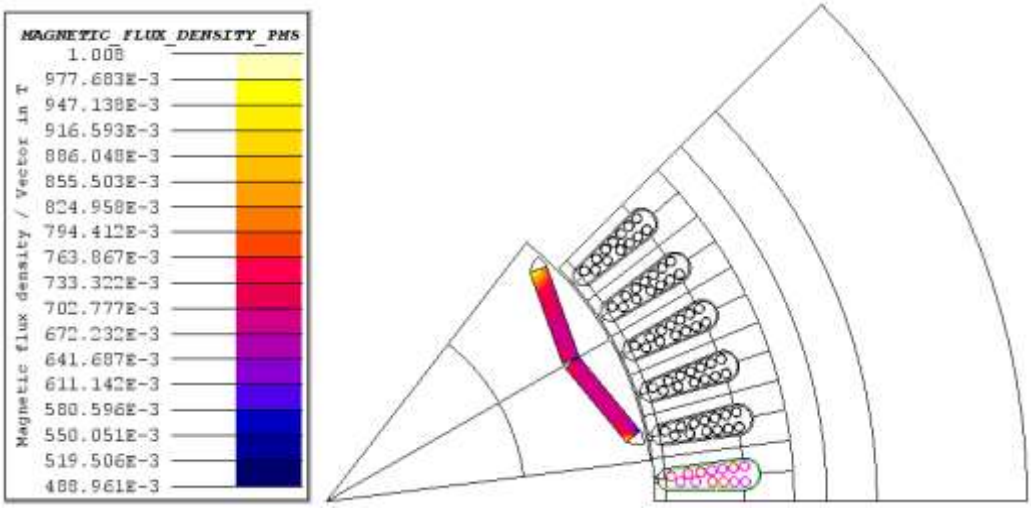
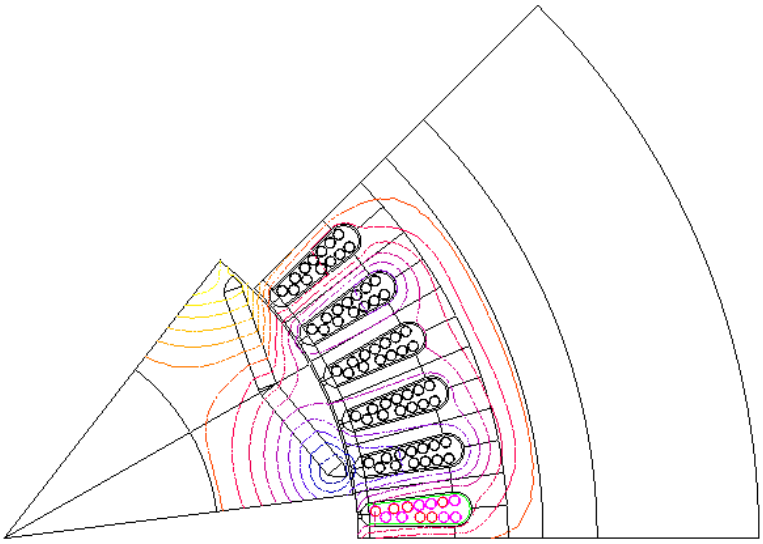



Chart of the magnetic flux density on the PMs regions(80 [A])



Equi-flux lines on the 2D computation domain (80 [A])

## 4.3.2. Evaluation of current density and Joule losses

**Current density and Joule losses**

The current density and the Joule losses are located in the stator windings and in the PMs (eddy current losses) and they contribute to the motor heating.

**Results**

The charts of the current density on the stator conductors and PMs regions are shown in the figures below for the two distinct peak values of the stator currents. All the results correspond to the time instant t = 0.001 sec.
We can notice that the current the current density values are much higher in the stator conductors than in the PMs regions.

**Results for $I_{1max}$ = 100 [A] (CASE_1_MG1.FLU)**

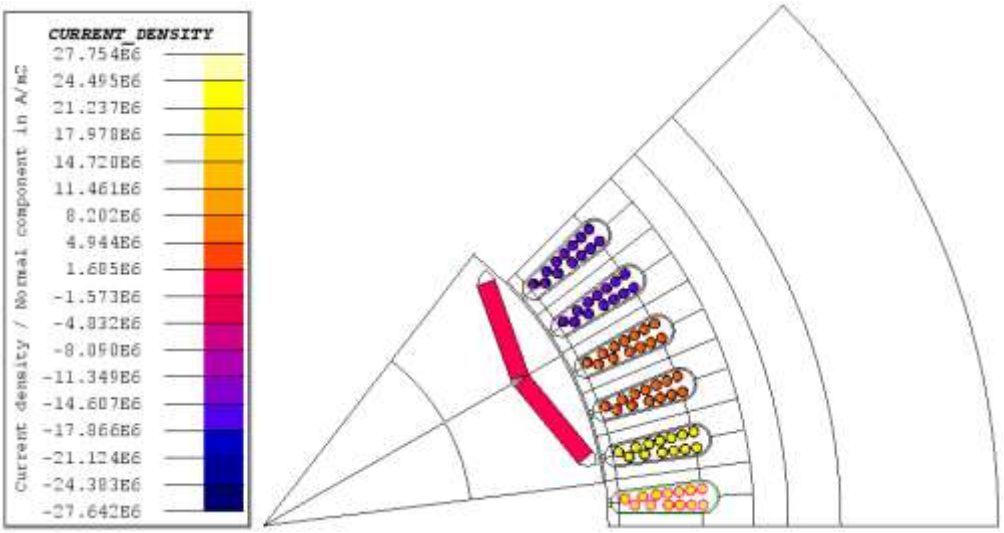



Chart of current density on the 2D computation domain (100 [A])

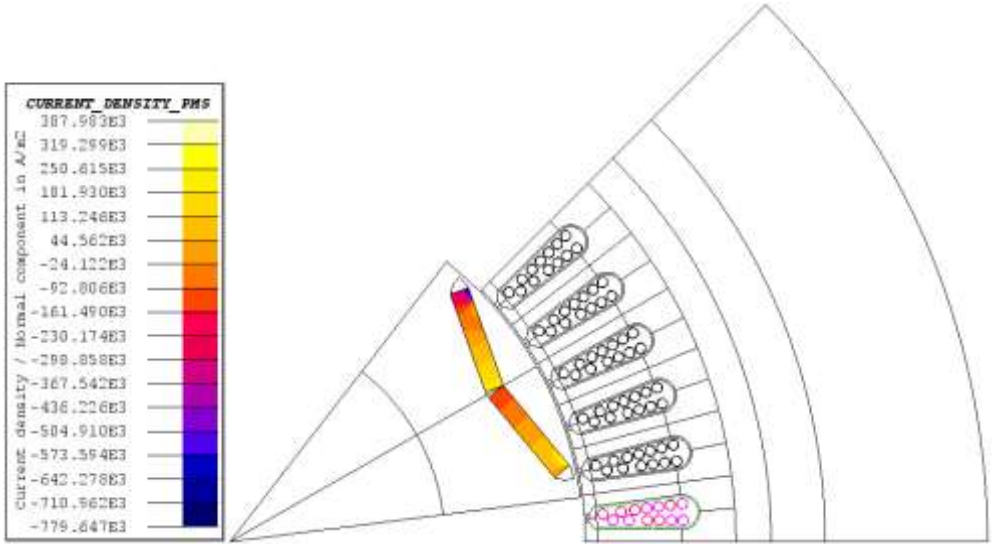



*Chart of current density on the PMs regions(100 [A])*

**Results for $I_{2max}$ = 80 [A] (CASE_1_MG2.FLU)**

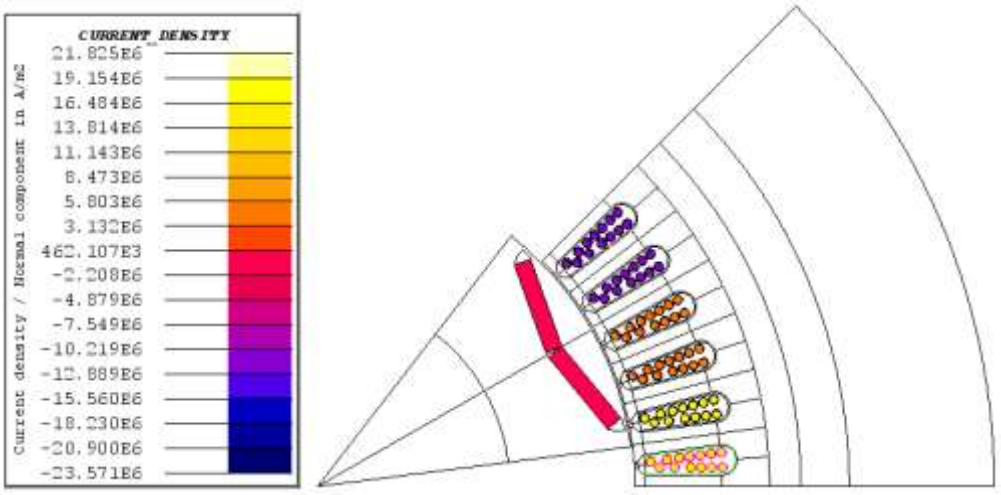



Chart of current density on the 2D computation domain (80 [A])

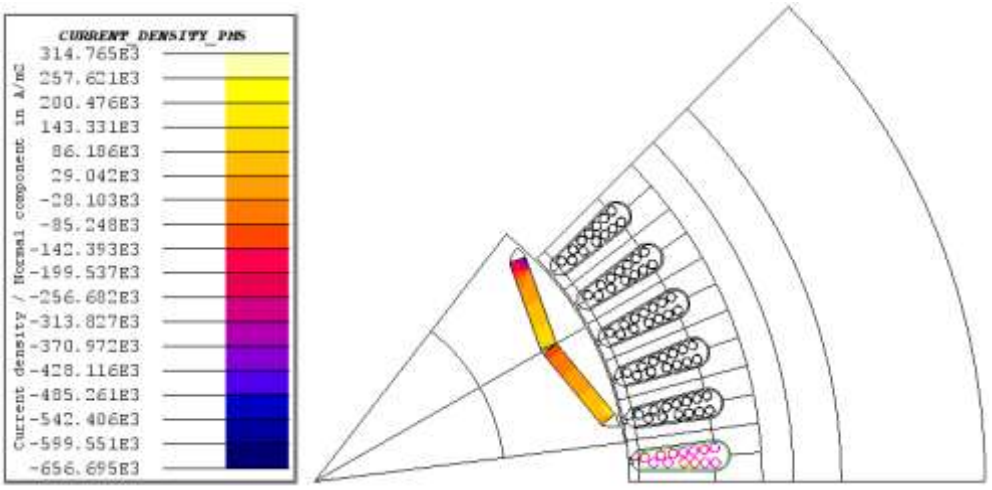



*Chart of current density on the PMs regions (80 [A])*

The numerical results of Joule losses are shown in the table below for the two values of the stator peak current. As expected we can see that the Joule losses increase with the stator peak current value. The Joule losses in the PMs are very small in both cases. They are obtained by computing the average values of the concerned quantities over a full electric cycle. Some of the values will be used for the thermal analysis of the motor (case 2).

| **Results of Joule losses** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **For peak current $I_{1max}$ = 100 [A]** | | | | **For peak current $I_{1max}$ = 80 [A]** | | | |
| **Stator Joule losses [W]** | **PM1 Joule losses [W]** | **PM2 Joule losses [W]** | **Total Joule losses [W]** | **Stator Joule losses [W]** | **PM1 Joule losses [W]** | **PM2 Joule losses [W]** | **Total Joule losses [W]** |
| 103.364 | 0.219 | 0.245 | 103.828 | 66.27 | 0.071 | 0.115 | 66.456 |

*Continued on next page*

☞ **Advanced → Sensor → New** 

☞ **Advanced → Sensor → Display curve** (mean value)

**Note**

The numerical results of the Joule losses shown above correspond to the computation domain which is 1/8 of the whole motor. In order to compute the total Joule losses for the whole motor these results should be multiplied by 8. There are Joule losses also in the end winding resistances (R_EW_1, R_EW_2 and R_EW_3) shown in the circuit associated to the studied motor but their contribution to the 2D thermal analysis of the machine is not taken into account.
To compute the Joule losses in the PMs regions with a better accuracy (e.g. in case of high speed PM motors when these losses are important) a refined mesh in those regions is recomended.

## 4.3.3. Evaluation of iron losses

**Iron losses**

The iron losses are located in the stator and rotor magnetic cores and they are computed by the Bertotti model. The coefficients used for these calculations corresponding to the electric steel laminations made of FLU_M270_35A material are given in the table below.

| Bertotti iron losses | | | | | | | |
|---|---|---|---|---|---|---|---|
| Coefficients | | | | | Regions | Time evolution | |
| Hysteresis loss coeff. | Classical loss coeff. | Loss in excess coeff. | Thickness of lamination | Stacking factor | | Limit min | Limit max |
| 130.346 | 1923077 | 0.357 | 3.5E-4 | 0.97 | STATOR and/or ROTOR | 0.001 | 0.0135 |

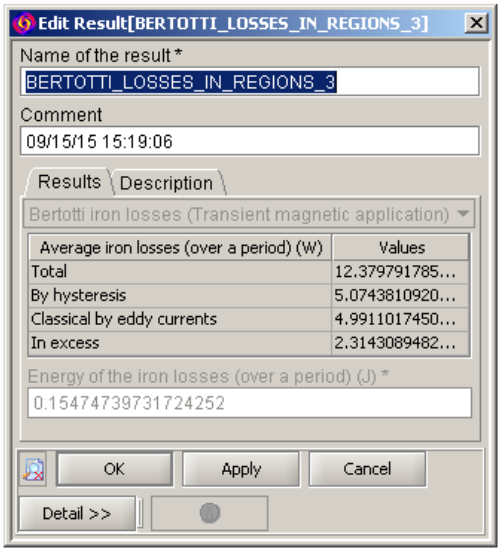
| Exponents | | | |
|---|---|---|---|
| B exponent for hysteresis loss term | f exponent for hysteresis loss term | B exponent for classical loss term | B exponent for loss in excess term |
| 2 | 1 | 2 | 1.5 |

☞ Computation → Iron losses computation → Bertotti iron losses

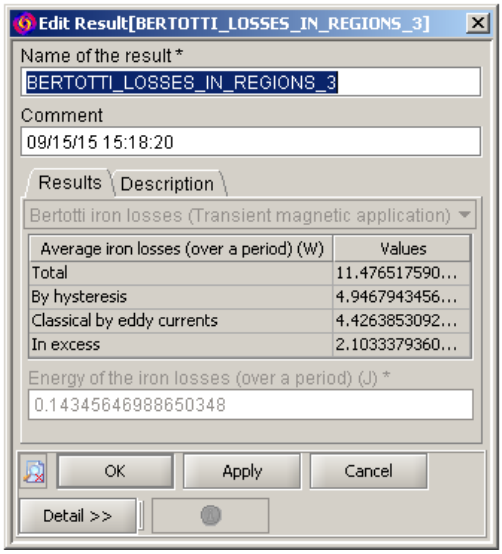
**Result (1)**

The iron losses are computed over a full electric cycle defined between the limits 0.001 sec. and 0.0135 sec. which means a period T = 0.0125 sec. and a frequency of supply currents f = 1/T = 80 [Hz]. The electric cycle is a time period after which the steady state of the electric quantities is reached. The commands used for iron losses computation from Computation menu are: *Iron losses computation → Bertotti iron losses*.

The total iron losses (developed in the rotor and stator magnetic cores) are shown below for the two values of the peak currents. We can notice that the iron losses increase with the peak value of the stator currents ($P_{Fe1} \approx 12.38$ [W] for $I_{1max} = 100$ [A] and $P_{Fe2} \approx 11.48$ [W] for $I_{2max} = 80$ [A]).




$I_{1max}$ = 100 [A]




$I_{2max}$ = 80 [A]

**Result (2)**

The iron losses can be computed separately for the rotor and stator magnetic core and the results are show in the table below. Some of the values will be used for the thermal analysis of the motor (case 2).

| Results of Bertotti iron losses | | | | | |
|---|---|---|---|---|---|
| For peak current $I_{1max}$ = 100 [A] | | | For peak current $I_{1max}$ = 80 [A] | | |
| Rotor core losses [W] | Stator core losses [W] | Total core losses [W] | Rotor core losses [W] | Stator core losses [W] | Total core losses [W] |
| 1.77 | 10.61 | 12.38 | 1.6 | 9.88 | 11.48 |

**Note**

The numerical results of the iron losses presented above correspond to the computation domain which is 1/8 of the whole motor. In order to compute the total iron losses for the whole motor we have to multiply these results with 8.

# 4.4. Case 2: Heat transfer analysis. Parametric study

**Introduction**

The second case deals with the Steady State Thermal analysis of the studied brushless IPM motor.

The **Steady State Thermal** analysis is carried out without multiphysics coupling. The electromagnetic and thermal physical phenomena are considered **independent**.

The thermal sources for the thermal analysis are taken from the **Transient / Time Dependent Magnetic Analysis** of the motor. The purpose of this study is to evaluate the temperature distribution in the motor (after steady state is reached) corresponding to the Joule and iron losses calculated during the transient / time dependent magnetic analysis of the machine.

**A parametric study** will be carried out in order to analyze the influence of the heat transfer properties imposed on the outer boundary of the stator on the temperature distribution in the motor (i.e. different flow rates of the cooling liquid used to cool down the motor).

**Flux projects**

The Flux projects are presented in the table below.

| Application | | Starting Flux project | Working Flux project |
|---|---|---|---|
| 1 | Steady State Thermal | CASE_2_TH_INI.FLU (rotor, stator and airgap geometry, mesh and physical description) | CASE_2_TH.FLU (computation results) |

**Contents**

This chapter contains the following topics:

# 4.5. Case 2: Presentation and analysis

**Case 2**

For the case 2 we will solve the *steady state thermal problem* associated to the studied brushless IPM motor and we will evaluate the temperature distribution in the machine. The sources of thermal field are the Joule losses (in the stator windings and PMs) and iron losses (in the rotor and stator magnetic cores) calculated in the previous case for peak stator current $I_{1max}$ = 100 [A].
*A parametric study* will be carried out for different values of the **convection coefficient *h*** (in the range 100 ÷ 1200 [$Wm^{-2}°C^{-1}$] ) imposed on the outer surface of the stator modeling thus various flow rates of the cooling liquid used to evacuate the heat in excess from the motor. Thirteen computations will be carried out using a constant step of 100 [$Wm^{-2}°C^{-1}$]. The reference value of the convection coefficient is 1000 [$Wm^{-2}°C^{-1}$].

**Contents**

This chapter contains the following topics for thermal analysis (for electromagnetic analysis see §4.1):

## 4.5.1. Define the physical application

**Goal**

First, the physical application is defined. The required physical application is the Transient Thermal 2D application.

**Data**

The characteristics of the application are presented in the table below.

| Steady State Thermal 2D application | | | |
|---|---|---|---|
| Definition | | | Unit |
| 2D domain type | Length unit | Depth of the domain | Temperature unit |
| 2D plane | MILLIMETER | 75 | CELSIUS_DEGREE |

☞ Application → Define → Thermal → Steady State Thermal 2D

## 4.5.2. Define physical aspect of periodicities

**Goal**

The physical aspects of the periodicity created during the geometry description should be defined.

**Data**

The characteristics of the periodicity are presented in the table below. The periodicity type is *Even*, different from case 1 (where *Odd* periodicity was imposed) because in case of thermal problems the temperature values on the linked boundaries are positive.

| **Periodicity - Rotation about Z axis with number of repetitions** | | | |
|---|---|---|---|
| ***Name (automatic)*** | **Geometrical type** | | **Even or odd periodicity/number of modeled poles** |
| | **Repetition number of the periodicity about Z*** | **Offset angle with respect to the X line (ZOX) plane** | |
| *PeriodicityNumberZaxis* | 8 | 0 | **Even (cyclic boundary condition)** |

☞ **Physics → Periodicity → New** 

## 4.5.3. Create Input/Output parameters

**Goal**

In order to carry out a parametric analysis of the solving scenario, we will create one Input/Output parameter. This parameter is named CONV and it represents the convection coefficient used to model different flow rates of the cooling liquid used to evacuate the heat in excess from the studied motor.

**Data**

The characteristics of the I/O parameter CONV are presented in the table below.

| I/O parameters | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type of I/O parameter** | **Reference value** |
| CONV | Convection coefficient | Parameter controlled via a scenario | 1000 |

☞ Parameter/Quantity → I/O parameter → New 

## 4.5.4. Create the materials

**Materials**

The following materials are used in this case :

| Material | | |
|---|---|---|
| **Name** | **Comment** | **k(T)** |
| AIR_AIR_GAP | Airgap region | See following blocks |
| AIR_CLASSICAL | Air regions | See following blocks |
| FLU_COPPER | Stator conductors | See following blocks |
| IMPREGNATION | Stator impregnation material | See following blocks |
| INSULATION | Insulation of stator conductors | See following blocks |
| IRON | Stator and rotor magnetic cores | See following blocks |
| LINER | Slot liner | See following blocks |
| NDFEB | Permanent magnets | See following blocks |

**k(T): AIR_AIR_GAP**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding constant value is: $k = 0.025+3\cdot10^{-3}$ [$Wm^{-1}°C^{-1}$]

**k(T): AIR_CLASSICAL**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding mathematical formula is: $k = 0.025$ [$Wm^{-1}°C^{-1}$]

**k(T): FLU_COPPER**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding mathematical formula is: $k = 394$ [$Wm^{-1}°C^{-1}$]

**k(T): IMPREGNATION**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding mathematical formula is: $k = 0.2$ [$Wm^{-1}°C^{-1}$]

**k(T): INSULATION**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding mathematical formula is: $k = 0.21$ [$Wm^{-1}°C^{-1}$]

**k(T): IRON**

The model used for the thermal conductivity is the following model:
Isotropic constant thermal conductivity.
The corresponding mathematical formula is: $k = 55$ [$Wm^{-1}°C^{-1}$]

| | |
|---|---|
| **k(T): LINER** | The model used for the thermal conductivity is the following model:<br>Isotropic constant thermal conductivity.<br>The corresponding mathematical formula is: $k = 0.21$ [$Wm^{-1}°C^{-1}$] |
| **k(T): NDFEB** | The model used for the thermal conductivity is the following model:<br>Isotropic constant thermal conductivity.<br>The corresponding constant value is: $k = 9$ [$Wm^{-1}°C^{-1}$]<br>☞ **Physics → Material → New**  |

## 4.5.5. Create and assign face regions

**Goal**

Fifteen face regions, necessary for the physical description, are created and assigned to corresponding faces. The face regions are the same as for the electromagnetic analysis (case 1) with some exceptions. First of all the INFINITE and STATOR_AIR face regions are completely removed along with the points and lines that define them. Then STATOR_COND_ PHASE1 is defined so as to regroup the following 26 face regions used in the transient / time dependent magnetic analysis: PHASE1_COND_1, PHASE1_COND_2 ... PHASE1_COND_26.
In a similar way are defined the STATOR_COND_ PHASE2 and STATOR_COND_ PHASE3 face regions so as to regroup PHASE2_COND_1, PHASE2_COND_2 ... PHASE2_COND_26 and PHASE3_COND_1, PHASE3_COND_2 ... PHASE3_COND_26 face regions used in the transient / time dependent magnetic analysis respectively.

**Data**

The characteristics of the face regions are presented in the table below. The last column of the table contains the total heat source in each region expressed in [W], taken from the transient / time dependent magnetic analysis (case 1).

| Surface regions (thermal conducting regions) | | | | |
|---|---|---|---|---|
| **Name** | **Comment** | **Color** | **Material** | **Total heat losses in the region by formula with I/O parameters [W]** |
| IMPREGNATION | Insulation surface | Turquoise | IMPREGNATION | - |
| INSULATION | Insulation surface | Black | INSULATION | - |
| LINER | Insulation surface | Magenta | LINER | - |
| MAGNET1_1_POLE_1 | Permanent magnet surface | Red | NDFEB | 0.219 |
| MAGNET2_1_POLE_1 | Permanent magnet surface | Red | NDFEB | 0.245 |
| PRESLOT | Insulation surface | Yellow | AIR_CLASSICAL | - |
| ROTOR | Magnetic core surface | Blue | IRON | 1.77 |
| ROTOR_AIR | Air surface | White | AIR_CLASSICAL | - |
| SHAFT | Air surface | Yellow | AIR_CLASSICAL | - |

| | | | | |
|---|---|---|---|---|
| STATOR | Magnetic core surface | Blue | IRON | 10.61 |
| ROTATING_AIRGAP | Air surface | Turquoise | AIR_AIR_GAP | - |
| STATOR_ COND _PHASE1 | Copper conductor surfaces | Red | FLU_COPPER | 103.364/3 |
| STATOR_ COND _PHASE2 | Copper conductor surfaces | Red | FLU_COPPER | 103.364/3 |
| STATOR_ COND _PHASE3 | Copper conductor surfaces | Red | FLU_COPPER | 103.364/3 |
| WEDGE | Insulation surface | White | IMPREGNATION | - |

☞ **Physics → Face region → New** 

☞ **Physics → Face region → Assign regions to faces (completion mode)** 

## 4.5.6. Create and assign line regions

**Goal**

One line region, used to model the heat exchange between the outer surface of the stator core and a cooling liquid, should be created and assigned to the corresponding line.

**Data**

The characteristics of the line region are presented in the table below. The features of the line region correspond to a convection type heat exchange between the stator outer surface and a flowing liquid with an average temperature of 70 [°C], used to cool down the motor. The convection coefficient is parameterized, the name of the parameter being CONV (previously defined).

| Line region | | |
|---|---|---|
| **Name** | **Comment** | **Color** |
| EXTERNAL | For heat transfer | Turquoise |

| Thermal definition | | | | | | |
|---|---|---|---|---|---|---|
| **Type** | **Thermal exchange coefficients** | | | **External temperature** | | |
| | **Definition** | **Convection h [$Wm^{-2}$ $°C^{-1}$]** | **Emissivity ε** | **Definition** | **Temp. value** | **Unit** |
| Region with surface of thermal exchange and heat source | Formula with I/O parameters | CONV | 0 | Formula with I/O parameters | 70 | CELSIUS_ DEGREE |

☞ Physics → Line region → New 

☞ Physics → Line region → Assign regions to lines 

# 4.6. Case 2: Physical description and solving process

**Goal**

The *steady state thermal analysis* of the motor is carried out based on the thermal sources (Joule losses and iron losses) computed previously in case 1 for a peak stator current $I_{1max}$ = 100 [A].
A *parametric study* is carried out for different values of the Input/Output parameter CONV (convection coefficient) used to model various heat transfer conditions between the outer surface of the motor and a cooling liquid flowing at different rates.

The thermal Flux project derived from the initial project CASE_2_TH_INI.FLU will be saved under the name CASE_2_TH.FLU.

**Data**

The values of the I/O parameter CONV and the characteristics of the solving scenario SCENARIO_TH are presented in the tables below.

| I/O parameter | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type of I/O parameter** | **Reference value** |
| CONV | Convection coefficient | Parameter controlled via a scenario | 1000 |

| Solving scenario | | | |
|---|---|---|---|
| **Name** | **Comment** | **Controllable parameters** | |
| SCENARIO_TH | Different values of convection coefficient | **Physic** | |
| | | **Parameter name** | **Reference value** |
| | | CONV | 1000 |

| Solving scenario | | | | |
|---|---|---|---|---|
| **Parameter control: CONV (Reference value: 1000)** | | | | |
| **Control type** | **Method** | **Lower limit** | **Higher limit** | **Step value** |
| Multi-values | Step value | 100 | 1200 | 100 |

| | |
|---|---|
| ☞ Parameter/Quantity → I/O Parameter → New |  |
| ☞ Solving → Solving scenario → New |  |

# 4.7. Case 2: Results post-processing

**Goal**

This section explains how to analyze the principal results of **case 2** stored in the Flux projects named **CASE_2_TH.FLU**.

**Principal results**

The principal results that can be analyzed from the Flux project mentioned above are the following:

-Temperature distribution on the study domain for the reference value of the convection coefficient,

-Temperature versus convection coefficient in different points of the study domain.

**Result (1)**

The following chart shows the temperature distribution in the study domain for the reference value of the convection coefficient (CONV = 1000). The temperature of PMs is about 108 °C.
The hottest regions of the studied motor are the stator conductors that reach about 128 °C.

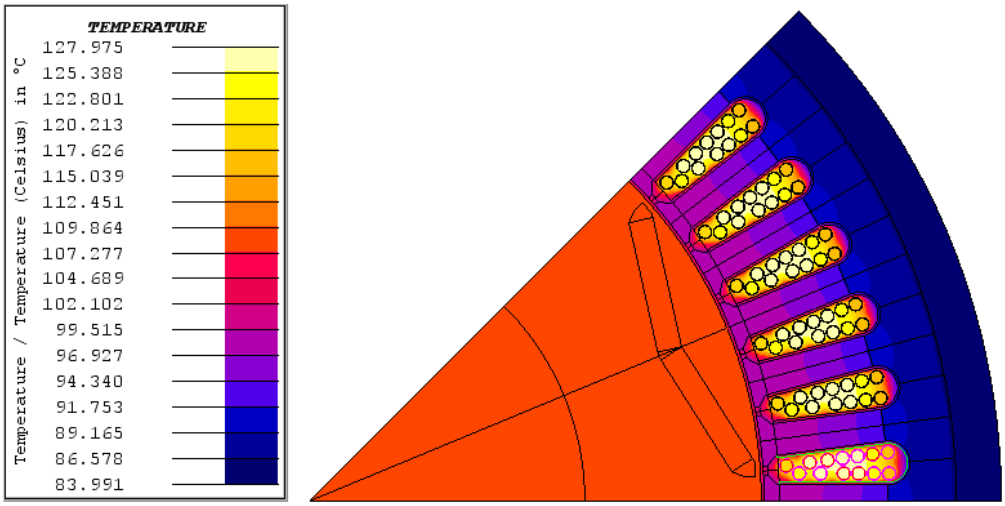



Chart of temperature in [°C] on the 2D computation domain

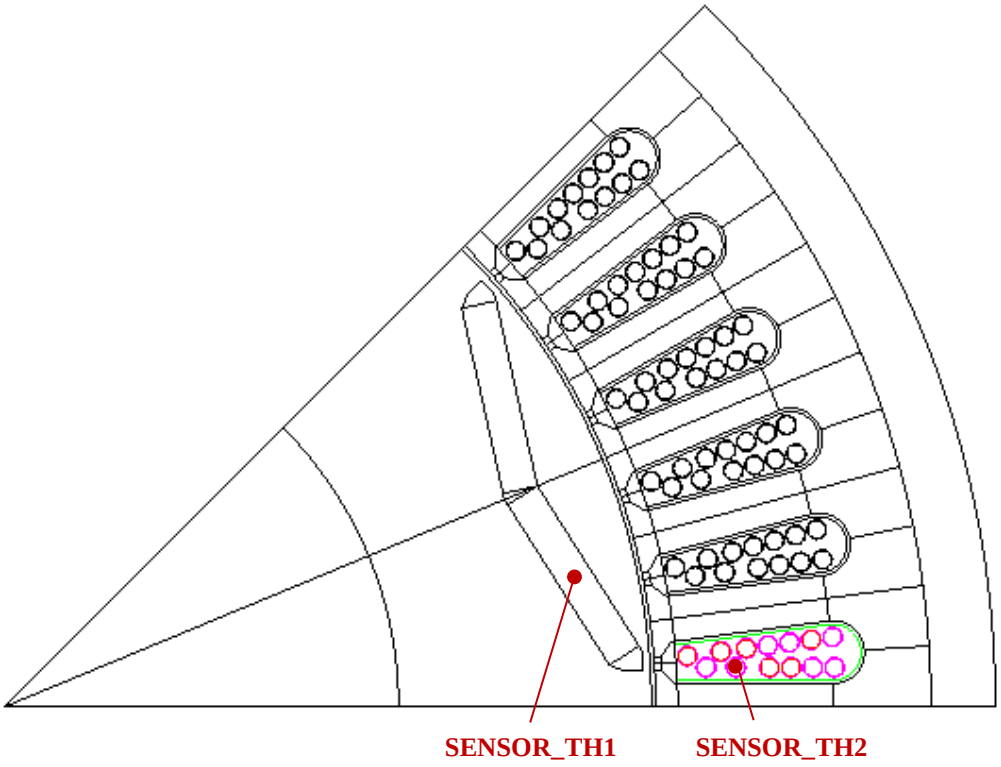
☞ Graphic → Isovalues → New 

**Action**

Two sensors will be created to evaluate the variation of temperature at two distinct points of the computation domain versus the convection coefficient value. The points of interest are situated in the center of a PM and in the center of a stator conductor.

The sensors are composed of:
- a support (a point defined by its coordinates)
- a formula (the temperature)

The points of evaluation are presented in the figure below.




The characteristics of the sensor are presented in the table below.

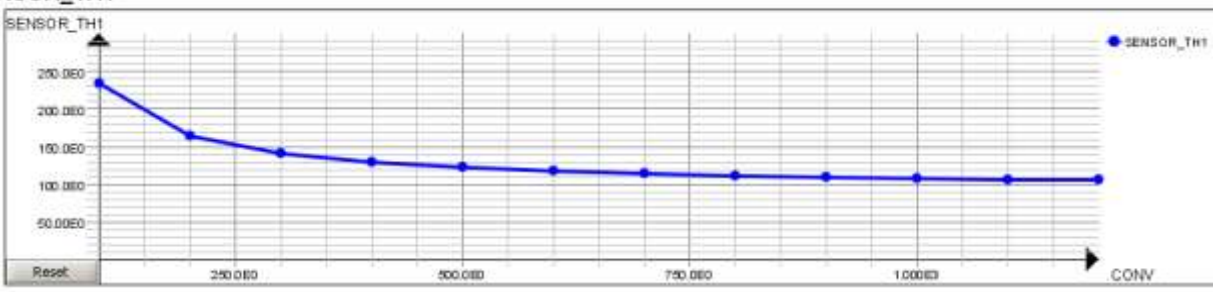
| Sensor: Point (a spatial quantity on a point) | | | | | |
|---|---|---|---|---|---|
| Name | Type | Formula | Support: point | | |
| | | | Coord. system | Coordinates | |
| | | | | X (mm) | Y (mm) |
| SENSOR_TH1 | Point | TKelvin | XY1 | 80.88 | 18.66 |
| SENSOR_TH2 | Point | TKelvin | XY1 | 104.03 | 5.56 |

*Continued on next page*

**Result (2)**

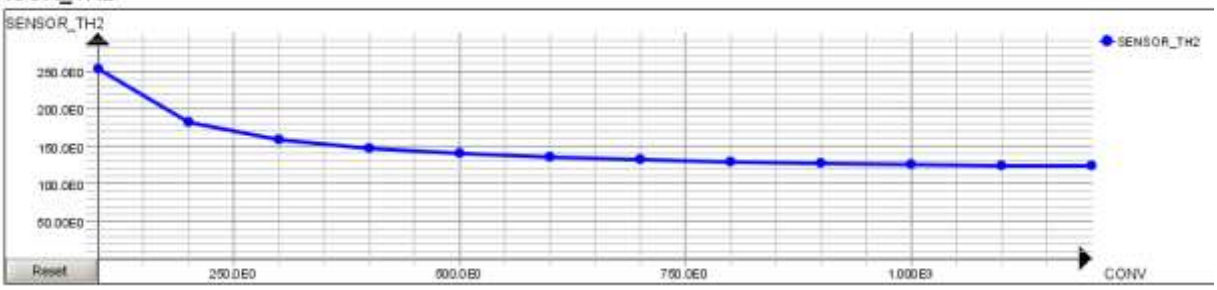
The curves below represent the temperature variation versus convection coefficient obtained by evaluating the two sensors previously created. These results emphasize that the influence of convection coefficient on the temperature in the PMs and in the stator conductors is important until a value of roughly 500 [$Wm^{-2}C^{-1}$]. After this value the temperature decrease is negligible.

SENSOR_TH1




Temperature in [°C] evaluated by SENSOR_TH1 (in a PM region)

SENSOR_TH2




Temperature in [°C] evaluated by SENSOR_TH2 (in a stator conductor region)

☞ **Advanced → Sensor → New**

☞ **Advanced → Sensor → Display curve** (mean value)

# 5. Case 3: Multiphysics coupling

**Case 3**

The third case is an analysis based on a multiphysics coupling between a *Transient / time dependent Magnetic application (after steady state is reached)* and a *Steady State Thermal application* managed by the user or by a python file.

Successive exchanges between the *Transient / time dependent Magnetic application* and the *Steady State Thermal application* are carried out until numerical convergence is reached.

The physical phenomena (electromagnetic and thermal) are **interdependent**.

The purpose of this study is to compute :

- the volume density of Joule losses and iron losses dependent on the temperature distribution in the Transient / time dependent magnetic application and
- the chart of the temperature dependent on the active power losses in the Steady State Thermal application.

**Flux projects**

The Flux projects are presented in the table below.

| Application | | Starting Flux project | Working Flux project |
|---|---|---|---|
| 1 | Transient / time dependent magnetic | CASE_3_MG_INI.FLU (stator, rotor, airgap, surrounding air geometry, mesh and physical description) | CASE_3_MG.FLU (exchange tools, computation results) |
| 2 | Steady State Thermal | CASE_3_TH_INI.FLU (stator, rotor, airgap geometry, mesh and physical description) | CASE_3_TH.FLU (exchange tools, computation results) |

**Contents**

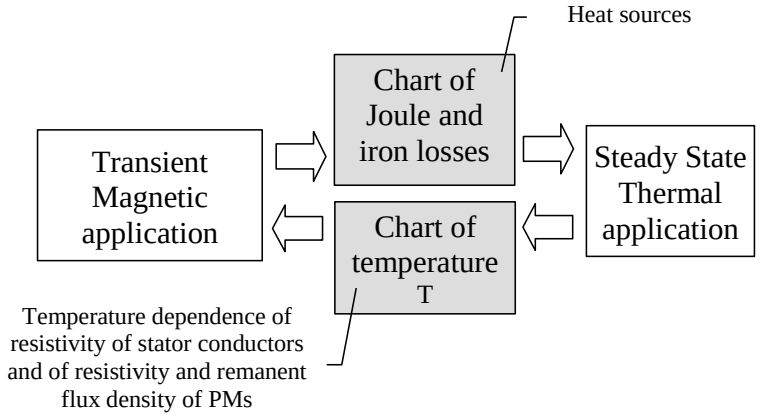
This chapter contains the following topics:

# 5.1. Case 3: General process description

**Case 3**

A multiphysics coupling between a Transient / time dependent Magnetic application (after steady state is reached) and a Steady State Thermal application is carried out. The temperature dependence of resistivity of stator conductors and of resistivity and remanent magnetic flux density of PMs are taken into account.

The operating principle of the multiphysics coupling is recalled in the figure below.




The iterative process of exchanges between the two applications stops when the results are convergent.

**Data transfer and exchange tools**

The simulation involves two data transfers:

- from the Transient / time dependent Magnetic application to the Steady State Thermal application
- from the Steady State Thermal application to the Transient / time dependent magnetic application

To manage two interdependent data transfers, the user needs:

- several multipoint supports
- several multiphysics spatial quantities

**Synchro-nization**

The process of data exchange between the two projects (two interdependent data transfers) must be synchronized

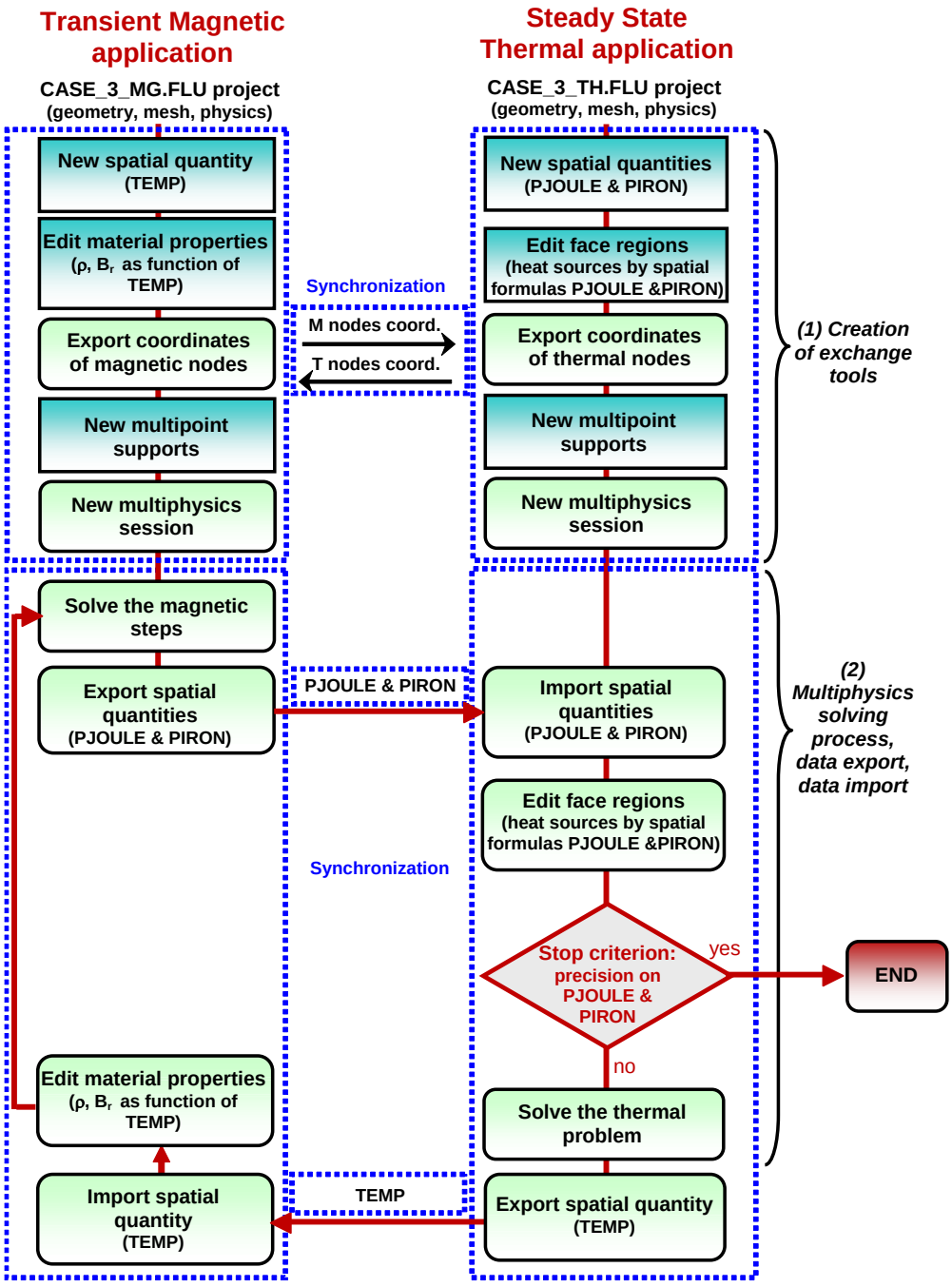
- either by the user
- or via command files

The different steps of two approaches are detailed in next paragraphs.

*Continued on next page*

**Flowchart**

The synopsis of data exchange and synchronization between the two projects is presented in the flowchart below.



## 5.2. Case 3: Process description by the user

**Goal**

A multiphysics coupling between the Transient / time dependent magnetic application and the Steady State Thermal application can be carried out by the user.

The process description is presented in the table below.

| Step | Action (Transient / time dependent magnetic application) | Action (Steady State Thermal application) |
|---|---|---|
| **1** | Verify/modify the Transient / time dependent magnetic application (optional). | |
| 2 | Verify/modify solving options of the magnetic project (optional). | |
| 3 | Create a spatial quantity (TKELVIN) to store the temperature values from the thermal project | |
| 4 | Create a sensor T_INT_PHASE (integral of TKELVIN on the face regions of stator conductors of phase 1) | |
| 5 | Create the following I/O parameters:<br>• S_26COND – area of 26 stator conductors of a phase<br>• RES_EW_COIL_1, RES_EW_COIL_2, RES_EW_COIL_3 – resistance of end winding as function of temperature<br>• ANGPOS_ROTOR_DEG – angular position of the rotor in degree | |
| 6 | Create the following spatial quantities:<br>• BR_NDFEB – remanent flux density of PMs dependent on temperature<br>• BX_1, BY_1 – components along OX and OY of remanent flux density of first PM region dependent on temperature<br>• BX_2, BY_2 – components along OX and OY of remanent flux density of second PM region dependent on temperature<br>• RESISTIVITY_NDFEB – resistivity of PMs dependent on temperature<br>• RESISTIVITY_CU – resistivity of copper for stator conductors dependent on temperature | |
| 7 | Create temperature dependent materials:<br>• COPPER_TEMP – copper with temperature dependent resistivity<br>• NDFEB_1TEMP – NdFeB magnet with temperature dependent resistivity and remanent flux density for first PM region<br>• NDFEB_2TEMP – NdFeB magnet with temperature dependent resistivity and remanent flux density for second PM region | |

| | | |
|---|---|---|
| 8 | Modify the circuit elements (end winding resistors) and the face regions (stator conductors and PMs) with temperature dependent properties:<br>• Resistors: R_EW_1, R_EW_2, R_EW_3<br>• Face regions: PHASE1_COND_1, PHASE1_COND_2 ..., PHASE3_COND_26, MAGNET1_1_POLE_1, MAGNET2_1_POLE_1 | |
| 9 | **Open a multiphysics session** | |
| 10 | Create new multiphysics formulas:<br>• PJ_PREDEFINED and PJOULE – Joule losses<br>• PIRON – iron losses | |
| 11 | Export the magnetic mesh nodes coordinates into data exchange files (MESH_MAG_PHASE1_COND1.DEX, MESH_MAG_PHASE1_COND2.DEX, … MESH_MAG_PHASE3_COND26.DEX, MESH_MAG_MAGNET_1.DEX, MESH_MAG_MAGNET_2.DEX, MESH_MAG_ROTOR.DEX, MESH_MAG_STATOR.DEX). | |
| 12 | Create magnetic iteration synchronization file. Wating for the thermal iteration synchronization file to continue the magnetic computation | |
| 13 | | Verify/modify the Steady State Thermal application (optional) |
| 14 | | Verify/modify solving options of the thermal project (optional). |
| 15 | | Create two spatial quantities (PJOULE and PIRON) to store the density of Joule and iron losses from the electromagnetic project. |
| 16 | | Modify the thermal sources of face regions using the spatial quantities (PJOULE and PIRON): STATOR_COND_PHASE1, STATOR_COND_PHASE1, STATOR_COND_PHASE3, MAGNET1_1_POLE_1, MAGNET2_1_POLE_1, ROTOR , STATOR |
| 17 | | **Open a multiphysics session** |
| 18 | | Create a new multiphysic formula named TEMP |
| 19 | | Export the thermal mesh nodes coordinates into data exchange files (MESH_TH_COND_PHASE1.DEX, MESH_TH_COND_PHASE2.DEX, MESH_TH_COND_PHASE3.DEX, MESH_TH_MAGNET1.DEX, MESH_TH_MAGNET2.DEX, MESH_TH_ROTOR.DEX, MESH_TH_STATOR.DEX). |

| | | |
|---|---|---|
| 20 | | Create thermal iteration synchronization file. Wating for the magnetic iteration synchronization file to continue the thermal computation |
| 21 | | Create multipoint supports (PHASE1_COND1_MAG, PHASE1_COND2_MAG, … PHASE3_COND26_MAG, MAGNET1_MAG, MAGNET2_MAG, ROTOR_MAG, STATOR_MAG) to import the magnetic mesh nodes coordinates from the files (MESH_MAG_PHASE1_COND1.DEX, MESH_MAG_PHASE1_COND2.DEX, … MESH_MAG_PHASE3_COND26.DEX, MESH_MAG_MAGNET_1.DEX, MESH_MAG_MAGNET_2.DEX, MESH_MAG_ROTOR.DEX, MESH_MAG_STATOR.DEX). |
| 22 | | Create new exported data (TEMP) from the multipoint supports (PHASE1_COND1_ MAG, PHASE1_COND2_ MAG, ... PHASE3_COND26_MAG, MAGNET1_MAG, MAGNET2_ MAG, ROTOR_ MAG, STATOR_ MAG) into data exchange files (TEMPERATURE_PHASE1_COND1.DEX, TEMPERATURE_PHASE1_COND2.DEX, … TEMPERATURE_PHASE3_COND26.DEX, TEMPERATURE_MAGNET_1.DEX, TEMPERATURE_MAGNET_2.DEX, TEMPERATURE_ROTOR.DEX, TEMPERATURE_STATOR.DEX). |
| 23 | | Create new imported data (PJOULE and PIRON) from the data exchange files (PJ_COND_PHASE_1.DEX, PJ_COND_PHASE_2.DEX, PJ_COND_PHASE_3.DEX, PJ_MAGNET_1.DEX, PJ_MAGNET_2.DEX, PIRON_ROTOR.DEX, PIRON_STATOR.DEX). |
| 24 | | **Create and run cosimulation**. |
| 25 | Create multipoint supports (COND_PHASE1_TH, COND_PHASE2_TH, COND_PHASE3_TH, MAGNET1_TH, MAGNET2_TH, ROTOR_TH, STATOR_TH) to import the thermal mesh nodes coordinates (MESH_TH_COND_PHASE1.DEX, MESH_TH_COND_PHASE2.DEX, MESH_TH_COND_PHASE3.DEX, MESH_TH_MAGNET1.DEX, MESH_TH_MAGNET2.DEX, MESH_TH_ROTOR.DEX, MESH_TH_STATOR.DEX). | |

| | | |
|---|---|---|
| 26 | Create new exported data (PJOULE and PIRON) from the multipoint supports (COND_PHASE1_TH, COND_PHASE2_TH, COND_PHASE3_TH, MAGNET1_TH, MAGNET2_TH, ROTOR_TH, STATOR_TH) | |
| 27 | Create new imported data (TKELVIN) from the data exchange files (TEMPERATURE_PHASE1_COND1.DEX, TEMPERATURE_PHASE1_COND2.DEX, … TEMPERATURE_PHASE3_COND26.DEX, TEMPERATURE_MAGNET_1.DEX, TEMPERATURE_MAGNET_2.DEX, TEMPERATURE_ROTOR.DEX, TEMPERATURE_STATOR.DEX). | |
| 28 | **Create and run cosimulation.** | |

**Action (1)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

After loading the project CASE_3_MG_INI.FLU and saving it under the name CASE_3_MG.FLU the application can be verified or modifed. The characteristics of the transient / time dependent magnetic application are presented in the table below.

| Transient Magnetic 2D application | | | | |
|---|---|---|---|---|
| **Definition** | | | **Coils coefficient** | **Type of initialization** |
| **2D Domain type** | **Length unit** | **Depth of the domain** | | |
| 2D plane | MILLIMETER | 75 | Automatic coefficient | Initialized by static computation |

☞ Application → Edit current application

**Action (2)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

The characteristics of the solving options are presented in the table below.

| Solving options for nonlinear system solvers | | |
|---|---|---|
| **Precision for Newton-Raphson** | **Maximum number of iterations for Newton-Raphson** | **Method for relaxation factor** |
| 1.0E-4 | 100 | Fujiwara method (computation) |

☞ Solving → Solving process options → Edit 

**Action (3)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***
A New spatial quantity TKELVIN for temperature is created.

The characteristics of this spatial quantity are presented in the table below.

| **Spatial quantity TKELVIN** for **temperature** | |
|---|---|
| **Name (predefined)** | **Default value** |
| TKELVIN | 293.15 |

☞ **Parameter/Quantity → Spatial quantity → New spatial quantity TKELVIN for temperature**

The transfer of the temperature between two applications is only carried out in Kelvin; the conversion of the unit for temperature in predefined material models is managed by Flux. (see §3.4).

**Action (4)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***
Create the sensor T_INT_PHASE to evaluate the integral of TKELVIN on the face regions corresponding to the stator conductors of phase 1. By dividing this result with the total area of the 26 conductors of a phase (this area is stored in the parameter S_26COND created by Action (5)) the average temperature of the conductors of a phase is obtained. The information obtained for a phase will be used for all the three phases of the machine (due to symmetries) to update the values of the end winding resistances with the temperature.

| **Sensor T_INT_PHASE** | | | | | |
|---|---|---|---|---|---|
| **Name** | **Type of sensor** | **Type of support** | **Face regions** | **Spatial formula** | **Type of integration** |
| T_INT_PHASE | Integral on face | Face regions | PHASE1_COND_1<br>PHASE1_COND_2 ...<br>PHASE1_COND_26 | TKELVIN | Integration in the plane |

☞ **Advanced → Sensor → New** 

**Action (5)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***
Create the following I/O parameters:

| I/O Parameters | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type of physical parametera** | **Expression** |
| S_26COND | Area of 26 stator conductors of a phase | Parameter defined by a formula | 0.00012766 |
| RES_EW_COIL_1 | End winding resistance as function of temperature | Parameter defined by a formula | 0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15)) |
| RES_EW_COIL_2 | End winding resistance as function of temperature | Parameter defined by a formula | 0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15)) |
| RES_EW_COIL_3 | End winding resistance as function of temperature | Parameter defined by a formula | 0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15)) |
| ANGPOS_ROTOR_DEG | Rotor angular position in degree | Parameter defined by a formula | AngPos(ROTOR) |

☞ Parameter/Quantity → I/O Parameter → New 

**Action (6)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***
New spatial quantities for multiphysics should be created. They are used to define the temperature dependent materials.
The characteristics of the spatial quantities are presented in the table below.

| **Spatial quantity** for **multiphysics** | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type** | **Value or spatial formula** |
| BR_NDFEB | Remanent flux density of PMs dependent on temperature | Spatial formula | 1.2*(1-0.0013* (TKelvin-293.15)) |
| BX_1 | Component along OX of remanent flux density of first PM region dependent on temperature | Spatial formula | BR_NDFEB*Cosd(22.5+10+IO(ANGPOS_ROTOR_DEG)) |
| BY_1 | Component along OY of remanent flux density of first PM region dependent on temperature | Spatial formula | BR_NDFEB*Sind(22.5+10+IO(ANGPOS_ROTOR_DEG)) |
| BX_2 | Components along OX of remanent flux density of second PM region dependent on temperature | Spatial formula | BR_NDFEB*Cosd(22.5-10+IO(ANGPOS_ROTOR_DEG)) |
| BY_2 | component along OY of remanent flux density of second PM region dependent on temperature | Spatial formula | BR_NDFEB*Sind(22.5-10+IO(ANGPOS_ROTOR_DEG)) |

| | | | |
|---|---|---|---|
| RESISTIVITY_NDFEB | Resistivity of PMs dependent on temperature | Spatial formula | 1.4*(1+0.0014* (TKelvin-293.15))*1E-6 |

| | | | |
|---|---|---|---|
| RESISTIVITY_CU | Resistivity of stator conductors dependent on temperature | Spatial formula | 1.7*(1+0.00427* (TKelvin-293.15))*1E-8 |

☞ Parameter/Quantity → Spatial quantity → New

---

**Action (7)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

Materials with temperature dependent electric and magnetic properties are created as described in the tables below.

| **B(H)** magnetic properties of materials | | |
|---|---|---|
| **Name** | **Comment** | **Magnetic property** |
| | | **Linear isotropic/ relative permeability** |
| COPPER_TEMP | Material for the stator conductors | 1 |

| **Name** | **Comment** | **Spatial linear magnet** | | |
|---|---|---|---|---|
| | | **Spatial remanent flux density [T]** | **Taking the movement of mechanical sets into account** | Relative permeability by spatial formula |
| NDFEB_1TEMP | Material for first PM region | Vec3(BX_1, BY_1, 0) | yes | 1.05 |
| NDFEB_2TEMP | Material for second PM region | Vec3(BX_2, BY_2, 0) | yes | 1.05 |

| **J(E)** electrical property of materials | | |
|---|---|---|
| **Name** | **Comment** | **Spatial isotropic resistivity** |
| | | **Resistivity by spatial formula [Ωm]** |
| COPPER_TEMP | Material for the stator conductors | RESISTIVITY_CU |
| NDFEB_1TEMP | Material for first PM region | RESISTIVITY_NDFEB |
| NDFEB_2TEMP | Material for second PM region | RESISTIVITY_NDFEB |

☞ Physics → Material → New 


---

**Action (8)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

Certain circuit elements (end winding resistors) and face regions (stator conductors and PMs) have temperature dependent properties. Their properties should be changed according to the data in the table below.

| Modification of circuit elements | |
|---|---|
| **Resistor name** | **Model of resistance** |
| R_EW_1 | RES_EW_COIL_1 |
| R_EW_2 | RES_EW_COIL_2 |
| R_EW_3 | RES_EW_COIL_3 |

☞ Physics → Electrical components → Resistor → Edit 

| Modification of face regions | | | | |
|---|---|---|---|---|
| **Face region** | | | **Type of the conductor** | **Associated solid conductor** |
| **Name** | **Type** | **Material of the region** | | |
| PHASE1_COND_1 | Solid conductor region | COPPER_TEMP | Circuit | PHASE1_COND_1/ Positive orientation of the conductor |
| PHASE1_COND_2 | Solid conductor region | COPPER_TEMP | Circuit | PHASE1_COND_2/ Positive orientation of the conductor |
| ... | ... | ... | ... | ... |
| PHASE3_COND_26 | Solid conductor region | COPPER_TEMP | Circuit | PHASE3_COND_26 / Positive orientation of the conductor |
| MAGNET1_1_POLE_1 | Solid conductor region | NDFEB_1TEMP | No circuit/ Open circuit conductor | - |
| MAGNET2_1_POLE_1 | Solid conductor region | NDFEB_2TEMP | No circuit/ Open circuit conductor | - |

☞ Physics → Face region → Edit 

**Action (9)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

A multiphysics solving session is open to carry out the data transfer between the magnetic and thermal applications.

The characteristics of the multiphysics solving session are presented in the table below.

| Multi physic solving session (existing scenario) | |
|---|---|
| **Scenario name** | **Save solved project as** |
| SCENARIO_MG | Current project |

☞ Solving → Multi physic solving session (existing scenario) 

**Action (10)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

Several spatial quantities should be created. They are used to store the density of Joule and iron losses from the magnetic problem.
The characteristics of the spatial quantities are presented in the table below.

| Multiphysics formula | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type** | **Value or spatial formula** |
| PJ_PREDEFINED | Joule losses | Spatial formula | 0 |
| PJOULE | Density of Joule losses | Spatial formula | 0 |
| PIRON | Density of iron losses | Spatial formula | 0 |

☞ Coupling tools → Multiphysics formula → New 

**Action (11)** ***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

The coordinates of mesh nodes of electromagnetic application are exported into data exchange files (MESH_MAG_PHASE1_COND1.DEX, MESH_MAG_PHASE1_COND2.DEX,... MESH_MAG_PHASE3_COND26.DEX, MESH_MAG_MAGNET_1.DEX, MESH_MAG_MAGNET_2.DEX, MESH_MAG_ROTOR.DEX, MESH_MAG_STATOR.DEX).

The characteristics of the mesh nodes exportation are presented in the table below.

| Export coordinates of region nodes | | | | | | |
|---|---|---|---|---|---|---|
| **Region** | | **File** | | **Length unit** | **Coord. system** | **Position of the mechanical set** |
| **Type** | **Name** | **Type** | **Name** | | | |
| Face region | PHASE1_ COND_1 | DEX | MESH_MAG_ PHASE1_COND1 | Millimeter | XY1 | Reference position |
| Face region | PHASE1_ COND_2 | DEX | MESH_MAG_ PHASE1_COND2 | Millimeter | XY1 | Reference position |
| ... | ... | ... | ... | ... | ... | ... |
| Face region | PHASE3_ COND_26 | DEX | MESH_MAG_ PHASE3_COND26 | Millimeter | XY1 | Reference position |
| Face region | MAGNET1_ 1_POLE_1 | DEX | MESH_MAG_ MAGNET_1 | Millimeter | XY1 | Reference position |
| Face region | MAGNET2_ 1_POLE_1 | DEX | MESH_MAG_ MAGNET_2 | Millimeter | XY1 | Reference position |
| Face region | ROTOR | DEX | MESH_MAG_ ROTOR | Millimeter | XY1 | Reference position |
| Face region | STATOR | DEX | MESH_MAG_ STATOR | Millimeter | XY1 | Reference position |

☞ Parameter/Quantity → Export nodes of regions → Export coordinates of region nodes 

**Action (12)**

***Steady State Thermal application – CASE_3_MG.FLU:***
A magnetic iteration synchronization file named *Synchro_M.txt* should be created in the MAGNETIC folder.

**Action (13)**

***Steady State Thermal application – CASE_3_TH.FLU:***
After loading the project CASE_3_TH_INI.FLU and saving it under the name CASE_3_TH.FLU the application can be verified or modifed. The characteristics of the thermal application are presented in the tables below.

| **Steady State Thermal 2D application** | | | |
|---|---|---|---|
| **Definition** | | | **Temperature unit** |
| **2D Domain type** | **Length unit** | **Depth of the domain** | |
| 2D plane | MILLIMETER | 75 | CELSIUS_DEGREE |

**Action (14)**

***Steady State Thermal application – CASE_3_TH.FLU:***
The characteristics of the solving options of the thermal project are presented in the table below.

| **Solving options for nonlinear system solvers** | | |
|---|---|---|
| **Precision for Newton-Raphson** | **Maximum number of iterations for Newton-Raphson** | **Method for computing relaxation factor** |
| 1.0E-4 | 100 | Fujiwara method (computation) |

☞ Solving → Solving process options 

**Action (15)**

***Steady State Thermal application – CASE_3_TH.FLU:***
Two spatial quantities (PJOULE and PIRON) are created to store the density of Joule losses and of iron losses from the electromagnetic project.

The characteristics of the spatial quantities are presented in the table below.

| **Spatial quantity** for **multiphysics** | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type** | **Value or spatial formula** |
| PJOULE | Power density values of Joule losses | Real scalar | 0 |
| PIRON | Power density values of iron losses | Real scalar | 0 |

☞ Parameter/Quantity → Spatial quantity → New 

**Action (16)**

***Steady State Thermal application – CASE_3_TH.FLU:***

The spatial quantities PJOULE and PIRON are used to define the volume density of losses (Joule and iron losses) - heat sources for the thermal computations. The face regions with heat sources are modified as presented in the table below.

| Face region | | | | | |
|---|---|---|---|---|---|
| **Name** | **Comment** | **Type** | **Material** | **Possible thermal source** | |
| | | | | **Type** | **Formula in W/m3** |
| STATOR_COND_PHASE1 | Stator phase1 conductors with Joule losses | Thermal conducting region with heat thermal source | FLU_COPPER | Volume density by formula with spatial quantities | PJOULE |
| STATOR_COND_PHASE2 | Stator phase2 conductors with Joule losses | Thermal conducting region with heat thermal source | FLU_COPPER | Volume density by formula with spatial quantities | PJOULE |
| STATOR_COND_PHASE3 | Stator phase3 conductors with Joule losses | Thermal conducting region with heat thermal source | FLU_COPPER | Volume density by formula with spatial quantities | PJOULE |
| MAGNET1_1_POLE_1 | PM first region with Joule eddy current losses | Thermal conducting region with heat thermal source | NDFEB | Volume density by formula with spatial quantities | PJOULE |
| MAGNET2_1_POLE_1 | PM second region with Joule eddy current losses | Thermal conducting region with heat thermal source | NDFEB | Volume density by formula with spatial quantities | PJOULE |
| STATOR | Stator core with iron losses | Thermal conducting region with heat thermal source | IRON | Volume density by formula with spatial quantities | PIRON |
| ROTOR | Rotor core with iron losses | Thermal conducting region with heat thermal source | IRON | Volume density by formula with spatial quantities | PIRON |

☞ Physics → Face region → Edit 

**Action (17)**

***Steady State Thermal application – CASE_3_TH.FLU:***

A multiphysics solving session is open to carry out the data transfer between the two thermal and magnetic applications.

The characteristics of the multiphysics solving session are presented in the table below.

| Multi physic solving session (new scenario) | |
|---|---|
| **Scenario name** | **Save solved project as** |
| SCENARIO_THMG | Current project |

☞ Solving → Multi physic solving session (new scenario) 

**Action (18)**

***Steady State Thermal application – CASE_3_TH.FLU:***

A multi-physic formula (TEMP) is created to evaluate and export the temperature to the electromagnetic project.

The characteristics of the spatial quantity are presented in the table below.

| Multiphysics formula | | | |
|---|---|---|---|
| **Name** | **Comment** | **Type** | **Value or spatial formula** |
| TEMP | Temperature | Real scalar | 0 |

☞ Parameter/Quantity → Multiphysic formula 

**Action (19)**

***Steady State Thermal application – CASE_3_TH.FLU:***

The coordinates of mesh nodes of thermal application are exported into data exchange files (MESH_TH_COND_PHASE1.DEX, MESH_TH_COND_PHASE2.DEX, MESH_TH_COND_PHASE3.DEX, MESH_TH_MAGNET1.DEX, MESH_TH_MAGNET2.DEX, MESH_TH_ROTOR.DEX, MESH_TH_STATOR.DEX).

The characteristics of the mesh nodes exportation are given in the table below.

| Export coordinates of mesh nodes | | | | | |
|---|---|---|---|---|---|
| **Region** | | **File** | | **Length unit** | **Coord. system** |
| **Type** | **Name** | **Type** | **Name** | | |
| Face region | STATOR_COND_ PHASE_1 | DEX | MESH_TH_COND_ PHASE1 | millimeter | XY1 |
| Face region | STATOR_COND_ PHASE_2 | DEX | MESH_TH_COND_ PHASE2 | millimeter | XY1 |
| Face region | STATOR_COND_ PHASE_3 | DEX | MESH_TH_COND_ PHASE3 | millimeter | XY1 |
| Face region | MAGNET1_1_POLE_1 | DEX | MESH_TH_MAGNET1 | millimeter | XY1 |
| Face region | MAGNET2_1_POLE_1 | DEX | MESH_TH_MAGNET2 | millimeter | XY1 |
| Face region | ROTOR | DEX | MESH_TH_ROTOR | millimeter | XY1 |
| Face region | STATOR | DEX | MESH_TH_STATOR | millimeter | XY1 |

Parameter/Quantity → Export nodes of regions → Export coordinates of region nodes

**Action (20)**

***Steady State Thermal application – CASE_3_TH.FLU:***
A magnetic iteration synchronization file named *Synchro_T.txt* should be created in the THERMAL folder.

**Action (21)**

***Steady State Thermal application – CASE_3_TH.FLU:***

Several multipoint supports are created (PHASE1_COND1_MAG, PHASE1_COND2_MAG, … PHASE3_COND26_MAG, MAGNET1_MAG, MAGNET2_MAG, ROTOR_MAG, STATOR_MAG) to import the coordinates of magnetic mesh nodes from the following DEX files: MESH_MAG_PHASE1_COND1.DEX, MESH_MAG_PHASE1_COND2.DEX, … MESH_MAG_PHASE3_COND26.DEX, MESH_MAG_MAGNET_1.DEX, MESH_MAG_MAGNET_2.DEX, MESH_MAG_ROTOR.DEX, MESH_MAG_STATOR.DEX.

The characteristics of the multipoint supports are presented in the table below.

*Continued on next page*

| Multipoint support | | | | | |
|---|---|---|---|---|---|
| **Name** | **File** | | **Length unit** | **Coord. system** | **Support** |
| | **Type** | **Name** | | | |
| PHASE1_COND1_MAG | DEX | MESH_MAG_PHASE1_COND1 | millimeter | XY1 | Create a fixed support |
| PHASE1_COND2_MAG | DEX | MESH_MAG_PHASE1_COND1 | millimeter | XY1 | Create a fixed support |
| ... | ... | ... | ... | ... | ... |
| PHASE3_COND26_MAG | DEX | MESH_MAG_PHASE1_COND1 | millimeter | XY1 | Create a fixed support |
| MAGNET1_MAG | DEX | MESH_MAG_ MAGNET_1 | millimeter | XY1 | Create a fixed support |
| MAGNET2_MAG | DEX | MESH_MAG_ MAGNET_2 | millimeter | XY1 | Create a fixed support |
| ROTOR_MAG | DEX | MESH_MAG_ ROTOR | millimeter | XY1 | Create a fixed support |
| STATOR_MAG | DEX | MESH_MAG_ STATOR | millimeter | XY1 | Create a fixed support |

☞ Coupling tools → Multipoint support → New multipoint support by importation of a list of points 

**Action (22)**

***Steady State Thermal application – CASE_3_TH.FLU:***

The values of temperature (TEMP) are exported from the multipoint supports containing the magnetic nodes (PHASE1_COND1_MAG, PHASE1_COND2_MAG, … PHASE3_COND26_MAG, MAGNET1_MAG, MAGNET2_MAG, ROTOR_MAG, STATOR_MAG) into data exchange files (TEMPERATURE_PHASE1_COND1.DEX, TEMPERATURE_PHASE1_COND2.DEX, … TEMPERATURE_PHASE3_COND26.DEX, TEMPERATURE_MAGNET_1.DEX,TEMPERATURE_MAGNET_2.DEX, TEMPERATURE_ROTOR.DEX, TEMPERATURE_STATOR.DEX).

The characteristics of the exportation are presented in the table below.

| New exported data | | | | | | | |
|---|---|---|---|---|---|---|---|
| Support | | File | | Length unit | Coord. system | Multiphysics formula | Position |
| Type | Name | Type | Name | | | | |
| Multipoint | PHASE1_ COND1_MAG | DEX | TEMPERATURE_ PHASE1_COND1 | mm | XY1 | TEMP | Reference position |
| Multipoint | PHASE1_ COND2_MAG | DEX | TEMPERATURE_ PHASE1_COND2 | mm | XY1 | TEMP | Reference position |
| ... | ... | ... | ... | ... | ... | ... | ... |
| Multipoint | PHASE3_ COND26_MAG | DEX | TEMPERATURE_ PHASE3_COND26 | mm | XY1 | TEMP | Reference position |
| Multipoint | MAGNET1_ MAG | DEX | TEMPERATURE_ MAGNET_1 | mm | XY1 | TEMP | Reference position |
| Multipoint | MAGNET2_ MAG | DEX | TEMPERATURE_ MAGNET_2 | mm | XY1 | TEMP | Reference position |
| Multipoint | ROTOR_MAG | DEX | TEMPERATURE_ ROTOR | mm | XY1 | TEMP | Reference position |
| Multipoint | STATOR_ MAG | DEX | TEMPERATURE_ STATOR | mm | XY1 | TEMP | Reference position |

☞ Coupling tools → Exported data → New 

**Action (23)**

***Steady State Thermal application – CASE_3_TH.FLU:***

The values of volume density of Joule losses and iron losses (PJOULE and PIRON) are imported from the following data exchange files: PJ_COND_PHASE_1.DEX, PJ_COND_PHASE_2.DEX, PJ_COND_PHASE_3.DEX, PJ_MAGNET_1.DEX, PJ_MAGNET_2.DEX, PIRON_ROTOR.DEX, PIRON_STATOR.DEX.

The characteristics of the data importation are presented in the table below.

| New imported data | | | | | | |
|---|---|---|---|---|---|---|
| Name of the import | Spatial quantity to import | Support of the spatial quantity to import | | Length unit | System of coord. | Import. method |
| | | Type | Surfacic region | | | |
| PJ_COND_ PHASE_1 | PJOULE | Face region | STATOR_COND_ PHASE_1 | mm | XY1 | Node to node |
| PJ_COND_ PHASE_2 | PJOULE | Face region | STATOR_COND_ PHASE_2 | mm | XY1 | Node to node |
| PJ_COND_ PHASE_3 | PJOULE | Face region | STATOR_COND_ PHASE_3 | mm | XY1 | Node to node |
| PJ_MAGNET_1 | PJOULE | Face region | MAGNET1_1_ POLE_1 | mm | XY1 | Node to node |
| PJ_MAGNET_2 | PJOULE | Face region | MAGNET2_1_ POLE_1 | mm | XY1 | Node to node |
| PIRON_ROTOR | PIRON | Face region | ROTOR | mm | XY1 | Node to node |
| PIRON_STATOR | PIRON | Face region | STATOR | mm | XY1 | Node to node |

☞ Coupling tools → Imported data → New 

**Action (24)**

***Steady State Thermal application – CASE_3_TH.FLU:***

A cosimulation enity named COSIMULATION_ THMG should be created and run. The characteristics of this entity are presented in the table below.

| Cosimulation characteristics - thermal application | | | | |
|---|---|---|---|---|
| **Name of the cosimulation** | **Defining a multiphsics coupling** | **Exchange directory** | **Defining the loop-interuption mode** | **Relative accuracy to obtain on imported data (in %)** |
| COSIMULATION_ THMG | FLUX-FLUX cosimulation | .../Echange | Convergence evaluation performed by this project | 1 |
| **Exported data** | | **Imported data** | | |
| TEMPERATURE_PHASE1_COND1, TEMPERATURE_PHASE1_COND2, … TEMPERATURE_PHASE3_COND26, TEMPERATURE_MAGNET_1, TEMPERATURE_MAGNET_2, TEMPERATURE_ROTOR, TEMPERATURE_STATOR | | PJ_COND_ PHASE_1, PJ_COND_ PHASE_2, PJ_COND_ PHASE_2, PJ_MAGNET_1, PJ_MAGNET_2, PIRON_ROTOR, PIRON_STATOR | | |

☞ Coupling tools → Cosimulation → New 

**Action (25)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

Several multipoint supports are created (COND_PHASE1_TH, COND_PHASE2_TH, COND_PHASE3_TH, MAGNET1_TH, MAGNET2_TH, ROTOR_TH, STATOR_TH) to import the coordinates of thermal nodes from the following DEX files:
MESH_TH_COND_PHASE1.DEX, MESH_TH_COND_PHASE2.DEX, MESH_TH_COND_PHASE3.DEX, MESH_TH_MAGNET1.DEX, MESH_TH_MAGNET2.DEX, MESH_TH_ROTOR.DEX, MESH_TH_STATOR.DEX.
The characteristics of the multipoint supports are presented in the table below.

| Multipoint support | | | | |
|---|---|---|---|---|
| Name | File | | Length unit | Coord. system |
| | Type | Name | | |
| COND_PHASE1_TH | DEX | MESH_TH_COND_PHASE1 | millimeter | XY1 |
| COND_PHASE2_TH | DEX | MESH_TH_COND_PHASE2 | millimeter | XY1 |
| COND_PHASE3_TH | DEX | MESH_TH_COND_PHASE3 | millimeter | XY1 |
| MAGNET1_TH | DEX | MESH_TH_MAGNET1.DEX | millimeter | XY1 |
| MAGNET2_TH | DEX | MESH_TH_MAGNET2.DEX | millimeter | XY1 |
| ROTOR_TH | DEX | MESH_TH_ROTOR | millimeter | XY1 |
| STATOR_TH | DEX | MESH_TH_STATOR | millimeter | XY1 |

☞ Coupling tools → Multipoint support → New multipoint support by importation of a list of points 

**Action (26)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

The values of volume density of Joule losses and iron losses (PJOULE and PIRON) are exported from the following multipoint supports: COND_PHASE1_TH, COND_PHASE2_TH, COND_PHASE3_TH, MAGNET1_TH, MAGNET2_TH, ROTOR_TH, STATOR_TH.
The characteristics of the data exportation are presented in the table below

| New exported data | | | | | | |
|---|---|---|---|---|---|---|
| Name of the export | Support of the spatial quantity to export | Multipoint support | Length unit | Coord. system | Position of the mech. set | Multiphys. formula |
| PJ_COND_ PHASE_1 | Exporting on a multipoint support | COND_PHASE1 _TH | mm | XY1 | Reference position | PJOULE |
| PJ_COND_ PHASE_2 | Exporting on a multipoint support | COND_PHASE2 _TH | mm | XY1 | Reference position | PJOULE |
| PJ_COND_ PHASE_3 | Exporting on a multipoint support | COND_PHASE3 _TH | mm | XY1 | Reference position | PJOULE |
| PJ_MAGNET_ 1 | Exporting on a multipoint support | MAGNET1_TH | mm | XY1 | Reference position | PJOULE |
| PJ_MAGNET_ 2 | Exporting on a multipoint support | MAGNET2_TH | mm | XY1 | Reference position | PJOULE |
| PIRON_ ROTOR | Exporting on a multipoint support | ROTOR_TH | mm | XY1 | Reference position | PIRON |
| PIRON_ STATOR | Exporting on a multipoint support | STATOR_TH | mm | XY1 | Reference position | PIRON |

**Action (27)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

The temperature values (TKELVIN) are updated by data importation from the data exchange files (TEMPERATURE_PHASE1_COND1.DEX, TEMPERATURE_PHASE1_COND2.DEX, … TEMPERATURE_PHASE3_COND26.DEX, TEMPERATURE_MAGNET_1.DEX,TEMPERATURE_MAGNET_2.DEX, TEMPERATURE_ROTOR.DEX, TEMPERATURE_STATOR.DEX).

The characteristics of the updates are presented in the table below.

☞ Coupling tools → Exported data → New 

| New imported data | | | | | | |
|---|---|---|---|---|---|---|
| **Name of the import** | **Spatial quantity to import** | **Support of the spatial quantity to import** | | **Length unit** | **System of coord.** | **Import. method** |
| | | **Type** | **Surfacic region** | | | |
| TEMPERATURE_ PHASE1_COND1 | TKELVIN | Face region | PHASE1_COND_1 | mm | XY1 | Node to node |
| TEMPERATURE_ PHASE1_COND2 | TKELVIN | Face region | PHASE1_COND_2 | mm | XY1 | Node to node |
| ... | TKELVIN | Face region | ... | mm | XY1 | Node to node |
| TEMPERATURE_ PHASE3_COND26 | TKELVIN | Face region | PHASE3_COND_26 | mm | XY1 | Node to node |
| TEMPERATURE_ MAGNET_1 | TKELVIN | Face region | MAGNET1_1_ POLE_1 | mm | XY1 | Node to node |
| TEMPERATURE_ MAGNET_2 | TKELVIN | Face region | MAGNET2_1_ POLE_1 | mm | XY1 | Node to node |
| TEMPERATURE_ ROTOR | TKELVIN | Face region | ROTOR | mm | XY1 | Node to node |
| TEMPERATURE_ STATOR | TKELVIN | Face region | STATOR | mm | XY1 | Node to node |

☞ Coupling tools → Imported data → New 

**Action (28)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

A cosimulation entity named COSIMULATION_ MGTH should be created and run. The characteristics of this entity are presented in the table below.

| Cosimulation characteristics - thermal application | | | | |
|---|---|---|---|---|
| **Name of the cosimulation** | **Defining a multiphsics coupling** | **Data exchange strategy** | **Averaging interval (select one electrical period)** | **Exchange directory** |
| COSIMULATION_ MGTH | FLUX-FLUX cosimulation | Steady (exchanging averaged values) | TIME: Limit min: 0.001 Limit max: 0.0135 | .../Echange |

| **Defining the loop-interruption mode** | **Imported data** | **Exported data** |
|---|---|---|
| Convergence evaluation performed by this project | TEMPERATURE_PHASE1_COND1, TEMPERATURE_PHASE1_COND2, … TEMPERATURE_PHASE3_COND26, TEMPERATURE_MAGNET_1, TEMPERATURE_MAGNET_2, TEMPERATURE_ROTOR, TEMPERATURE_STATOR | PJ_COND_ PHASE_1, PJ_COND_ PHASE_2, PJ_COND_ PHASE_2, PJ_MAGNET_1, PJ_MAGNET_2, PIRON_ROTOR, PIRON_STATOR |

☞ Coupling tools → Cosimulation → New 

## 5.3. Case 3: Process description via command files

**Goal**

A multiphysics coupling between a Transient / time dependent Magnetic application and a Steady State Thermal application is carried out via command files.

The process description is presented in the table below.

| Step | Action |
|---|---|
| 1 | Save the sequences of the multiphysics process description carried out by the user in command files |
| 2 | Execute the command files |

**Action (1)**

The sequences of Flux commands during the multiphysics coupling process, carried out by the user (see the previous paragraph), are saved into the following command files: **CASE_3_MG.py** for transient / time dependent magnetic computation and **CASE_3_TH.py** for steady state thermal computation.

The main content of the **CASE_3_MG.py** command file is presented in the table below.

| Command | Function |
|---|---|
| `#! Flux2D 12.0` | indication on the executable program. |
| `loadProject('CASE_3_MG_INI.FLU')` | loading of Flux project. |
| `saveProjectAs('CASE_3_MG.FLU')` | save the Flux project under a different name. |
| `CreateTemperature(defaultValue=293.15)` | creation of spatial quantity TKELVIN. |

| | |
|---|---|
| lastInstance = SensorIntegralFace(name='T_INT_PHASE', spatialFormula='TKELVIN', integrationType=PlaneIntegration(), support=SupportIntegralRegionFace(region=[ RegionFace['PHASE1_COND_1'], RegionFace['PHASE1_COND_10'], RegionFace['PHASE1_COND_11'], RegionFace['PHASE1_COND_12'], RegionFace['PHASE1_COND_13'], RegionFace['PHASE1_COND_14'], RegionFace['PHASE1_COND_15'], RegionFace['PHASE1_COND_16'], RegionFace['PHASE1_COND_17'], RegionFace['PHASE1_COND_18'], RegionFace['PHASE1_COND_19'], RegionFace['PHASE1_COND_2'], RegionFace['PHASE1_COND_20'], RegionFace['PHASE1_COND_21'], RegionFace['PHASE1_COND_22'], RegionFace['PHASE1_COND_23'], RegionFace['PHASE1_COND_24'], RegionFace['PHASE1_COND_25'], RegionFace['PHASE1_COND_26'], RegionFace['PHASE1_COND_3'], RegionFace['PHASE1_COND_4'], RegionFace['PHASE1_COND_5'], RegionFace['PHASE1_COND_6'], RegionFace['PHASE1_COND_7'], RegionFace['PHASE1_COND_8'], RegionFace['PHASE1_COND_9']]), evaluationMode=preStepEvaluation(defaultValue='293.15')) | creation of the I/O parameter named T_INT_PHASE used to evaluate the integral of TKELVIN on the face regions of stator conductors of PHASE1. |
| lastInstance = VariationParameterFormula(name='S_26COND', formula='0.00012766') | creation of I/O parameter to evaluate the area of 26 stator conductors on a phase. |
| lastInstance = VariationParameterFormula(name='RES_EW_COIL_1', formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))') | creation of I/O parameter to define the resistance of end winding per phase 1 as function of temperature. |
| lastInstance = VariationParameterFormula(name='RES_EW_COIL_2', formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))') | creation of I/O parameter to define the resistance of end winding per phase 2 as function of temperature. |
| lastInstance = VariationParameterFormula(name='RES_EW_COIL_3', formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))') | creation of I/O parameter to define the resistance of end winding per phase 3 as function of temperature. |
| VariationParameterFormula(name='ANGPOS_ROTOR_DEG',formula='AngPos(ROTOR)') | creation of I/O parameter to define the rotor angular position in degree |
| lastInstance = SpatialParameterFormula(name='BR_NDFEB', SpatialFormula='1.2*(1-0.0013*(TKelvin-293.15))') | creation of remanent flux density of PMs dependent on temperature. |
| lastInstance = SpatialParameterFormula(name='BX_1', SpatialFormula='BR_NDFEB*Cosd(22.5+10 +IO(ANGPOS_ROTOR_DEG))') | creation of component along OX of remanent flux density dependent on temperature for first PM region. |
| lastInstance = SpatialParameterFormula(name='BY_1', SpatialFormula='BR_NDFEB*Sind(22.5+10+IO(ANGPOS_ROTOR_DEG))') | creation of component along OY of remanent flux density dependent on temperature for first PM region. |

| | |
|---|---|
| `lastInstance = SpatialParameterFormula(name='BX_2', SpatialFormula='BR_NDFEB*Cosd(22.5-10+IO(ANGPOS_ROTOR_DEG))')` | creation of component along OX of remanent flux density dependent on temperature for second PM region. |

| | |
|---|---|
| `lastInstance = SpatialParameterFormula(name='BY_2', SpatialFormula='BR_NDFEB*Sind(22.5-10+IO(ANGPOS_ROTOR_DEG))')` | creation of component along OY of remanent flux density dependent on temperature for second PM region. |
| `lastInstance = SpatialParameterFormula(name='RESISTIVITY_NDFEB', SpatialFormula='1.4*(1+0.0014*(Tkelvin-293.15))*1E-6')` | creation of resistivity of PMs dependent on temperature. |
| `lastInstance = SpatialParameterFormula(name='RESISTIVITY_CU', SpatialFormula='1.7*(1+0.00427*(Tkelvin-293.15))*1E-8')` | creation of temperature dependent resistivity of copper for stator conductors. |
| `startMacroTransaction()` | start the modification of resistances of the end winding resistors. |
| `Resistor['R_EW_1'].resistance='RES_EW_COIL_1'` | assign the temperature dependent resistance RES_EW_COIL_1 to the end winding resistor R_EW_1. |
| `Resistor['R_EW_2'].resistance='RES_EW_COIL_2'` | assign the temperature dependent resistance RES_EW_COIL_2 to the end winding resistor R_EW_2. |
| `Resistor['R_EW_3'].resistance='RES_EW_COIL_3'` | assign the temperature dependent resistance RES_EW_COIL_3 to the end winding resistor R_EW_3. |
| `endMacroTransaction()` | end the modification of resistances of the end winding resistors. |
| `RegionFace['PHASE1_COND_1'].magneticTransient2D=MagneticTransient2DfaceSolidConductor(material=Material['COPPER_TEMP'], circuitType=Circuit(orientation=PositiveOrientation(), component=SolidConductor2Terminals['PHASE1_COND_1']))` | assign to the face region PHASE1_COND1 the material COPPER_TEMP with a temperature dependent resistivity. |
| `RegionFace['PHASE1_COND_2'].magneticTransient2D=MagneticTransient2DfaceSolidConductor(material=Material['COPPER_TEMP'], circuitType=Circuit(orientation=PositiveOrientation(), component=SolidConductor2Terminals['PHASE1_COND_2']))` | assign to the face region PHASE1_COND2 the material COPPER_TEMP with a temperature dependent resistivity. |
| … | … |
| `RegionFace['PHASE3_COND_26'].magneticTransient2D=MagneticTransient2DfaceSolidConductor(material=Material['COPPER_TEMP'], circuitType=Circuit(orientation=PositiveOrientation(), component=SolidConductor2Terminals['PHASE3_COND_26']))` | assign to the face region PHASE3_COND26 the material COPPER_TEMP with temperature dependent resistivity. |
| `RegionFace['MAGNET1_1_POLE_1'].magneticTransient2D=MagneticTransient2DfaceSolidConductor(material=Material['NDFEB_1TEMP'], circuitType=OpenCircuit())` | assign to the face region MAGNET1_1_POLE_1 the material NDFEB_1TEMP with temperature dependent resistivity and remanent flux density. |

| | |
|---|---|
| `RegionFace['MAGNET2_1_POLE_1'].magneticTransient2D=MagneticTransient2DfaceSolidConductor(material=Material['NDFEB_2TEMP'], circuitType=OpenCircuit())` | assign to the face region MAGNET2_1_POLE_1 the material NDFEB_2TEMP with temperature dependent resistivity and remanent flux density. |
| `Scenario['SCENARIO_MG'].openSessionMultiPhysics(projectName='CASE_3_MG.FLU')` | open a multiphysics solving session. |

| | |
|---|---|
| `lastInstance = JouleLossesMPSensor(name='PJ_PREDEFINED')` | create a spatial quantity for the evaluation of Joule losses. |
| `lastInstance = SpatialFormulaMPSensor(name='PJOULE', FormulaMPSensor='PJ_PREDEFINED')` | create a spatial quantity (PJOULE) for the evaluation of Joule losses. |
| `lastInstance = BertottiIronLossesMPSensor(name='PIRON', coefficients=BertottiCoefficients(hysteresisLossCoefficient=130.346, classicalLossCoefficient=1923077.0, excessLossCoefficient=0.357, sheetIronThickness=3.5E-4, stackingFactor=0.97, hysteresisLossTermBExponent=2.0, hysteresisLossTermFrequencyExponent=1.0, classicalLossTermBExponent=2.0, classicalLossTermFrequencyExponent=2.0, excessLossTermBExponent=1.5, excessLossTermFrequencyExponent=1.5), orientation=SheetIronOrientationXY(coordSys=CoordSys['XY1']))` | create a spatial quantity (PIRON) for the evaluation of iron losses. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_PHASE1_COND1'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['PHASE1_COND_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of magnetic mesh nodes of face region PHASE1_COND1 by a data exchange file MESH_MAG_PHASE1_COND1.DEX. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_PHASE1_COND2'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['PHASE1_COND_2'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of magnetic mesh nodes of face region PHASE1_COND2 by a data exchange file MESH_MAG_PHASE1_COND2.DEX. |
| … | … |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_PHASE3_COND26'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['PHASE3_COND_26'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of magnetic mesh nodes of face region PHASE3_COND26 by a data exchange file MESH_MAG_PHASE3_COND26.DEX. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_MAGNET_1.DEX'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['MAGNET1_1_POLE_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of magnetic mesh nodes of face region MAGNET1_1_POLE_1 by a data exchange file MESH_MAG_MAGNET_1.DEX. |

| | |
|---|---|
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_MAGNET_2.DEX'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['MAGNET2_1_POLE_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of magnetic mesh nodes of face region MAGNET2_1_POLE_1 by a data exchange file MESH_MAG_MAGNET_2.DEX. |

| | |
|---|---|
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_ROTOR.DEX'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['ROTOR'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of magnetic mesh nodes of face region ROTOR by a data exchange file MESH_MAG_ROTOR.DEX. |
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_MAG_STATOR.DEX'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['STATOR'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of magnetic mesh nodes of face region STATOR by a data exchange file MESH_MAG_STATOR.DEX. |
| f1=open('Synchro_M.txt','a')<br>f1.close() | create the magnetic iteration synchronization file to continue the thermal computation → the thermal problem can continue the computation. |
| waitForFile(fileName="../THERMAL/Synchro_T.txt", existFile="exist") | waiting for the thermal iteration synchronization file to continue the magnetic computation. |
| MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE1_TH', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE1.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE2_TH', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE2.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |

| | |
|---|---|
| MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE3_TH',<br>file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE3.DEX'),<br>lengthUnit=LengthUnit['METER'],<br>mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),<br>multipointSupportColor=Color['Turquoise'],<br>multipointSupportVisibility=Visibility['VISIBLE'],<br>coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |

| | |
|---|---|
| MultipointSupportImportingCoordinates(multipointSupportName='MAGNET1_TH',<br>file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_MAGNET1.DEX'),<br>lengthUnit=LengthUnit['METER'],<br>mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=ReferencePosition(),<br>mechanicalSet=MechanicalSet['ROTOR']),<br>multipointSupportColor=Color['Turquoise'],<br>multipointSupportVisibility=Visibility['VISIBLE'],<br>coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='MAGNET2_TH',<br>file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_MAGNET2.DEX'),<br>lengthUnit=LengthUnit['METER'],<br>mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=ReferencePosition(),<br>mechanicalSet=MechanicalSet['ROTOR']),<br>multipointSupportColor=Color['Turquoise'],<br>multipointSupportVisibility=Visibility['VISIBLE'],<br>coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='ROTOR_TH',<br>file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_ROTOR.DEX'),<br>lengthUnit=LengthUnit['METER'],<br>mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=ReferencePosition(),<br>mechanicalSet=MechanicalSet['ROTOR']),<br>multipointSupportColor=Color['Turquoise'],<br>multipointSupportVisibility=Visibility['VISIBLE'],<br>coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='STATOR_TH',<br>file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_STATOR.DEX'),<br>lengthUnit=LengthUnit['METER'],<br>mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),<br>multipointSupportColor=Color['Turquoise'],<br>multipointSupportVisibility=Visibility['VISIBLE'],<br>coordSys=CoordSys['XY1']) | create a multipoint support to import the thermal mesh nodes coordinates by a data exchange file. |

| | |
|---|---|
| lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_1', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PJOULE'], multipointSupport=MultipointSupport['COND_PHASE1_TH']) | export the volume density of Joule losses of the stator conductors on phase 1 by a multipoint support. |

| | |
|---|---|
| lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_2', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PJOULE'], multipointSupport=MultipointSupport['COND_PHASE2_TH']) | export the volume density of Joule losses of the stator conductors on phase 2 by a multipoint support. |
| lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_3', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PJOULE'], multipointSupport=MultipointSupport['COND_PHASE3_TH']) | export the volume density of Joule losses of the stator conductors on phase 3 by a multipoint support. |
| lastInstance = CosimExportMultipointSupport(name='PJ_MAGNET_1', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PJOULE'], multipointSupport=MultipointSupport['MAGNET1_TH']) | export the volume density of Joule losses of PM region 1 by a multipoint support. |
| lastInstance = CosimExportMultipointSupport(name='PJ_MAGNET_2', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PJOULE'], multipointSupport=MultipointSupport['MAGNET2_TH']) | export the volume density of Joule losses of PM region 2 by a multipoint support. |
| lastInstance = CosimExportMultipointSupport(name='PIRON_ROTOR', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PIRON'], multipointSupport=MultipointSupport['ROTOR_TH']) | export the volume density of iron losses of rotor magnetic core by a multipoint support. |
| lastInstance = CosimExportMultipointSupport(name='PIRON_STATOR', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['PIRON'], multipointSupport=MultipointSupport['STATOR_TH']) | export the volume density of iron losses of rotor magnetic core by a multipoint support. |

| | |
|---|---|
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND1',spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['PHASE1_COND_1']) | import the temperature of a stator conductor by a multipoint support. |

| | |
|---|---|
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND2',spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['PHASE1_COND_2']) | import the temperature of a stator conductor by a multipoint support. |
| ... | ... |
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND26',spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['PHASE3_COND_26']) | import the temperature of a stator conductor by a multipoint support. |
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_MAGNET_1', spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['MAGNET1_1_POLE_1']) | import the temperature of PM region 1 by a multipoint support. |
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_MAGNET_2', spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['MAGNET2_1_POLE_1']) | import the temperature of PM region 2 by a multipoint support. |
| lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_ROTOR', spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['ROTOR']) | import the temperature of rotor magnetic core by a multipoint support. |

| | |
|---|---|
| `lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_STATOR', spatialParameter=SpatialParameter['TKELVIN'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['STATOR'])` | import the temperature of stator magnetic core by a multipoint support. |

| | |
|---|---|
| `lastInstance = FluxCosimulation(name='Cosimulation_MGTH', transientComputationType=EstablishedTransientType(parameterSet=[SetParameterXVariable(paramEvol=VariationParameter['TIME'], limitMin=0.001, limitMax=0.0135)]), exchangeDirectory='../Exchange', outData=CosimExportedData[ALL], inData=CosimImportedData[ALL], convergenceCriterion=ThirdPartEvalConvergence())` | create a cosimulation entity. |
| `solveCosimulation()` | run the cosimulation entity to solve the multiphysics problem. |
| `Scenario['SCENARIO_MG'].closeSessionMultiPhysics()` | close the multiphysics solving session. |
| `saveProjectAs('CASE_3_MG.FLU')` | save the magnetic project under the name CASE_3_MG.FLU. |

**Action (2)** The main content of the thermal command file (**CASE_3_TH.py**) is presented in the table below.

| Command | Function |
|---|---|
| `#! Flux2D 11.2` | indication on the executable program. |
| `loadProject('CASE_3_TH_INI.FLU')` | loading of Flux project. |
| `saveProjectAs('CASE_3_TH.FLU')` | save the Flux project under a different name. |
| `lastInstance = SpatialParameterTabulated(name='PJOULE', parameterType=ScalarReal(), defaultValue='0')` | create the spatial quantity PJOULE. |
| `lastInstance = SpatialParameterTabulated(name='PIRON', parameterType=ScalarReal(), defaultValue='0')` | create the spatial quantity PIRON. |
| `startMacroTransaction()` | start the odification of region faces. |
| `RegionFace['STATOR_COND_PHASE_1'].thermalSteady2D=Thermal2DfaceThermalConductor(material=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))` | modify the heat source of a face region corresponding to a stator conductor. |
| `RegionFace['STATOR_COND_PHASE_2'].thermalSteady2D=Thermal2DfaceThermalConductor(material=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))` | modify the heat source of a face region corresponding to a stator conductor. |

| | |
|---|---|
| `RegionFace['STATOR_COND_PHASE_3'].thermalSteady2D=Thermal2DfaceThermalConductor(material=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))` | modify the heat source of a face region corresponding to a stator conductor. |

| | |
|---|---|
| `RegionFace['MAGNET1_1_POLE_1'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['NDFEB'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))` | modify the heat source of a face region corresponding to a PM region. |
| `RegionFace['MAGNET2_1_POLE_1'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['NDFEB'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))` | modify the heat source of a face region corresponding to a PM region. |
| `RegionFace['ROTOR'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['IRON'], heat=VolumeHeatSourceSpatialFormula(value='PIRON'))` | modify the heat source of a face region corresponding to the rotor core region. |
| `RegionFace['STATOR'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['IRON'], heat=VolumeHeatSourceSpatialFormula(value='PIRON'))` | modify the heat source of a face region corresponding to the stator core region. |
| `endMacroTransaction()` | end the modification of face regions. |
| `Scenario(name='Scenario_THMG')` | create a new solving scenario. |
| `Scenario['SCENARIO_THMG'].openSessionMultiPhysics(projectName='CASE_3_TH.FLU')` | open a multiphysics solving session. |
| `lastInstance = TemperatureMPSensor(name='TEMP')` | create a spatial quantity (TEMP) for the evaluation of temperature. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_COND_PHASE1'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['STATOR_COND_PHASE_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of mesh nodes in a data exchange file. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_COND_PHASE2'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['STATOR_COND_PHASE_2'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of mesh nodes in a data exchange file. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_COND_PHASE3'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['STATOR_COND_PHASE_3'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of mesh nodes in a data exchange file. |
| `result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_MAGNET1'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['MAGNET1_1_POLE_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'])` | export the coordinates of mesh nodes in a data exchange file. |

| | |
|---|---|
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_MAGNET2'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['MAGNET2_1_POLE_1'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of mesh nodes in a data exchange file. |

| | |
|---|---|
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_ROTOR'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['ROTOR'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of mesh nodes in a data exchange file. |
| result = ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_STATOR'), mechanicalSetPosition=ReferencePosition(), regionFace=RegionFace['STATOR'], lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1']) | export the coordinates of mesh nodes in a data exchange file. |
| f=open('Synchro_T.txt','a')<br>f.close() | creation of a thermal iteration synchronization file.<br>→ the magnetic problem can continue the computation. |
| waitForFile(fileName="../MAGNETIC/Synchro_M.txt",existFile="exist") | waiting for the magnetic iteration synchronization file to continue the magnetic computation. |
| MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND1_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND1.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND2_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND2.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |
| ... | ... |

| | |
|---|---|
| MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND26_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND26.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |

| | |
|---|---|
| MultipointSupportImportingCoordinates(multipointSupportName='MAGNET1_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_MAGNET_1.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='MAGNET2_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_MAGNET_2.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='ROTOR_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_ROTOR.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |
| MultipointSupportImportingCoordinates(multipointSupportName='STATOR_MAG', file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_STATOR.DEX'), lengthUnit=LengthUnit['METER'], mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(), multipointSupportColor=Color['Turquoise'], multipointSupportVisibility=Visibility['VISIBLE'], coordSys=CoordSys['XY1']) | create the multipoint support to import the magnetic nodes coordinates by data exchange file. |

| | |
|---|---|
| lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND1', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['TEMP'], multipointSupport=MultipointSupport['PHASE1_COND1_MAG']) | create the multipoint support to export the temperature by data exchange file. |
| lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND2', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['TEMP'], multipointSupport=MultipointSupport['PHASE1_COND2_MAG']) | create the multipoint support to export the temperature by data exchange file. |

| | |
|---|---|
| ... | ... |
| lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND26', lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], position=ReferencePosition(), MPSensor=multiphysicsSensor['TEMP'], multipointSupport=MultipointSupport['PHASE3_COND26_MAG']) | create the multipoint support to export the temperature by data exchange file. |
| lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_1', spatialParameter=SpatialParameter['PJOULE'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['STATOR_COND_PHASE_1']) | import (update) the values of volume density of Joule losses. |
| lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_2', spatialParameter=SpatialParameter['PJOULE'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['STATOR_COND_PHASE_2']) | import (update) the values of volume density of Joule losses. |
| lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_3', spatialParameter=SpatialParameter['PJOULE'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['STATOR_COND_PHASE_3']) | import (update) of the values of volume density of Joule losses. |
| lastInstance = CosimImportDataRegSurf(name='PJ_MAGNET_1', spatialParameter=SpatialParameter['PJOULE'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['MAGNET1_1_POLE_1']) | import (update) the values of volume density of Joule losses. |

| | |
|---|---|
| `lastInstance = CosimImportDataRegSurf(name='PJ_MAGNET_2', spatialParameter=SpatialParameter['PJOULE'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['MAGNET2_1_POLE_1'])` | import (update) the values of volume density of Joule losses. |
| `lastInstance = CosimImportDataRegSurf(name='PIRON_ROTOR', spatialParameter=SpatialParameter['PIRON'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['ROTOR'])` | import (update) the values of volume density of iron losses. |
| `lastInstance = CosimImportDataRegSurf(name='PIRON_STATOR', spatialParameter=SpatialParameter['PIRON'], position=ReferencePosition(), lengthUnit=LengthUnit['METER'], coordSys=CoordSys['XY1'], importMethod=NodeToNode(), regionFace=RegionFace['STATOR'])` | import (update) the values of volume density of iron losses. |
| `lastInstance = FluxCosimulation(name='Cosimulation_THMG', exchangeDirectory='../Exchange', outData=CosimExportedData[ALL], inData=CosimImportedData[ALL], convergenceCriterion=FluxEvalConvergence(threshold=1.0))` | create a cosimulation entity. |
| `solveCosimulation()` | run the cosimulation entity to solve the multiphysics problem. |
| `Scenario['SCENARIO_THMG'].closeSessionMultiPhysics()` | close the multiphysics solving session. |
| `saveProjectAs('CASE_3_TH.FLU')` | save the thermal project under the name CASE_3_TH.FLU. |

**Action (3)**

In order to execute the command file **CASE_3_MG.py** in batch mode the Python file **MAIN_MG.py** should be launched (this file will launch the file **MAIN_EX_MG.py** that will launch at its turn the file **CASE_3_MG.py**).
Similarly to execute the command file **CASE_3_TH.py** in batch mode the **MAIN_TH.py** Python file should be launched (this file will launch the file **MAIN_EX_TH.py** that will launch at its turn the file **CASE_3_TH.py**).

## 5.4. Case 3: Results post-processing

**Goal**

The principal results of **case 3** are analyzed as follows:

- in case of **Transient / time dependent Magnetic** application:
  - the color-shade of magnetic flux density
  - the equi-flux lines
- in the **Transient State Thermal** application:
  - the volume density of Joule losses in [W/m$^3$] - average value over an electric cycle - heat source for the thermal analysis
  - the volume density of iron losses in [W/m$^3$] - average value over an electric cycle - heat source for the thermal analysis
  - the color-shade of temperature in [K]

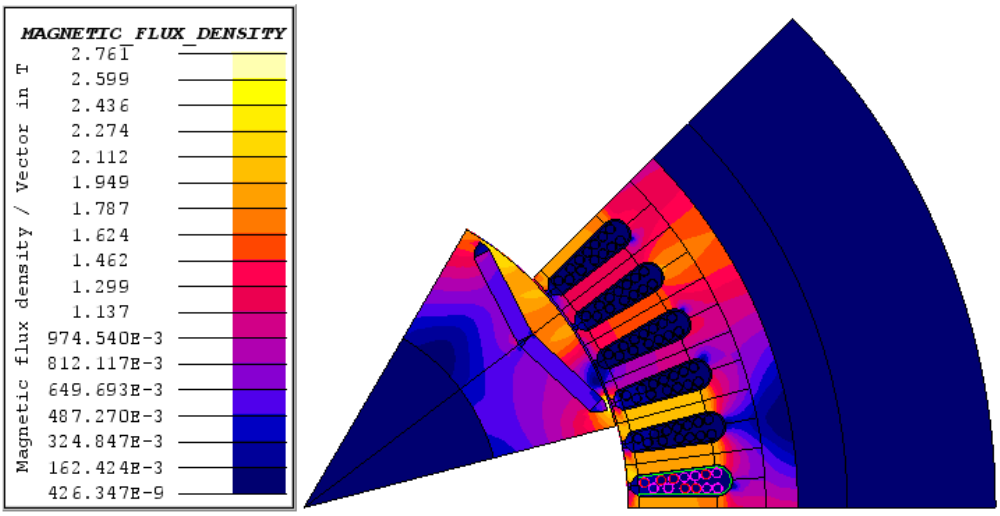
**Action (1)**

***Transient / time dependent magnetic application – CASE_3_MG.FLU:***

The graphical representation of the magnetic flux density (color-shade) and equi-flux lines should be done by exploiting the numerical results stored in CASE_3_MG.FLU Flux project.

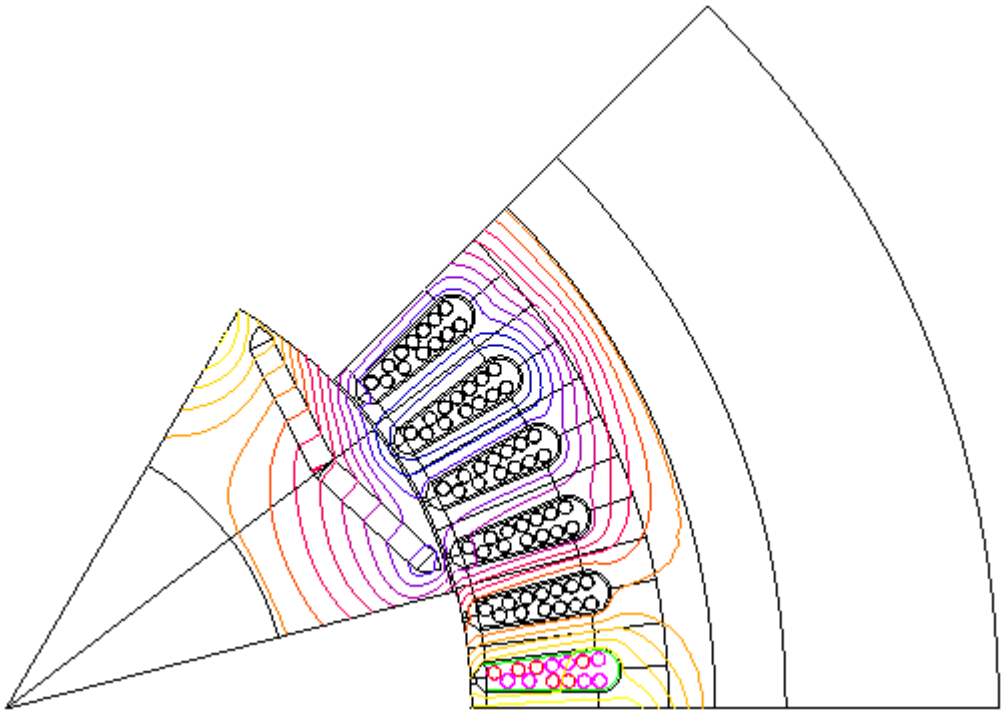
**Result (1)**

The following charts shows the magnetic flux density and the equi-flux lines at time instant t = 0.001 sec.




Color-shade representation of magnetic flux density. Transient / time dependent magnetic analysis (t = 0.001 sec.)

*Continued on next page*



Equi-flux lines. Transient / time dependent magnetic analysis (t = 0.001 sec.)

**Action (2)**

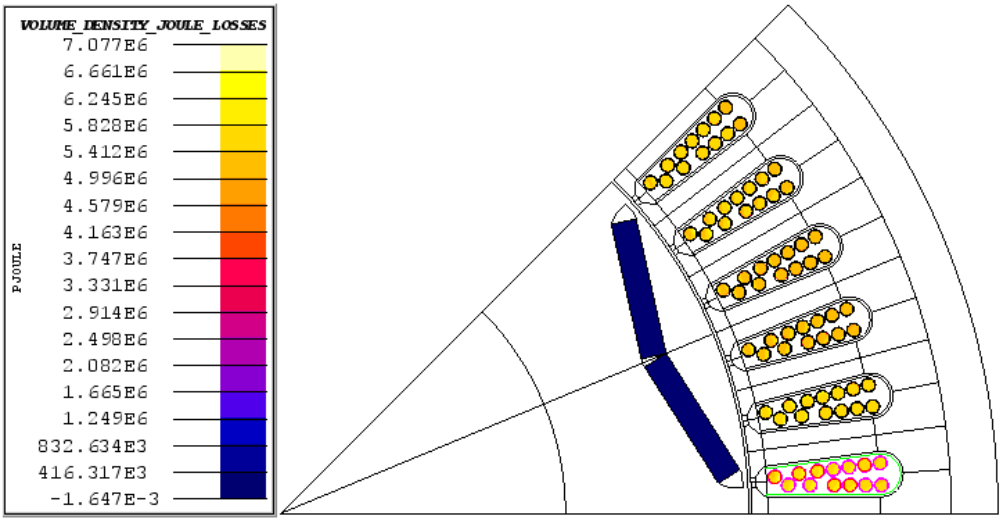
***Steady State Thermal application – CASE_3_TH.FLU:***

The color-shade representation of the volume density of Joule and iron losses and the temperature (color-shade) should be done by exploiting the numerical results stored in CASE_3_TH.FLU Flux project.

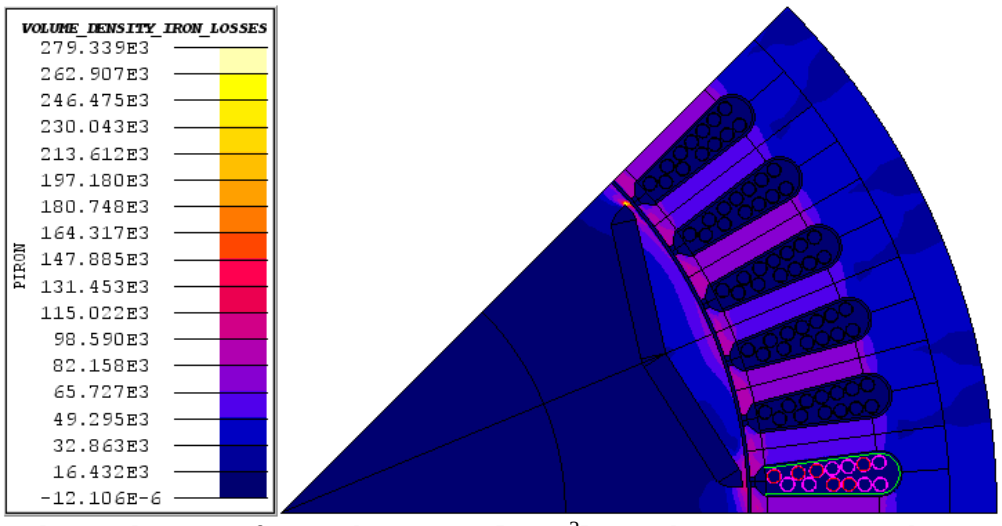
*Continued on next page*

**Result (2)**

The following charts represent the volume density of the Joule losses in the solid conductors (i.e. stator conductors and PMs) expressed in [W/m$^3$], the iron losses in the rotor and stator magnetic cores expressed in [W/m$^3$] and the temperature in [°C] on the 2D computation domain. The volume densities of the Joule and iron losses are the average values over an electric cycle.

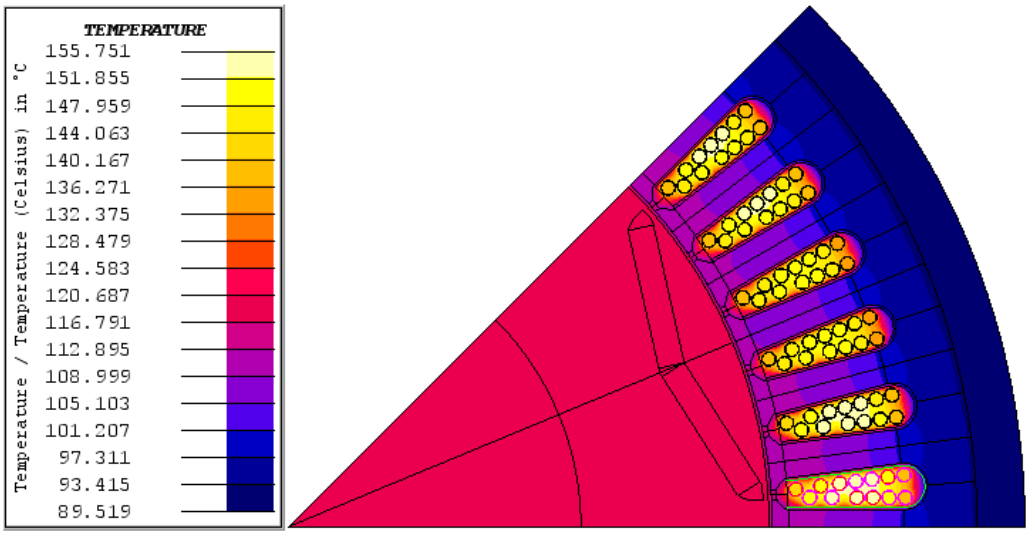



Volume density of Joule losses in [W/m$^3$] on the computation domain.
Steady state thermal 2D analysis




Volume density of iron losses in [W/m$^3$] on the computation domain.
Steady state thermal 2D analysis

*Continued on next page*




*Temperature in [°C] on the computation domain. Steady state thermal 2D analysis*

| | | |
|---|---|---|
| ☞ | **Graphic → Isovalues → New** | |
| ☞ | **Graphic → Isolines → New** | |

# 6. Annex

**Presentation**

The content of the command files used to carry out the multiphysics coupling case 3 are presented in the next sections.

**Contents**

This chapter contains the following topics:

# 6.1. Command files for the case 3

## 6.1.1. Command files for the Transient / time dependent magnetic application

**MAIN_MG.py**

The **MAIN_MG.py** command file is *the first to be launched* for the multiphysics coupling (transient / time dependent magnetic - steady state thermal). The content of this file is presented below. This file will launch in batch mode the command file **MAIN_EX_MG.py**.

```
#! Flux2D 12.0
try :
    executeBatchSpy('MAIN_EX_MG.py')
except :
    resetError()
closeProject()
exit()
```

**MAIN_EX_MG.py**

The **MAIN_EX_MG.py** command file will launch the file **CASE_3_MG.py** that contains the multiphysics coupling commands. The content of **MAIN_EX_MG.py** file is presented below.

```
#! Preflu2D 11.2
try :
   loadProject('CASE_3_MG_INI.FLU')
except :
   resetError()
try :
    executeBatchSpy('CASE_3_MG.py')
except :
   resetError()
```

*Continued on next page*

**CASE_3_MG.py** The **CASE_3_MG.py** command file corresponds to the transient / time dependent magnetic application and it is used to carry out the multiphysics coupling (transient / time dependent magnetic - steady state thermal). The content of this file is detailed below.

```
#! Flux2D 11.2

# CASE 3: MULTIPHYSICS COUPLING / TRANSIENT MAGNETIC APPLICATION

from math import *
import os
import sys

try :
################ STEP1: PHYSICS PREPARATION ###################

saveProjectAs('CASE_3_MG.FLU')

CreateTemperature(defaultValue=293.15)

## Creation of a sensor to compute the integral of temperature on the conductors of a
## phase. The temperature will be practically the same on all the pases. Integration
## on a plane

lastInstance=SensorIntegralFace(name='T_INT_PHASE',spatialFormula='TKELVIN',
integrationType=PlaneIntegration(),
support=SupportIntegralRegionFace(region=[RegionFace['PHASE1_COND_1'],
RegionFace['PHASE1_COND_10'],
RegionFace['PHASE1_COND_11'],
RegionFace['PHASE1_COND_12'],
RegionFace['PHASE1_COND_13'],
RegionFace['PHASE1_COND_14'],
RegionFace['PHASE1_COND_15'],
RegionFace['PHASE1_COND_16'],
RegionFace['PHASE1_COND_17'],
RegionFace['PHASE1_COND_18'],
RegionFace['PHASE1_COND_19'],
RegionFace['PHASE1_COND_2'],
RegionFace['PHASE1_COND_20'],
RegionFace['PHASE1_COND_21'],
RegionFace['PHASE1_COND_22'],
RegionFace['PHASE1_COND_23'],
RegionFace['PHASE1_COND_24'],
RegionFace['PHASE1_COND_25'],
RegionFace['PHASE1_COND_26'],
RegionFace['PHASE1_COND_3'],
RegionFace['PHASE1_COND_4'],
RegionFace['PHASE1_COND_5'],
RegionFace['PHASE1_COND_6'],
RegionFace['PHASE1_COND_7'],
RegionFace['PHASE1_COND_8'],
RegionFace['PHASE1_COND_9']]),
evaluationMode=preStepEvaluation(defaultValue='293.15'))
```

*Continued on next page*

```
## Creation of I/O parameters

lastInstance = VariationParameterFormula(name='S_26COND', formula='0.00012766')

lastInstance=VariationParameterFormula(name='RES_EW_COIL_1',
formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))')

lastInstance=VariationParameterFormula(name='RES_EW_COIL_2',
formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))')

lastInstance=VariationParameterFormula(name='RES_EW_COIL_3',
formula='0.034*(1+0.00427*((T_INT_PHASE/S_26COND)-293.15))')

VariationParameterFormula(name='ANGPOS_ROTOR_DEG',formula='AngPos(ROTOR)')

## Creation of spatial quantities for multiphysics

lastInstance=SpatialParameterFormula(name='BR_NDFEB',
SpatialFormula='1.2*(1-0.0013*(TKelvin-293.15))')

lastInstance=SpatialParameterFormula(name='BX_1',
SpatialFormula='BR_NDFEB*Cosd(22.5+10+IO(ANGPOS_ROTOR_DEG))')

lastInstance=SpatialParameterFormula(name='BY_1',
SpatialFormula='BR_NDFEB*Sind(22.5+10+IO(ANGPOS_ROTOR_DEG))')

lastInstance=SpatialParameterFormula(name='BX_2',
SpatialFormula='BR_NDFEB*Cosd(22.5-10+IO(ANGPOS_ROTOR_DEG))')

lastInstance=SpatialParameterFormula(name='BY_2',
SpatialFormula='BR_NDFEB*Sind(22.5-10+IO(ANGPOS_ROTOR_DEG))')

lastInstance=SpatialParameterFormula(name='RESISTIVITY_NDFEB',
SpatialFormula='1.4*(1+0.0014*(TKelvin-293.15))*1E-6')

lastInstance=SpatialParameterFormula(name='RESISTIVITY_CU',
SpatialFormula='1.7*(1+0.00427*(TKelvin-293.15))*1E-8')

## Creation of materials with temperature dependent properties for multiphysics

Material(name='NDFEB_1TEMP',
propertyBH=PropertyBhMagnetSpatial(br='Vec3(BX_1,BY_1,0)',
applyingMovement='OUI',mur='1.05'),
propertyJE=PropertyJeLinearSpatial(rho='RESISTIVITY_NDFEB'))

Material(name='NDFEB_2TEMP',
propertyBH=PropertyBhMagnetSpatial(br='Vec3(BX_2,BY_2,0)',
applyingMovement='OUI',mur='1.05'),
propertyJE=PropertyJeLinearSpatial(rho='RESISTIVITY_NDFEB'))

Material(name='COPPER_TEMP',
propertyBH=PropertyBhLinear(mur='1'),
propertyJE=PropertyJeLinearSpatial(rho='RESISTIVITY_CU'))

## Modifying the properties of circuit components as functions of temperature
## (TKELVIN):

startMacroTransaction()
Resistor['R_EW_1'].resistance='RES_EW_COIL_1'
Resistor['R_EW_2'].resistance='RES_EW_COIL_2'
Resistor['R_EW_3'].resistance='RES_EW_COIL_3'
endMacroTransaction()
```

*Continued on next page*

```
## Modifying the properties face regions as functions of temperature (TKELVIN):

## phase 1

RegionFace['PHASE1_COND_1'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_1']))

RegionFace['PHASE1_COND_2'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_2']))

RegionFace['PHASE1_COND_3'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_3']))

RegionFace['PHASE1_COND_4'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_4']))

RegionFace['PHASE1_COND_5'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_5']))

RegionFace['PHASE1_COND_6'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_6']))

RegionFace['PHASE1_COND_7'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_7']))

RegionFace['PHASE1_COND_8'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_8']))

RegionFace['PHASE1_COND_9'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_9']))

RegionFace['PHASE1_COND_10'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_10']))

RegionFace['PHASE1_COND_11'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_11']))

RegionFace['PHASE1_COND_12'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_12']))

RegionFace['PHASE1_COND_13'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_13']))
```

*Continued on next page*

```
RegionFace['PHASE1_COND_14'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_14']))

RegionFace['PHASE1_COND_15'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_15']))

RegionFace['PHASE1_COND_16'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_16']))

RegionFace['PHASE1_COND_17'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_17']))

RegionFace['PHASE1_COND_18'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_18']))

RegionFace['PHASE1_COND_19'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_19']))

RegionFace['PHASE1_COND_20'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_20']))

RegionFace['PHASE1_COND_21'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_21']))

RegionFace['PHASE1_COND_22'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_22']))

RegionFace['PHASE1_COND_23'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_23']))

RegionFace['PHASE1_COND_24'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_24']))

RegionFace['PHASE1_COND_25'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_25']))

RegionFace['PHASE1_COND_26'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE1_COND_26']))
```

*Continued on next page*

```
## phase 2

RegionFace['PHASE2_COND_1'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_1']))

RegionFace['PHASE2_COND_2'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_2']))

RegionFace['PHASE2_COND_3'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_3']))

RegionFace['PHASE2_COND_4'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_4']))

RegionFace['PHASE2_COND_5'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_5']))

RegionFace['PHASE2_COND_6'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_6']))

RegionFace['PHASE2_COND_7'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_7']))

RegionFace['PHASE2_COND_8'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_8']))

RegionFace['PHASE2_COND_9'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_9']))

RegionFace['PHASE2_COND_10'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_10']))

RegionFace['PHASE2_COND_11'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_11']))

RegionFace['PHASE2_COND_12'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_12']))

RegionFace['PHASE2_COND_13'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_13']))
```

*Continued on next page*

```
RegionFace['PHASE2_COND_14'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_14']))

RegionFace['PHASE2_COND_15'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_15']))

RegionFace['PHASE2_COND_16'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_16']))

RegionFace['PHASE2_COND_17'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_17']))

RegionFace['PHASE2_COND_18'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_18']))

RegionFace['PHASE2_COND_19'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_19']))

RegionFace['PHASE2_COND_20'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_20']))

RegionFace['PHASE2_COND_21'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_21']))

RegionFace['PHASE2_COND_22'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_22']))

RegionFace['PHASE2_COND_23'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_23']))

RegionFace['PHASE2_COND_24'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_24']))

RegionFace['PHASE2_COND_25'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_25']))

RegionFace['PHASE2_COND_26'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE2_COND_26']))
```

*Continued on next page*

```
## phase 3

RegionFace['PHASE3_COND_1'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_1']))

RegionFace['PHASE3_COND_2'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_2']))

RegionFace['PHASE3_COND_3'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_3']))

RegionFace['PHASE3_COND_4'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_4']))

RegionFace['PHASE3_COND_5'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_5']))

RegionFace['PHASE3_COND_6'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_6']))

RegionFace['PHASE3_COND_7'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_7']))

RegionFace['PHASE3_COND_8'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_8']))

RegionFace['PHASE3_COND_9'].magneticTransient2D=MagneticTransient2DFaceSolidConductor(
material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_9']))

RegionFace['PHASE3_COND_10'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_10']))

RegionFace['PHASE3_COND_11'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_11']))

RegionFace['PHASE3_COND_12'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_12']))

RegionFace['PHASE3_COND_13'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_13']))
```

*Continued on next page*

```
RegionFace['PHASE3_COND_14'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_14']))

RegionFace['PHASE3_COND_15'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_15']))

RegionFace['PHASE3_COND_16'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_16']))

RegionFace['PHASE3_COND_17'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_17']))

RegionFace['PHASE3_COND_18'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_18']))

RegionFace['PHASE3_COND_19'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_19']))

RegionFace['PHASE3_COND_20'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_20']))

RegionFace['PHASE3_COND_21'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_21']))

RegionFace['PHASE3_COND_22'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_22']))

RegionFace['PHASE3_COND_23'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_23']))

RegionFace['PHASE3_COND_24'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_24']))

RegionFace['PHASE3_COND_25'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_25']))

RegionFace['PHASE3_COND_26'].magneticTransient2D=MagneticTransient2DFaceSolidConductor
(material=Material['COPPER_TEMP'],
circuitType=Circuit(orientation=PositiveOrientation(),
component=SolidConductor2Terminals['PHASE3_COND_26']))
```

*Continued on next page*

```
## magnets

RegionFace['MAGNET1_1_POLE_1'].magneticTransient2D=MagneticTransient2DFaceSolidConduct
or(material=Material['NDFEB_1'], circuitType=OpenCircuit())

RegionFace['MAGNET2_1_POLE_1'].magneticTransient2D=MagneticTransient2DFaceSolidConduct
or(material=Material['NDFEB_2'], circuitType=OpenCircuit())

############### STEP2: PREPARATION OF PARAMETERS AND MULTIPHYSICS FORMULA FOR COUPLING

Scenario['SCENARIO_MG'].openSessionMultiPhysics(projectName='CASE_3_MG.FLU')

## Create spatial quantities for multiphysics coupling

lastInstance = JouleLossesMPSensor(name='PJ_PREDEFINED')

lastInstance = SpatialFormulaMPSensor(name='PJOULE', FormulaMPSensor='PJ_PREDEFINED')

lastInstance = BertottiIronLossesMPSensor(name='PIRON',
coefficients=BertottiCoefficients(hysteresisLossCoefficient=130.346,
classicalLossCoefficient=1923077.0,
excessLossCoefficient=0.357,
sheetIronThickness=3.5E-4,
stackingFactor=0.97,
hysteresisLossTermBExponent=2.0,
hysteresisLossTermFrequencyExponent=1.0,
classicalLossTermBExponent=2.0,
classicalLossTermFrequencyExponent=2.0,
excessLossTermBExponent=1.5,
excessLossTermFrequencyExponent=1.5),
orientation=SheetIronOrientationXY(coordSys=CoordSys['XY1']))

############### STEP3: EXPORT OF MAGNETIC MESH NODES COORDINATES ###################

## phase 1

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE1_COND1'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND2'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_2'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND3'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_3'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND4'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_4'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE1_COND5'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_5'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND6'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_6'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND7'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_7'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND8'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_8'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND9'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_9'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND10'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_10'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND11'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_11'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND12'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_12'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND13'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_13'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE1_COND14'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_14'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND15'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_15'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND16'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_16'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND17'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_17'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND18'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_18'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND19'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_19'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND20'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_20'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND21'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_21'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND22'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_22'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE1_COND23'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_23'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND24'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_24'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND25'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_25'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE1_COND26'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE1_COND_26'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

## phase 2

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND1'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND2'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_2'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND3'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_3'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND4'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_4'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND5'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_5'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE2_COND6'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_6'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND7'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_7'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND8'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_8'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND9'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_9'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND10'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_10'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND11'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_11'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND12'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_12'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND13'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_13'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND14'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_14'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE2_COND15'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_15'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND16'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_16'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND17'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_17'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND18'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_18'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND19'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_19'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND20'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_20'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND21'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_21'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND22'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_22'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND23'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_23'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE2_COND24'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_24'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND25'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_25'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE2_COND26'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE2_COND_26'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

## phase 3

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND1'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND2'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_2'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND3'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_3'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND4'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_4'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND5'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_5'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND6'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_6'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE3_COND7'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_7'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND8'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_8'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND9'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_9'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND10'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_10'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND11'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_11'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND12'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_12'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND13'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_13'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND14'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_14'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND15'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_15'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE3_COND16'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_16'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND17'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_17'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND18'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_18'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND19'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_19'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND20'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_20'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND21'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_21'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND22'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_22'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND23'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_23'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND24'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_24'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/
MESH_MAG_PHASE3_COND25'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_25'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_PHASE3_COND26'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['PHASE3_COND_26'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

## rest of regions

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_MAGNET_1.DEX'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['MAGNET1_1_POLE_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_MAGNET_2.DEX'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['MAGNET2_1_POLE_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_ROTOR.DEX'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['ROTOR'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_M
AG_STATOR.DEX'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['STATOR'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

############### STEP4: CREATION OF MULTIPOINT SUPPORT AFTER MAGNETIC NODES EXPORT ###

## Creation of a magnetic iteration synchronization file
## The thermal problem can continue the computation

f1=open('Synchro_M.txt','a')
f1.close()

print 'wait for response from thermal application'

## Waiting for the thermal iteration synchronization file to continue the magnetic
## computation

waitForFile(fileName="../THERMAL/Synchro_T.txt",existFile="exist")
```

*Continued on next page*

```
## Create multipoint supports for multiphysics study

MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE1_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE1.DEX
'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE2_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE2.DEX
'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='COND_PHASE3_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_COND_PHASE3.DEX
'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='MAGNET1_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_MAGNET1.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=Refer
encePosition(),
mechanicalSet=MechanicalSet['ROTOR']),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='MAGNET2_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_MAGNET2.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=Refer
encePosition(),
mechanicalSet=MechanicalSet['ROTOR']),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='ROTOR_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_ROTOR.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithMechanicalSet(mechanicalSetPosition=Refer
encePosition(),
mechanicalSet=MechanicalSet['ROTOR']),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='STATOR_TH',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_TH_STATOR.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
############### STEP5: DATA EXPORT CREATION ###################

lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PJOULE'],
multipointSupport=MultipointSupport['COND_PHASE1_TH'])

lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PJOULE'],
multipointSupport=MultipointSupport['COND_PHASE2_TH'])

lastInstance = CosimExportMultipointSupport(name='PJ_COND_PHASE_3',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PJOULE'],
multipointSupport=MultipointSupport['COND_PHASE3_TH'])

lastInstance = CosimExportMultipointSupport(name='PJ_MAGNET_1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PJOULE'],
multipointSupport=MultipointSupport['MAGNET1_TH'])

lastInstance = CosimExportMultipointSupport(name='PJ_MAGNET_2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PJOULE'],
multipointSupport=MultipointSupport['MAGNET2_TH'])

lastInstance = CosimExportMultipointSupport(name='PIRON_ROTOR',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PIRON'],
multipointSupport=MultipointSupport['ROTOR_TH'])

lastInstance = CosimExportMultipointSupport(name='PIRON_STATOR',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['PIRON'],
multipointSupport=MultipointSupport['STATOR_TH'])
```

*Continued on next page*

```
############### STEP6: DATA IMPORT CREATION ###################

## phase 1

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND1',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_1'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND2',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_2'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND3',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_3'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND4',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_4'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND5',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_5'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND6',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_6'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND7',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_7'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND8',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_8'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND9',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_9'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND10',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_10'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND11',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_11'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND12',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_12'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND13',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_13'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND14',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_14'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND15',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_15'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND16',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_16'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND17',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_17'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND18',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_18'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND19',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_19'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND20',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_20'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND21',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_21'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND22',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_22'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND23',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_23'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND24',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_24'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND25',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_25'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE1_COND26',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE1_COND_26'])


## phase 2

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND1',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_1'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND2',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_2'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND3',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_3'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND4',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_4'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND5',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_5'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND6',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_6'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND7',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_7'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND8',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_8'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND9',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_9'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND10',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_10'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND11',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_11'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND12',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_12'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND13',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_13'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND14',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_14'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND15',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_15'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND16',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_16'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND17',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_17'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND18',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_18'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND19',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_19'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND20',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_20'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND21',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_21'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND22',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_22'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND23',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_23'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND24',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_24'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND25',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_25'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE2_COND26',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE2_COND_26'])


## phase3

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND1',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_1'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND2',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_2'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND3',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_3'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND4',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_4'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND5',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_5'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND6',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_6'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND7',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_7'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND8',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_8'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND9',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_9'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND10',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_10'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND11',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_11'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND12',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_12'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND13',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_13'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND14',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_14'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND15',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_15'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND16',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_16'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND17',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_17'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND18',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_18'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND19',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_19'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND20',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_20'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND21',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_21'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND22',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_22'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND23',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_23'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND24',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_24'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND25',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_25'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_PHASE3_COND26',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['PHASE3_COND_26'])

## rest of regions

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_MAGNET_1',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['MAGNET1_1_POLE_1'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_MAGNET_2',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['MAGNET2_1_POLE_1'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_ROTOR',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['ROTOR'])

lastInstance = CosimImportDataRegSurf(name='TEMPERATURE_STATOR',
spatialParameter=SpatialParameter['TKELVIN'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['STATOR'])

############### STEP7: COSIMULATION CREATION AND SOLVING ###################

lastInstance = FluxCosimulation(name='Cosimulation_MG',
transientComputationType=EstablishedTransientType(parameterSet=[SetParameterXVariable(
paramEvol=VariationParameter['TIME'],limitMin=0.001,limitMax=0.0135)]),
exchangeDirectory='../Exchange', outData=CosimExportedData[ALL],
inData=CosimImportedData[ALL],convergenceCriterion=ThirdPartEvalConvergence())

solveCosimulation()

Scenario['SCENARIO_MG'].closeSessionMultiPhysics()

saveProjectAs('CASE_3_MG.FLU')

except :
    traceback.print_exc(file=sys.stderr)
    ef = open('hs_err_pp_res','a')
    ef.close()
    resetError()
    closeProject()
    exit()
```

*Continued on next page*

## 6.1.2. Command files for the Steady state thermal application

**MAIN_TH.py**

The **MAIN_TH.py** command file is *the second to be launched* for the multiphysics coupling (transient / time dependent magnetic - steady state thermal). The content of this file is presented below. This file will launch in batch mode the command file **MAIN_EX_TH.py**.

```
#! Flux2D 12.0
try :
    executeBatchSpy('MAIN_EX_TH.py')
except :
    resetError()
closeProject()
exit()
```

**MAIN_EX_TH.py**

The **MAIN_EX_TH.py** command file will launch the file **CASE_3_TH.py** that contains the multiphysics coupling commands. The content of **MAIN_EX_TH.py** file is presented below.

```
#! Preflu2D 11.2
try :
   loadProject('CASE_3_TH_INI.FLU')
except :
   resetError()
try :
    executeBatchSpy('CASE_3_TH.py')
except :
   resetError()
```

*Continued on next page*

**CASE_3_TH.py** The **CASE_3_TH.py** command file corresponds to the steady state thermal application and it is used to carry out the multiphysics coupling (transient / time dependent magnetic - steady state thermal). The content of this file is detailed below.

```
#! Preflu2D 11.2

# CASE 3: MULTIPHYSICS COUPLING / STEADY STATE THERMAL APPLICATION

from math import *
import os
import sys

try :

############### STEP1: PHYSICS PREPARATION ###################

saveProjectAs('CASE_3_TH.FLU')

lastInstance = SpatialParameterTabulated(name='PJOULE',
parameterType=ScalarReal(),
defaultValue='0')

lastInstance = SpatialParameterTabulated(name='PIRON',
parameterType=ScalarReal(),
defaultValue='0')

## Modify the heat sources of the face regions

startMacroTransaction()

RegionFace['STATOR_COND_PHASE_1'].thermalSteady2D=Thermal2DFaceThermalConductor(materi
al=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))

RegionFace['STATOR_COND_PHASE_2'].thermalSteady2D=Thermal2DFaceThermalConductor(materi
al=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))

RegionFace['STATOR_COND_PHASE_3'].thermalSteady2D=Thermal2DFaceThermalConductor(materi
al=Material['FLU_COPPER'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))

RegionFace['MAGNET1_1_POLE_1'].thermalSteady2D=Thermal2DFaceThermalConductor(material=
Material['NDFEB'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))

RegionFace['MAGNET2_1_POLE_1'].thermalSteady2D=Thermal2DFaceThermalConductor(material=
Material['NDFEB'], heat=VolumeHeatSourceSpatialFormula(value='PJOULE'))

RegionFace['ROTOR'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['I
RON'], heat=VolumeHeatSourceSpatialFormula(value='PIRON'))

RegionFace['STATOR'].thermalSteady2D=Thermal2DFaceThermalConductor(material=Material['
IRON'], heat=VolumeHeatSourceSpatialFormula(value='PIRON'))

endMacroTransaction()
```

*Continued on next page*

```
############### STEP2:CREATE MULTIPHYSICS FORMULA FOR TEMPERATURE ###################

Scenario(name='Scenario_THMG')

Scenario['SCENARIO_THMG'].openSessionMultiPhysics(projectName='CASE_3_TH.FLU')

lastInstance = TemperatureMPSensor(name='TEMP')

############### STEP3: EXPORT OF THERMAL MESH NODES COORDINATES ###################

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_CO
ND_PHASE1'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['STATOR_COND_PHASE_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_CO
ND_PHASE2'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['STATOR_COND_PHASE_2'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_CO
ND_PHASE3'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['STATOR_COND_PHASE_3'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_MA
GNET1'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['MAGNET1_1_POLE_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_MA
GNET2'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['MAGNET2_1_POLE_1'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_RO
TOR'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['ROTOR'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])

result=ExportNodeCoordinatesRegionFace(file=exportDexFluxFile(fileName='../Mesh/MESH_TH_ST
ATOR'),
mechanicalSetPosition=ReferencePosition(),
regionFace=RegionFace['STATOR'],
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
############### STEP4: CREATION OF MULTIPOINT SUPPORT AFTER MAGNETIC NODES EXPORT ###

# Creation of a thermal iteration synchronization file
# The magnetic problem can continue the computation

f=open('Synchro_T.txt','a')
f.close()

print 'Wait for response from Magnetic application'

# Waiting for the magnetic iteration synchronization file to continue the thermal
# computation

waitForFile(fileName="../MAGNETIC/Synchro_M.txt",existFile="exist")

## Create multipoint supports for multiphysics study

## phase 1

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND1_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND1.D
EX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND2_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND2.D
EX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND3_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND3.D
EX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND4_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND4.D
EX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND5_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND5.D
EX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND6_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND6.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND7_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND7.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND8_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND8.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND9_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND9.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND10_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND10.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND11_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND11.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND12_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND12.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND13_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND13.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND14_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND14.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND15_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND15.
DEX'),lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND16_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND16.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND17_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND17.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND18_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND18.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND19_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND19.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND20_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND20.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND21_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND21.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND22_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND22.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND23_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND23.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND24_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND24.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND25_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND25.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE1_COND26_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE1_COND26.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

## phase 2

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND1_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND1.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND2_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND2.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND3_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND3.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND4_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND4.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND5_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND5.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND6_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND6.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND7_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND7.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND8_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND8.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND9_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND9.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND10_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND10.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND11_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND11.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND12_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND12.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND13_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND13.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND14_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND14.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND15_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND15.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND16_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND16.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND17_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND17.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND18_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND18.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND19_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND19.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND20_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND20.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND21_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND21.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND22_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND22.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND23_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND23.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND24_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND24.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND25_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND25.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE2_COND26_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE2_COND26.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

## phase 3

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND1_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND1.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND2_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND2.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND3_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND3.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND4_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND4.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND5_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND5.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND6_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND6.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND7_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND7.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND8_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND8.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND9_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND9.D
EX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND10_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND10.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND11_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND11.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND12_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND12.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND13_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND13.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND14_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND14.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND15_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND15.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND16_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND16.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND17_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND17.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND18_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND18.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND19_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND19.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND20_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND20.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND21_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND21.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND22_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND22.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND23_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND23.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND24_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND24.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND25_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND25.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='PHASE3_COND26_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_PHASE3_COND26.
DEX'), lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])
```

*Continued on next page*

```
## rest of regions

MultipointSupportImportingCoordinates(multipointSupportName='MAGNET1_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_MAGNET_1.DEX')
, lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='MAGNET2_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_MAGNET_2.DEX')
, lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='ROTOR_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_ROTOR.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])

MultipointSupportImportingCoordinates(multipointSupportName='STATOR_MAG',
file=coordinatesImportDexFluxFile(coordinatesFileName='../Mesh/MESH_MAG_STATOR.DEX'),
lengthUnit=LengthUnit['METER'],
mechanicalSetDependency=MultipointSupportWithoutMechanicalSet(),
multipointSupportColor=Color['Turquoise'],
multipointSupportVisibility=Visibility['VISIBLE'],
coordSys=CoordSys['XY1'])


################ STEP5: DATA EXPORT ENTITIES CREATION ####################

## phase 1

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND1_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND2_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND3',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND3_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND4',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND4_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND5',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND5_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND6',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND6_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND7',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND7_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND8',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND8_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND9',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND9_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND10',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND10_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND11',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND11_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND12',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND12_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND13',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND13_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND14',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND14_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND15',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND15_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND16',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND16_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND17',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND17_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND18',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND18_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND19',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND19_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND20',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND20_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND21',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND21_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND22',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND22_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND23',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND23_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND24',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND24_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND25',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND25_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE1_COND26',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE1_COND26_MAG'])

## phase 2

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND1_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND2_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND3',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND3_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND4',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
PSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND4_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND5',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND5_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND6',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND6_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND7',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND7_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND8',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND8_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND9',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND9_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND10',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND10_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND11',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND11_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND12',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND12_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND13',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND13_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND14',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND14_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND15',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND15_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND16',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND16_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND17',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND17_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND18',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND18_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND19',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND19_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND20',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND20_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND21',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND21_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND22',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND22_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND23',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND23_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND24',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND24_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND25',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND25_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE2_COND26',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE2_COND26_MAG'])

## phase3

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND1_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND2_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND3',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND3_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND4',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND4_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND5',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND5_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND6',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND6_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND7',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND7_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND8',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND8_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND9',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND9_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND10',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND10_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND11',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],

multipointSupport=MultipointSupport['PHASE3_COND11_MAG'])
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND12',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],

multipointSupport=MultipointSupport['PHASE3_COND12_MAG'])
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND13',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],

multipointSupport=MultipointSupport['PHASE3_COND13_MAG'])
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND14',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND14_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND15',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND15_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND16',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND16_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND17',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND17_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND18',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND18_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND19',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND19_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND20',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND20_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND21',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND21_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND22',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND22_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND23',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND23_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND24',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND24_MAG'])
```

*Continued on next page*

```
lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND25',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND25_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_PHASE3_COND26',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['PHASE3_COND26_MAG'])

## rest of supports:

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_MAGNET_1',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['MAGNET1_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_MAGNET_2',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['MAGNET2_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_ROTOR',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['ROTOR_MAG'])

lastInstance = CosimExportMultipointSupport(name='TEMPERATURE_STATOR',
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
position=ReferencePosition(),
MPSensor=multiphysicsSensor['TEMP'],
multipointSupport=MultipointSupport['STATOR_MAG'])


############### STEP6: DATA IMPORT ENTITIES CREATION ###################

lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_1',
spatialParameter=SpatialParameter['PJOULE'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['STATOR_COND_PHASE_1'])

lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_2',
spatialParameter=SpatialParameter['PJOULE'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['STATOR_COND_PHASE_2'])
```

*Continued on next page*

```
lastInstance = CosimImportDataRegSurf(name='PJ_COND_PHASE_3',
spatialParameter=SpatialParameter['PJOULE'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['STATOR_COND_PHASE_3'])

lastInstance = CosimImportDataRegSurf(name='PJ_MAGNET_1',
spatialParameter=SpatialParameter['PJOULE'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['MAGNET1_1_POLE_1'])

lastInstance = CosimImportDataRegSurf(name='PJ_MAGNET_2',
spatialParameter=SpatialParameter['PJOULE'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['MAGNET2_1_POLE_1'])

lastInstance = CosimImportDataRegSurf(name='PIRON_ROTOR',
spatialParameter=SpatialParameter['PIRON'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['ROTOR'])

lastInstance = CosimImportDataRegSurf(name='PIRON_STATOR',
spatialParameter=SpatialParameter['PIRON'],
position=ReferencePosition(),
lengthUnit=LengthUnit['METER'],
coordSys=CoordSys['XY1'],
importMethod=NodeToNode(),
regionFace=RegionFace['STATOR'])


################ STEP7: COSIMULATION CREATION AND SOLVING ###################

lastInstance = FluxCosimulation(name='Cosimulation_TH',
exchangeDirectory='../Exchange',
outData=CosimExportedData[ALL],
inData=CosimImportedData[ALL],
convergenceCriterion=FluxEvalConvergence(threshold=1.0))

solveCosimulation()

Scenario['SCENARIO_THMG'].closeSessionMultiPhysics()

saveProjectAs('CASE_3_TH.FLU')

except :

ef = open('hs_err_pp_res','a')
ef.close()
resetError()
closeProject()
exit()
```

*Continued on next page*

# 6.2. List of multiphysics commands

## 6.2.1. Description of multiphysics commands

**Description of commands** The description of each multiphysic command is presented in the table below.

| command | description |
|---|---|
| Multiphysic solving session (new scenario) | Open a multiphysic session and create a new scenario. |
| Multiphysic solving session (existing scenario) | Open a multiphysic session with an existing scenario.<br>This command can be use to begin or to continue a session |
| Close a multiphysics session | Close the multiphysic session.<br>Allow to return of standard Flux context. |
| Define next steps | In multiphysic session, the user can choose the next step to compute.<br>This step can be a predefine step in scenario or an added step.<br><br>note : a multiphysic scemario can't be multi values, only mono value |
| Activate the next step | Activate the next step of the list of scenario.<br>The next computation can be executed. |
| Solve the current step | Solve the current step. |
| Solve several steps (only for transient application) | Solve successively several steps.<br>This command is only for transient application. |
| Get the status of the last processed step | Get the status of the last processed step.<br>This command is usually used by python command (not used by menu). |
| Reset the current step | The descritpion of this command depend of th type of multiphysic use :<br>• Spatial quantity: reset the chart of values by nodes.<br>• Parameter E/S: return on values of last step. |
| Initiate the current step | Allow to achieve the pre-solving processes ( mesh …). |
| Finalize the current step | Allow to achieve the post solving (user/predefined sensor…). |
| Delete the n last steps | Delete all the last computed steps |
| New spatial quantity Tkelvin for temperature | Create a spatial quantity TKELVIN corresponding of the temperature.<br>This spatial quantity is predefined. |
| New (constante) tabulated spatial quantity by storage of values | Create a tabulated spatial quantity by stockage of values. |
| New (constante) tabulated spatial quantity by importation | Create a tabulated spatial quantity by importation. |
| Update multiphysics spatial quantity by importation | Update of a multiphysic spatial quantity by importation of chart of values by nodes. |
| Update current value of mutliphysics parameter | Allow to modifiy the current value of a multiphysic parameter.<br>This command can be only used before the solving. |

*Continued on next page*

| | |
|---|---|
| Export spatial quantity and formula starting from different types of supports | Export of spatial quantity or a formule on each nodes of a support in a file.<br>This command is used after a computation.<br>The created file can be exchanged with another project. |
| Export coordinates of region nodes | Export coordinates of nodes of a region to add these nodes on a multipoint support in another project. |
| New multipoint support by importation of a list of points | Create a multipoint support by importation of a list of points.<br>This command. |

## 6.2.2. Access path menu for multiphysics commands

**Access path menu**

The access menu of each multiphysic command is presented in the table below.

| commands | Access path by menu | |
|---|---|---|
| | Standard Flux context | Multiphysic session |
| Multiphysic solving session (new scenario) | Solving | |
| Multiphysic solving session (existing scenario) | Solving | |
| Close a multiphysics session | | Project |
| Define next steps | | Multiphysics solving |
| Activate the next step | | |
| Solve the current step | | |
| Solve several steps (only for transient application) | | |
| Get the status of the last processed step | | |
| Reset the current step | | |
| Initiate the current step | | |
| Finalize the current step | | |
| Delete the n last steps | | |
| New spatial quantity Tkelvin for temperature | Parameter/Quantity → Spatial Quantity | |
| New (constante) tabulated spatial quantity by storage of values | Parameter/Quantity → Store quantity or Parameter/Quantity → Spatial Quantity | |
| New (constante) tabulated spatial quantity by importation | Parameter/Quantity → Import parameter and quantity or Parameter/Quantity → Spatial Quantity | |
| Update multiphysics spatial quantity by importation | | Parameter/Quantity → Import parameter and quantity or Parameter/Quantity → Spatial Quantity |
| Update current value of mutliphysics parameter | | Parameter/Quantity → Import parameter and quantity or Parameter/Quantity → I/O parameter |
| Exporter spatial quantity and formula starting from different types of supports | Parameter/Quantity → Export quantity | |
| Export coordinates of region | Parameter/Quantity → Export nodes of region | |
| New multipoint support by importation of a list of points | Parameter/Quantity → Multipoint support | |