

## Liste des macros de précompilation

---

### Résumé :

Ce document liste les macros présentes dans le code et, pour la plupart, déterminées lors de la phase de précompilation.

## Table des Matières

---

1 Classification.....	3
2 Configuration.....	3
3 Fonctionnalités.....	4
4 Interfaces C-Fortran.....	4
5 Système.....	4
6 Utilisation par le développeur.....	5
6.1 Types à utiliser en C.....	5
6.2 Types à utiliser en Fortran.....	5
6.3 Gestion d'erreur et d'exception en C.....	5
6.4 Macros utilitaires en fortran.....	6

## 1 Classification

L'utilisation des macros du précompilateur permet d'associer un texte à un identificateur. Dans Code\_Aster, on utilise les macros pour adapter le code à l'environnement de la machine (architecture, bibliothèques disponibles, etc.).

On présente par la suite les macros regroupées par catégorie.

### Remarque

*On ne décrit pas ici les macros placées de manière conventionnelle dans la définition des fichiers d'entête qui permettent d'éviter les inclusions récursives. Ces macros doivent reprendre le nom du fichier (`ASTER_CONSTANT_H` pour `aster_constant.h`) et ne sont utilisées que dans le fichier en question.*

*On ne détaille pas non plus certaines macros qui ne sont utilisées que localement dans un module par commodité de programmation.*

## 2 Configuration

On regroupe dans cette catégorie les macros liées à la configuration préalable à la compilation d'une version. On les retrouve dans les entêtes `asterc_config.h` (et presque toutes dans `asterf_config.h`) qui sont produits automatiquement lors de l'étape `waf configure`.

Les macros précédées de (\*) sont déterminées lors de la phase de configuration mais ne sont pas ajoutées dans les fichiers d'entête `asterc_config.h` et `asterf_config.h`. Elles sont passées en ligne de commande avec `-D<nom-macro>`.

- `__POSIX / __WINDOWS` : exclusif, dit si la plate-forme est de type Unix ou Windows.
- `__USE_64_BITS` : dit si la plate-forme est 64 bits
- `LINUX, LINUX64, SOLARIS, SOLARIS64` : utiliser pour distinguer certains systèmes parmi les différents Unix.
- `__NO_UNDERSCORE` : utilisé pour les appels C-Fortran, ne pas ajouter le '\_' en fin de routine.
- `__USE_MPI` : indique que l'on utilise le parallélisme MPI.
- `__USE_OPENMP` : indique que l'on utilise les directives OpenMP.
- `__DISABLE_HDF5` : indique que la bibliothèque `hdf5` n'est pas disponible. Par défaut, on suppose que `hdf5` est disponible (comme si `HAVE_HDF5` était défini).
- `__DISABLE_MED` : indique que la bibliothèque `med` n'est pas disponible. Par défaut, on suppose que `med` est disponible (comme si `HAVE_MED` était défini).
- `__HAVE_MUMPS` : indique que la bibliothèque `mumps` est disponible.
- `MUMPS_VERSION` : version de la bibliothèque `mumps`.
- `__HAVE_METIS` : indique que la bibliothèque `metis` est disponible.
- `__DISABLE_SCOTCH` : indique que la bibliothèque `scotch` n'est pas disponible. Par défaut, on suppose que `scotch` est disponible (comme si `HAVE_SCOTCH` était défini).
- `__HAVE_PETSC` : indique que la bibliothèque `PETSc` est disponible.
- (\*) `__USE_INTEL_IFORT` : indique que l'on utilise le compilateur Intel. Les fonctionnalités spécifiques à Intel doit utiliser `__USE_INTEL_IFORT && HAVE_xxx` parce que `aslint` ne doit pas l'utiliser, car indépendant du compilateur. Cette macro est si besoin passée en ligne de commande avec `-D__USE_INTEL_IFORT`.
- `HAVE_TRACEBACKQQ` : indique que la fonction `tracebackqq` est disponible (nécessite `__USE_INTEL_IFORT`).
- `HAVE_BACKTRACE` : indique que la fonction `backtrace` est disponible.
- `__WITHOUT_PYMOD` : utiliser pour la construction séparée d'une bibliothèque contenant l'ensemble des modules objets et de modules python sous forme de bibliothèques dynamiques.
- `__MAIN` : nom attendu pour le programme principal (peut être `main`, ou `MAIN` précédé et/ou suivi ou non de '\_').
- `STRINGIFY_USE_QUOTES, STRINGIFY_USE_OPERATOR` : indique comment le préprocesseur fortran convertit du code en chaîne de caractères.

## Définition des types :

- `ASTER_INT4_SIZE` : taille du type fortran `integer(kind=4)`.
- `ASTERC_FORTRAN_INT4` : type C correspondant à l'entier fortran `integer(kind=4)`.
- `ASTER_INT_SIZE` : taille du type fortran `integer`.
- `ASTERC_FORTRAN_INT` : type C correspondant à l'entier fortran `integer`.
- `ASTER_REAL4_SIZE` : taille du type fortran `real(kind=4)`.
- `ASTERC_FORTRAN_REAL4` : type C correspondant à l'entier fortran `real(kind=4)`.
- `ASTER_REAL8_SIZE` : taille du type fortran `real(kind=8)`.
- `ASTERC_FORTRAN_REAL8` : type C correspondant à l'entier fortran `real(kind=8)`.
- `ASTER_COMPLEX_SIZE` : taille du type fortran `complex(kind=8)` (`complex*16` du fortran77).
- `ASTERC_STRING_SIZE` : type de la taille des chaînes de caractères en C, échangées avec le fortran.
- `MED_INT_SIZE` : taille du type entier utilisé dans med.

## 3 Fonctionnalités

On retrouve dans cette catégorie les macros liées à l'activation, la désactivation de certaines fonctionnalités et leur paramétrage.

- `_DISABLE_MATHLIB_FPE` : permet d'ignorer les erreurs numériques (floating point exceptions) à certains endroits du code, entre deux appels à la fonction `matfpe`, notamment autour des appels aux routines `blas/lapack`. C'est le comportement par défaut.
- `_ENABLE_MATHLIB_FPE` : modifie le comportement par défaut, n'intercepte pas ce type d'erreur.
- `ASTER_DISABLE_MPI_CHECK` : désactive la vérification des communications MPI.
- `_NO_EXPIR` : permet de supprimer l'alarme prévenant qu'une version a plus de 15 mois.
- `USE_ASSERT` : permet d'activer les assertions dans la partie C écrites avec `AS_ASSERT`.
- `TEST_STRICT` : lorsque cette macro est activée, on ne tient pas compte de la valeur du mot-clé `TOLE_MACHINE` dans les fonctions de test (`TEST_RESU` et dérivés). Un message d'information le rappelle.

## 4 Interfaces C-Fortran

Dans `definition.h`, de nombreuses macros sont définies afin d'éviter les erreurs lors du passage d'argument entre le C et le Fortran.

Par exemple :

```
#define CALL_JEEXIN(a,b) CALLSP(JEEXIN, jeexin, a, b)
```

Ceci permet d'appeler depuis le C la subroutine fortran JEEXIN dont le premier argument est une chaîne de caractères (S) et le second un pointeur (P) vers un entier ou un réel.

L'intérêt est que selon la plate-forme, il faut ajouter ou non un '\_' à la fin du nom de la subroutine. De même, il peut être nécessaire de passer la longueur de la chaîne de caractères juste après la variable ou en dernier position.

NB : suite au passage en Fortran 90, on pourrait améliorer le passage des chaînes de caractères.

## 5 Système

On regroupe dans cette catégorie les macros liées au système d'exploitation et l'architecture de la machine. Les macros précédées de (\*) sont activées automatiquement dans `aster_depend.h` en fonction de l'étape de configuration ou de la plate-forme.

L'objectif est qu'elles soient toutes déterminées à la configuration.

- (\*) `_STRLEN_AT_END` : utilisé pour les appels C-Fortran, indique que la longueur de la chaîne de caractères doit être reporté après les arguments et non juste après l'argument de type chaîne.
- (\*) `GNU_LINUX` : indique que l'on utilisera plus tard (dans le gestion des signaux) `_GNU_SOURCE=1` pour accéder à des fonctions hors standard POSIX.
- `_NAN_BULL` : définit une valeur particulière pour le « réel non défini », `R8UND` dans `envima.c`.

Types déterminés lors de la configuration, redéfinis pour des raisons pratiques et de lisibilité :

- (\*) `STRING_SIZE` : type de la taille des chaînes de caractères en C.
- (\*) `ASTERINTEGER4` : type C correspondant à l'entier fortran `integer(kind=4)`.
- (\*) `ASTERINTEGER` : type C correspondant à l'entier fortran `integer`.
- (\*) `ASTERREAL4` : type C correspondant à l'entier fortran `real(kind=4)`.
- (\*) `ASTERDOUBLE` : taille du type fortran `real(kind=8)`.
- (\*) `ASTERC_FORTRAN_REAL8` : type C correspondant à l'entier fortran `real(kind=8)`.
- `REAL_NB_CHIFFRES_SIGNIFICATIFS` : nombre de chiffres significatifs pour les réels.
- `INTEGER_NB_CHIFFRES_SIGNIFICATIFS` : nombre de chiffres significatifs pour les entiers.
- `OPT_TAILLE_BLOC_MULT_FRONT` : taille optimale des blocs pour le solveur mult-front.
- `USE_STDCALL` : dit si l'on utilise `__stdcall` pour l'appels des fonctions fortran depuis le C, pour Windows.

## 6 Utilisation par le développeur

Dans la programmation, on évite d'utiliser des macros de très bas niveau.

Par exemple, en C, on n'utilise jamais `ASTER_INT_SIZE` (le nombre d'octets utilisés pour un entier) mais le type associé. Et pour ce type, on n'utilise pas `ASTERC_FORTRAN_INT` mais le nom « interne » : `INTEGER`.

De même en fortran, on n'utilise pas directement `ASTER_INT_SIZE`, mais les valeurs définies dans `aster_types.h`.

### 6.1 Types à utiliser en C

On inclut le fichier d'entête général `aster.h` (qui inclus `aster_depend.h`). On a alors accès aux types pré-cités : `INTEGER`, `INTEGER4`, `DOUBLE`, `REAL4`, `STRING_SIZE`.

### 6.2 Types à utiliser en Fortran

On inclut le fichier d'entête `aster_types.h` qui définit :

- `aster_int_kind`
- `aster_int`

ainsi que des fonctions de conversion `to_aster_int`.

Des types et fonctions de conversion similaires sont définis : `med_int`, `mpi_int`, `mumps_int`, `blas_int` et donc `to_mpi_int`, etc.

On prend soin d'utiliser ces types dans les fonctions qui échangent des données avec des bibliothèques externes.

### 6.3 Gestion d'erreur et d'exception en C

En fortran, on dispose de la macro `ASSERT` qui vérifie une condition et arrête en erreur quand celle-ci n'est pas vérifiée en indiquant le lieu dans le source.

Pour émettre un message d'erreur en C, on utilise la macro `MYABORT` qui permet de lever une exception pour arrêter proprement le calcul.

On émule dans le code C le mécanisme d'exception en s'appuyant sur les fonctions système `setjmp` et `longjmp`.

L'écriture est simplifiée par l'utilisation des macros : `try`, `except`, `exceptAll`, `endTry`. Ainsi que `raiseException` et `raiseExceptionString` pour lever une exception.

Voir le module `aster_exceptions.c` pour un exemple.

## 6.4 Macros utilitaires en fortran

- `ASSERT(condition)` : émet un message d'erreur si la condition n'est pas vérifiée en indiquant le nom du fichier et la ligne de code où la vérification a été faite.
- `TO_STRING(code)` : remplace `code` par une chaîne de caractères, utilisable dans d'autres macros (exemple dans `ASSERT`).