AN INTRODUCTION TO MFRONT: HOW TO IMPLEMENT MATERIAL PROPERTIES

thomas.helfer@cea.fr
maxence.wangermez@cea.fr

Maxence Wangermez, Thomas Helfer, 04/06/2024

PREAMBLE

MFront is available here:

https://github.com/thelfer/tfel

The documentation is available here:

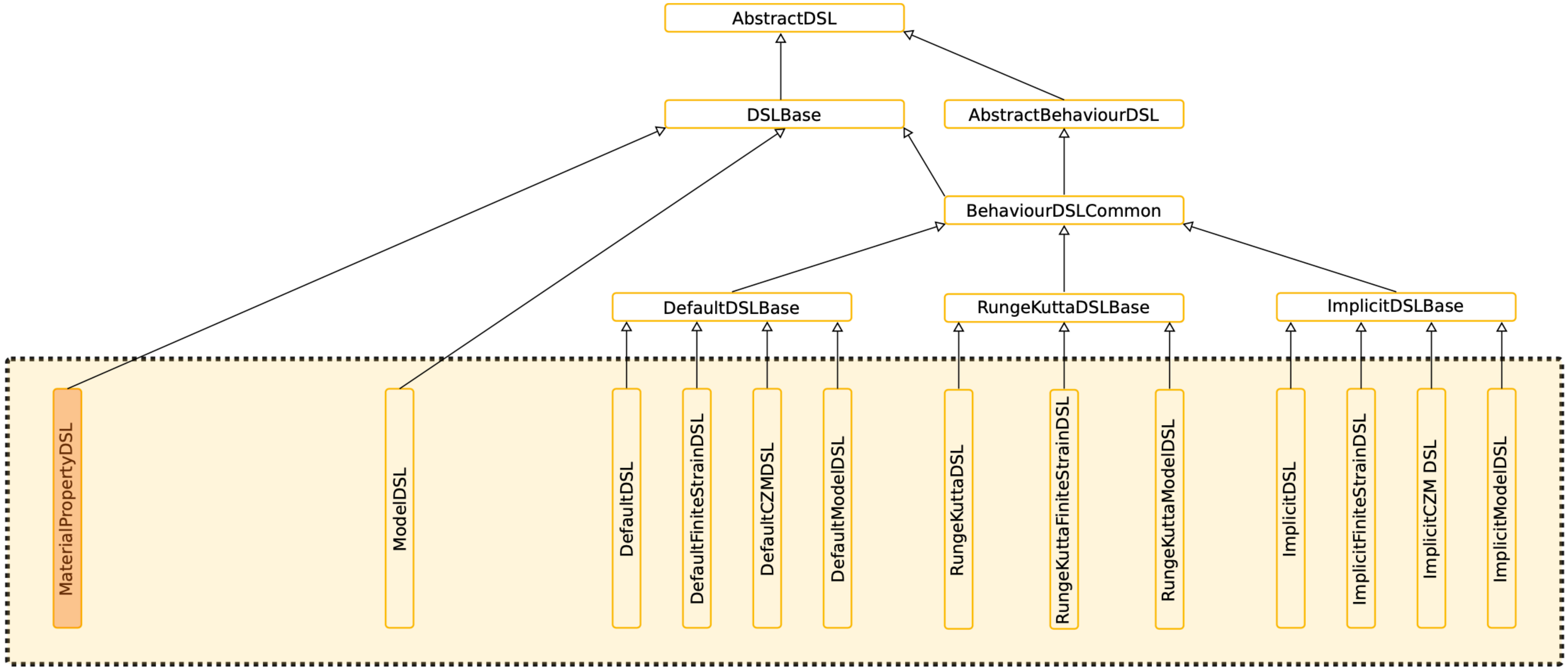
https://thelfer.github.io/tfel/web/index.html

This presentation is based on a tutorial available here:

https://thelfer.github.io/tfel/web/material-properties.html

and on the ongoing **MFrontbook**

PREAMBLE


AbstractDSL
DSLBase
AbstractBehaviourDSL
BehaviourDSLCommon
DefaultDSLBase
RungeKuttaDSLBase
ImplicitDSLBase
MaterialPropertyDSL
ModelDSL
DefaultDSL
DefaultFiniteStrainDSL
DefaultCZMDSL
DefaultModelDSL
RungeKuttaDSL
RungeKuttaFiniteStrainDSL
RungeKuttaModelDSL
ImplicitDSL
ImplicitFiniteStrainDSL
ImplicitCZM DSL
ImplicitModelDSL

OUTLINE

SOME ADVICES AND GOOD PRACTICES TO WORK WITH MFRONT

1. First writing and use of a MFront file with the Python interface
2. Analysis of the MFront file content
3. Improvement and best practices (quality assurance)

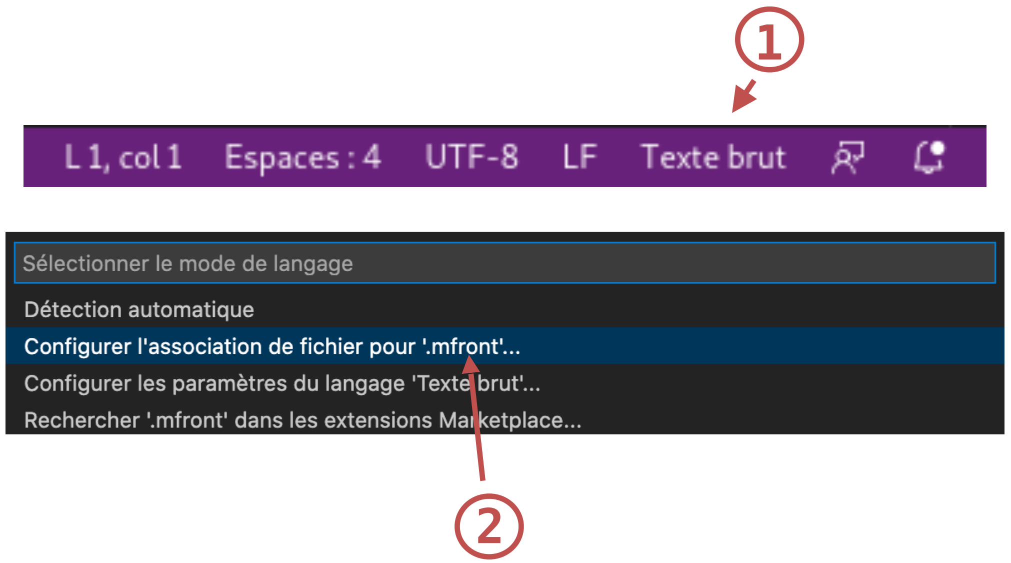
1. FIRST WRITING AND USE OF A MFRONT FILE WITH THE PYTHON INTERFACE

Material Properties

FIRST FILE WRITING - CONFIGURATION

Utiliser votre éditeur de texte avec la **coloration syntaxique du C++**

Exemple : Configuration de Visual Studio Code (une fois pour toutes) :



L 1, col 1   Espaces : 4   UTF-8   LF   Texte brut

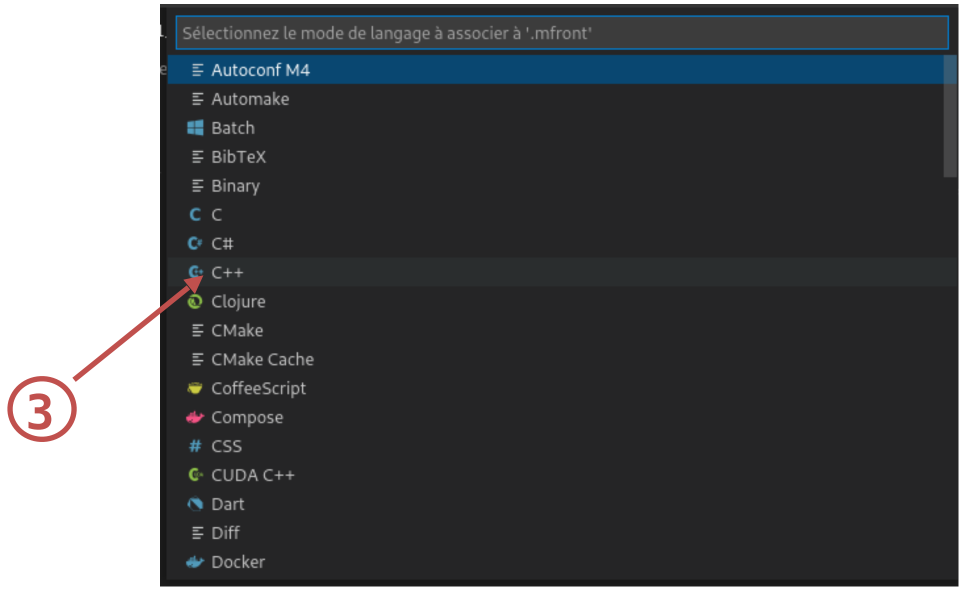
Sélectionner le mode de langage

Détection automatique

Configurer l'association de fichier pour '.mfront'...

Configurer les paramètres du langage 'Texte brut'...

Rechercher '.mfront' dans les extensions Marketplace...



Sélectionnez le mode de langage à associer à '.mfront'

Autoconf M4
Automake
Batch
BibTeX
Binary
C
C#
C++
Clojure
CMake
CMake Cache
CoffeeScript
Compose
CSS
CUDA C++
Dart
Diff
Docker

FIRST FILE WRITING - MFRONT FILE

Young's modulus of Uranium Dioxide[1] :

$$E(T, f) = 2.2693\,10^{11}\,(1 - 2.5\,f)\,(1 - 6.786\,10^{-5}\,T - 4.23\,10^{-8}\,T^2)$$

[1] MARTIN, DG. The elastic constants of polycrystalline UO2 and (U,Pu) mixed oxides: A review and recommendations. High Temperatures. High Pressures. 1989. Vol. 21, no. 1, p. 13–24.

First `MyFirstMFrontFile.mfront` file:

FIRST FILE WRITING - MFRONT FILE

Young's modulus of Uranium Dioxide[1] :

$$E(T,f) = 2.2693\,10^{11}\,(1-2.5\,f)\,(1-6.786\,10^{-5}\,T-4.23\,10^{-8}\,T^2)$$

[1] MARTIN, DG. The elastic constants of polycrystalline UO2 and (U,Pu) mixed oxides: A review and recommendations. High Temperatures. High Pressures. 1989. Vol. 21, no. 1, p. 13–24.

First `MyFirstMFrontFile.mfront` file:

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
        res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

FIRST FILE WRITING - COMPILATION

Step 1 - Generate the corresponding C++ files associated to the chosen interface:

TRY...

```
$ mfront --interface=python MyFirstMFrontFile.mfront
```

Creation of two folders: src/ and include/ with the C++ sources

These two directories are not working directories, since they are often **deleted.**

Step 2 - Generate the dynamic library :

TRY...

```
$ mfront --obuild
Treating target : all
The following library has been built :
- materiallaw.so :  MyFirstYoungModulusOfUraniumDioxide
```

Name of the library generated by MFront in the src folder

Name of the material property it contains

Both steps can be performed in a single command:

```
$ mfront --obuild --interface=python MyFirstMFrontFile.mfront
```

FIRST FILE WRITING - USE WITH PYTHON INTERFACE

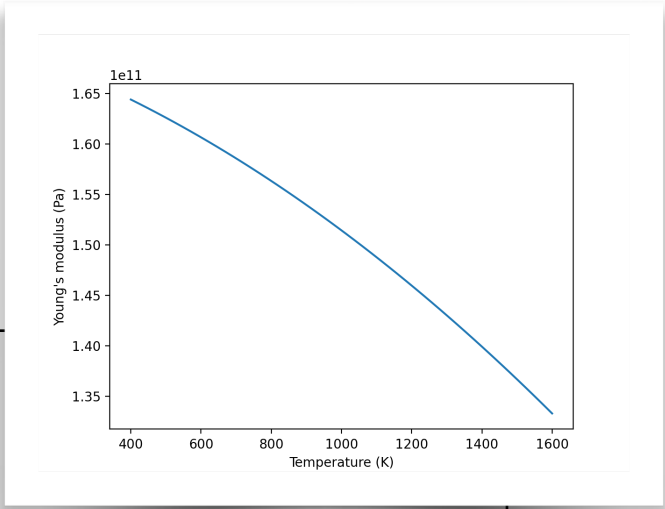
Use the library with Python:

Add the src directory to your PYTHONPATH :

```
$ export PYTHONPATH="${PYTHONPATH}:/my/src/path"
```

or get the subdirectory in your python script:

```
import src.materiallaw as ml
```



```
import materiallaw as ml
import numpy as np
from matplotlib import pyplot as plt

T = np.linspace(400, 1600)
E = np.array([ml.MyFirstYoungModulusOfUraniumDioxide(Ti, 0.1) for Ti in T])
plt.xlabel("Temperature (K)")
plt.ylabel("Young's modulus (Pa)")
plt.plot(T, E)
plt.show()
```

The order of the arguments corresponds to the order of declaration of the @Input (Temperature, Porosity) in the MFront file.

2. ANALYSIS OF THE FIRST FILE CONTENT

ANALYSIS OF THE FIRST FILE CONTENT - FIRST KEYWORD

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
        res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@DSL (Domain Specific Language): Tell MFront how to interpret the file :

Material Property (**today**)	-> **MaterialLaw**
Material Behaviour	-> Implicit, ImplicitParser, RungeKutta, etc.
Point-wise model	-> Model, DefaultModel, ImplicitModel

ANALYSIS OF THE FIRST FILE CONTENT - FIRST KEYWORD

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
      res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@DSL (Domain Specific Language): Tell MFront how to interpret the file :

- Material Property (**today**) -> **MaterialLaw**
- Material Behaviour -> Implicit, ImplicitParser, RungeKutta, etc.
- Point-wise model -> Model, DefaultModel, ImplicitModel

To display the list of available DSLs:

```
$ mfront --list-dsl
```

ANALYSIS OF THE FIRST FILE CONTENT - @DSL - DOMAIN SPECIFIC LANGUAGE

Each **DSL** has its conventions and **keywords**, fortunately, they are often common to several DSLs.

To display the list of keywords associated with the DSL MaterialLaw :

```
$ mfront --help-keywords-list=MaterialLaw
```

Each keyword in a DSL is documented:

To display the documentation for the @Law keyword of the MaterialLaw DSL:

```
$ mfront --help-keyword=MaterialLaw:@Law
```

```
The `@Law` keyword allows the user to associate a name to the material
law being treated. This keyword is followed by a name.
```

ANALYSIS OF THE FIRST FILE CONTENT - SECOND KEYWORD

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
      res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@Law: defines the name of the material property. Here `MyFirstYoungModulusOfUraniumDioxide` is not a very good choice for the material property name…

To display the documentation of the @Input keyword related to the MaterialLaw DSL:

```
$ mfront --help-keyword=MaterialLaw:@Law
```

ANALYSIS OF THE FIRST FILE CONTENT - THIRD KEYWORD

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
        res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@Input: name the input parameters of the material property. No information for the reader…

To display the documentation of the @Input keyword related to the MaterialLaw DSL:

```
$ mfront --help-keyword=MaterialLaw:@Input
```

ANALYSIS OF THE FIRST FILE CONTENT - FOURTH MOT-CLEF

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
     res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@Function: C++ code block used to implement the material property. By default, the block returns the result of the material property in the variable `res` !

3. IMPROVEMENTS AND BEST PRACTICES

CHANGE THE NAME OF THE MATERIAL PROPERTY LAW

```
@DSL MaterialLaw;
@Law MyFirstYoungModulusOfUraniumDioxide;
@Input T, f;
@Function {
      res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

@Law: defines the name of the material property. Here `MyFirstYoungModulusOfUraniumDioxide` which is not a very explicit name…

BEST PRACTICE BEST PRACTICE

It is recommended to use explicit name of the material property. For example, add a reference to the original paper from which the law is taken.

```
@Law YoungModulus_Martin1989; // name of the material property
```

FILE NAME CONSISTENT WITH THE MATERIAL PROPERTY

Choose an explicit name of the file (no need to be identical to @Law name):

```
@DSL MaterialLaw;
@Law YoungModulus_Martin1989; // name of the material property
@Input T, f;
@Function {
      res = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

It is a good practice to change the name of the MFront file to be consistent with the material property. For example:

`UO2_YoungModulus_Martin1989.mfront`

MFRONT FILE ANALYSIS - PAY ATTENTION TO PREVIOUS IMPLEMENTATIONS

Pay attention ! If the file is compiled after changing the name of the @Law:

```
@DSL MaterialLaw;
@Law YoungModulus_Martin1989; // name of the material property
@Input T, f;
@Function {
        res = 2.2693e11 * (1 − 2.5 * f) * (1 − 6.786e−05 * T − 4.23e−08 * T * T);
}
```

Compiling the Mfront file:

```
$ mfront --obuild --interface=python UO2_YoungModulus_Martin1989.mfront
Treating target : all
The following library has been built :
- materiallaw.so :  MyFirstYoungModulusOfUraniumDioxide YoungModulus_Martin1989
```

Two material properties in the materiallaw.so library

ASSOCIATE A MATERIAL NAME WITH THE MATERIAL PROPERTY

It is possible to enter the name of the modeled material using the **@Material** keyword:

```
@Material UO2; // material name
```

Associating a material name with the material property has two effects:

- changes the name of the function (`UO2_YoungModulus_Martin1989`)
- change le nom de la librairie (`uo2.so` instead of `materiallaw.so`)

```
$ mfront --obuild --interface=python UO2_YoungModulus_Martin1989.mfront
Treating target : all
The following libraries have been built :
- materiallaw.so : MyFirstYoungModulusOfUraniumDioxide YoungModulus_Martin1989
- uo2.so : UO2_YoungModulus_Martin1989
```

CHANGE THE NAME OF THE OUTPUT

It is possible to change the name of the default output to a more explicit name using the **@Output** keyword:

```
@Output E;
@Function {
    E = 2.2693e11 * (1 - 2.5 * f) * (1 - 6.786e-05 * T - 4.23e-08 * T * T);
}
```

DOCUMENT THE NAMES OF THE INPUTS

The names of the inputs are not explicit (f : porosity ??). To improve the clarity and unambiguity of the MFront file, it is possible to use the TFEL Glossary.

```
@Input T,f;
T.setGlossaryName("Temperature");
f.setGlossaryName("Porosity");
```

This also helps the interoperability of the library since the TFEL glossary defines a set of uniquely defined names that can be used to qualify a variable.

If the variable name does not exist in the TFEL glossary : use the method **setEntryName(str)**

Glossary documentation: https://thelfer.github.io/tfel/web/glossary.html

USE OF PARAMETERS

Entering quantities in the form of parameters facilitates:

- sensitivity studies
- taking into account the propagation of uncertainties in the data
- potential re-identifications

```
@Parameter E0 = 2.2693e11;
@Parameter dE_dT = -1.53994698e7;
@Parameter d2E_dT2 = -1.9198278e4;
@Parameter f0 = 0.4;

@Function {
    E = (1 - f / f0) * (E0 + dE_dT * T + (d2E_dT2 / 2) * T * T);
}
```

MODIFICATION OF PARAMETERS

Modification of the parameters with a .txt file:

- the file contains lines: <parameter name> <new parameter value>
- the expected file name in the current directory is given by command:

```
$ mfront-query --parameters-file UO2_YoungModulus_Martin1989.mfront

UO2_YoungModulus_Martin1989-parameters.txt
```

TRY…

UO2_YoungModulus_Martin1989-parameters.txt

```
# new material parameters
E0         2.2693e11
dE_dT     -1.53994698e7
d2E_dT2 -1.9198278e4
f0        0.4
```

PHYSICAL BOUNDS AND STANDARD BOUNDS (VERY IMPORTANT)

Two types of bounds:

- **Physical bounds** : ⛔
  - prohibit non-physical values (negative temperatures, negative porosity, etc.)
  - **<u>always stop</u>** the calculation if an input value is not physical

- **Standards bounds**: ⚠
  - manage the cases when implemented laws are extrapolated (relevance of the models outside the validity limits)
  - depending on the option, does **<u>nothing</u>**, displays a **<u>warning</u>** or **<u>stops</u>** the calculation if an input value is outside the standard limits.

USE OF PHYSICAL BOUNDS

Physical bounds:
- prohibit non-physical values (negative temperatures, negative porosity, etc.)
- **<u>always stop</u>** the calculation if an input value is not physical

It is possible to define physical bounds with the keyword **@PhysicalBounds:**

```
@PhysicalBounds T in [0:*[;
@PhysicalBounds f in [0:1];
```

TRY…

Evaluate the Young Modulus of UO2 for a temperature from 400 to 1600 K and a porosity ratio of 1.1 with your Python script.

USE OF THE PHYSICAL BOUNDS

Let’s make a test with f > 1:

```
from uo2 import UO2_YoungModulus_Martin1989 as E_UO2
import numpy as np
from matplotlib import pyplot as plt
T = np.linspace(400, 1600)
E = np.array([E_UO2(Ti, 1.1) for Ti in T])
plt.xlabel("Temperature (K)")
plt.ylabel("Young's modulus (Pa)")
plt.plot(T, E)
plt.show()
```

```
Traceback (most recent call last):
  File ".../YoungModulus_PhyBounds.py", line 6, in <module>
    E = np.array([E_UO2(Ti, 1.1) for Ti in T])
  File ".../YoungModulus_PhyBounds.py", line 6, in <listcomp>
    E = np.array([E_UO2(Ti, 1.1) for Ti in T])
RuntimeError: UO2_YoungModulus_Martin1989 : f is beyond its physical upper bound (1.1>1).
```

PHYSICAL BOUNDS - AUTOMATIC DECLARATION

The physical bounds are automatically declared using the TFEL glossary and the @UnitSystem keyword:

```
T.setGlossaryName("Temperature");
f.setGlossaryName("Porosity");
@UnitSystem SI;
```

```
Traceback (most recent call last):
  File ".../YoungModulus_PhyBounds.py", line 6, in <module>
    E = np.array([E_UO2(Ti, 1.1) for Ti in T])
  File ".../YoungModulus_PhyBounds.py", line 6, in <listcomp>
    E = np.array([E_UO2(Ti, 1.1) for Ti in T])
RuntimeError: UO2_YoungModulus_Martin1989 : f is beyond its physical upper bound (1.1>1).
```

USE OF STANDARDS BOUNDS

Standards bounds:

- manage the cases when implemented laws are extrapolated (relevance of the models outside the validity limits)
- depending on the option, does **nothing**, displays a **warning** or **stops the calculation** if an input value is outside the standard limits.

Standard bounds are defined with the keyword **@Bounds:**

```
@Bounds T in [273.15:2610.15]; // Validity range
```

TRY...

Make a test with a temperature T > 2610.15 K

USE OF STANDARDS BOUNDS

Let’s make a test with T > 2610.15 K :

```
from uo2 import UO2_YoungModulus_Martin1989 as E_UO2
import numpy as np
from matplotlib import pyplot as plt
T = np.linspace(400, 3000)
E = np.array([E_UO2(Ti, 0.5) for Ti in T])
plt.xlabel("Temperature (K)")
plt.ylabel("Young's modulus (Pa)")
plt.plot(T, E)
plt.show()
```

The default setting for MFront is to do nothing. Three possible behaviors:

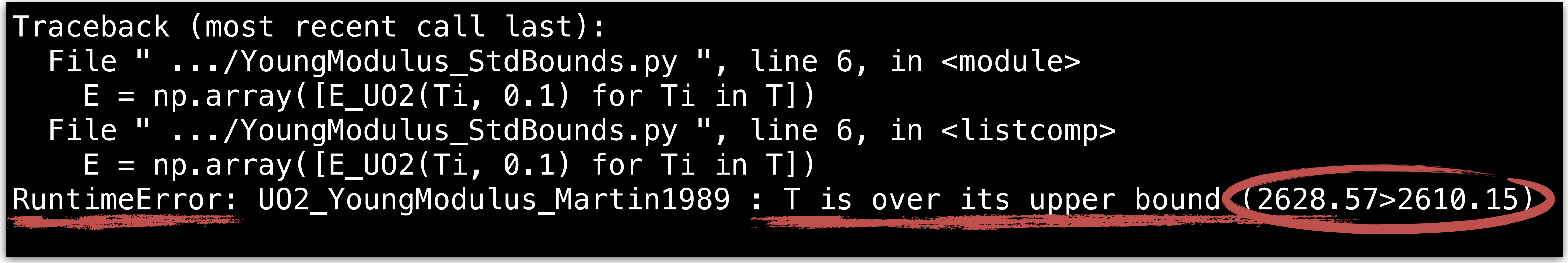
- None : **does nothing** (default)
- Warning : **notifies** the user
- Strict : **stops** the calculation

Different actions on each interface.
In practice, managed at a higher level.

USE OF STANDARDS BOUNDS

To test the behavior of MFront when the standard bounds are exceeded, it is possible to change it from the python script:

```
import os
os.environ['PYTHON_OUT_OF_BOUNDS_POLICY'] = 'STRICT'
```



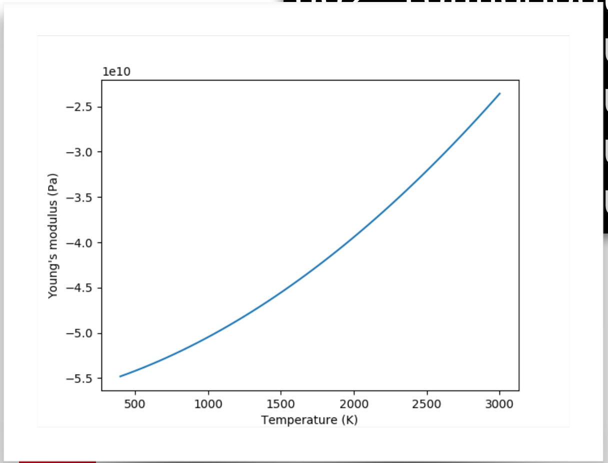
```
Traceback (most recent call last):
  File " .../YoungModulus_StdBounds.py ", line 6, in <module>
    E = np.array([E_UO2(Ti, 0.1) for Ti in T])
  File " .../YoungModulus_StdBounds.py ", line 6, in <listcomp>
    E = np.array([E_UO2(Ti, 0.1) for Ti in T])
RuntimeError: UO2_YoungModulus_Martin1989 : T is over its upper bound (2628.57>2610.15)
```

USE OF STANDARDS BOUNDS - PYTHON EXAMPLE

To test the behaviour of MFront when the standard bounds are exceeded, it is possible to activate warnings from the python script :

```python
import os
os.environ['PYTHON_OUT_OF_BOUNDS_POLICY'] = 'WARNING'
```

```
UO2_YoungModulus_Martin1989 : T is over its upper bound (2628.57>2610.15).
UO2_YoungModulus_Martin1989 : T is over its upper bound (2681.63>2610.15).
UO2_YoungModulus_Martin1989 : T is over its upper bound (2734.69>2610.15).
UO2_YoungModulus_Martin1989 : T is over its upper bound (2787.76>2610.15).
                ulus_Martin1989 : T is over its upper bound (2840.82>2610.15).
                ulus_Martin1989 : T is over its upper bound (2893.88>2610.15).
                ulus_Martin1989 : T is over its upper bound (2946.94>2610.15).
                ulus_Martin1989 : T is over its upper bound (3000>2610.15).
```



USE OF QUANTITIES

The **@UseQT** keyword allows to do a dimensional analysis at compile time. For example, summing a stress and a strain generates an error at compile time (no influence on the execution performance):

```
const auto a = strain{1e-8};
const auto b = stress{10e6};
const auto c = a + b;
```

```
TFEL/Math/Quantity/qtOperations.hxx:302:19: error: static assertion failed:
invalid operation
    static_assert(std::is_same_v<UnitType, UnitType2>, "invalid operation");
```

USE OF QUANTITIES

The **@UseQT** keyword allows to do a dimensional analysis at compile time. For example, summing a stress and a strain generates an error at compile time (no influence on the execution performance):

```
@UseQt true;
@Input temperature T;
@Input real f;
@Parameter stress E0 = 2.2693e11;
@Parameter real f0 = 0.4;
@Parameter derivative_type<stress, temperature> dE_dT = −1.53994698e7;
@Parameter derivative_type<stress, temperature, temperature> d2E_dT2 = −1.9198278e4;
@Output stress E;
```

List of predefined types:

https://thelfer.github.io/tfel/web/mfront-types.html

CONCLUSION

```
@DSL MaterialLaw;
@Author Author Name;
@Date 11/02/20;
@UseQt true;
@Law YoungModulus_Martin1989; // name of the material property
@Material UO2; // material name
@Input temperature T;
@Input real f;
T.setGlossaryName("Temperature"); // T stands for the temperature
f.setGlossaryName("Porosity"); // f stands for the porosity
@PhysicalBounds T in [0:*[;
@PhysicalBounds f in [0:1];
@Bounds T in [273.15:2610.15]; // validity range
@Bounds f in [0:0.4]; // validity range
@Parameter stress E0 = 2.2693e11;
@Parameter real f0 = 0.4;
@Parameter derivative_type<stress, temperature> dE_dT = -1.53994698e7;
@Parameter derivative_type<stress, temperature, temperature> d2E_dT2 = -1.9198278e4;
@Output stress E;
@Function {
    E = (1 - f / f0) * (E0 + dE_dT * T + (d2E_dT2 / 2) * T * T);
}
```