

TRUST Baltik Project Tutorial V1.9.6

CEA Saclay

Support team: trust@cea.fr

May 26, 2025

- 1 TRUST initialization
- 2 Eclipse initialization
- 3 Create a Baltik project
- 4 Modify the cpp sources
- 5 Parallel exercise
- 6 Validation form and test cases
- 7 Code coverage exercise
- 8 Tools
- 9 For more

- 1 **TRUST initialization**
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

TRUST initialization

TRUST commands

- Load TRUST environment:
 - On CEA Saclay PCs, TRUST versions are available with:
source /home/trust_trio-public/env_TRUST-1.9.5.sh
 - On your own computer, download and install the latest version of TRUST in your local folder \$MyPathToTRUSTversion (unless this was done), then write on the terminal:
source \$MyPathToTRUSTversion/env_TRUST.sh

Ensure that the configuration is ok and locate the sources:

```
$ echo $TRUST_ROOT
```

- Now, copy a TRUST test case that we will need later:

```
$ mkdir -p Formation_TRUST/yourname
```

```
$ cd Formation_TRUST/yourname
```

```
$ trust -copy upwind
```

```
$ cd upwind
```

Replace "format lml" by "format lata" in the data file

- 1 TRUST initialization
- 2 **Eclipse initialization**
 - **Download eclipse**
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Download Eclipse

Download Eclipse

- Visit the Eclipse Foundation website:
<http://www.eclipse.org/downloads/eclipse-packages/>
- In "More Downloads", select version **Eclipse 2022-09 (4.25)**.
- Select **Eclipse IDE for C/C++ Developers** → **Linux 64-bits**
- Download the **eclipse-cpp-2022-09-R-linux-gtk-x86_64.tar.gz** package in your directory `Formation_TRUST/yourname`

Untar the downloaded archive

```
$ cd Formation_TRUST/yourname
$ tar xfz eclipse-*.tar.gz
$ cd eclipse
```

- 1 TRUST initialization
- 2 **Eclipse initialization**
 - Download eclipse
 - **Create TRUST platform project under Eclipse**
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Create a TRUST platform project under Eclipse (I)

Launch Eclipse

```
$ mkdir -p Formation_TRUST/yourname/workspace  
$ cd Formation_TRUST/yourname/eclipse  
$ ./eclipse &
```

- Workspace: Browse the directory Formation_TRUST/yourname/workspace
- Welcome : close x button

Create the project

- Create a preconfigured TRUST project:
\$ cd Formation_TRUST/yourname
\$ trust -eclipse-trust
- Then, follow the instructions displayed on the terminal to import TRUST sources.

Create a TRUST platform project under Eclipse (II)

Configure the project and launch a computation

- From the "Project Explorer" tab, right click on your TRUST project → "Debug As" → "Debug Configurations..."
⇒ Click on the triangle on the left of "C/C++ Application" → Select the debug configuration already created with trust -eclipse-trust
 - The "Main" tab tells Eclipse which binary will be used:
⇒ Project: your project's name
⇒ "C/C++ Application": points to the TRUST \$exec_debug
 - The "Arguments" tab tells Eclipse which datafile to run:
⇒ "Program arguments" → specifies datafile's name (here upwind)
⇒ "Working directory" → contains path to datafile
- ⇒ "Debug" : your datafile will be run with the specified executable

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 **Create a Baltik project**
 - **Creation of a Baltik project**
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Creation of a Baltik project

Create an empty Baltik

- Create a directory for your project:
`$ cd Formation_TRUST/yourname`
- Create your project from a basic project template using TRUST commands:
`$ trust -baltik my_project`
`$ cd my_project`
`$ ls -l`

You can see that you have now:

- three directories: share, src and tests, and
- one "project.cfg" file.
- one "README.BALTIK" file.
- one "configure" script.

Add sources to your Baltik

- Copy the following TRUST .cpp file into your baltik project:
`$ mkdir -p src/Trust_fixes`
`$ cp $TRUST_ROOT/src/MAIN/mon_main.cpp src/Trust_fixes/`

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 **Create a Baltik project**
 - Creation of a Baltik project
 - **Creation of your git repository**
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

Create your git repository

Git commands

You will now create a git repository to manage your developments.

- Initialize an empty git repository:
`$ git init`
- Display your working tree status:
`$ git status`
- You can see 3 file and src directory on the "untracked" files section. It means that they are not yet followed by the git repository.
- Add src and project.cfg to your git repository in order to prepare a commit:
`$ git add src project.cfg`
- Now, you can commit your files to add it to your git repository:
`$ git commit -m "Initial commit"`
Remark: If you are not able to commit files, you should first configure your username and email in git with :
`git config --global user.name "Your Name"`
`git config --global user.email you@example.com`

Create your git repository

Git commands

- Display your working tree status:
`$ git status`
Only README.BALTIK and configure script (automatically generated) are not added to your git repository.
- Display the list of commits:
`$ git log`

Baltik commands

- Edit your project file "project.cfg" to specify name, author and executable.
- Then configure your project:
`$ baltik_build_configure -execute`
this command launches both scripts: the "baltik_build_configure" and "configure".

Create your git repository

Git commands

- Check the status of your git repository with the "--ignored" option to see the status of all files:
`$ git status`
- You can see that
 - "project.cfg" has been modified.
 - there are new untracked files: these files are not on the git repository
- To see only the changes on the git repository files:
`$ git status -uno`
- Track changes via gitk (GUI interface of Git):
`$ gitk &`
You can see information about your first commit and actual untracked changes.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 **Create a Baltik project**
 - Creation of a Baltik project
 - Creation of your git repository
 - **Builds**
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

Make a basic build

- To make a basic build:
`$ cd Formation_TRUST/yourname/my_project`
- Configure your project:
`$./configure`
- Build your project in different modes:
 - Build an optimized (-O3 option) version:
`$ make optim`
 - Build a debug (-g -O0 option with asserts) version:
`$ make debug`
- Initialize your baltik project environment:
`$ source env_my_project.sh`
- Check that executables are available:
`$ ls $exec`
`$ ls $exec_opt`
`$ ls $exec_debug`

Other builds

- List other options available for the make command:
`$ make help`
- Build an :
 - semi-optimized -O3 option and assert enabled mode:
`$ make semi_optim`
`$ ls $exec_semi_optim`
 - optimized binary for profiling (option -pg -O3):
`$ make prof`
`$ ls $exec_pg`
 - optimized binary for test coverage (option -gcov -O3):
`$ make gcov`
`$ ls $exec_gcov`

Notice that TRUST optimized binary for profiling or a TRUST optimized binary for test coverage must exist in order to be able to compile your baltik's profiling or test coverage executable.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 **Create a Baltik project**
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - **Using Eclipse**
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

Create a basic BALTIK project without dependency (I)

Initialize baltik environnement

```
$ source env_my_project.sh
```

Launch Eclipse

```
$ cd Formation_TRUST/yourname/eclipse  
$ ./eclipse &
```

Create the project

```
$ trust -eclipse-baltik
```

then follow the instructions displayed on the terminal.

Create a basic BALTIK project without dependency (II)

Launch a computation

- From the "Project Explorer" tab, right click MY_BALTIK → "Debug As" → "Debug Configurations..."
⇒ C/C++ Application → Select the configuration containing your baltik's name
 - In the "Main" tab:
 - ⇒ Project: MY_BALTIK
 - ⇒ C/C++ Application: contains path to \$exec_debug
 - ⇒ "Apply"
 - In the "Arguments" tab:
 - ⇒ Program arguments: contains datafile's name (upwind)
 - ⇒ Working directory: path to datafile's directory
 - ⇒ "Apply"
- ⇒ Debug

Useful shortcuts in sources

Shortcuts

- Open a cpp file from Project Explorer tab:
Double click on TRUST → Kernel → Framework → Probleme_base.cpp
or : Ctrl+shift+R then type Probleme_base.cpp
- In the cpp file: Right click on method "postraiter()"
 - ⇒ F3: Opens Declaration
 - ⇒ F4: Open Type Hierarchy
 - ⇒ Ctrl+Alt+H: Open Call Hierarchy
 - ⇒ "Ctrl+PageUp" and "Ctrl+PageDown": Move from a tab to another
- you can also:
 - ⇒ search files by name using: "Ctrl+shift+R"
 - ⇒ search attributes/methods/functions/... using: "Ctrl+shift+T"

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 **Modify the cpp sources**
 - **Create a new cpp class**
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Create a new cpp class

Baltik commands

- Create a new folder for your own classes:
`$ mkdir -p $project_directory/src/my_module`
`$ cd $project_directory/src/my_module`
- Create your first class "my_first_class" with template:
`$ baltik_gen_class my_first_class`

Git commands

- Display the status of your repository:
`$ git status .`
- Add your new class to your git repository to follow your modifications:
`$ git add my_first_class.*`
`$ git commit -m "Add my_first_class src"`

Create a new cpp class

Baltik commands

- Have a look at the 2 files `my_first_class.h|cpp`.
- **Important remark:** Each time a source file is added to the project, you need to reconfigure your project to take new files into account when building the executable:

```
$ cd $project_directory  
$ ./configure
```
- Build your project with Eclipse or in the terminal.
- Edit the 2 files with `gedit|vim` or use Eclipse.

Eclipse

- Edit the 2 files with Eclipse.
- For Eclipse use, you have to update your project to see your new files:
 - "Index/Rebuild" from "my_project" of "Project Explorer"
 - Click on "►" button of "my_project" in the "Project Explorer"

Create a new cpp class

Baltik commands

- We want to change the inheritance of the class in order that it inherits from "Interprete" class instead of "Objet_U".
"Interprete" class is the base class of all the keywords doing tasks when read from the datafile, example (read, associate, solve, ...).
So:
 - add an "#include <Interprete.h>" in my_first_class.h,
 - replace "Objet_U" to "Interprete" in the .h and .cpp files,
 - Interprete class is an abstract class, it contains a pure virtual method which should be implemented!
 - rebuild your application and an error will occur!
- Look at the "Interprete" class:
 - in Eclipse: highlight the string "Interprete" and push the F3 button of your keyboard to open the declaration file of this class

Create a new cpp class

Baltik commands

- Look at "interpreter()" method
This method is called each time a keyword is read from the datafile (eg: "Read_file dom dom.geom", "Solve pb",...).
- Define the public method "interpreter(Entree&)" in the include file and implement it (just print a message with "Cerr" like "- My first keyword!") into the cpp file.
"Entree" is a TRUST class to read an input stream (from a file for example):
Entree& interpreter(Entree&) override;
- Rebuild your project and fix errors until the debug binary of your project is generated

Test your new class

- Copy a test case to the build folder of your Baltik project:
\$ cd \$project_directory/build/
\$ trust -copy Cx
ERROR...

Create a new cpp class

Test your new class

- The error occurs because this test case is not in your baltik but in TRUST project. To be able to copy it, you have to load the full environment (TRUST+your baltik).

```
$ source ../full_env_my_project.sh
```

```
$ trust -copy Cx
```

```
$ cd Cx
```
- Edit the data file:

```
$ gedit Cx.data &
```

Add keywords "my_first_class" and "End" after the line where the problem is discretized.
- Run this datafile with your baltik binary and check that this new keyword is recognized (see next slide).

Create a new cpp class

With Eclipse:

- In the project explorer, right click on "my_project" and select "Debug As/Debug configurations..."
- In "Main" tab, check "Disable auto build" then click on "Apply"
- In "Arguments" tab, fill "Program arguments:" with "Cx"
- "Working directory:" Copy the path to datafile matching
\$project_directory/build/Cx
- "Apply" and "Debug"
- Click on "Yes" to switch to the debug view
- Click on "Resume" button (or F8) to run the calculation until the end

On a terminal, you can only run:

```
$ cd $project_directory/build/Cx/  
$ exec=$exec_debug trust Cx
```

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 **Modify the cpp sources**
 - Create a new cpp class
 - **Modify your cpp class(Part 1)**
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Modify your cpp class (Part 1)

- Now, instead of reading:

```
my_new_keyword
```

we want you to read this syntax from the datafile:

```
my_new_keyword { domain dom_name option 0 }
```

- The recommended way is to use **Param** objects.
- As an example, see the next slide to see how to use **Param** to read the syntax:

```
my_new_keyword { problem pb_name option 1 }
```

Modify your cpp class (Part 1)

Example of Param use

```
#include <Param.h>
Entree& Class::interpreter(Entree& is)
{
    int opt = 0;
    Nom pb_name;
    Cerr << "Reading parameters of A from a stream (cin or file)" << finl;
    Param param(que_suis_je());
    // Register parameters to be read:
    param.ajouter("option",&opt);
    param.ajouter("problem",&pb_name,Param::REQUIRED);
    // Read now the parameters from the stream is and produces an error
    // if unknown keyword is read or if braces are not found at the
    // beginning and the end:
    param.lire_avec_accolades_depuis(is);
    ...
    return is;
}
```


Modify your cpp class (Part 1)

- As said before, we want to read this syntax from the data:

```
my_first_class { domain dom option 0 } # dom is domain name #
```

- here is how to do:

```
Entree& Class::interpreter(Entree& is)
{
    Nom domain_name;
    ... // read params from the datafile
    // Retrieve the domain object name given by the user, and find
    // back the actual instance in memory:
    Objet_U& obj = Interpreter::objet(domain_name);
    // Sanity check: if read domain name does not correspond to a
    domain, exit
    if(!sub_type(Domaine, obj))
        Process::exit("Object passed to 'Class' is of wrong type!");
    // Fill the local reference to domain:
    Domaine& domaine_ref_ = ref_cast(Domaine, obj);
    ...
}
```

Modify your cpp class (Part 1)

- Fix your implementation of "interpreter()" method (add missing includes)
- Add a print at the end of the method "interpreter(Entree&)" and find how to print the domain name:

```
Cerr << "Option number " << option_number << " has been  
read on the domain named " << ??? << finl;
```

Modify your cpp class (Part 1)

- With Eclipse:
 - Build/fix/re-build your project:
→ "Project" and "Build project"
 - Run the test case:
→ "Run" and "Debug"
- Or in a terminal:
 - Build/fix/re-build your project:

```
$ cd $project_directory  
$ make debug
```
 - Run the test case:

```
$ cd $project_directory/build/Cx/  
$ export exec=$exec_debug  
$ trust Cx
```

In this case, TRUST runs with `exec_debug`.

Modify your cpp class (Part 2)

Display information about domain boundaries

- Edit the "my_first_class.cpp" file and add into the "interpreter()" method a loop on the boundaries.

Look for help inside the "Domaine", "Bord", "Frontiere" classes in Eclipse or in the HTML documentation to access to the:

- Number of boundaries (**nb_bords()** method)
- Boundaries (**bord(int)** method)
- Name of the boundaries (**le_nom()** method)
- Number of faces of each boundary (**nb_faces()** method)

Print these information with something like:

```
Cerr << "The boundary named " << ??? << " has " << ??? << "
faces." << finl;
```

Modify your cpp class (Part 2)

Compute the sum of the control volumes of a domain discretized in VEF

- Information about control volumes is in the "Domaine_VF" class (discretized domain) which can't be accessed from the domain, but only from the problem.

So, you need to read another parameter from your datafile:

```
my_first_class { domain dom option 0 problem pb }
```

- Add the read of a new parameter problem into "my_first_class.cpp" file (see the example at the beginning of this part).
- Remember the "equation" or "problem" UML diagram of the presentation's slides.
- Look for help inside the "Domaine_VF", "Probleme_base" and "Domaine_dis_base" in Eclipse or the HTML documentation to access to the:
 - discretized domain (**domaine_dis()** method)
 - control volumes (**volumes_entrelaces()** method)

Modify your cpp class (Part 2)

- You will need to cast the discretized domain returned by the **domaine_dis()** method into a "Domaine_VF" object.
- Print the size of the control volumes array with something like:

```
Cerr << control_volumes.size() << finl;
```

Where control_volumes is a **DoubleVect** returned by the **Domaine_VF::volumes_entrelaces()** method.

- Now, compute and print the sum of the control volumes with a "for" loop.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 **Modify the cpp sources**
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - **Add XD tags (keyword documentation)**
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Create automated documentation

- We want now to add XD tags to create the automated documentation of your new code.
- First, we have to create this documentation for the first time.

```
$ cd $project_directory  
$ make gui
```
- Open the documentation file:

```
$ evince $project_directory/build/xdata/XTriou/doc.pdf &
```
- Now will add XD tags (keyword documentation) in cpp files.
- For this open the help of the TRAD_2 syntaxe:

```
$ gedit $project_directory/build/xdata/XTriou/doc_TRAD_2 &
```


Create automated documentation

- Add a first tag (in comments) into your cpp file just after the opening brace of the 'interpreter_()' method:

```
// XD english_class_name base_class_name TRUST_class_name  
mode description
```

where "english_class_name" and "TRUST_class_name" can be "my_first_class".

- The "base_class_name" is the name of the section in which will appear the information of your new class in the 'doc.pdf' file.
- The "mode" is to choose with the help of the doc_TRAD_2 file. Here set it to "-3".

Create automated documentation

- Then add at the end of the lines of type "param.ajouter...", an XD comment like:

```
param.ajouter(...); // XD_ADD_P type description
```

where "type" can be (cf 'doc_TRAD_2' file): 'int', 'floattant', 'chaine', 'rien'...

- rebuild the documentation:
\$ make gui
- Check that the documentation of your new class is in the new doc:
\$ evince \$project_directory/build/xdata/XTriou/doc.pdf &
- To check that the GUI is validated:
\$ make check_gui
- Notice that you must have XD commands in all your cpp classes.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 **Modify the cpp sources**
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - **Adding prints**
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

Adding prints

- Edit the "\$project_directory/src/Trust_fixes/mon_main.cpp" file of your baltik project using text editor or Eclipse.
- Add these lines after "Process::imprimer_ram_totale(1);" :
std::cout << "Hello World to cout." << std::endl;
std::cerr << "Hello World to cerr." << std::endl;
Cout << "Hello World to Cout." << finl;
Cerr << "Hello World to Cerr." << finl;
Process::Journal() << "Hello World to Journal." << finl;

in a terminal: Rebuild the code

```
$ cd $project_directory  
$ make debug optim
```

Adding prints

- Create an empty data file:

```
$ mkdir -p $project_directory/build/hello  
$ cd $project_directory/build/hello  
$ touch hello.data
```

- Run the code

- sequentially:

```
$ trust hello
```

- in parallel:

```
$ trust hello 4
```

and see the differences.

- "Cout" is equivalent to "std::cout" on the master process only. Use this output for infos about the physics (convergence, fluxes,...).
 - "Cerr" is equivalent to "std::cerr" on the master process only. Use this output for warning/errors only.
 - "finl" is equivalent to "std::endl" + "flush()" on the master process.
 - "Journal()" prints to "datafile_000n.log" files. Use this output during parallel development to print plumbing infos which would be hidden during production runs.

Adding prints

- During a parallel run, the "Journal()" output can be disabled.

To verify this, first clean your folder:

```
$ ls *.log
```

```
$ trust -clean
```

and run computation with -journal=0 option

```
$ trust hello 4 -journal=0
```

```
$ ls *.log
```

- Other options are available. To get it, run:

```
$ trust hello.data -help_trust
```

Adding prints

Printing into a file

- Now, we will print the control volumes sum into a file for test case Cx.
- We want to write in a file with name similar to:
DataFileName_result.txt
where "DataFileName" is the name of the data file (eg: Cx).
- For that, you will:
 - create an object of the class **Nom** and fill it by collecting the datafile's name using **Objet_U::nom_du_cas()** method.
 - complete the datafile's name with the string "_result.txt" thanks to the "operator" += method of the class **Nom**.
 - create the output file with the **SFichier** class and print the sum into it.
- Compile your project and run Cx datafile:
\$ cd \$project_directory/build/Cx/
\$ exec=\$exec_debug trust Cx
- Then open the "Cx_result.txt" file.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 **Parallel exercise**
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Parallel exercise

Part 1

- Run your test case Cx in parallel mode:

```
$ cd $project_directory/build/Cx/  
$ trust -partition Cx 2 # Partition in 2 subdomains  
$ trust PAR_Cx 2 # 2 processes used
```
- Compare the files: Cx_result.txt, PAR_Cx_result.txt.
Differences come from the fact that the 2 processors write into the file one after the other one. So the final content will be the value calculated on the last processor which will access to the file.
- You can try to launch one more time the calculation, the result may differ.
- To have the entire sum, you can apply the **mp_sum()** method on the sum obtained and add the print in the .txt file.
- Compare it to the sum obtained in the sequential run.
- It is better but we counted several times faces that belongs to the joint and to the virtual zones.

Parallel exercise

Part 1

- To parallelize the algorithm, rewrite it with the help of the **mp_somme_vect(DoubleVect&)** method.
- Add this print in the .txt file.
- You should find the same value for the sequential and parallel calculation.

Part 2 (Optional)

- Create a "verifie" script to check the resulting value (sequential then parallel).
- Add a call to "compare_sonde" in your "verifie" script...

Part 3

- To validate parallelization in TRUST, you can use the command "compare_lata":
\$ ls *lata
\$ compare_lata Cx.lata PAR_Cx.lata

Parallel exercise

Part 3

- You can see that there is no differences and the maximal relative error encountered is about 4.e-12.
- Performances \$ ls *TU
\$ meld Cx.TU PAR_Cx.TU &
\$ meld Cx_csv.TU PAR_Cx_csv.TU &

Part 4 Debug

- Copy a debug test case:
\$ cd \$project_directory/build
\$ trust -copy Debug_VEF
\$ cd Debug_VEF
- Open the Debug_VEF.data file and search the "Debug" command.
- Sequential run:
\$ trust Debug_VEF
- You get "seq" and "faces" files.

Parallel exercise

Part 4 Debug

- Partitioning step and creation of the parallel data file:
`$ trust -partition Debug_VEF 2`
- Verify the parallel data file, you must have now "Debug pb seq faces 1.e-6 1".
- Run in parallel:
`$ trust PAR_Debug_VEF 2`
- You get debug*.log and DEBOG files.
- If a value of an array differs between the two calculations and the difference is greater than 1.e-6 then "ERROR" message appears in the log files else we will get "OK" (cf debug.log).
- Add a debug instruction in your file mon_main.cpp located in
`$project_directory/Trust_fixes`, after the "Hello world" prints put:
`double var = 2.5;`
`Debug::verifier("- Debug test message",var);`
- Do not forget to add the `"#include <Debug.h>"`!

Parallel exercise

Part 4 Debog

- Then compile and do the sequential run.
- You can see a first message.
- Then do the parallel run and check the debug.log file.
- Be careful the debug instruction in the data file must be between the "Discretize" and "Read pb" lines.
- For more information:
 - \$ `trust -doc &`
 - Open the TRUST Generic Guide
 - Click onto the TRUST Reference Manual
 - Search for "Debog" keyword.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 **Validation form and test cases**
 - **"Run_fiche"**
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

New share/Validation/Rapports_automatiques

Baltik commands

- Go to the directory where to create your notebook:
\$ cd \$project_directory
\$ cd share/Validation/Rapports_automatiques
- Create a new directory for your new validation form:
\$ mkdir -p upwind/src
- Add the needed files (data file, mesh & .ipynb file):
\$ cp Formation_TRUST/yourname/upwind/upwind.data upwind/src
\$ cp Formation_TRUST/yourname/upwind/upwind.geo upwind/src
- Create a new Jupyter validation form:
\$ cd upwind
\$ trust -jupyter
- Now you have a upwind.ipynb file (i.e. a new Jupyter notebook).

New share/Validation/Rapports_automatiques

Git commands

- Add it to your git repository:
\$ git add upwind
\$ git commit -m "New validation notebook"

Baltik commands

- Run this Jupyter notebook:
\$ cd upwind/
\$ Run_fiche
- Build directly a PDF report from the notebook:
\$ Run_fiche -export_pdf
- Open the pdf report:
\$ evince build/rapport.pdf &

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 **Validation form and test cases**
 - "Run_fiche"
 - **Validation test case**
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

New tests/Reference/Validation

Baltik commands

- Create automatically the non-regression test case:

```
$ cd $project_directory
```

```
$ make check_optim
```

Creation of upwind_jdd1

Creation of upwind_jdd1/lien_fiche_validation

Extracting test case (upwind.data) ...End.

Creation of the file upwind_jdd1.lml.gz

...

- → You can see in the report table that PAR_upwind_jdd1 has crashed:
"CORE" message.

New tests/Reference/Validation

Git commands

- Lets check the git status before solving this problem:
`$ git status -uno`
- A new test case based on your validation form has been created in the directory:
`$project_directory/tests/Reference/Validation/upwind_jdd1`

Baltik commands

- Now we want to correct the error, so copy the test case:
`$ cd $project_directory/build`
`$ trust -copy upwind_jdd1`
ERROR...
- We have to re-run the configure script to take into account the new test case:
`$ cd $project_directory`
`$./configure`
`$ cd build`
`$ trust -copy upwind_jdd1`

New tests/Reference/Validation

Baltik commands

- Now we will analyse the error:

```
$ cd upwind_jdd1  
$ trust -partition upwind_jdd1  
$ trust PAR_upwind_jdd1 2
```
- Correct the data file PAR_upwind_jdd1.data and re-run it.
- If it's ok, update the data file in
`$project_directory/share/Validation/Rapports_automatiques/upwind/src`
("Scatter ../upwind/DOM.Zones dom" → "Scatter DOM.Zones dom")
- To Relaunch the last test cases which do not run:

```
$ cd $project_directory  
$ make check_last_pb_optim  
Changement du jeu de donnees...  
suite a une modification d'un jeu de donnees de la fiche de validation associee.  
...  
Successful tests cases :1/1
```

New tests/Reference/Validation

Git commands

- Add this non-regression test in configuration:
\$ git status -uno
\$ git add
tests/Reference/Validation/upwind_jdd1/upwind_jdd1.data
- Commit the modifications on your git repository:
\$ git commit -m "New reference test"
\$ git log

New tests/Reference/Validation

Baltik commands

- To run all the non regression tests with a optimized binary:
`$ make check_all_optim`
- To run all the non regression tests with a debug binary:
`$ make check_all_debug`
- To create an archive to share your work:
`$ make distrib`
`$ ls`
- You have now an archive in tar.gz format of your baltik project.

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 **Code coverage exercise**
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Code coverage exercise

- We want to run test cases using rational Runge-Kutta scheme of ordre 2.
 - For this go to the Doxygen documentation of RRK2 class to see the methods of this class.
 - Use the "trust -check function|class|class::method" command to find and launch tests cases.
 - For example:

```
$ trust -check RRK2::RRK2
```


- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 **Tools**
 - **Debug with GDB**
 - Find memory bugs with valgrind
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Debug with GDB

With Eclipse:

Run a test case with GDB:

- "Debug As" and "Debug configurations..." from "my_project"
- in "Arguments", "Program arguments:" upwind
- "Working directory:" Formation_TRUST/yourname/upwind/
- "Apply" and "Debug"

For more information about GDB commands, refer to the help menu.

Or in a terminal:

- Run a test case with GDB:

```
$ cd Formation_TRUST/yourname/upwind/  
$ exec=$exec_debug trust -gdb upwind
```
- You are now in GDB.
- Add a breakpoint and stop into the SSOR preconditionner:

```
(gdb) break SSOR::ssor
```

Debug with GDB

- Run the test case:
(gdb) run upwind
- Have a look at the stack
(gdb) where
- Go to the next instruction:
(gdb) n
- Print an array:
(gdb) print tab1
- Or print `matrice.tab1_` if "optimized out" message printed:
(gdb) print tab1[10]
- Print only a value of an array:
(gdb) dumpint tab1 # Dump the array
(gdb) print tab1.size_array() # Array size
(gdb) up
(gdb) list 100

Debug with GDB

- Print lines after the 100th line:

```
(gdb) print matrice
```

```
(gdb) print matrice.que_suis_je() # Kind of matrix ?
```

```
(gdb) print matrice.que_suis_je().nom_ # Kind of matrix ?
```

```
(gdb) up 6 # Move up 6 levels
```

```
(gdb) list 865
```

- Print others variables:

```
(gdb) # Pressure field
```

```
(gdb) print la_pression->que_suis_je().nom_
```

```
(gdb) # Pressure values (DoubleTab)
```

```
(gdb) print la_pression->valeurs()
```

```
(gdb) # DoubleTab dimension
```

```
(gdb) print la_pression->valeurs().nb_dim()
```

```
(gdb) # Dump the field values
```

```
(gdb) pttab la_pression->valeurs()
```

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 **Tools**
 - Debug with GDB
 - **Find memory bugs with valgrind**
- 9 For more
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

Find memory bugs with valgrind

- Run a test case with Valgrind:

```
$ cd $project_directory  
$ source env_my_project.sh  
$ cd build/Cx/  
$ VALGRIND=1 trust Cx
```
- The Valgrind messages appear on the screen with the beginning of each line the same number. For example:

```
$ ==26645== ...
```
- The last line indicates if errors have occurred. An example with 0 error:

```
$ ==26645== ERROR SUMMARY: 0 errors from 0 contexts  
(suppressed: 0 from 0)
```
- Now we will modify the sources in your baltik project to generate a Valgrind error on the Cx test case.

Find memory bugs with valgrind

- Edit the "my_first_class.cpp" file and remove initialization the following line (before computing the sum of control volumes):
double sum = 0;
- Edit "my_first_class.h" and add in protected:
double sum; // do not initialize it to 0
- Rebuild your project and run the test case:

```
$ cd $project_directory
```

```
$ make debug optim
```

```
$ cd build/Cx/
```

```
$ exec=$exec_opt trust Cx # in mode optim, no error!
```

```
$ exec=$exec_debug trust Cx # in mode debug, no error too
```

```
$ VALGRIND=1 exec=$exec_opt trust Cx # in mode valgrind
```

Valgrind detects that there is a "Conditional jump or move depends on uninitialised value(s)"

On the other hand, in this case, there are errors.

```
$ ==28400== ERROR SUMMARY: 772 errors from 114 contexts
```

NB: In valgrind+debug mode, you'll get the file+line number

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 **For more**
- 10 Appendix
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - TPP files in Eclipse

For more

- You can find the commented solution of the exercise:
`$ cd $TRUST_ROOT/doc/TRUST/exercices/my_first_class`
- You can practice on a tutorial:
`$ cd $TRUST_ROOT/doc/TRUST/exercices/
$ evince equation_convection_diffusion/rapport.pdf &`

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 **Appendix**
 - **Configuring TRUST project in Eclipse**
 - Configuring BALTIK project in Eclipse
 - TPP files in Eclipse

Create a TRUST platform project under Eclipse (I)

On a terminal

Load TRUST environment and copy "upwind" test case as described on p.4:

```
$ echo $TRUST_ROOT/src  
$ echo $exec_debug
```

Launch Eclipse

```
$ mkdir -p Formation_TRUST/yourname/workspace  
$ cd Formation_TRUST/yourname/eclipse  
$ ./eclipse &
```

- Workspace: Browse the directory Formation_TRUST/yourname/workspace
- Welcome : close x button

Create a TRUST platform project under Eclipse (II)

Create the project

- File → New → C/C++ Project → C++ Managed Build
 - ⇒ Project name: TRUST-X.Y.Z (e.g.: TRUST-1.9.5)
 - ⇒ Project type: "Executable" → "Empty Project"
 - ⇒ Toolchains: "Linux GCC"
 - ⇒ Finish

Import TRUST source files into the project

- From the "Project Explorer" tab, right click on TRUST-X-Y-Z → "Import..."
 - ⇒ General → File System → Next
 - ⇒ From directory: copy the string matching \$TRUST_ROOT/src/
 - ⇒ Check "Select All"
 - ⇒ Into folder: TRUST-X.Y.Z
 - ⇒ Finish
 - ⇒ Wait to have 100% at the bottom right corner of the window (C/C++ indexer).

Create a TRUST platform project under Eclipse (III)

Configure the project and launch a computation

- From the "Project Explorer" tab, right click on TRUST-X.Y.Z → Properties
⇒ Builders: uncheck "CDT Builder" → OK → apply and close
 - From the "Project Explorer" tab, right click on TRUST-X.Y.Z → "Debug As" → "Debug Configurations..."
⇒ Right click on "C/C++ Application" → New configuration
 - In the "Main" tab (tell Eclipse which binary will be used):
⇒ Project: TRUST-X.Y.Z
⇒ "C/C++ Application": copy the string matching \$exec_debug
⇒ "Apply"
 - In the "Arguments" tab (tell Eclipse which datafile to run):
⇒ "Program arguments" → specify datafile's name (here upwind)
⇒ "Working directory" → uncheck "Use default" and type path to datafile
⇒ "Apply"
- ⇒ "Debug": your datafile will be run with the specified executable

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 **Appendix**
 - Configuring TRUST project in Eclipse
 - **Configuring BALTIC project in Eclipse**
 - TPP files in Eclipse

Create a basic BALTIC project without dependency (I)

Initialize baltik environnement

```
$ source env_my_project.sh  
$ echo $project_directory/src
```

Launch Eclipse

```
$ cd Formation_TRUST/yourname/eclipse  
$ ./eclipse &
```

Create the project

- File → New → Project → C/C++ → "Makefile Project with Existing Code"
 - ⇒ Project name: MY_BALTIC
 - ⇒ Existing Code Location: copy string matching \$project_directory/src
 - ⇒ Toolchain for Indexer Settings: "Linux GCC"
 - ⇒ Finish
 - ⇒ Wait to have 100% at the bottom right corner of the window (C/C++ indexer).

Create a basic BALTIK project without dependency (II)

Configure the BALTIK project and link it with TRUST

- From the "Project Explorer" tab, right click on MY_BALTIK → Properties
 - ⇒ Builders: check "CDT Builder"
 - ⇒ C/C++ Build :
 - Builder Settings: Build directory: `${workspace_loc:/MY_BALTIK}/../` or copy the string matching `$project_directory/`
 - Behavior: check "Build (Incremental build)": debug optim (instead of all)
 - ⇒ Project References: check your TRUST project → Apply and Close

Build the BALTIK project

From the "Project Explorer" tab, right click MY_BALTIK → Index → Rebuild
⇒ Wait to have 100% at the bottom right corner of the window (C/C++ indexer).
Right click MY_BALTIK → Build Project

Create a basic BALTIK project without dependency (III)

Launch a computation

- From the "Project Explorer" tab, right click MY_BALTIK → "Debug As" → "Debug Configurations..."
 - ⇒ C/C++ Application → New configuration
 - In the "Main" tab:
 - ⇒ Project: MY_BALTIK
 - ⇒ C/C++ Application: `${workspace_loc:/MY_BALTIK}/../my_project` or copy the string matching `$exec_debug`
 - ⇒ "Apply"
 - In the "Arguments" tab:
 - ⇒ Program arguments → specify the name of your datafile (upwind)
 - ⇒ Working directory → uncheck "Use default" and type path to datafile's directory
 - ⇒ "Apply"
 - ⇒ Debug

- 1 TRUST initialization
- 2 Eclipse initialization
 - Download eclipse
 - Create TRUST platform project under Eclipse
- 3 Create a Baltik project
 - Creation of a Baltik project
 - Creation of your git repository
 - Builds
 - Using Eclipse
- 4 Modify the cpp sources
 - Create a new cpp class
 - Modify your cpp class(Part 1)
 - Modify your cpp class(Part 2)
 - Add XD tags (keyword documentation)
 - Adding prints
- 5 Parallel exercise
- 6 Validation form and test cases
 - "Run_fiche"
 - Validation test case
- 7 Code coverage exercise
- 8 Tools
 - Debug with GDB
 - Find memory bugs with valgrind
- 9 For more
- 10 **Appendix**
 - Configuring TRUST project in Eclipse
 - Configuring BALTIC project in Eclipse
 - **TPP files in Eclipse**

TPP files in Eclipse

tpp format in TRUST is not natively recognized by eclipse and code is not highlighted. If you want to edit tpp files, you can:

- open Eclipse
- click on "Window" then select "Preferences"
- search for "File Associations" and add *.tpp to the list
- search for "File types" and add *.tpp to pattern the select for type "C++ header file"
- save preferences

Launch Eclipse

```
$ mkdir -p Formation_TRUST/yourname/workspace  
$ cd Formation_TRUST/yourname/eclipse  
$ ./eclipse &
```

- Workspace: Browse the directory Formation_TRUST/yourname/workspace
- Welcome : close x button