Règles de programmation

Résumé :

Ce document présente les règles retenues par l'équipe de Développement d'Aster (EDA) pour l'écriture du code source de Code_Aster.

Le respect de ces règles a deux objectifs principaux :

- assurer une bonne portabilité du code,
- assurer une bonne lisibilité (et donc maintenabilité) du texte source.

Le contrôle du respect des règles de programmation est effectué lors de la soumission des développements (lors de l'exécution des commandes `hg commit` et `hg submit`) par l'outil `aslint` (du dépôt `devtools` qui peut aussi être utilisé de manière autonome).
Les erreurs (messages de type E ou C comme *erreur* ou *convention*) émises par aslint doivent impérativement être corrigées par le développeur.
Les alarmes (messages de type W comme *warning*) doivent être supprimées autant que possible pour l'amélioration du code et ne doivent être acceptées uniquement s'il n'est pas possible de faire autrement.

Ce document fournit des éléments complémentaires aux messages de type *warning* émis par `aslint` pour en expliquer l'objectif.
On y décrit aussi des conventions ou conseils qui ne peuvent pas être vérifiés automatiquement.

1 Vérifications automatiques

L'ensemble des vérifications effectuées par `aslint` sont répertoriées :
- en interne EDF : http://aster-services.der.edf.fr/mercurial/messages_index.html
- sur internet : https://bitbucket.org/code_aster/codeaster-src/wiki/AslintMessagesIdentifiers

Chaque message est identifié par une lettre suivie de quatre chiffres.

La lettre détermine la gravité du messages :
- E signifie *erreur* : c'est un erreur qu'on ne peut ignorer ;
- C signifie *convention* : c'est une erreur par convention. Par exemple, une alarme du compilateur que l'on juge trop grave dans code_aster.
- W signifie *warning* : il est habituellement possible de les supprimer en respectant de bonnes pratiques ;
- I signifie *information* : il s'agit simplement d'une information sur l'analyse du source.

Les deux premiers chiffres sont utilisés pour indiquer le langage concerné et le type de messages.

On ne détaille pas ici les différents messages relevés par `aslint`.
Il est impératif de corriger les erreurs E. On n'a pas le choix !
Il est impératif de corriger les erreurs par Convention.
Il est fortement recommandé d'éliminer un maximum de Warnings pour améliorer le code.

2 Règles complémentaires

2.1 Modules Fortran

Lors de la compilation, il est nécessaire de compiler les modules dans l'ordre imposé par leurs inter-dépendances, et ensuite, de compiler les routines qui utilisent ces modules.

Pour identifier rapidement et de manière automatique les modules, on impose donc le nommage suivant :
- `xxxx_type.F90` pour les modules qui définissent des types dérivés (un type par module) et les méthodes élémentaires pour la création des objets de ce type ;
- `xxxx_module.F90` pour les « méthodes » sur les types.

Tous les types dérivés doivent être documentés (comme le sont les structures de données Jeveux).

2.2 Constantes

On évite l'utilisation abondante de `#define`.
Il faut plutôt définir des paramètres (`parameter`) regroupés thématiquement dans un module.

3 Contourner la loi

Lors de la soumission de développements (avec `hg submit`), il arrive que l'on veuille passer outre l'émission de certains messages d'erreur (par convention, erreurs C) pour une ***bonne raison***.
Il est possible de faire en sorte que les erreurs soient dégradées en simples warnings.

Pour cela, on écrit dans le fichier `~/.hgrc`, dans la section `[aster]` :
```
check.submit = warn
```

Il faudra ensuite expliquer à l'intégrateur la *bonne raison* qui justifie ce contournement.