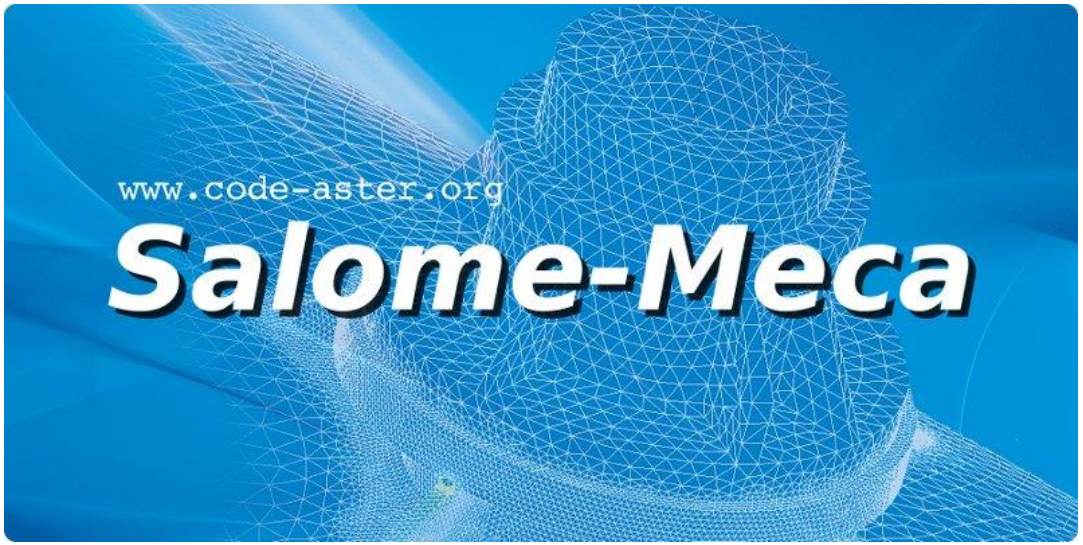
Development in code_aster
Description of the commands syntax



Code_Aster, Salome-Meca course material

Code_Aster supervisor

A Code_Aster commands file is exclusively written in Python.

Code_Aster extends the language with its commands.

Example:

```
acce2 = CALC_FONCTION(FFT=_F(FONCTION=acce,
                             SYME='OUI'))
```

- Commands results are called « concepts ».
- All keywords of of each command are described in a « catalog »

  Type of value, number of values, compulsory or optional…

Roles of the supervisor:

- Executing the commands
- Give the fortran/python code access to the content of the keywords

Syntax description – header (1)

[d5.01.01], [d5.01.02]

The catalog is written in Python, under the `code_aster/Cata/` directory

- Cata/DataStructure.py: declaration of the types of concepts

  Definitions are imported from Cata/Legacy/DS/.py*

- Cata/Commons: definitions shared between several commands
- Cata/Commands: definition of the commands syntax

Assign the operator (as worker):

```
DEFI_FONCTION=OPER(nom="DEFI_FONCTION",
```

*Type of command (OPER or PROC), use the **same name**, it is compulsory*

```
    op=3, ...
```

Number of the fortran subroutine that will be called: bibfor/op/op0003.F90

```
CALC_FONCTION=MACRO(nom="CALC_FONCTION",
```

*Type of command, use the **same name**, it is compulsory*

```
    op=OPS('Macro.calc_fonction_ops.calc_fonction'),
```

Textual reference to the Python source code to allow outside syntax checking (eficas)

Import will be: from Macro.calc_fonction_ops import calc_fonction_ops

Source file is bibpyt/Macro/calc_fonction_ops.py and it defines a function called calc_fonction_ops

Syntax description – header (2)

[d5.01.01], [d5.01.02]

Doc string

```
fr=tr("Effectue des opérations mathématiques sur des fonctions"),
```

Description of the command, tr() supports external translation (native language is french)

Creation of the result

```
reentrant='n',
```

Tell if the result is a new object ('n'), or if the command always changes an argument ('o') or ('f') sometimes.

A `reuse` keyword must be added, hidden.

Type of the result

```
sd_prod=fonction_sdaster,
```

Type of the result, class declared in Cata/DataStructure.py

```
sd_prod=defi_fonction_prod,
```

Function that returns the type of the result computed from the arguments/keywords content

Example:

```
def defi_fonction_prod(VALE, VALE_C, **args):
    if VALE is not None:
        return fonction_sdaster
    if VALE_C is not None:
        return fonction_c
    raise AsException('unsupported keywords')
```

Add *self* as first argument for MACRO

It must raise AsException(msg) if the type can not be set. Return None if there is no result.

Keywords

[d5.01.01], [d5.01.02]

Simple keyword (SIMP):

```
METHODE=SIMP(statut='f', typ='TXM', defaut="PROL_ZERO", into=("PROL_ZERO",
  "TRONCATURE", "COMPLET"), fr=tr('…') ),
```

statut: 'o' obligatory, 'f' optional (default)

typ: 'TXM' for text, 'I' for integer, 'R' for float, 'C' for complex or objects.

defaut: default value

into: authorized values

*min/max: minimal/maximal number of values. max='**' means unlimited number of values.*

val_min/val_max: minimal/maximal value of the given values. val_min=0. means positive value expected.

Validators allow more checkings: NoRepeat(), LongStr(low, high)…

May also define another result using `typ=CO`. Usage: `MODELE=CO('model')`

Factor keyword (FACT):

```
FFT=FACT(statut='f', fr=tr("Transformée de Fourier ou de son inverse"),
    FONCTION=SIMP(statut='o', typ=(fonction_sdaster, fonction_c),
),
```

A logical group of one or more simple keywords.

statut: 'o' obligatory, 'f' optional (default), 'd' present by default

Conditional blocks

[d5.01.01], [d5.01.02]

Conditional blocks (BLOC):

```
FONCTION=SIMP(…),
b_syme=BLOC(condition="AsType(FONCTION)==fonction_c",
    SYME=SIMP(statut='f', typ='TXM', into=('OUI', 'NON'), defaut='OUI'),
),
```

The keywords inside the block will exist/be authorized if the condition is True.

The condition is a string because it will be evaluated during the execution.

The condition is evaluated at the same level: here, FONCTION must exist at the same level as b_syme.

If an optional keyword is not provided by the user, its value is None. Example: « if FONCTION is not None ».

Rules

```
regles=(UN_PARMI('DERIVE', 'INTEGRE',…, 'REGR_POLYNOMIALE'),),
```

All keywords must be in the same level as regles*: at top level, in a BLOC or in a FACT.*

Rules about the presence of the keywords:

UN_PARMI, AU_MOINS_UN, EXCLUS, ENSEMBLE, PRESENT_PRESENT, PRESENT_ABSENT

End of presentation

Is something missing or unclear in this document?

Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about Code_Aster training materials.

Do not hesitate to share with us your comments on the Code_Aster forum dedicated thread.