Manuel de Descriptif informatique
Fascicule D1.02 : Atelier de Génie Logiciel d' *Aster*
Document : D1.02.01

Manuel d'utilisation de l'AGLA (Atelier de Génie Logiciel d'*Aster*)

Résumé :

Ce document décrit l'organisation du source de *Code_Aster* et les règles que les développeurs doivent respecter afin d'y apporter des modifications ou des ajouts. Pour cela, des outils de développement et de gestion du source constituent l'Atelier de Génie Logiciel d'*Aster*, les commandes de l'AGLA sont décrites en vue de leur utilisation directe sur la machine de développement (par l'administrateur des sources) mais aussi à travers l'interface du *Code_Aster* Astk [U1.04.00] (par l'ensemble des développeurs).

Table des matières

1 L'environnement de développement du *Code_Aster*

La version de référence de *Code_Aster* est localisée sur une machine unique du réseau informatique de EDF R&D. Toutes les opérations de développements et les outils de l'AGLA sont sur cette machine.

Les comptes des développeurs sont identifiés sur le fichier `/aster/agla/identAster`. Seuls les titulaires de ces comptes sur la machine de référence ont accès à l'onglet AGLA dans l'interface Astk.

2 Les composantes du code source d'*Aster*

Code_Aster est composé essentiellement de 6 types de source :

le source de type `FORTRAN`	fichier dont l'extension vaut `.f`
le source de type `F90` ou `FORTRAN90`	fichier dont l'extension vaut `.F`
le source de type `CATALO`	description des éléments finis, des grandeurs qu'ils portent et des options qu'ils calculent - fichier dont l'extension vaut `.cata`
le source de type `CATAPY`	description du langage de commandes - fichier dont l'extension vaut `.capy`
le source de type `PYTHON`	macro-commandes et opérateurs nativement écrits en langage Python ; code source de la couche de supervision d'exécution - fichier dont l'extension vaut `.py`
les cas-tests qui sont considérés comme faisant partie intégrante d'une version d'*Aster* et suivent les mêmes règles de développement que le code source	

S'ajoutent également à cela, mais de façon marginale, du langage C (`.c`) et des fichiers d'entête (`.h`) , des catalogues matériau et des fichiers de données GIBI ou MED.

2.1 Le mécanisme des cartes

Pour associer de l'information nécessaire à l'AGLA à chaque unité de source, une notion de carte est utilisée. Une carte est un commentaire normalisé qui va permettre de conserver de l'information et de piloter des actions de l'AGLA. Selon le type de source, la syntaxe est adaptée en fonction du langage utilisé. Certaines cartes sont obligatoires, d'autres sont optionnelles. Toutes les cartes ne sont pas reconnues dans tous les types de fichiers.

Il y a une contrainte générale pour toutes les cartes : elles doivent être sur une seule ligne. Dans certains cas, il est possible de répéter une carte.

Carte	Statut	Description
`INFO`	`O`	Pour indiquer si l'unité est ajoutée ou modifiée. Cette carte sert également à designer des informations qui ne peuvent être déduites, comme la bibliothèque d'appartenance du module. Cette carte permet également de stocker les informations permettant de détecter la version officielle de l'unité. Selon le type de source, les informations conservées sont un peu différentes.
`TITRE`	`O`	Titre d'un cas-test.
`TOLE`	`F`	Permet d'autoriser la violation de certaines règles de programmation. `CRS_XXX` et `CRP_XXX`.
`RESPONSABLE`	`F`	Permet d'associer une unité à un responsable afin de bien contrôler l'évolution de certaines unités sensibles.

`O` : obligatoire `F` : facultatif

Prise en comptes des différentes cartes en fonction du type de source (Oui ou Non) :

Carte	FORTRAN*	CATALOGU	CASTEST	C	PYTHON	CMAT	DATG
INFO	O	O	O	O	O	O	O
TITRE	N	N	O	N	N	N	N
	O	N	N	N	N	N	N
RESPONSABLE	O	O	O	O	O	O	O

•fortran77 et fortran90

Seules les cartes RESPONSABLE ont les mêmes informations pour tous les types de source. La seule différence est la manière d'indiquer un commentaire selon le type de source. Pour ajouter ou modifier le responsable associé à une unité de source :

FORTRAN	C RESPONSABLE
CATALO	% RESPONSABLE
CATAPY	# RESPONSABLE
PYTHON	# RESPONSABLE
CASTEST	% RESPONSABLE
C	/* RESPONSABLE */
CMAT	% RESPONSABLE
DATG	* RESPONSABLE
F90	! RESPONSABLE

La carte est ajoutée en spécifiant le caractère suivie du champ RESPONSABLE, elle sera alors automatiquement complétée par le nom du compte suivi du nom du propriétaire (identifié dans le fichier identAster) lors de la mise à jour du source. Il est possible de fournir une carte complète, dans ce cas le nom du compte et le nom du propriétaire sont conservés tels quels.
Les autres cartes seront présentées par type de source.

Le mécanisme d'intégration des sources lors de l'ajout d'un fichier vérifie la présence du mot clé COPYRIGHT parmi les commentaires et en cas d'absence un bloc normalisé est ajouté. Si la présence d'un COPYRIGHT est détectée, il est est laissé en l'état dans le source.

2.2 Le source FORTRAN

Le source de type FORTRAN doit bien évidemment suivre la syntaxe du FORTRAN77 mais certaines règles restrictives sont apportées pour *Code_Aster*. Elles sont énoncées dans le document [D2.01.01].

Sur la forme, un ficher FORTRAN d'*Aster* est défini par :

au maximum **72** caractères par ligne,
des caractères autorisés et **conseillés** :
- les lettres majuscules et les chiffres **A-Z 0-9**,
- le caractère blanc ou espace,
- les caractères suivants ,) (* + - = / ' & § ! % : " . ? $ < > _| .

des caractères autorisés mais **déconseillés** (pour des problèmes de transfert entre machines) :
- les lettres minuscules **a-z**.
- les caractères suivants **; é è # \ [ ] @**.

des caractères **interdits** (pour des problèmes de transfert entre machines) :
- tous les autres caractères, y compris le caractère de tabulation,

une ou plusieurs routines FORTRAN par fichier,
une ligne INFO AJOUT/MODIF par routine.

Pour la gestion du projet *Aster* une ligne commentaire -dite "carte INFO"- est présente dans chaque routine FORTRAN. Cette ligne doit être **unique**.

Les développeurs ne doivent la modifier sous aucun prétexte. Elle permet de connaître la bibliothèque à laquelle est rattachée la routine mais sert aussi à vérifier la cohérence des restitutions.

Lorsque l'on veut rajouter une routine il faut mettre la ligne INFO suivante dans la routine :

```
C AJOUT nom_de_bibliotheque
```

C en première colonne,
AJOUT (en majuscules obligatoirement !),
le nom d'une bibliothèque FORTRAN *Aster* où la routine doit être ajoutée (cette bibliothèque doit déjà exister –cf. organisation Source *Aster*- il s'agit du nom du répertoire).

Remarque :

Une routine que l'on veut ajouter ne doit pas déjà exister dans Aster (soit en FORTRAN soit en C soit en FORTRAN90*), en effet une seule librairie est construite à partir de l'ensemble des sources, il ne peut donc exister qu'un seul point d'entrée.*

Une fois la routine ajoutée à *Code_Aster*, la ligne INFO est gérée et mise à jour par l'AGLA. Elle possède alors la forme suivante :

```
C MODIF NOM_BIB DATE jj/mm/aaaa AUTEUR LEFEBVRE J-P.LEFEBVRE
```

Elle contient alors :

le nom de la bibliothèque FORTRAN où se trouve la routine (NOM_BIB),
la date de sa dernière modification (*jj/mm/aaaa*),
le user sur la machine de développement et le nom de l'auteur de la dernière modification.

La tolérance peut porter sur CRP, ou CRS. On peut mélanger les différents types sur une même ligne. On peut mettre plusieurs lignes :

```
C CRP_ xxx CRS_ XXX
```

exemple :
La routine JEVEUO fait partie de la bibliothèque JEVEUX, elle est stockée dans le fichier : /aster/v10/NEW10/bibfor/jeveux/jeveuo.f

```
      SUBROUTINE JEVEUO ( NOMLU , CEL  , JCTAB )
C            CONFIGURATION MANAGEMENT OF EDF VERSION
C MODIF JEVEUX  DATE 26/07/2010   AUTEUR LEFEBVRE J-P.LEFEBVRE
C ======================================================================
C COPYRIGHT (C) 1991 - 2010  EDF R&D                  WWW.CODE-ASTER.ORG
C THIS PROGRAM IS FREE SOFTWARE; YOU CAN REDISTRIBUTE IT AND/OR MODIFY
C IT UNDER THE TERMS OF THE GNU GENERAL PUBLIC LICENSE AS PUBLISHED BY
C THE FREE SOFTWARE FOUNDATION; EITHER VERSION 2 OF THE LICENSE, OR
C (AT YOUR OPTION) ANY LATER VERSION.
C
C THIS PROGRAM IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL, BUT
C WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
C MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE GNU
C GENERAL PUBLIC LICENSE FOR MORE DETAILS.
C
C YOU SHOULD HAVE RECEIVED A COPY OF THE GNU GENERAL PUBLIC LICENSE
C ALONG WITH THIS PROGRAM; IF NOT, WRITE TO EDF R&D CODE_ASTER,
C    1 AVENUE DU GENERAL DE GAULLE, 92141 CLAMART CEDEX, FRANCE.
C ======================================================================
C TOLE CRP_18 CRS_508 CRS_512
C RESPONSABLE LEFEBVRE J-P.LEFEBVRE
```

Si le fichier contient plusieurs routines, ces dernières sont éclatées en autant de sous-fichiers portant le nom de chaque routine.

2.3 Les cartes TOLE et les outils de vérification

Les fichiers de type FORTRAN sont soumis à deux outils de vérification lors de leur restitution.
L'outil CRS (Contrôle des Règles de Syntaxe) permet de s'assurer que le source Fortran répond bien aux préconisations [D2.02.01]. L'outil CRP (Contrôle des Règles de Progammation) permet de s'assurer du respect de bonnes règles en mode d'inter-compilation (en particulier du type et et de la

longueur des arguments des routines). Ces différentes règles peuvent toutefois être contournées, sur justification, en introduisant une carte commentaire particulière qui sera prise en compte par les outils de l'agla.

Liste des erreurs CRS pouvant l'objet d'une carte C TOLE :

```
CRS_230   Module contains no executable statements
CRS_505   Names longer than 6 chars (nonstandard):
CRS_507   Identifiers which are not unique in first six chars
CRS_508   Mixed types equivalenced (not portable):
CRS_512   Variables set but never used
CRS_513   Variables used before set
CRS_602   character variable length exceeds 255
CRS_745   wrong number of arguments for intrinsic function (ABORT, ...)
CRS_1322  Nonstandard syntax: internal file cannot be used with list-
          directed I/O
CRS_1404  Warning: Nonstandard syntax: local array cannot have variable
          size
```

Liste des erreurs CRP pouvant l'objet d'une carte C TOLE :

```
CRP_4   : l'appel n'est tolere que sur justification et dans JEVEUX
CRP_6   : l'appel n'est tolere que sur justification
CRP_7   : ne pas utiliser la declaration EXTERNAL
CRP_12  : l'instruction STOP est interdite
CRP_15  : l'instruction ENTRY est interdite
CRP_18  : l'instruction EQUIVALENCE est interdite
CRP_20  : le nb de lignes d'un sp ne doit pas depasser 500
CRP_21  : le nb d'arguments d'un sp ne doit pas depasser 20
```

2.4 Le source FORTRAN90

Le source de type FORTRAN90 respecte la syntaxe du FORTRAN 90, il est essentiellement utilisé pour assurer l'interface d'appel à la librairie associée au solveur MUMPS. Le source étant analysé par le pré-processeur du compilateur, on utilise les directives pour appeler ou non certaines librairies (MUMPS, MED, PETSC, …). Les fichiers doivent impérativement posséder le suffixe .F.
L'absence d'outil de contrôle de la programmation (CRP/CRS) pour le FORTRAN90 et les éventuels conflits de gestion mémoire nous incitent à limiter l'usage de ce langage au stricte minimum.

La carte INFO est gérée dans les fichiers sources de type FORTRAN90, le bloc COPYRIGHT est traité lors de l'ajout des routines. La carte INFO est du type !& AJOUT ou !& MODIF.

Il n'y a aucune restriction concernant les caractères utilisés dans le corps du fichier source, ni sur la longueur des lignes.

On ne gère qu'une seule unité par fichier, chaque fichier doit contenir une déclaration du type SUBROUTINE. Après ajout dans les fichiers sources officiels, la carte INFO contient MODIF et un bloc de COPYRIGHT est ajouté si aucun COPYRIGHT n'est présent lors de l'ajout.

2.5 Le source CATALO

Contient le source du type catalogue d'*Aster*. Le nom du fichier associé doit posséder un suffixe du type .cata.

Comme pour le FORTRAN certaines restrictions sont apportées, essentiellement à cause des problèmes de transfert de fichier entre machines.

Sur la forme, un fichier CATALOGU est défini par :

au maximum **256** caractères par ligne,

des caractères **autorisés** :
- les lettres majuscules et les chiffres **A-Z 0-9**,
- les caractères suivants ,) (* + - = / ' & § ! % ; é è à _ .

des caractères **interdits** (pour des problèmes de transfert entre machines) :
- tous les autres caractères, y compris le caractère de tabulation,

un ou plusieurs sous-catalogues par fichier,
une ligne `INFO AJOUT/MODIF` en tête de chaque sous-catalogue.

La ligne `INFO` d'un fichier `CATALOGU` est semblable à celle des fichiers `FORTRAN` mais elle a un rôle plus fort. Elle permet de délimiter une unité de type `CATALOGUE` et de déterminer son nom. Il ne doit donc pas y avoir de commande ou de commentaire concernant une unité avant sa ligne `INFO`. La carte `INFO` est du type `%& AJOUT` ou `%& MODIF` .

Pour rajouter une unité de type catalogue il faut mettre la ligne `INFO` suivante en **tête de l'unité** :

```
%& AJOUT nom_d_unite nom_de_catalogue
```

`%` en première colonne,
`&` en deuxième colonne,
`AJOUT` (en majuscules obligatoirement !),
le nom d'un catalogue *Aster* où l'unité doit être ajoutée (ce catalogue doit déjà exister -cf organisation Source *Aster*- il s'agit du nom du répertoire).

Une fois l'unité ajoutée dans les sources de *Code_Aster,* la ligne `INFO` est gérée et mise à jour par l'AGLA. Elle possède alors la forme suivante :

```
 %& MODIF OPTIONS  DATE 05/10/2010   AUTEUR SELLENET N.SELLENET
% RESPONSABLE PELLET
%            CONFIGURATION MANAGEMENT OF EDF VERSION
% ======================================================================
% COPYRIGHT (C) 1991 - 2010  EDF R&D                  WWW.CODE-ASTER.ORG
% THIS PROGRAM IS FREE SOFTWARE; YOU CAN REDISTRIBUTE IT AND/OR MODIFY
% IT UNDER THE TERMS OF THE GNU GENERAL PUBLIC LICENSE AS PUBLISHED BY
% THE FREE SOFTWARE FOUNDATION; EITHER VERSION 2 OF THE LICENSE, OR
% (AT YOUR OPTION) ANY LATER VERSION.
%
% THIS PROGRAM IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL, BUT
% WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
% MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE GNU
% GENERAL PUBLIC LICENSE FOR MORE DETAILS.
%
% YOU SHOULD HAVE RECEIVED A COPY OF THE GNU GENERAL PUBLIC LICENSE
% ALONG WITH THIS PROGRAM; IF NOT, WRITE TO EDF R&D CODE_ASTER,
%    1 AVENUE DU GENERAL DE GAULLE, 92141 CLAMART CEDEX, FRANCE.
% ======================================================================
VARI_ELGA_ELNO
```

Elle contient alors :

le nom du catalogue où se trouve l'unité,
la date de sa dernière modification,
le user de la machine de développement Aster et le nom de l'auteur de la dernière modification.
éventuellement une carte `RESPONSABLE`,
le bloc `COPYRIGHT`,
le nom du catalogue.

Si le fichier contient plusieurs catalogues, ces derniers sont éclatés en autant de sous-fichiers portant le nom de chaque catalogue. Après ajout dans les fichiers sources officiels, la carte `INFO` contient `MODIF` et un bloc de `COPYRIGHT` est ajouté si aucun `COPYRIGHT` n'est présent lors de l'ajout.

2.6 Les sources CATAPY

Contient le source du type catalogue python de commandes d'*Aster*. Le nom du fichier associé doit posséder un suffixe du type `.capy`.

Comme pour les autres sources, certaines restrictions sont apportées, essentiellement à cause des problèmes de transfert de fichier entre machines.

Sur la forme, un fichier CATAPY est défini par :

au maximum **256** caractères par ligne,
des caractères **autorisés** :
- les lettres majuscules et minuscules et les chiffres **A-Z, a-z, 0-9**,
- les caractères suivants ,) (* + - = / \' & $ < >§ ! %: ; é è à _ . ",][}{#@|?^~ç
- les lettres minuscules ou majuscules accentuées,

des caractères **interdits** (pour des problèmes de transfert entre machines) :
- tous les autres caractères, y compris le caractère de tabulation,

un ou plusieurs sous-catalogues par fichier,
une ligne INFO AJOUT/MODIF en tête de chaque sous-catalogue.

La ligne INFO d'un fichier CATALOGU est semblable à celle des fichiers FORTRAN mais elle a un rôle plus fort. Elle permet de délimiter une unité de type CATALOGUE et de déterminer son nom. 3 sous-répertoires sont utilisés pour stocker les différents catalogues :

ENTETE : dont le contenu sera placé en tête du catalogue,
COMMUN : qui permet de mutaliser des blocs de commandes,
COMMANDE : qui contient la définition des commandes.

Pour rajouter une unité de type catalogue il faut mettre la ligne INFO suivante en **tête de l'unité** :

```
#& AJOUT ENTETE
import Accas

#& AJOUT COMMUN
def C_NOM_CHAM_INTO() : return  ("ACCE",    #COMMUN#

#& AJOUT COMMUN
def C_TYPE_CHAM_INTO( type_cham=None ) : #COMMUN#

# AJOUT COMMANDE
AFFE_CARA_ELEM=OPER(nom="AFFE_CARA_ELEM",op=  19,sd_prod=cara_elem,
```

La détermination du nom de catalogue est effectuée à l'aide d'une expression régulière du type :

```
"#&[ ]+MODIF[ ]+[A-Za-z]+[ ]+DATE[ ]+[0-9][0-9]"/"[0-9][0-9]"/"[0-9][0-9]+
[ ]+"AUTEUR"[ ]+[A-Za-z.0-9]+
|#&[ ]+AJOUT[ ]+[A-Za-z]+[ ]*
|^[A-Za-z_0-9]*[ ]*[=][ ]*(OPER|PROC|MACRO|FORM)[ ]*\(
|import Accas
|def[ ]+[A-Za-z_]+\([A-Za-z_= ]*\)[ ]*(:){1}[ ]*(return)*[ ]*(SIMP|FACT|
BLOC|.*#COMMUN#){1}
```

Les sources PYTHON

Contient le source du type python utilisé pour le superviseur de commandes ou pour construire des macros-commandes. Le nom du fichier associé doit posséder un suffixe du type .py.

Comme pour les autres sources, certaines restrictions sont apportées, essentiellement à cause des problèmes de transfert de fichier entre machines.

Sur la forme, un fichier PYTHON est défini par :

au maximum **1024** caractères par ligne,
des caractères **autorisés** :
- les lettres majuscules et minuscules et les chiffres **A-Z, a-z, 0-9**,

- les caractères suivants ,) (* + - = / \' & $ < >§ ! %: ; é è à _ . ",][}{#@|?^~ç
- les lettres minuscules ou majuscules accentuées,

des caractères **interdits** (pour des problèmes de transfert entre machines) :

- tous les autres caractères, y compris le caractère de tabulation,

un ou plusieurs sous-fichiers par fichier,
une ligne `INFO AJOUT/MODIF` en tête de chaque sous-fichier.

La carte `INFO` est du type `#@ AJOUT` ou `#@ MODIF` .

Attention :

Lors de la création d'un nouveau répertoire dans `bibpyt`, l'administrateur doit créer auparavant le fichier `__init__.py` (il ne contient que la carte `info` et le bloc `COPYRIGHT`).

2.7 Le source CASTEST

Le source de type `CASTEST` suit la même syntaxe que les fichiers `CATALOGU` avec comme particularités :

la première commande doit être :

```
DEBUT ( CODE : ( NOM : 'nom_du_cas_test'
```

Pour l'AGLA une restriction est apportée par rapport à la syntaxe reconnue par le superviseur. Les mots-clés `DEBUT, CODE et NOM` ainsi que le nom du cas-test doivent être **sur une seule ligne**.
la dernière procédure doit être :

```
FIN () ;
```

Le fichier contenant le cas-test doit avoir le nom suivant (en minuscules) : `nom_du_cas_test.comm`

Il peut exister un ou plusieurs fichiers de commandes supplémentaires nommés de la façon suivante :

```
nom_du_cas_test.com[0-9]
```

Mais les carte `INFO` et `RESPONSABLE` figurent uniquement dans le fichier `.comm`.

Les fichiers associés (paramètres, maillages et autres données) prennent alors les noms suivants :

nom_du_cas_test .para	paramètres d'exécution du cas-test
nom_du_cas_test .mail	maillage de type Aster
nom_du_cas_test .msup	maillage de type SuperTab (IDEAS MASTER SERIES)
nom_du_cas_test .mgib	maillage de type GIBI
nom_du_cas_test .msh	maillage de type gmsh
nom_du_cas_test .mmed	maillage de type MED
nom_du_cas_test .datg	données pour créer un maillage de type GIBI ou autre
nom_du_cas_test .ensi/	répertoire de données de type EnSight
nom_du_cas_test . *ul*	fichier dont l'extension est le numéro d'unité logique FORTRAN qui doit lui être associé pour être lu par Aster

Lorsque l'on restitue un fichier de type maillage `GIBI` (`.mgib`), il faut obligatoirement restituer en même temps un fichier de type données `GIBI` (`.datg`) permettant de reconstituer le fichier `.mgib`.

Le nom du cas-test doit respecter la règle de nommage suivante -avec au maximum 8 caractères- :

des lettres,
des chiffres,
une lettre.

Sur la forme, un fichier de type `CASTEST` est défini par :

au maximum **256** caractères par ligne,

des caractères **autorisés** :
- les lettres majuscules, minuscules et les chiffres **A-Z a-z 0-9**,
- les caractères suivants ,) (* + - = / \' & $ < >§ ! %: ; é è à _ . ",][}{#@|?^~ç

des caractères **interdits** (pour des problèmes de transfert entre machines) :
- tous les autres caractères,

un seul cas-test par fichier,
une ligne `INFO AJOUT/MODIF` et une ligne `INFO TITRE`.

Les fichiers de type `CASTEST` comportent deux lignes `INFO`. Une ligne `AJOUT/MODIF` semblable aux précédentes et une ligne `TITRE` qui doit décrire la fonction du cas-test.

Pour ajouter un cas-test à une version d'*Aster* il faut mettre les lignes `INFO` suivantes en tête de l'unité :

```
% AJOUT
% TITRE Explication du role du cas-test
```

Avec toujours le `%` en première colonne et `AJOUT` et `TITRE` en **majuscules**.

Lors d'une modification il faut bien évidemment mettre à jour la ligne `TITRE` si cela est nécessaire.

Une fois le cas-test ajouté au source de *Code Aster* la ligne `INFO AJOUT/MODIF` est gérée et mise à jour par l'AGLA. Elle possède alors la forme suivante :

```
% MODIF DATE jj/mm/aaaa AUTEUR  GENIAUT S.GENIAUT
```

Elle contient alors :

la date de sa dernière modification,
le `user` de la machine de développement et le nom de l'auteur de la dernière modification.

Exemple de fichier de commande :

```
# MODIF  DATE 03/01/2011   AUTEUR GENIAUT S.GENIAUT
# RESPONSABLE SFAYOLLE S.FAYOLLE
# TITRE  : ENDOMMAGEMENT D UNE PLAQUE PLANE SOUS SOLLICITATIONS VARIEES
# ssns106g.para = tps_job 120 mem_job 64Mo ncpus 1 liste_test S
#            CONFIGURATION MANAGEMENT OF EDF VERSION
# ==================================================================
# COPYRIGHT (C) 1991 - 2011  EDF R&D                  WWW.CODE-ASTER.ORG
# THIS PROGRAM IS FREE SOFTWARE; YOU CAN REDISTRIBUTE IT AND/OR MODIFY
...
DEBUT(CODE=_F(NOM='SSNS106G',
              NIV_PUB_WEB='INTERNET',),) ;
```

Lors de l'ajout ou de la modification du fichier de commande, le fichier `.para` contenant les paramètres de soumission associés est recopié après le titre.
Le fichier .para contient des mots-clés et des valeurs associées permettant de fixer :
`tps_job` : temps CPU de l'exécution en secondes,
`mem_job` : limite mémoire en Mégaoctets Mo
`ncpus` : nombre de CPUs OpenMP
`nproc_mpi` : nombre de CPUs MPI
`nnoeu_mpi` : nombre de noeuds MPI
`liste_test` : groupe auquel est affecté le test, S (liste complète).

2.8 Le source C

Sur la forme, un fichier `C` d'*Aster* est défini par :

une ou plusieurs fonctions `C` (formant alors un groupe cohérent) par fichier (avec les instructions préprocesseur la concernant -c'est-à-dire sans référence à un fichier `header` personnel-),

des instructions du type `#ifdef` permettant une compilation conditionnelle suivant les plates-formes cibles,
un nom de fonction ou de groupe de fonctions doit avoir au maximum 8 caractères,
extension du fichier : `.c` ou `.h`,
une ligne `INFO AJOUT/MODIF` par fonction.

Pour la gestion du projet *Aster* une ligne commentaire -dite "ligne `INFO`"- est présente dans chaque fonction C. Cette ligne doit être **unique**.

Les développeurs ne doivent la modifier sous aucun prétexte. Elle permet de connaître la bibliothèque à laquelle est rattachée la fonction mais sert aussi à vérifier la cohérence des restitutions.

Lorsque l'on veut rajouter une fonction il faut mettre la ligne `INFO` suivante dans le fichier (sur une seule ligne) :

```
/* AJOUT nom_fonction nom_de_bibliotheque */
```

`/*` en première colonne,
`AJOUT` (en majuscules obligatoirement !),
le nom de la fonction,
le nom d'une bibliothèque C *Aster* où la fonction doit être ajoutée (cette bibliothèque doit déjà exister -cf organisation Source *Aster*-),
`*/` pour fermer le commentaire sur la même ligne.

Remarque :

Une fonction que l'on veut ajouter ne doit pas déjà exister dans Aster (soit en FORTRAN , soit en FORTRAN90, soit en `C`).

Une fois la fonction ajoutée au *Code_Aster*, la ligne `INFO` est gérée et mise à jour par l'AGLA. Elle possède alors la forme suivante :

```
/* MODIF nom_fonc NOM_BIB DATE jj/mm/aaaa AUTEUR SELLENET N.SELLENET */
```

Elle contient alors :

le nom de la fonction ou du groupe de fonctions,
le nom de la bibliothèque `C` où se trouve la fonction,
la date de sa dernière modification,
le `user` de la machine de développement et le nom de l'auteur de la dernière modification.

Note :

Pour qu'une *fonction* `C` puisse être appelée en `FORTRAN` il faut respecter certaines conventions dépendant du type de plate-forme. C'est le rôle des fonctions définies dans le fichier `include/definition.h` et `include/aster.h` qui permettent de réordonner les arguments en fonction de la plate-forme.

2.9 Le source CMAT

Le source `CMAT` est un fichier de commandes Aster respectant certaines règles strictes. Un fichier de ce type permet de créer une structure de donnée matériau. Ce fichier sert de donnée à la commande `INCLUDE_MATERIAU` [U4.43.02].

La syntaxe de la carte `INFO` pour un ajout est :

```
% AJOUT
```

Note :

Comme ce type de fichier ne dépend que de la responsabilité d'une seule personne, le mécanisme de notation n'a pas été activé pour ce type de source. Par contre, il peut être intéressant de lui adjoindre systématiquement une carte `RESPONSABLE` .

2.10 Le source DATG

Le source DATG est un fichier de données GIBI. Il sert, en général, de donnée à une macro-commande (pour ASPIC et ASCOUF par exemple). Cette fonction a été étendue et ce type de fichier peut contenir les données nécessaire à la construction d 'un maillage gmsh.

La syntaxe de la carte INFO pour un ajout est :

```
* AJOUT
```

Note :

Comme ce type de source ne dépend que de la responsabilité de peu de personnes, le mécanisme de notation n'a pas été activé. Par contre, il peut être intéressant de lui adjoindre systématiquement une carte RESPONSABLE .

3 Les sources de référence d'*Aster* sur la machine de développement

3.1 Règle de dénomination des fichiers et répertoires

Règle : Tous les noms de fichier ou de répertoire sont en minuscules.

Exception (confirmant la règle précédente) :
Les répertoires relatifs à la version d'*Aster* sont en **majuscules** (NEW9, OLD9, STA10, ...)

3.2 Organisation des répertoires

Tous les fichiers de référence d'*Aster* ne sont modifiables que par l'administrateur, mais accessibles en lecture par tous.

L'organisation est identique pour toutes les versions. Les différents états d'une version (NEWX, STAX et OLDX) sont regroupés dans un répertoire : /aster/vX/.
Les versions parallèles MPI ou instrumentées en vue du « profiling », sont construites à partir des mêmes fichiers sources mais disposent d'un sous-répertoire dédié (mpi, prof). Par commodité les exécutables sont déposés au même niveau que la version standard. Un répertoire est dédié aux fichiers de cas tests, les données sont des liens pointant sur la version séquentielle.

/aster/v9/ NEW9/ OLD9/ STA9/		Sources des versions 9
/aster/v10/..		
	NEW10/	*Sources de la version NEW10*
	asteru	*Exécutable Aster compilé en version optimisée*
	asterd	*Exécutable Aster compilé en version instrumentée pour le déboggage*
	astest/	*Cas-tests de la version*
	bibc/bibli/fonction.c bibc/include/fic.h	*Toutes les fonctions C classées par bibliothèques*
	bibfor/bibli/routine.f	*Toutes les routines FORTRAN par bibliothèques*

		(sauf fermetur)
	bibpyt/bibli/fonction.py	*Toutes les fonctions PYTHON par bibliothèques*
	bibobj.a	*Bibliothèque ar FORTRAN (sauf fermetur) et C*
	bibdbg.a	*Bibliothèque ar FORTRAN (sauf fermetur) et C en mode debug*
	bibobj_old.a	*Bibliothèque ar FORTRAN et C de la mise à jour précédente*
	bibdbg_old.a	*Bibliothèque ar FORTRAN et C en mode debug de la mise à jour précédente*
	catalo/ ../catalogue/ ../sous-catalogue.cata	*Les sous-catalogues classés par catalogues*
	catapy/ ../catalogue/ ../sous-catalogue.capy	*Les commandes classées par catalogues*
	catobj/ ../commande/ ../../cata.py ../../cata.pyc ../cata_ele.pickled ../elements	*Les catalogues compilés*
	config.txt	*Fichier décrivant l'environnement de construction de la version*
	crd/ *v.sv.n*	*Comptes rendus des opérations de mise à jour*
	crp/	*Informations pour l'inter-compilation (outil CRP)*
	datg/	*Données GIBI*
	diffc/bibli/fonction.c	*Différences anciennes versions / version de référence C*
	diffcat/catalogue/sct.cata	*Différences anciennes versions / version de référence COMMANDES*
	diffpyt/*catalogue*/sct.*capy*	*Différences anciennes versions / version de référence COMMANDES*
	diffconf/	*Différences anciennes versions/version de référence du fichier de configuration*
	diffct/	*Différences anciennes versions/version de référence des cas-tests*
	diffdatg/	*Différences anciennes versions/version de référence des DATG*
	diffmat/	*Différences anciennes versions/version de référence des CMAT*
	bibpyt/*bibli* /*fct.py*	*Différences anciennes versions/version de référence du PYTHON*
	diffsub/*bibli*/routine.f	*Différences anciennes versions/version de référence (sauf fermetur)*
	diff90/*bibli*/*routine.F*	*Différences anciennes versions/version de référence fortran90*
	fermetur/ *routine*	*Les routines FORTRAN de fermeture*

	fermobj	*Bibliothèque ar FORTRAN de fermeture*
	fermdbg	*Bibliothèque ar FORTRAN de fermeture en mode debug*
	fermold	*Ancienne bliothèque ar FORTRAN de fermeture*
	fort/ *routine*	*Toutes les routines FORTRAN (sauf fermetur) sous forme de liens*
	histor/ *v.sv.n*	*Historiques des versions*
	materiau/	*Catalogue matériau*
	versio	*Version et sous-version*

Contenu du répertoire /aster/v10/ astest/	
liste	*Fichier contenant la liste de tous les cas-tests de la version et leur répartition dans les différentes listes (restreinte, complète,) performance,validation avec les différents paramètres de passage (temps et mémoire)*
liste_ct.debug	*Liste complète de cas-tests en mode debogage (asterd)*
liste_ct.long	*Liste longue de cas-tests (finmaj)*
liste_ct.miss	*Liste de cas-tests faisant appel au logiciel MISS3D*
liste_ct.mpi	*Liste de cas-tests parallèles utilisant MPI*
liste_ct.mpro	*Liste de cas-tests parallèles utilisant OpenMP*
liste_ct.perf	*Liste de cas-tests dits de performance*
liste_ct.perfw	*Liste de cas-tests dits de performance à lancer le week-end*
liste_ct.rest	*Liste restreinte de cas-tests (asrest)*
liste_ct.tout	*Liste complète de cas-tests (finmaj)*
liste_ct.vali	*Liste de cas-tests issus de la validation indépendante*
liste_ct.zmat	*Liste de cas-tests faisant appel au logiciel ZMAT*
*.code	*Trace des commandes et des mots-clés utilisés lors de l'exécution*
*.comm	*Fichiers de commandes Aster*
*.com[0-9]	*Commandes Aster à enchaîner lors d'une exécution avec plusieurs fichiers (poursuites)*
*.datg	*Données GIBI pour le fichier .mgib correspondant, ou autre données géométriques.*
*.ensi/	*Répertoire de données EnSight*
*.mail	*Maillage au format Aster*
*.mess	*Fichier de messages*
*.mgib	*Maillage au format GIBI*
*.mmed	*Maillage au format MED*
*.msh	*Maillage au format GMSH*
*.msup	*Maillage au format IDEAS*
*.para	*paramètres d'exécution*
*.resu	*Fichier de résultat*
*. *ul*	*Fichier à associer à l'unité logique FORTRAN ul*

4 La gestion des unités de source : les fichiers UNIGEST

Les fichiers de référence d'*Aster* n'étant pas modifiables par les développeurs les ordres UNIGEST permettent de supprimer une unité de code source ou de changer sa bibliothèque de rattachement. Ces ordres sont regroupés dans un fichier (ils ne sont pas mélangeables avec du code source).

Les ordres UNIGEST disponibles :

FORSUPPR *nom_de_routine nom_de_bibli*
F90SUPPR *nom_de_routine nom_de_bibli*
pour les routines que l'on souhaite supprimer, en indiquant leur nom et la bibliothèque d'appartenance,
CSUPPR *nom_de_fonction nom_de_bibli*
pour les fonctions C que l'on souhaite supprimer, en indiquant leur nom et la bibliothèque d'appartenance. Cette commande fonctionne aussi pour les fichiers d' « include » .h,
CATSUPPR *nom_de_sous_catalogue nom_de_catalogue*
pour les éléments de catalogues (de commande ou d'élément) que l'on souhaite supprimer, en indiquant leur nom et le catalogue d'appartenance,
PYSUPPR *nom_de_fonction nom_de_bibli*
pour les fonctions PYTHON que l'on souhaite supprimer en indiquant leur nom et la bibliothèque d'appartenance,
TESSUPPR *nom_de_cas_test* ou *nom_de_fichier_de_cas_test*
pour les cas-tests que l'on souhaite supprimer, en indiquant leur nom. On peut supprimer un des fichiers constituant le test (un maillage, un fichier de commandes supplémentaire, etc ..), on indique alors le nom complet du fichier,
FORDEPLA *nom_de_routine bibli1 bibli2*
F90DEPLA *nom_de_routine bibli1 bibli2*
pour les routines que l'on souhaite déplacer d'une bibliothèque *bibli1* vers une bibliothèque *bibli2*.

Il est à noter que pour ce type de fichier :

les ordres UNIGEST doivent être en **majuscules** (pour les noms d'unité et de bibliothèque cela n'a pas d'importance),
il n'y a pas de notion de commentaire,
pour qu'il n'y ait pas d'incohérence il ne peut pas y avoir plus d'un ordre UNIGEST concernant la même unité,
il ne doit pas y avoir de ligne vide.

Les possibilités de déplacement et de suppression ne sont pas données pour tous les types de source car certaines actions sont considérées comme marginales. Dans le cas où une action de ce type est néanmoins nécessaire, elle est réalisée 'à la main' par l'administrateur des sources.

5 Généralités sur les outils de l'AGLA

Tous les outils de l'AGLA peuvent êtres appelés "en interactif" dans un "script shell" ou en "batch".
Il est fortement recommandé d'utiliser l'interface d'accès astk qui enrobe l'appel de ces différentes commandes et s'appuie sur le profile d'exécution pour renseigner les différents paramètres.

5.1 Le code de retour des outils de l'AGLA

Tous les outils de l'AGLA retournent un code d'erreur. Ce code est affiché dans le fichier de sortie standard et dans le fichier des messages des outils de l'AGLA. Il peut également être récupéré et testé (la syntaxe dépend du shell utilisé -voir plus loin-).

Signification du code d'erreur :

0 si tout est OK,
2 si un message d'alarme a été émis,
4 si une erreur grave a été rencontrée.

5.2 Pour appeler un des outils à partir du Bourne ou du Korn-Shell

Pour connaître le shell utilisé faire :

```
echo $SHELL
   si /bin/sh   ->  Bourne-Shell
   si /bin/csh  ->  C-Shell
   si /bin/ksh  ->  Korn-Shell
   si /bin/bash ->  Bash-Shell
```

Pour appeler un des outils en Bourne-Shell ou en Bash_shell:

```
/aster/agla/bin/nom_outil parametres
```

Pour ne pas avoir à donner tout le "path" il faut rajouter /aster/agla/bin/ à la variable PATH dans le fichier $HOME/.profile :

```
PATH=$PATH:/aster/agla/bin
export PATH
```

Pour récupérer le code de retour d'un outil de l'AGLA (juste après son exécution) :

```
echo $?
```

5.3 Pour appeler un des outils à partir du C-Shell

Pour appeler un des outils en C-Shell : C'est le shell par défaut sur la machine de développement Aster:

```
/aster/agla/bin/nom_outil parametres
```

Pour ne pas avoir à donner tout le "path" il faut rajouter /aster/agla/bin/ à la variable path dans le fichier $HOME/.cshrc :

```
set path=($path /aster/agla/bin)
```

On peut alors invoquer un outil de l'AGLA par :

```
nom_outil parametres
```

Pour récupérer le code de retour d'un outil de l'AGLA (juste après son exécution) :

```
echo $status
```

5.4 Les paramètres

5.4.1 Identification des paramètres (valeurs par défaut)

Certains des paramètres de l'AGLA sont obligatoires d'autres sont optionnels. Les paramètres optionnels seront représentés entre crochets [parametre_optionnel]. La reconnaissance des paramètres par les outils de l'AGLA s'effectue sur leur position dans la liste des paramètres. Donc si l'on veut modifier la valeur du troisième paramètre par défaut mais pas celle des deux premiers il faut quand même préciser leurs valeurs.

Exemple :

Avec `outil para1 para2 [ para3 [ para4 [ para5 ] ] ]`

Si l'on veut modifier la valeur par défaut de *para4* mais pas de *para5* et *para3* il faudra quand même invoquer :

```
outil val_para1 val_para2 val_para3 val_para4
```

5.4.2 Noms de fichier et de répertoire

Lorsque le paramètre est un nom de fichier ou de répertoire, il peut être indiqué en relatif (par rapport au répertoire courant) ou avec le chemin d'accès complet. Il faut cependant être prudent avec la notation relative lors d'une utilisation en batch (le répertoire courant est alors un répertoire temporaire).

6 Identification d'un développeur

Presque tous les outils de l'AGLA ont besoin de connaître l'identité du développeur appelant. Le mécanisme retenu se base sur le `user` de l'utilisateur sur la machine de développement (serveur de référence).

Donc pour que les opérations soient effectuées à **son** profit, il faut invoquer les outils de l'AGLA à partir d'une session `UNIX` ouverte avec **son** `userid`.

Pour pouvoir utiliser les outils de l'AGLA il faut y être autorisé. Contacter l'administrateur si un des outils ne reconnaît pas le `user` utilisé. La présence du `user` dans le ficher `/aster/agla/identAster` donne les autorisations nécessaires, il faut aussi que être reconnu de l'outil de gestion des fiches REX (retour d'expérience).

7 Le système de verrouillage des outils de l'AGLA

Lors de certaines opérations comme la mise à jour d'une version ou sa stabilisation, l'accès aux outils de l'AGLA est interdit car la référence n'est pas dans un état cohérent.

Le développeur est prévenu par un message sur le verrouillage de l'AGLA avec des informations du type :

```
aster M.ADMINISTRATEUR le 03/12/1998 a 07:00:01 avec majnew.exe
```

8 La notation des sources *Aster*

Pour pouvoir modifier, ou ajouter du code source à *Aster* il faut le noter au préalable. La notation concerne une unité pour une seule version.

Une notation n'est valable que 3 mois. Elle ne peut être résiliée que par le développeur l'ayant notée, ou exceptionnellement par l'Administrateur (pour cause de vacances, ...).

Plusieurs développeurs peuvent noter une unité, cependant seul le développeur l'ayant notée le premier (on parle alors de notation prioritaire) est autorisé à restituer cette unité.

Lorsqu'une unité est restituée, la mise à jour supprime toutes les notations concernant cette unité (la notation prioritaire comme les notations non prioritaires).

Les notations sont regroupées par types :

les unités compilées `FORTRAN, FORTRAN90` et `C`,
`CATALOGUE` (commandes python et éléments),
`PYTHON`,
`CASTEST`.

Exemple de notations :

```
MMELIN                          NEW10   19:21:04   05/10/2010 desoza T.DESOZA
NMCEAI                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
NMCENI                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
NMCESE                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
NMCETA                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
NMDOPI                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
NUEQCH                          NEW10   08:55:48   05/11/2010 abbas  M.ABBAS
IMPDEF                          NEW9    15:07:11   05/11/2010 micol  A.MICOL
ALGOCG                          NEW10   13:12:12   10/11/2010 abbas  M.ABBAS
ALGOCL                          NEW10   13:12:12   10/11/2010 abbas  M.ABBAS
ALGOCO                          NEW10   13:12:12   10/11/2010 abbas  M.ABBAS
ALGOCP                          NEW10   13:12:12   10/11/2010 abbas  M.ABBAS
```

Certains types de source ont été exclus du mécanisme de notation car ils ne dépendent que d'un nombre restreint de responsables, il n'y a donc pas de risque de collision entre plusieurs développements simultanés. C'est le cas actuellement des sources CMAT et DATG.

8.1 asno : notation d'unités

8.1.1 Fonctionnalité

Cette commande permet de noter des unités de source *Aster*. asno ne peut noter à la fois qu'un seul type de fichier (pas de mélange possible).

Pour chaque notation effectuée un message informe si c'est une notation :

prioritaire (qui permet de restituer l'unité),
non prioritaire (qui ne donne aucun droit) avec la liste de tous les développeurs l'ayant également notée.

8.1.2 Mode d'appel

asno mon_fichier [version [message]]

mon_fichier désigne un fichier étant soit du type FORTRAN ou FORTRAN90 ou C ou CATALOGUE ou PYTHON ou UNIGEST ou CASTEST.

version est un paramètre qui porte le nom de la version de référence concernée (NEW2 à NEW10). La valeur prise par défaut est actuellement NEW10.

message désigne un fichier en sortie où seront inscrits les messages d'alarmes et d'erreurs émis par asno. Le nom pris par défaut est : message_asno.

8.2 asdeno : dénotation d'unités

8.2.1 Fonctionnalité

Dénoter des unités de source à partir d'un fichier source.

8.2.2 Mode d'appel

asdeno *mon_fichier [version [message]]*

mon_fichier désigne un fichier contenant soit du code FORTRAN, FORTRAN90 ou C, soit du code CATALOGUE, soit du type UNIGEST ou CASTEST.

version est un paramètre qui porte le nom de la version de référence concernée (NEW2 à NEW10). La valeur prise par défaut est actuellement NEW10.

message désigne un fichier en sortie où seront inscrits les messages d'alarmes et d'erreurs émis par asdeno. Le nom pris par défaut est : message_asdeno.

8.3 xasdeno : dénotation d'unités à partir de leur nom

8.3.1 Fonctionnalité

Dénoter des unités de source à partir d'un fichier contenant des noms de module et d'un type de module.

8.3.2 Mode d'appel

```
xasdeno mon_fichier type [ version [ message ] ]
```

mon_fichier désigne un fichier contenant un nom de module par ligne les noms de module doivent faire référence à des unités du même type.

version est un paramètre qui porte le nom de la version de référence concernée (NEW2 à NEW10). La valeur prise par défaut est NEW10.

type : pour désigner le type d'unité à dénoter parmi f (FORTRAN), F (FORTRAN90), c, h, cata (catalogue d'élément), capy (catalogue de commande), test (cas-test).

message désigne un fichier en sortie où seront inscrits les messages d'alarmes et d'erreurs émis par xasdeno. Le nom pris par défaut est : message_xasdeno.

8.4 asqui : liste de notation d'unités

8.4.1 Fonctionnalité

Connaître les développeurs qui ont noté des unités de source (par rapport à un fichier donné).

8.4.2 Mode d'appel

```
asqui mon_fichier [ message ]
```

mon_fichier désigne un fichier contenant soit du code FORTRAN ou FORTRAN90 ou C, soit du code PYTHON, soit du code CATALOGUE , soit du type UNIGEST ou CASTEST.

message désigne un fichier en sortie où seront inscrits les messages d'alarmes et d'erreurs émis par asqui. Le nom pris par défaut est : message_asqui.

8.5 asquit : liste de toutes les notations

8.5.1 Fonctionnalité

Connaître toutes les unités de source notées. Les notations sont recopiées dans le fichier de messages.

8.5.2 Mode d'appel

```
asquit [ message ]
```

message : définition identique à asno. Le nom pris par défaut est : message_asquit.

9 La restitution des sources *Aster*

Après la notation, les modifications et ajouts, il faut restituer le source dans la version de référence d'*Aster*.

Pour un développement, on pourra procéder en trois étapes :

asverif : vérification du source (facultative)
pre_eda : présentation des développements (obligatoire)
asrest : restitution du source (obligatoire)

Lorsque des développements sont dépendants il faut pouvoir les rendre simultanément afin de tester le code. Cela oblige un des développeurs à centraliser les développements et à effectuer la restitution pour les autres développeurs. Les développeurs doivent alors dénoter leurs sources et les fournir celui qui centralise les développements.

9.1 asverif : vérification de source

9.1.1 Fonctionnalité

Vérifier qu'un ensemble de sources peut être introduit dans une version de référence du *Code Aster*.

Il y a vérification :

que les contraintes énoncées en [D1.02.01 §2] sont respectées,
que les règles décrites dans le document [D2.02.01] sont respectées à l'aide de deux outils : CRS (Contrôle de Règles de Syntaxe) et CRP (Contrôle de Règles de Programmation),
que les unités à ajouter n'existent pas déjà et que celles à modifier existent.

Cet outil effectue une partie des tâches réalisées par le module de restitution de source asrest qui, contrairement au module présent, doit être utilisé obligatoirement par toute personne souhaitant restituer un développement.

asverif n'effectue pas de notation sur les sources transmis.

9.1.2 Mode d'appel

```
asverif repertoire [ version [ message ] ]
```

repertoire est le nom d'un répertoire qui contient un répertoire par développeur désirant restituer du source. Les noms de ces répertoires doivent être composés à partir du user développeur sur la machine de développement Aster pour déterminer l'auteur du source restitué. Dans chacun de ces répertoires on peut trouver, suivant les cas, les fichiers :
repertoire/user(s)/

histor	dans lequel on trouve les numéros des fiches REX traitées (**obligatoire**).
fortran	dans lequel se trouve du source FORTRAN.
f90	dans lequel se trouve du source FORTRAN90.
catalogu	dans lequel se trouve des éléments de catalogues d'éléments.
catalopy	dans lequel se trouve des éléments de catalogues de commandes python.
python	dans lequel se trouve des fichiers sources python.
unigest	dans lequel on indique les unités de source que l'on souhaite déplacer d'une bibliothèque à une autre ou supprimer.
test/	répertoire dans lequel se trouvent des fichiers de commande de cas-tests *.comm , et éventuellement *.para , *.mail , *.ensi/ , …
c/	répertoire dans lequel se trouvent des fichiers de fonctions C *.c *.h .
materiau/	*.NOMI , *.REFE , *.MINI , *.MAXI : fichiers catalogue matériau. Les noms doivent être en majuscules.
datg/	*.datg : fichiers de données GIBI.

message : définition identique à asno. Par défaut : message_asverif.

version est un paramètre qui porte le nom de la version de référence concernée (NEW2 à NEW10). Par défaut : NEW10 actuellement.

9.1.3 Les codes retours 2

Les outils de l'AGLA émettent des alarmes, mais toutes les alarmes n'ont pas la même importance. Pour que l'EDA puisse valider certaines alarmes pouvant avoir des conséquences sur la bonne évolution du code, elles émettent un code retour 2. Les messages d'alarme ayant émis un code retour 2 sont rappelées en fin de phase **asverif**.

Cas qui conduisent à l'émission d'un code retour 2 :

ajout, modification, suppression de carte TOLE dans le source FORTRAN,
utilisation de caractères déconseillés,
ajout, modification d'une carte RESPONSABLE,
restitution par un autre développeur que le responsable de l'unité.

9.2 pre_eda et asrest : restitution de source

9.2.1 Fonctionnalité

Vérifier qu'un ensemble de sources peut être introduit dans une version de référence du *Code Aster* et préparer la mise à jour éventuelle.

Ces deux outils effectuent un certain nombre de vérifications, effectuent les compilations nécessaires (FORTRAN, FORTRAN90, C et catalogues) puis il lancent le passage des cas-tests ajoutés ou modifiés (pre_eda) ou bien les cas-tests ajoutés et la liste restreinte de référence dans un job (asrest).

L'outil pre_eda prépare la restitution et permet de présenter les développement lors de la réunion hebdomadaire de l'équipe de développement.

Cet outil doit être obligatoirement utilisé par toute personne souhaitant restituer un développement.

Tâches principales effectuées par pre_eda et asrest :

vérifications effectuées par asverif,
vérification de la cohérence avec la version de référence, pour détecter une modification de l'unité à vérifier entre l'extraction du source de la référence et la restitution envisagée,
notation des sources à restituer si cela n'est pas déjà fait,
mise à jour des lignes INFO (la date mentionnée dans la ligne INFO est donc la date de l'asrest et non celle de la mise à jour d'*Aster*),
compilation du FORTRAN et du FORTRAN90 et du C en mode non debug (même compilation que l'exécutable utilisateur),
création d'un nouvel exécutable *Aster* (édition des liens) avec les bibliothèques objets,
compilation des catalogues (avec le nouvel exécutable pour les éléments),
si toutes les étapes précédentes se sont bien déroulées, lancement des tests fournis (pre_eda) ou bien lancement des tests de la liste restreinte plus les cas-tests restitués (asrest).

Attention :

> *Pour qu'une restitution soit prise en compte lors de la mise à jour il faut que asrest renvoie le code d'erreur 0 . Si le code d'erreur est 2 , il faut justifier en réunion de présentation (EDA) les alarmes). Les messages d'erreurs associés à un code retour 2 sont rappelés juste avant le lancement des cas-tests. Si le code d'erreur est supérieur à 2 , la restitution est ignorée par la mise à jour. Un code retour 4 peut-être levé sous certaines conditions après accord lors de la présentation en EDA.*

9.2.2

9.2.3 Mode d'appel

asrest *repertoire* [*version* [*message* [classe_job]]]

repertoire est le nom d'un répertoire. Il a la même fonction que dans asverif. On peut donc y trouver, suivant les cas, dans des répertoires user sur la machine de développement, les fichiers :

histor,
fortran,
f90
catalogu,
catalopy
python
unigest,
c/

test/
- *.comm : Fichier de commande du cas-test,
- *.com[0-9] : Fichier de commande du cas-test enchaîné,
- *.mail : Maillage *Aster* du cas-test,
- *.mgib : Maillage GIBI du cas-test,
- *.datg : Données GIBI du cas-test .datg (obligatoire s'il y a un .mgib).
- *.msup : Maillage I-DEAS du cas-test,
- *.msh : Maillage GMSH du cas-test,
- *.mmed : Maillage MED du cas-test,
- *.para : Paramètres d'exécution du cas-test.
- *.ensi/: Données EnSight du cas-test.
- *.miss/: Données MISS 3D du cas-test.
- *.*ul* : Fichier à associer à l'unité logique FORTRAN *ul*.

materiau/
- *.NOMI : valeurs nominale du matériau,
- *.REFE : valeurs de référence du matériau,
- *.MINI : valeurs minimales du matériau,
- *.MAXI : valeurs maximales du matériau.

datg/ Données GIBI pour les macro-commandes ou autres.
- *.datg : fichier de données GIBI ou autre.

Par rapport à asverif, on doit trouver dans le répertoire test/ tous les fichiers, nouveaux ou modifiés, concernant les cas-tests ajoutés ou modifiés.

version : est un paramètre qui porte le nom de la version de référence concernée (NEW2 à NEW10).

message : définition identique à asno. La valeur prise par défaut est message_asrest.

classe_job : classe des jobs sur la machine de développement soumis par as.tout.

Les fichiers générés à partir de cette restitution, qui serviront directement pour la mise à jour, se trouvent dans un répertoire de l'administrateur. Ces fichiers sont consultables par tous les développeurs. Ils sont dans le répertoire /aster/eda/*version*/*user*.

Le résultat du passage des tests qui échouent est déposé sous /aster/eda/*version*/*user*/*astout*.

Les fichiers créés sont :

```
/aster/eda/ version / user /
   histor                    fichier contenant l'extraction des fiches REX
   fortran
   catalogu
   test/
       nom_cas_test.comm
       nom_cas_test.para
       nom_cas_test.mail
       nom_cas_test.msh
       nom_cas_test.mmed
       nom_cas_test.msup
       nom_cas_test.mgib
       nom_cas_test.datg
       nom_cas_test.ensi/
       nom_cas_test.miss/
       nom_cas_test.ul
   unigest
    fortran
    bibfor/    bibli /    routine.f
    bibc/  bibli /    fonction.c ou include.h
    bibf90/    bibli /    routine.F
    repobj/   bibli/*.o
    asterd                    exécutable aster
   catapy/     bibli /    subcat.capy
    catalo/    bibli /    subcat.cata
    catobj/   cata.py
          cata.pyc
          elements
    bibpyt/    bibli/ *.py
    identifi/                 Copie du répertoire passé en argument (les fichiers sources y sont
                              déposés  dans l'état avant modification par les procédures de
                              restitution)
    liste_fiches              Liste des numéros de fiches REX correspondant à la restitution
    message                        Messages de l'asrest
    iret                      Code retour de la restitution
    recap                     Récapitulatif des unités manipulées
   astout/    liste_ct
              liste_ct.rest
              *.comm
              *.para
              *.mail
              *.msup
              *.mgib
              *.datg
              *.ensi/
              *.miss/
              *.ul
              resu_NOOK/ *.resu
                         *.code
                         flash/  *.error
                                 *.output
   datg/   *.datg
   materiau/
```

Pour `asrest`, seuls les résultats des jobs non OK sont conservés dans `astout/resu_NOOK/` et `astout/resu_NOOK/flash`.

9.2.4 Les codes retours 2 et 4

En plus des cas détectés par **averif**, voici les cas spécifiques à **asrest** qui conduisent à l'émission
- d'un code retour 2 :
  commande d'impression dans un cas test,
  aucun test (issu de la commande `TEST_RESU`) dans un cas-test,
- d'un code retour 4 (éventuellement autorisé lors de l'EDA):
  test en « `PAR_LOT='NON'` »,

messages d'alarmes supplémentaires lors de l'exécution.

10 Utilitaires *Aster*

10.1 `as.tout` : passage de cas-tests

10.1.1 Fonctionnalité

Enchaînement d'une liste de cas-tests avec analyse des résultats.

`as.tout` vérifie la cohérence des arguments transmis puis construit et soumet un `job (qastout)` de la même classe que celle des cas-tests. Ce `job` construit et soumet des chaînes de cas-tests et attend la fin de tous les cas-tests lancés pour mettre en forme les résultats et les envoyer dans le fichier `message`.

10.1.2 Mode d'appel

```
as.tout fichier_parametre [message]
```

`fichier_parametre` : Fichier contenant les différents paramètres d'exécution d'`as.tout`.

`message` : fichier des messages. Par défaut `message_astout`.

`fichier_parametre` contient 12 valeurs.

Un paramètre est le premier champ d'une ligne ne débutant pas par le caractère '`%`'. Il ne peut donc pas y avoir plusieurs paramètres sur la même ligne.

La reconnaissance des paramètres se fait sur l'ordre de déclaration. Si l'on veut avoir la valeur par défaut d'un paramètre il faut le remplacer par le caractère '`.`'. Les lignes vides sont autorisées.

Les paramètres sont les suivants :

nom du fichier contenant la liste des cas-tests (pas de valeur par défaut),
répertoire contenant les fichiers référence des cas-tests (fichiers de commandes, de maillages et paramètres avec la même organisation que dans la base de référence
- par défaut `/aster/vx/`*`version`*`/astest/-`),

répertoire contenant des fichiers de cas-tests qui surchargeront les fichiers de référence (pas de valeur par défaut),
répertoire recevant les résultats (pas de valeur par défaut), les fichiers `nom_cas_test.resu` et `nom_cas_test.code` y sont crées et s'il n'existe pas déjà un répertoire `flash/` est créé, il reçoit les fichiers `nom_cas_test.error` et `nom_cas_test.output`),
nom de l'exécutable *Aster* (par défaut `/aster/vx/`*`version`*`/asteru`),
répertoire contenant le catalogue des commandes compilées (par défaut : `/aster/vx/`*`version`*`/catobj/commande`),
fichier contenant le catalogue des éléments compilés (par défaut : `/aster/vx/`*`version`*`/catobj/elements`),
l'heure de départ des travaux batch (par exemple `06:00` pour démarrer les exécutions à 6 heures du matin),
la version *Aster* pour les cas-tests (sert à déterminer les noms par défaut, `NEW10` par défaut),
si `"DATE"` la date et l'heure est rajoutée en extension des résultats des jobs, sinon les extensions `.error` et `.output` sont ajoutées au nom du cas-test pour le résultat des jobs,
si `"RESOK"` les fichiers de résultat de tous les jobs lancés sont envoyés dans le répertoire résultat, pour toute autre valeur, seuls les résultats des jobs non `OK` sont conservés.

Les tests sont lancés avec les paramètres temps cpu et mémoire du fichier .para associé, pour pouvoir utiliser une version instrumentée en mode debug il peut être nécessaire d 'augmenter le temps en indiquant un facteur multiplicatif. Cette valeur peut-être positionnée en renseignant la variable

d'environnement FTPS. L'interface astk permet d'indiquer une valeur (menu Options paramètre facmtps). (Voir un exemple de ce fichier plus loin.)

10.1.3 Tâches réalisées

Résultats du lanceur de cas-tests qastout :

Soumission des cas-tests.
Attente de la fin de tous les cas-tests.
Mise en forme du résultat des cas-tests.

Pour chaque cas-test soumis il y a affichage :

d'un diagnostic :
- OK,
- OK_ALARME (il y a des alarmes Aster dans le fichier de messages),
- ARRET_ANORMAL (arrêt non géré par *Aster*),
- <F>_SUPERVISEUR (*Aster* interrompu par le superviseur),
- ERREUR_<F> (autres erreurs fatales détectées par *Aster*),
- ERREUR_<S> (écrasement mémoire détecté mais sans conséquence apparente),
- <E>_VOLATILE il reste des objets dans la base volatile,
- NOOK_TEST_RESU,
- DEFAUT_FICHIER (il manque un fichier ou un répertoire indispensable au cas-test),
- CPU_LIMITE (manque de temps pour finir le cas-test),
- ERR_NUMERIQUE (*floating point exception*),
- INTERRUPTION_JOB,
- PAS_DE_TEST_OK (uniquement dans le contexte d'un **asrest**, tout cas-test doit avoir obligatoirement au moins un test OK),

du temps CPU,
du temps système,
du temps total du cas-test,
du coût du cas-test.

Pour l'ensemble des cas-tests soumis affichage :

du temps CPU total,
du temps système total,
du temps total de tous les cas-tests,
du coût de tous les cas-tests.

Les fichiers de résultat des jobs non OK, sont systématiquement conservés dans le répertoire résultat, et selon le dernier paramètre du fichier d'arguments les fichiers des cas-tests OK le sont ou non.

11 Accès à l'AGLA via l'interface graphique astk

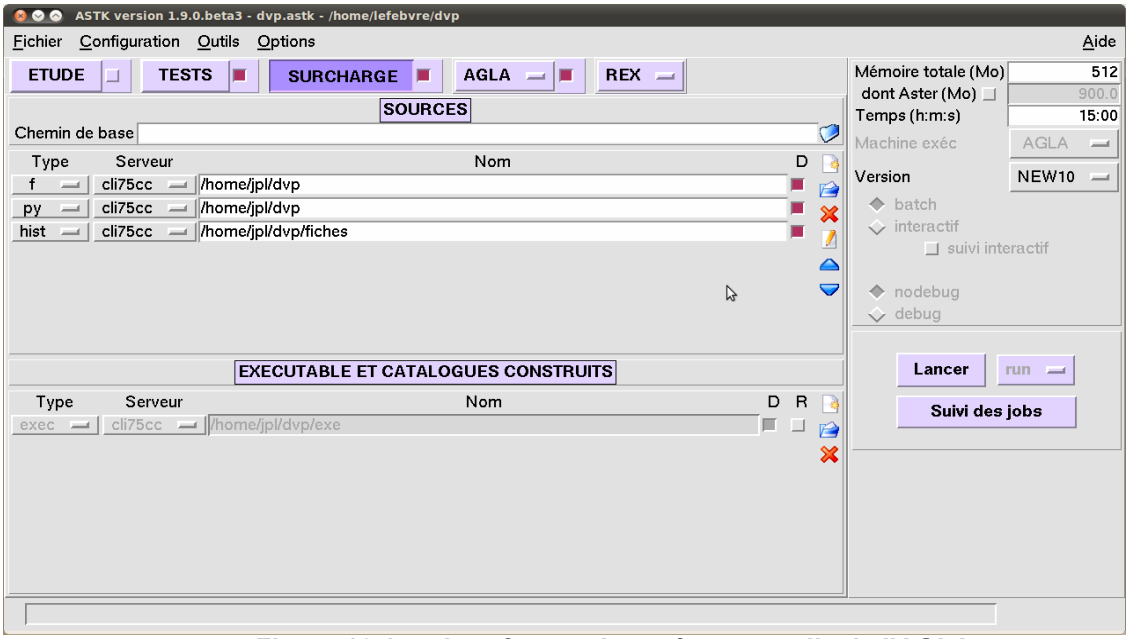

Figure 12.1-a : Interface astk accès aux outils de l'AGLA

Le menu **AGLA** n'est accessible que lorsque **le compte utilisateur est déclaré dans le fichier `/aster/agla/identAster`**. Les fichiers utilisés par les outils de l'AGLA sont ceux figurant les menus **TESTS** et **SURCHARGE** lorsque le bouton **D** est coché.
Les fichiers messages sont toujours mis sur la machine de développement et dupliqués sur la station de l'utilisateur. Ils sont accessibles par la fenêtre de suivi des jobs.

11.1 asno : notation de modules

Note les modules contenus dans tous les fichiers de type source *Aster* (`f, F90, c, py, cata, capy, test, unig`) qui ont l'attribut **D** dans le profil courant.

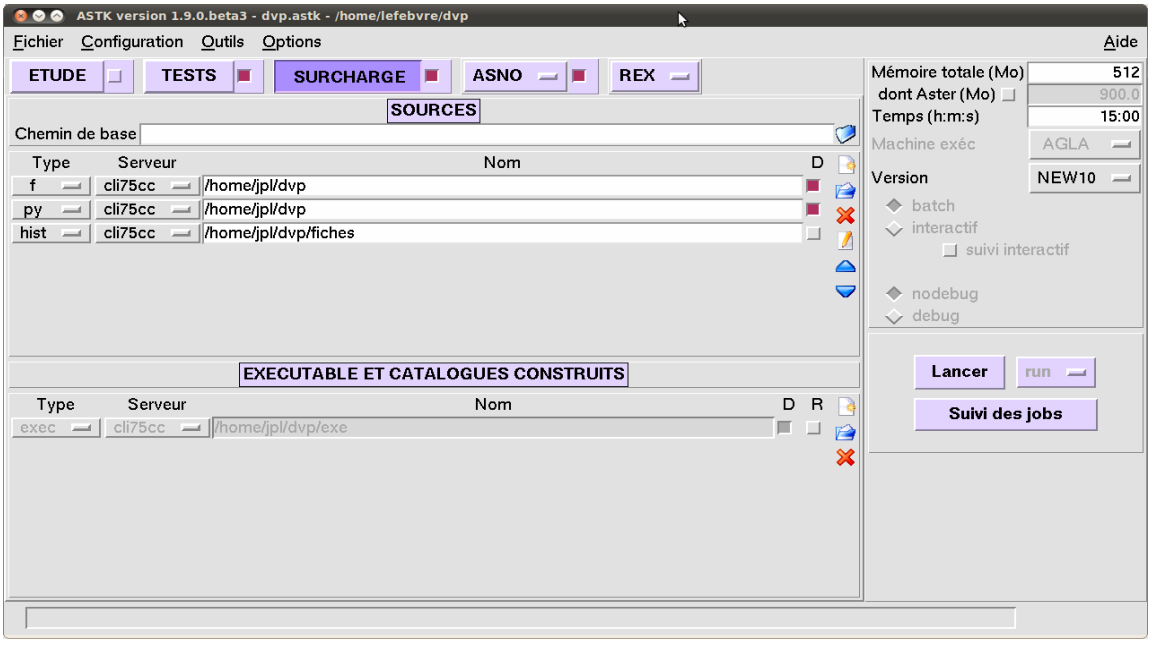

Figure 12.1-a : Fenêtre notations dans l'AGLA asno

Les paramètres du job sont renseignés dans la fenêtre principale d'astk :

la version de développement *Aster* concernée par la dénotation,
le temps (en s) et la mémoire (en Mo) donné spour le job.

Le bouton **Exécution** envoie un job sur la machine de développement. Ce job est pris en compte par le suiveur de job (pour visualiser les messages, supprimer le job ...).

11.2 `asdeno` : dénotation de modules

Permet de dénoter tous les modules de type source Aster (`f/f90/c` ou `py` ou `capy/cata` ou `test`) que l'on donne dans une liste. On ne peut dénoter qu'une seule sorte de source à la fois.
A partir de la fenre principale d'astk, une fenêtre dédiée s'ouvre lors du lancement (bouton l**ancer)** de la commnde **ASDENO:**

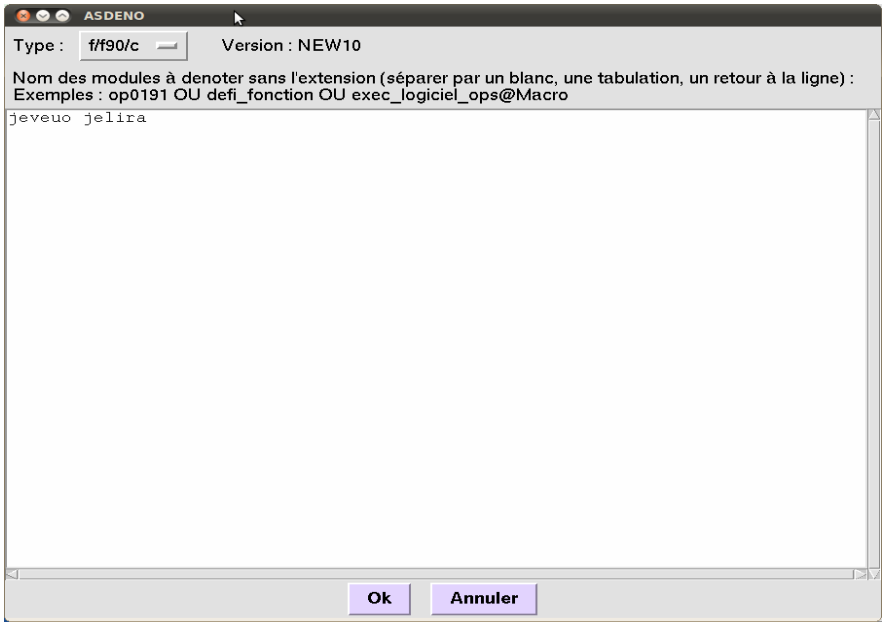

Figure 12.2-a : Fenêtre dénotations dans l'AGLA `asdeno`

La fenêtre permet d'indiquer le type de source *Aster* à dénoter (un parmi) :

- fortran, fortran90, c,
- catalogue de commande, catalogue d'éléments,
- source python,
- cas-test.

Elle affiche également une zone de texte éditable où l'on donne le nom des modules du type choisi que l'on veut dénoter. Les noms des modules peuvent êtres séparés par un ou plusieurs blancs, tabulation, retour chariot. Il est possible d'effectuer la dénotation (bouton **Exécution**) ou d'abandonner (bouton **Annuler**).

Les autres paramètres du job sont renseignés dans la fenêtre principale d'astk :

la version de développement *Aster* concernée par la dénotation,
le temps (en s) et la mémoire (en Mo) donné spour le job.

11.3 `asqui` : information sur certaines notations

Sur le même principe qu'`asno`, `asqui` donne les notations courantes des modules contenus dans les fichiers qui ont l'attribut **D** dans le profil courant.

11.4

11.5 asquit : informations sur toutes les notations

Sur le même principe qu'asno, asquit donne toutes les notations en cours dans l'AGLA.

11.6 asverif : vérification de la cohérence d'une restitution

Sur le même principe qu'asno, asverif effectue la vérification de cohérence sur tous les fichiers de type source du profil courant plus le(s) fichier(s) historique(s).

11.7 pre_eda : présentation d'une restitution de source *Aster*

Sur le même principe qu'asno, pre_eda effectue une restitution sur tous les fichiers de type source qui ont l'attribut **D** dans le profil courant.

La liste des cas-tests figurant dans la surcharge est lancée en batch, le fichier de message sur station sera mis à jour dès la fin des cas-tests et un code retour est attribué.
Le résultat est déposé sur le serveur sous /aster/pre_eda/*version*/*user*.

11.8 asrest : restitution de source *Aster*

Sur le même principe qu'asno, asrest effectue une restitution sur tous les fichiers de type source qui ont l'attribut **D** dans le profil courant.
Si la liste restreinte des cas-tests est lancée en batch , le fichier de message sur station sera mis à jour dès la fin des cas-tests et un code retour est attribué.
Le résultat est déposé sur le serveur sous /aster/eda/*version*/*user*.

11.9 as.tout : soumission d'une liste de cas-tests

Permet de soumettre une liste de cas-tests dont les noms sont dans le(s) fichier(s) de type list avec l'attribut **D**. Si un exécutable ou des commandes compilées ou des éléments compilés ou des fichiers python ont l'attribut **D**, ils sont pris en compte pour le passage des cas-tests. Les fichiers cas-tests de la référence sont surchargés par les fichiers correspondants qui sont dans les répertoires de type rep_test ayant l'attribut **D**.

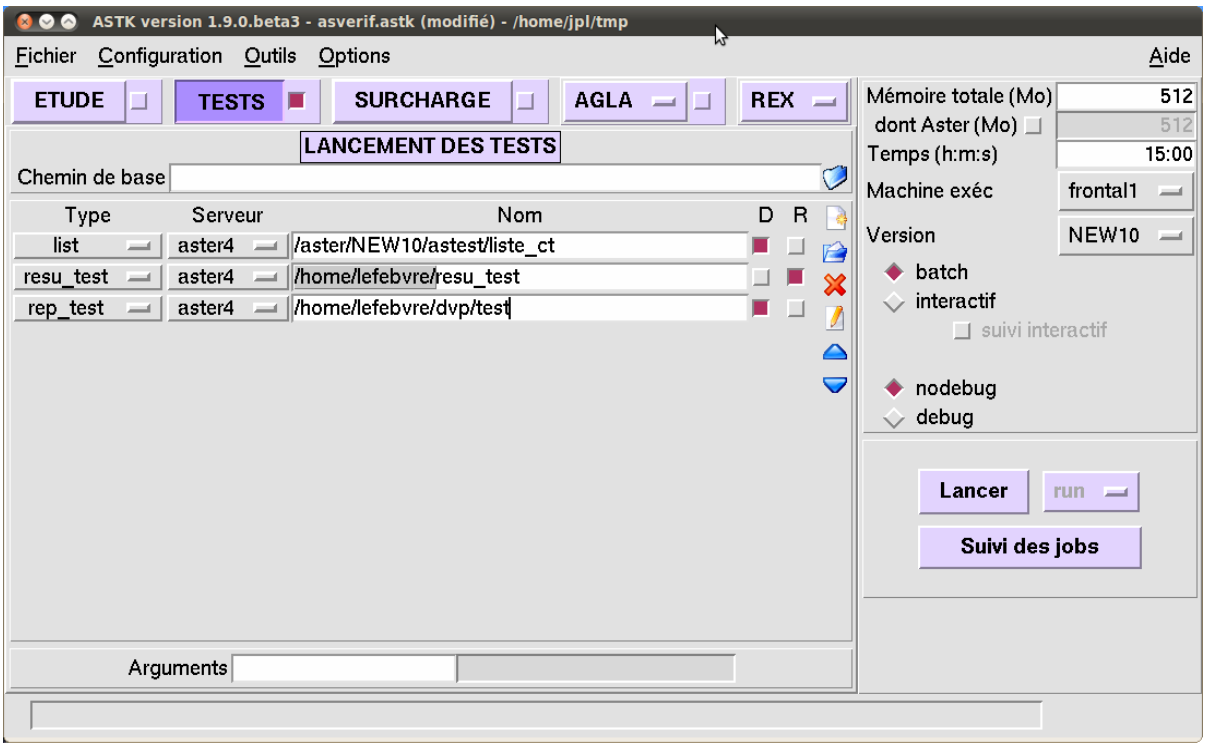

Figure 12.7-a : Fenêtre Passage de cas-tests dans l'AGLA as.tout

Le menu `Options` permet en plus de modifier les paramètres suivants :

- le nombre maximum de cas-test tolérés en échec avant d'interrompre le passage des autres cas-tests,
- le répertoire où les résultats des jobs de cas-test seront copiés (obligatoirement sur la machine d'exécution),
- si l'on rajoute la date aux noms des fichiers résultat,
- si l'on conserve le résultat des cas-tests OK ou non,
- un facteur multiplicatif appliqué sur la limite de temps CPU lue dans les fichiers de paramètres,
- l'option de debug `JEVEUX` (le temps maximum du cas-test est automatiquement multiplié par 3).

Comme pour `asrest`, lorsque tous les cas-tests sont passés, le fichier de message du suivi des jobs est mis à jour.

Les fichiers et répertoires suivants sont créés sur la machine de développement pour pouvoir être accessibles par le job de lancement des cas-tests :

repertoire_message`/liste_astout` (Liste des cas-tests à passer)
repertoire_message`/para_astout` (paramètres d'`as.tout`)