

# TRUST ICoCo Tutorial V1.9.3

CEA Saclay

*Support team: [trust@cea.fr](mailto:trust@cea.fr)*

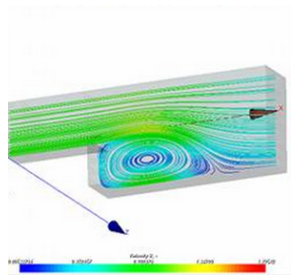
November 29, 2023

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
- 3 Basic test case
- 4 Coupled problem with ICoCo
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# Why code coupling?

## Code coupling ... what for ?

- Traditionnaly *numerical simulation codes* focus on a single physics
  - One code for thermics
  - Another code for mechanics,
  - Etc...
- Real life studies require the simulation of different physics
  - E.g. nuclear reactor simulations require a blend of: thermics, neutronics, mechanics...
- Solution? Code coupling!
  - Have different codes communicating one with another...
  - while each code deals with its own area of expertise



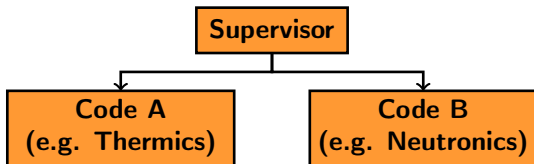
# Code coupling

## First approach

- An entity driving the complete computation is needed: the **supervisor**
  - Initializes code A and code B
  - Loop through code A and code B
  - Centralize exchanges and conversions between A and B

Supervisor is usually written from scratch as a C++ program, Python script, ...

- In a dummy approach : supervisor needs to know *both* A and B API.  
It becomes cumbersome if :
  - more than 2 codes to couple ...
  - single supervisor is meant to run with different pairs of codes (code C, D, E ...)



# Code coupling

Better: common interface!

**Idea:** define a unique *interface* to which each code must comply

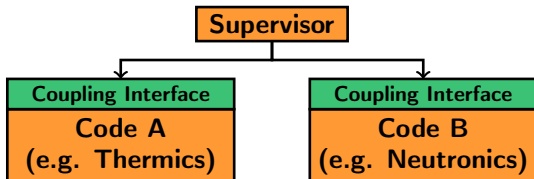
- Only an *interface*, i.e. a contract (=an API) that each code fulfills
- *Implementation* (=plugging wires) of this interface is tightly linked to the details of the code...

Advantages:

- All codes implementing the (unique) interface can be passed to the supervisor (almost) without modifying it! **Versatility**

Constraints:

- Need to implement the interface for each code to be coupled



# ICoCo, Interface for **C**ode **C**oupling

ICoCo is such a coupling interface:

- Stands for **I**nterface for **C**ode **C**oupling
- Written in C++ (and in swig since TRUST-1.8.3 to be used through Python)
- Initially designed for simulation codes exhibiting **iterative time loops**
- Presents a set of standard methods whose signature is fixed, with no default implementation:
  - initializeTimeStep()
  - validateTimeStep()
  - abortTimeStep()
  - ...
- Uses the notion of field for data exchange between codes
  - A field is a set of values supported by a mesh
  - A possible implementation: **MEDCouplingFieldDouble**, from SALOME's MEDCoupling library

## Further reading

### Meaningful documentation

- ICOCO presentation in TRUST documentation:  
**"An Interface for Code Coupling ICoCo v1.2"**  
load TRUST environnement  
`$ evince $TRUST_ROOT/doc/Kernel/ICoCo_V1.2.pdf &`
- Ask the "APIProblem.pdf" note to [trust@cea.fr](mailto:trust@cea.fr)

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization**
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!



# Initialization of TRUST environment

- Before starting this Tutorial,  
it is highly recommended to know some TRUST commands and hints.
- Source the TRUST environment:
  - On CEA Saclay PCs, TRUST versions are available with (e.g. X.Y.Z=1.8.3):  
**`source /home/triou/env_TRUST_X.Y.Z.sh`**
  - On your own computer, download and install the latest version of TRUST in your local folder \$MyPathToTRUSTversion (unless this was already performed), then write on the terminal:  
**`source $MyPathToTRUSTversion/env_TRUST.sh`**
- To check if the configuration is well and to locate sources:  
**`$ echo $TRUST_ROOT`**

# Initialization of ICoCo environment

- Source the ICoCo environment:

```
$ source $TRUST_ROOT/Outils/ICoCo/ICoCo_src/full_env_MEDICoCo.sh
```

- check if ICoCo is compiled:

```
$ ls $exec
```

- If you obtain "ls: cannot access \*\*/Outils/ICoCo/ICoCo\_src/ICoCo\_opt: No such file or directory", then compile ICoCo\_src:

```
$ cd $TRUST_ROOT/Outils/ICoCo/ICoCo_src
```

```
$ baltik_build_configure -execute
```

```
$ make optim debug
```

```
$ source full_env_MEDICoCo.sh
```

- If you do not have rights to compile ICoCo (Typically if TRUST install is made by someone else):

```
$ mkdir -p ICoCo
```

```
$ cp -r $TRUST_ROOT/Outils/ICoCo/ICoCo_src ICoCo/ICoCo_src
```

```
$ cd ICoCo/ICoCo_src
```

```
$ baltik_build_configure -execute ; make optim debug
```

```
$ source full_env_MEDICoCo.sh
```

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case**
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# Goal of this first exercise

The idea here is to:

- 1 Create a reference test case
  - To be launched with TRUST executable.
- 2 Update the test case in order to:
  - Launch it sequentially with ICoCo without information exchange.
  - Ensure that obtained results are identical to those obtained in step 1.
- 3 Realize your first information exchange between the supervisor and datafile
  - Impose a boundary condition from the supervisor and launch the calculation sequentially with ICoCo
  - Compare the results with those obtained with TRUST executable.
- 4 Make the test case running with ICoCo in parallel:
  - Launch the computation with ICoCo on 2 processes
  - Compare obtained results with those with TRUST executable.

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case**
  - TRUST: test case creation**
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# TRUST: test case creation

We will copy a reference test case from TRUST tests database that we launch with TRUST.

- Copy Vahl\_Davis\_hexa test case in your repository:  

```
$ mkdir ICoCo_exercises  
$ cd ICoCo_exercises  
$ trust -copy Vahl_Davis_hexa
```
- Rename the folder to distinguish your tests:  

```
$ mv Vahl_Davis_hexa Vahl_Davis_hexa_trust  
$ cd Vahl_Davis_hexa_trust
```
- Rename the datafile to be consistent with the folder's name:  

```
$ mv Vahl_Davis_hexa.data Vahl_Davis_hexa_trust.data
```
- Launch calculation:  

```
$ trust Vahl_Davis_hexa_trust
```
- You obtained a Vahl\_Davis\_hexa\_trust.lml file. We will need it in the following part.

# ICoCo: without exchange

## Adjusting the datafile

- Now let's do the same thing with ICoCo:

```
$ cd ..
```

- You are now in the folder "ICoCo\_exercises", create a new directory in order to launch your test case with ICoCo:

```
$ trust -copy Vahl_Davis_hexa
```

```
$ mv Vahl_Davis_hexa Vahl_Davis_hexa_ICoCo
```

- Rename the datafile to be consistent with the folder's name:

```
$ cd Vahl_Davis_hexa_ICoCo
```

```
$ mv Vahl_Davis_hexa.data Vahl_Davis_hexa_ICoCo.data
```

- Edit the datafile and add to it ICoCo instructions:

- *Add* the following line after "*dimension*" definition:

```
Nom ICoCoProblemName Lire ICoCoProblemName pb
```

- *Comment* the "*solve pb*" instruction at the end of the datafile.

# ICoCo: without exchange

## Creation of the main.cpp file

- Create the main.cpp which will launch the calculation:  
\$ file="main\_Vahl\_Davis\_hexa\_ICoCo.cpp"  
\$ cp \$TRUST\_ROOT/doc/TRUST/exercices/ICoCo/\$file main.cpp
- Open main.cpp file and see the main method which creates the objects and launches computation.
- You can use ICoCo with 1 or more processors, but in this part we use only one processor to solve the problem.

## Creation of the makefile

- Create a makefile for your calculation:  
\$ cp \$project\_directory/share/bin/create\_Makefile .  
\$ sh create\_Makefile
- Compile it:  
\$ make
- This creates an executable "couplage" and a datafile "couplage.data".



# ICoCo: without exchange

## Launch calculation

- Execute it:  
`$/couplage`
- You should obtain the same results as with TRUST executable. Compare TRUST and ICoCo results with:  
`$ compare_lata Vahl_Davis_hexa_ICoCo.lml  
../Vahl_Davis_hexa_trust/Vahl_Davis_hexa_trust.lml`
- Both lml files are identical!

## 1 Introduction to code coupling and ICoCo

## 2 TRUST and ICoCo initialization

- Initialization of TRUST environment
- Initialization of ICoCo environment

## 3 Basic test case

- TRUST: test case creation
- ICoCo: without exchange
- **ICoCo: first input**
- ICoCo: on 2 processes

## 4 Coupled problem with ICoCo

- TRUST: test case creation
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

## 5 ICoCo and TrioCFD

## 6 Using ICoCo API through Python

## 7 Conclusions and further simulations!

# ICoCo: first input

## Adjusting the main.cpp file

- Copy the following test case in your repository:  

```
$ cd ..  
$ cp -r Vahl_Davis_hexa_ICoCo Vahl_Davis_hexa_ICoCo_exchange  
$ cd Vahl_Davis_hexa_ICoCo_exchange
```
- Clean the repository and rename the datafile:  

```
$ trust -clean  
$ mv Vahl_Davis_hexa_ICoCo.data  
Vahl_Davis_hexa_ICoCo_exchange.data
```
- We will add a block in the main.cpp file to exchange values.
- Copy the following file in your folder:  

```
$ file="main_Vahl_Davis_hexa_ICoCo_exchange.cpp"  
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/$file main.cpp
```

# ICoCo: first input

## Adjusting the datafile

- Then modify the datafile Vahl\_Davis\_hexa\_ICoCo\_exchange.data to make the input:
  - Change the line:  
"Gauche Paroi\_temperature\_imposee Champ\_Front\_Uniforme 1 10."  
to  
"Gauche Paroi\_temperature\_imposee ch\_front\_input { nb\_comp 1 nom  
TEMPERATURE\_IN\_DOM probleme pb }"
  - You can see that the field "TEMPERATURE\_IN\_DOM" is the one employed in the main.cpp file.

# ICoCo: first input

## Launch calculation

- Now we can compile and launch the calculation:  
\$ make  
\$ ./couplage
- You may obtain the same results as with TRUST executable.
- Compare it:  
\$ compare\_lata Vahl\_Davis\_hexa\_ICoCo\_exchange.lml  
../Vahl\_Davis\_hexa\_trust/Vahl\_Davis\_hexa\_trust.lml
- The files are the same! **(but not for the first time step!!!!)**

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case**
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - **ICoCo: on 2 processes**
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# ICoCo: on 2 processes

## Adjusting the main.cpp file

- Copy the following test case in your repository:  
\$ cd ..  
\$ cp -r Vahl\_Davis\_hexa\_ICoCo\_exchange  
Vahl\_Davis\_hexa\_ICoCo\_para  
\$ cd Vahl\_Davis\_hexa\_ICoCo\_para
- Clean the repository and rename the datafile:  
\$ make clean  
\$ rm main.cpp Vahl\_Davis\_hexa\_ICoCo\_exchange.lml  
\$ mv Vahl\_Davis\_hexa\_ICoCo\_exchange.data  
Vahl\_Davis\_hexa\_ICoCo\_para.data
- Partition the domaine and create a parallel datafile:  
\$ trust -partition Vahl\_Davis\_hexa\_ICoCo\_para
- Copy the following file in your folder:  
\$ file="main\_Vahl\_Davis\_hexa\_ICoCo\_para.cpp"  
\$ cp \$TRUST\_ROOT/doc/TRUST/exercices/ICoCo/\$file main.cpp

# ICoCo: on 2 processes

## Adjusting the main.cpp file

- Open the main.cpp file and search for the MPI command lines.
- Open this file and look where:
  - the processors are added: search for "dom\_ids"
  - the names of the datafiles: search for "data\_file"
- Create the makefile and compile your new file:

```
$ sh create_Makefile  
$ make
```
- To run parallel, you have to use the following mpirun command:

```
$ mpirun -np 2 ./couplage
```
- Compare your results with the sequential ones:

```
$ compare_lata PAR_Vahl_Davis_hexa_ICoCo_para.lml  
../Vahl_Davis_hexa_exchange/Vahl_Davis_hexa_exchange.lml
```
- Sequential and parallel computation yield the same results!!



## Goal of this second exercise

The idea here is to:

- ① Create a reference test case treating a coupled problem
  - To be launched with TRUST executable in sequential and parallel.
- ② Create two datafiles, each one for a problem:
  - Launch it with TRUST first to ensure that datafiles do not contain errors
  - Launch it with ICoCo without any information exchange after updating datafiles.
- ③ Realize a one way coupling with ICoCo:
  - One way coupling means that one problem runs alone while some inputs of the second problem are provided by the first one
  - Launch the calculation sequentially with ICoCo
  - Compare the results with those obtained with TRUST executable.
- ④ Realize a two-way coupling with ICoCo:
  - Two-way means that each problem will have inputs provided by the other one.
  - Launch the computation with ICoCo.

## 1 Introduction to code coupling and ICoCo

## 2 TRUST and ICoCo initialization

- Initialization of TRUST environment
- Initialization of ICoCo environment

## 3 Basic test case

- TRUST: test case creation
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processes

## 4 Coupled problem with ICoCo

- **TRUST: test case creation**
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

## 5 ICoCo and TrioCFD

## 6 Using ICoCo API through Python

## 7 Conclusions and further simulations!

# First with TRUST

## Copy a coupled problem test case and launch it with TRUST

- Copy the coupled problem docond\_VEF\_3D from TRUST tests database:

```
$ cd ICoCo_exercises
```

```
$ trust -copy docond_VEF_3D
```

```
$ mv docond_VEF_3D docond_VEF_3D_trust
```

- Launch calculation:

```
$ cd docond_VEF_3D_trust
```

```
$ trust docond_VEF_3D
```

- Now, run the calculation in parallel also:

```
$ trust -partition docond_VEF_3D
```

```
$ trust PAR_docond_VEF_3D 2
```

- Compare the sequential & parallel results:

```
$ compare_lata docond_VEF_3D.1ml PAR_docond_VEF_3D.1ml
```

The results are the same. Differences are below the threshold:  $10^{-5}$ !

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 **Coupled problem with ICoCo**
  - TRUST: test case creation
  - **ICoCo: without exchange**
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# ICoCo: without exchange

## Separate the meshes of the coupled problem

- Create your ICoCo test case:

```
$ cd ICoCo_exercises
$ trust -copy docond_VEF_3D
$ mv docond_VEF_3D docond_VEF_3D_ICoCo
$ cd docond_VEF_3D_ICoCo
```
- Create separate mesh and calculation datafiles from docond\_VEF\_3D:

```
$ cp docond_VEF_3D.data docond_VEF_3D_mesh1.data
```
- Edit the file docond\_VEF\_3D\_mesh1.data, and:
  - Add a line containing "End" instruction after the partitionning block.
  - Remove all the lines below the added "End"
  - Remove the time scheme and the problems definition
  - Uncomment the partition block

Now, docond\_VEF\_3D\_mesh1.data datafile defines geometry, mesh and partitionning for both domains (solide and fluid).

# ICoCo: without exchange

## Separate the meshes of the coupled problem

- Create a datafile for each domain:  
`$ cp docond_VEF_3D_mesh1.data docond_VEF_3D_mesh2.data`
- In the file `docond_VEF_3D_mesh1.data`, keep only the information of the solid domain.
- In the file `docond_VEF_3D_mesh2.data`, keep only the information of the fluid domain.
- Run these datafiles to partition domains:  
`$ trust docond_VEF_3D_mesh1`  
`$ trust docond_VEF_3D_mesh2`  
You must these four .Zones files:  
DOM1\_0000.Zones DOM2\_0001.Zones  
DOM1\_0001.Zones DOM2\_0000.Zones

# ICoCo: without exchange

## Run with separated meshes

- In docond\_VEF\_3D.data datafile:
  - delete the mesh and partitioning blocks (which are now in the mesh datafiles)
  - Uncomment the 'scatter' block
- Launch the calculation in parallel:  
`$ trust docond_VEF_3D 2`
- Compare these results with previous parallel results:  
`$ compare_lata docond_VEF_3D.lml`  
`../docond_VEF_3D_trust/PAR_docond_VEF_3D.lml`  
The results are identical!
- Separate results in two lml files:
  - add "fichier pb1" in the "Post\_processing" block of the solid's problem,
  - add "fichier pb2" in the "Post\_processing" block of the fluid's problem,
- Run calculation to create these two files:  
`$ trust docond_VEF_3D 2`

# ICoCo: without exchange

## Separate the coupled problem into two new problems

- Create a datafile for the solid domain's problem:  
\$ cp docond\_VEF\_3D.data docond\_VEF\_3D\_dom1.data
- In docond\_VEF\_3D\_dom1.data datafile, remove the lines:
  - Probleme\_Couple pbc
  - Associate pbc pb1
  - Associate pbc pb2
  - "fichier pb1" and "fichier pb2"
- Change the following lines:
  - Associate pbc sch → Associate pb sch
  - Discretize pbc dis → Discretize pb dis
  - Solve pbc → Solve pb
- Create a datafile for the fluid domain's problem:  
\$ cp docond\_VEF\_3D\_dom1.data docond\_VEF\_3D\_dom2.data
- In docond\_VEF\_3D\_dom1.data, keep only the information about the solid domain (pb1, dom\_solide) and substitute pb1 → pb in the whole datafile



# Separate the coupled problem into two new problems

## Separate datafiles

- In `docond_VEF_3D_dom2.data`, keep only the information about the fluid domain (`pb2`, `dom_fluide`) and substitute `pb2` → `pb` in the whole datafile.
- Note that the coupling between solid and fluid problems is concretized by heat exchange between both domains. This coupling concerns the "`Paroi_echange1`" boundary of solid domain and "`Paroi_echange2`" boundary for of fluid domain.
- Modify `docond_VEF_3D_dom1.data` datafile to have:  
*Paroi\_echange1 paroi\_contact pb Paroi\_echange2*  
→  
*Paroi\_echange1 paroi\_temperature\_imposee Champ\_Front\_Uniforme 1 50.*
- Modify `docond_VEF_3D_dom2.data` datafile to have:  
*Paroi\_echange2 paroi\_contact pb Paroi\_echange1*  
→  
*Paroi\_echange2 paroi\_temperature\_imposee Champ\_Front\_Uniforme 1 50.*

# Separate the coupled problem into two new problems

## Running separately datafiles using trust

- Run docond\_VEF\_3D\_dom1.data and docond\_VEF\_3D\_dom2.data in parallel:  

```
$ trust docond_VEF_3D_dom1 2
```

```
$ trust docond_VEF_3D_dom2 2
```

The two problems must run.
- Notice that there is no coupling at all for the moment.

## Adjusting datafiles for ICoCo

- To create the ICoCo problem, just after 'dimension 3', add in docond\_VEF\_3D\_dom1.data and docond\_VEF\_3D\_dom2.data:  

```
Nom ICoCoProblemName Lire ICoCoProblemName pb
```
- Remove 'Solve pb' because the solving step will be made by ICoCo.

# Run with ICoCo

## Creation of the main.cpp file

- We have to create a new executable which will use our datafiles.
- Copy the following main.cpp file in your repository:  

```
$ file="main_docond_VEF_3D_ICoCo.cpp"  
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/$file main.cpp
```
- Open this file and look where:
  - the processors are added: search for "dom\_ids"
  - the names of the datafiles: search for "data\_file"
  - the loop to iterate on time steps: search for "while"

# Run with ICoCo

## Compiling and launching

- Create a makefile to compile your main.cpp file:  
`$ sh $project_directory/share/bin/create_Makefile 4`
- Compile the main.cpp file:  
`$ make`
- Launch calculation:  
`$ mpirun -np 4 ./couplage`
- Compare the results to the results of the coupled problem:  
`$ compare_lata pb1.lml docond_VEF_3D_dom1.lml`  
`$ compare_lata pb2.lml docond_VEF_3D_dom2.lml`
- As expected, there are differences between the results because there is no coupling here, we impose the temperature in the datafiles!

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 **Coupled problem with ICoCo**
  - TRUST: test case creation
  - ICoCo: without exchange
  - **ICoCo: one way coupling**
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# Run with ICoCo

## Adjusting datafiles

- Now, we want to send the temperature from the "Paroi\_echange2" boundary of the fluid domain to the Paroi\_echange1 boundary of the solid domain.

- Create a new directory:

```
$ cd ICoCo_exercises
```

```
$ cp -r docond_VEF_3D_ICoCo docond_VEF_3D_ICoCo_coupling1
```

```
$ cd docond_VEF_3D_ICoCo_coupling1
```

- In the docond\_VEF\_3D\_dom2.data datafile, add in the "Definition\_champs" block of the post-processings:

```
TEMPERATURE_OUT_DOM2 Interpolation {  
  localisation elem  
  domaine dom_fluide_boundaries_Paroi_echange2  
  source refChamp { Pb_Champ pb temperature }  
}
```

where "dom\_fluide\_boundaries\_Paroi\_echange2" is a predefined name for the boundary Paroi\_echange2 of the fluid's domain.

# Run with ICoCo

## Adjusting datafile

- In the docond\_VEF\_3D\_dom1.data datafile, change the boundary condition on the 'Paroi\_echange1' boundary to:  

```
Paroi_echange1 paroi_temperature_imposee ch_front_input {  
nb_comp 1 nom TEMPERATURE_IN_DOM1 probleme pb }
```
- Copy the main.cpp file for this exchange:  

```
$ file="main_docond_VEF_3D_ICoCo_coupling1.cpp"  
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/$file main.cpp
```
- Compare this main.cpp file with the previous one (use tkdiff, meld or diff):  

```
$ tkdiff main.cpp ../docond_VEF_3D_ICoCo/main.cpp
```
- You can find where the added fields in datafiles TEMPERATURE\_IN\_DOM1 and TEMPERATURE\_OUT\_DOM2 are used in a new part for exchanges.
- Notice that we use two new objects: one TrioDEC object and one TrioField object.
- Some comments are written to help you.

# Run with ICoCo

## Adjusting datafile

- Compile the main.cpp file:

```
$ make
```

- Launch calculation:

```
$ mpirun -np 4 ./couplage
```

- Compare results with those without coupling:

```
$ compare_lata docond_VEF_3D_dom1.lml
```

```
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom1.lml
```

```
$ compare_lata docond_VEF_3D_dom2.lml
```

```
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom2.lml
```

- The results on the domain dom2 are the same as this calculation made only one more postprocessing (TEMPERATURE\_OUT\_DOM2).

- But, we can see that the coupling works well because the results on the domain dom1 changes.



- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 **Coupled problem with ICoCo**
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - **ICoCo: two way coupling**
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# Run with ICoCo

## Adjusting datafile

- Two way coupling for thermal problems should use Dirichlet and Neumann's boundary conditions (using only Dirichlet boundary conditions for coupling both sides would not work).
- So we want to send the heat flux from the Paroi\_echange1 boundary of the solid domain's problem to the Paroi\_echange2 boundary of the fluid domain's problem.
- Create a new directory for this part:

```
$ cd ICoCo_exercises  
$ cp -r docond_VEF_3D_ICoCo_coupling1  
docond_VEF_3D_ICoCo_coupling2  
$ cd docond_VEF_3D_ICoCo_coupling2
```
- Inspire you from the previous part to make a Neumann boundary condition (heat flux imposed).

# Run with ICoCo

## Adjusting datafile

- In the docond\_VEF\_3D\_dom1.data datafile, add a "Definition\_champs" block in the post-processings:

```
FLUX_SURFACIQUE_OUT_DOM1 Interpolation {
  localisation elem
  domaine dom_solide_boundaries_Paroi_echange1
  source Morceau_equation {
    type operateur numero 0 option flux_surfacique_bords
    source refChamp { Pb_Champ pb temperature }
  }
}
```

where "dom\_solide\_boundaries\_Paroi\_echange1" is a predefined name for the boundary Paroi\_echange1 of the solid domain.

- In the docond\_VEF\_3D\_dom2.data datafile, switch the boundary condition on the 'Paroi\_echange2' boundary to:

```
Paroi_echange2 paroi_flux_impose ch_front_input {
  nb_comp 1 nom FLUX_SURFACIQUE_IN_DOM2 probleme pb }
```

# Run with ICoCo

## Adjusting datafile

- Modify the main.cpp file to add a new exchange:
  - Create a new TrioDEC object to do information exchange from domain dom2 to domain dom1:  
`TrioDEC dec_flux(dom2_ids, dom1_ids);`
  - Create a new TrioField object:  
`TrioField field_flux;`
  - Add code lines into the while loop to do information exchange.
- You can have a look at the main\_docond\_VEF\_3D\_ICoCo\_coupling2.cpp file for this exchange:  

```
$ file="main_docond_VEF_3D_ICoCo_coupling2.cpp"
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/$file .
```
- Compare it to the previous one using tkdiff (you can use meld, tkdiff or diff depending on which software is installed on your computer):  

```
$ tkdiff main_docond_VEF_3D_ICoCo_coupling2.cpp
../docond_VEF_3D_ICoCo_coupling1/main.cpp
```

# Run with ICoCo

## Adjusting datafile

- You can find the usage and synchronization of the new fields FLUX\_SURFACIQUE\_IN\_DOM2 and FLUX\_SURFACIQUE\_OUT\_DOM1.
- Compile your main.cpp file:  

```
$ make
```
- Launch calculation:  

```
$ mpirun -np 4 ./couplage
```
- Compare results with the first ones:  

```
$ compare_lata docond_VEF_3D_dom1.lml  
../docond_VEF_3D_ICoCo/pb1.lml  
$ compare_lata docond_VEF_3D_dom2.lml  
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom2.lml
```

# Run with ICoCo

## Adjusting datafile

- As you can see, the results are not the same because ICoCo coupling is different from coupled problem in TRUST. If this problem reaches convergence, results would be the same at least at the final time.
- For further learning, you can read the pdf report of "CouplageFluideSolide" validation form.

You can visualize this report by:

```
$ cd $project_directory/share/Validation  
$ cd Rapports_automatiques/CouplageFluideSolide  
$ Run_fiche -xpdf
```

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 Conclusions and further simulations!

# Why using TrioCFD in ICoCo?

## Why using TrioCFD?

Using TrioCFD allows you to get access to

- Turbulence models since its models are in TrioCFD since the version 1.8.0
- Access to other physics based models,
- Etc...

You can couple your code (with an ICoCo interface) with TrioCFD instead of TRUST.

The procedure is described in this exercise. However, it depends on the TrioCFD version you are using:

- if you are using your own version, you can modify it by adding ICoCo
- if you are using a network version, you can create a baltik that depends on both: TrioCFD and ICoCo

**In both cases, you should load the environment of your baltik.**



# What if you are using you own TrioCFD version?

If you want to use you own TrioCFD version?

a) Edit the project.cfg of TrioCFD:

```
> cd PathToTrioCFD  
> source env_TrioCFD.sh  
> echo "MEDICoCo : $TRUST_ROOT/Outils/ICoCo/ICoCo_src" » project.cfg
```

b) Configure and compile TrioCFD:

```
> ./configure  
> make optim debug  
> make module_optim module_debug
```

# What if you are using a network version of TrioCFD?

## If using a network version of TrioCFD?

a) Create a baltik project with dependency to TrioCFD:

```
> source env_TrioCFD.sh  
> cd PathWhereYouWantToCreateYourBaltik  
> triocfd -baltik BaltikName
```

b) Add ICoCo as a dependency of your baltik project:

```
> cd BaltikName  
> echo "MEDICoCo : $TRUST_ROOT/Outils/ICoCo/ICoCo_src" » project.cfg
```

c) Configure and compile your baltik project:

```
> ./configure  
> source env_*.sh  
> make optim debug  
> make module_optim module_debug
```

# Create a test case and run it with TrioCFD

## Copy a turbulent TrioCFD test case

```
> source $project_directory/full_env_*.sh
> trust -copy ThHyd_keps_VEF
> cp -r ThHyd_keps_VEF ThHyd_keps_VEF_ICoCo
> mv ThHyd_keps_VEF ThHyd_keps_VEF_TrioCFD
```

## Run the test case using TrioCFD executable

```
> cd ThHyd_keps_VEF_TrioCFD
```

Edit ThHyd\_keps\_VEF.data and:

- Remove lata block in the post-processings part.
- Replace lml by lata in the same block
- add "*format lata*" in the new lata block after the "*Definition\_champs*" block

```
> trust ThHyd_keps_VEF.data
```

```
> cd ..
```

Now you have reference lata results given by TrioCFD executable. We will compare it with the lata results issued from the run using ICoCo.

# Prepare the test for ICoCo run

Prepare the testcase for ICoCo run without exchange

```
> cd ThHyd_keps_VEF_ICoCo
```

Edit ThHyd\_keps\_VEF.data and:

- Remove the lata block in the post-processings part
- replace lml by lata
- add "*format lata*" in the new lata block after the "*Definition\_champs*" block
- Add the following line after the "*dimension*" definition:  
Nom ICoCoProblemName Lire ICoCoProblemName pb
- Comment the "*solve pb*" instruction at the end of the datafile.

```
> folder="$TRUST_ROOT/doc/TRUST/exercices/ICoCo"
```

```
> cp $folder/main_ThHyd_keps_VEF_ICoCo.cpp main.cpp
```

## Run the test case with ICoCo

Run the test case using ICoCo without any exchange

```
> sh $MEDICoCo_project_directory/share/bin/create_Makefile  
> make  
> ./couplage
```

If you get, during the make step, an error similar to:

**g++: error:**

**\$project\_directory/build/src/exec\_opt/\_TRUSTModule\_opt.so: No such file or directory**

be sure that you forget a step during the compilation of your baltik (or TrioCFD) project. You can do:

```
> cd $project_directory  
> make module_optim module_debug
```

and then come back to the the folder where you have your ICoCo problem and re-run:

```
> make  
> ./couplage
```

# Check results

Compare TrioCFD results with ICoCo results

```
> cd ThHyd_keps_VEF_ICoCo  
> compare_lata ThHyd_keps_VEF.lata  
../ThHyd_keps_VEF_TrioCFD/ThHyd_keps_VEF.lata  
Results are identical!!!
```

- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python**
- 7 Conclusions and further simulations!

## Using ICoCo API through Python

In addition to the MEDCoupling python API, it is possible since TRUST-1.8.3 to:

- use ICoCo API from python scripts
- exploit multiple processors from python thanks to mpi4py package

These new features can be used by loading a specific environment file:

```
> source $TRUST_ROOT/env_for_python.sh
```

```
> python
```

```
> import trusticoco as ti
```

```
> import medcoupling as mc
```

```
> from mpi4py import MPI
```

If you are interested in such coupling, you can copy the following test case from TRUST tests database:

```
> source $TRUST_ROOT/env_for_python.sh
```

```
> trust -copy docond_VEF_3D_ICoCo_py
```

and have a look at its prepare script.

If you are interested in ICoCo API through Python with baltiks, you can visit:

<https://github.com/cea-trust-platform/icoco-swig-baltik>



- 1 Introduction to code coupling and ICoCo
- 2 TRUST and ICoCo initialization
  - Initialization of TRUST environment
  - Initialization of ICoCo environment
- 3 Basic test case
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: first input
  - ICoCo: on 2 processes
- 4 Coupled problem with ICoCo
  - TRUST: test case creation
  - ICoCo: without exchange
  - ICoCo: one way coupling
  - ICoCo: two way coupling
- 5 ICoCo and TrioCFD
- 6 Using ICoCo API through Python
- 7 **Conclusions and further simulations!**

# Conclusions and further simulations!

## Conclusion

This tutorial gave you an idea on how to run TRUST test cases with:

- no exchange by ICoCo
- imposing values from ICoCo as boundary conditions in TRUST datafile
- exchanging values between two datafiles where coupling is realised with ICoCo

We have seen also how to run TrioCFD cases with ICoCo without any exchange, but we believe you are able to perform exchange in ICoCo with TrioCFD datafiles easily.

Finally, we recall that it is possible for you since TRUST 1.8.3 to run ICoCo API from Python.