Manuel de Descriptif Informatique
Fascicule D1.03 :
Document D1.03.01

Lancement d'une exécution sur un serveur centralisé Aster et réutilisation des outils d'`asterix` par d'autres applications : `lib_exec_aster` (4.0) et `lib_asterix` (4.0)

Résumé :

Ce document décrit les principes des bibliothèques `lib_exec_aster` et `lib_asterix` ainsi que les interfaces des fonctions mises à disposition. Ces bibliothèques permettent de soumettre des exécutions du *Code_Aster* sur un serveur centralisé Aster. La bibliothèque `lib_exec_aster` permet de soumettre une exécution sans nécessiter d'environnement graphique. La bibliothèque `lib_asterix` permet de le faire dans un environnement X11R5-Motif1.2. Elle permet également d'utiliser certains outils développés dans le cadre de l'interface X11-Motif du *Code_Aster* : asterix.

Les paragraphes 1 à 4 de cette note précisent les limites de fourniture des bibliothèques.

Table des matières

1 Principaux outils proposés dans les bibliothèques 5
1.1 Bibliothèque lib_exec_aster 5
1.1.1 Gestion de profils d'études de type asterix 5
1.1.2 Lancement d'une exécution Aster 5
1.1.3 La gestion des communications *remote* avec ou sans ASURE 5
1.2 Bibliothèque lib_asterix 5
1.2.1 Une boîte de sélection de fichier multi-machine : bsf 6
1.2.2 Lancement d'une exécution Aster 6
1.2.3 Suivi de jobs sur la machine d'exécution 6
1.2.4 Exécution d'une commande système interruptible 6
2 Livraison 6
2.1 Sur Sparc Sun sous SOLARIS 2.x 8
2.2 Sur HP 9000 sous HP-UX 10.x 8
2.3 Sur SGI sous IRIX 6.5 8
3 Anomalies, évolutions 8
4 Utilisation des bibliothèques 8
4.1 Utilisation de lib_exec_aster 8
4.2 Utilisation de lib_asterix 8
4.3 Sous Sun SOLARIS 9
4.4 Sous HP-UX 9
4.5 Sur SGI 9
5 Les Variables globales 10
5.1 lib_exec_aster 10
5.2 lib_asterix 10
6 Les services de lib_exec_aster 11
6.1 Initialisation de la bibliothèque 11
6.2 Quitter l'application 12
6.3 Fichier local / Fichier remote 12
6.4 rcp et rsh sécurisés via ASURE 13
6.4.1 Les paramètres ASURE pour lib_exec_aster 13
6.4.2 Fichier d'environnement ASURE pour lib_exec_aster 14
6.4.3 ASURE ENTRANT (extérieur -> EDF) 14
6.4.4 ASURE SORTANT (EDF -> extérieur) 15
6.4.5 DISPLAY ASURE 15
6.5 Gestion de profil d'étude ou d'exécution asterix 16
6.5.1 Structure de données pour un profil d'étude ou d'exécution 16
6.5.2 Normalisation d'un profil 19
6.6 La fonction sysint() 19

6.7 Soumission d'une étude Aster 20
6.7.1 Création/soumission d'un script d'exécution 20
6.7.2 Soumission d'un script d'exécution 21
6.8 Soumission à partir d'un fichier profil 23
6.9 Utilitaires en vrac 23
6.9.1 Version de lib_exec_aster 23
6.9.2 basename 23
6.9.3 dirname 24
6.9.4 Fichier sans son extension : QuelBase 24
6.9.5 Radical d'un fichier : radical 24
6.9.6 Extension d'un fichier : QuelExt 24
6.9.7 Composer un nom de fichier au format rcp : compNFC 25
6.9.8 Décomposer un nom de fichier au format rcp : decompNFC 25
6.9.9 Accoler un nom de répertoire et un nom de fichier : Ficsel 25
6.9.10 'Normalisation' d'une chaîne de caractères : strlennor 26
6.9.11 Longueur d'une chaîne 'normalisée' : strlennor2 26
6.9.12 Contenu d'un fichier dans une chaîne : filestr 26
6.9.13 Copie de fichier ou répertoire entre machines 27
7 Les services de lib_asterix 27
7.1 Initialisation de la bibliothèque 27
7.2 Quitter l'application 27
7.3 Boîte de sélection de fichier : bsf 28
7.3.1 Création et appel d'une bsf en dialogue modal 29
7.3.2 Création et appel d'une bsf en dialogue non modal 30
7.3.3 Commandes de lancement des éditeurs (jusqu'en 2.2) 31
7.3.4 Les commandes Unix de la bsf et les listes de PushButton associées 31
7.3.5 Conservation des commandes Unix et des répertoires rémanents par l'application. 32
7.4 Exécuter une commande système interruptible : sysint() 34
7.4.1 Bouton Interrompre 34
7.4.2 La fonction sysint() 35
7.4.3 Détection des erreurs : errshellRSH() 36
7.5 Suivi des *jobs* sur la machine d'exécution 36
7.5.1 Initialisation, création de la fenêtre de suivi 37
7.5.2 Afficher la fenêtre de suivi 38
7.5.3 Ajouter un job à gérer par le suivi 38
7.5.4 Les fichiers que le job doit envoyer sur le flasheur station 39
7.5.5 Sauvegarde des informations du Suivi des Jobs 39
7.5.6 Fermer l'application asjob à partir d'une autre application. 39
7.6 Soumission d'une étude Aster 39
7.7 Fenêtre de confirmation 40
7.8 Édition 40

7.8.1 Éditer un fichier 41
7.8.2 Editer tous les fichiers "éditables" des profils 41
7.9 Fenêtre de visualisation de texte 42
7.10 Quelques utilitaires en vrac 42
7.10.1 Version de lib_asterix 42
7.10.2 Nom de l'item courant d'un option-menu : historiqueOM 42
7.10.3 Choisir un item d'un option-menu : activeClick 43
7.10.4 Imprimer une chaîne de caractères : ImprString 43
7.10.5 MAJ d'une ressource XmNlabelString 43
7.10.6 Récupération d'une ressource XmNlabelString 43
7.10.7 Affichage du curseur montre dans une fenêtre 44
7.10.8 Retour à un curseur normal 44
8 Exemple d'utilisation 44
8.1 Exemple pour lib_exec_aster 44
8.2 Exemple pour lib_asterix 47
9 Différences d'utilisation avec les versions antérieures 50
9.1 Avec la version 1.1 50
9.2 Avec la version 2.2 50
9.3 Avec la version 3.0 50
9.4 Avec la version 3.1 50
9.5 Avec la version 3.2 51
9.6 Depuis la version 4.0 51
10 Bibliographie 52

Les principes généraux des outils d'asterix ainsi que les contraintes d'utilisation sont décrits dans le document "Description fonctionnelle de l'Interface d'Aster sur station de travail" [bib1].

Les bibliothèques livrées servent principalement à soumettre en batch une exécution Aster sur un serveur centralisé Aster. En effet le lancement d'Aster fait référence à des scripts, à une configuration spécifique des versions et à un gestionnaire de batch donné. Cela ne permet donc pas de lancer une exécution pour une version dite 'locale' d'Aster qui est gérée complètement différemment. Actuellement les serveurs centralisés pris en compte sont `claster`, `clcraya` et `cldragon` (sachant que l'accès Aster à `clcraya` est supprimé et que la machine `cldragon` n'existe plus).

1 Principaux outils proposés dans les bibliothèques

1.1 Bibliothèque `lib_exec_aster`

Cette bibliothèque contient tous les utilitaires de base pour soumettre une exécution batch d'Aster à un serveur centralisé Aster. Tous les utilitaires proposés sont indépendants d'un environnement graphique.

1.1.1 Gestion de profils d'études de type asterix

Un profil d'étude est une manière de décrire les fichiers de données et de résultats d'une exécution Aster ainsi que les paramètres de l'exécution (version Aster, mémoire, temps maximum ...). Ces fichiers sont créés et reconnus par asterix, mais ils servent également de support pour caractériser une exécution. Comme ils sont stockés au format ASCII, il est simple de les créer indépendamment d'asterix en respectant la syntaxe donnée. Les outils fournis permettent de créer et manipuler les profils d'étude asterix.

1.1.2 Lancement d'une exécution Aster

La soumission d'une exécution Aster se fait par rapport à un profil d'étude de type asterix. Des fichiers permettant de contrôler les exécutions sont créés sur la machine d'exécution (serveur Aster) et sur la machine de soumission (machine locale).

1.1.3 La gestion des communications *`remote`* avec ou sans ASURE

Les communications entre la machine de l'utilisateur et le serveur d'exécution Aster se font par l'intermédiaire des commandes UNIX `remote` (`rcp`, `rsh`). `lib_exec_aster` met à disposition un certain nombre d'outils pour enrober ces commandes. Il est notamment possible de les utiliser en traversant le coupe-feu EDF ASURE.

1.2 Bibliothèque `lib_asterix`

Cette bibliothèque reprend les outils de `lib_exec_aster` dans un contexte graphique, et ajoute un certain nombre d'utilitaires développés pour asterix.

1.2.1 Une boîte de sélection de fichier multi-machine : bsf

Cette boîte remplace la File Selection Box Motif. Elle permet de naviguer sur différentes machines par les mécanismes Unix `remote` (`remote shell` et `remote copy`).

Elle implique que chaque utilisateur ait un fichier `$HOME/.rhosts`

Elle permet d'éditer un fichier sélectionné. Pour cela des éditeurs sont pré définis.

Les fichiers manipulés le sont sous la forme `machine@user:nom_de_fichier`.

1.2.2 Lancement d'une exécution Aster

Il est possible de soumettre le passage d'une étude Aster à partir d'un profil d'étude et d'un profil d'exécution et de quelques paramètres.

Par rapport à `lib_exec_aster`, les jobs soumis sont pris en charge par le suiveur de job asjob.

1.2.3 Suivi de jobs sur la machine d'exécution

Le suivi des jobs comporte des mécanismes de gestion (interrogation, interruption, suppression, ...) et une application graphique (asjob) pour accéder interactivement à ces fonctionnalités. Les soumissions d'une étude Aster par les procédures fournies sont interfacées avec ce mécanisme de suivi.

Ce mécanisme utilise un répertoire sur la machine d'exécution (`$HOME/flasheur`, par défaut) et un répertoire sur station de travail (`$HOME/flash_Aster`, par défaut). On peut modifier ces répertoires de destination.

1.2.4 Exécution d'une commande système interruptible

Si l'on a défini un bouton Interrompre selon les recommandations données plus loin, la fonction `sysint()` permet d'exécuter une commande système pouvant être interrompue par l'utilisateur par un simple clic sur le bouton Interrompre.

2 Livraison

Sur la machine `claster`, dans `/aster/interface/lib/`, il y a un répertoire par environnement supporté. Chaque répertoire contient les bibliothèques partageables, les fichiers de ressources et un exemple d'utilisation de `lib_exec_aster` seul et `lib_asterix` avec XFaceMaker [§17].

La livraison comprend 4 bibliothèques partageables et deux fichiers d'en-tête :

Nom de fichier	Description
`lib/lib_asterix.sZ`	bibliothèque partageable `lib_asterix`
`lib/lib_exec_aster.sZ`	bibliothèque partageable `lib_exec_aster`
`lib/lib_util_exec_aster.sZ`	bibliothèque d'utilitaires `lib_exec_aster` sans comportement graphique (quand on utilise `lib_exec_aster` sans `lib_asterix`)
`lib/lib_util_exec_aster_X.sZ`	bibliothèque d'utilitaires `lib_exec_aster` avec comportement graphique (quand on utilise `lib_asterix`).
`include/lib_asterix.h`	Fichier d'en-tête pour `lib_asterix`
`include/lib_exec_aster.h`	Fichier d'en-tête pour `lib_exec_aster`

L'extension des bibliothèques partageable dépend du système d'exploitation.

Exemple d'utilisation de `lib_exec_aster`, dans le répertoire `ex_lib_exec_aster/` :

Nom de fichier	Description
`A_LIRE.txt`	Description des étapes pour compiler l'exemple.
`Makefile`	Fichier de `make` de l'exemple.
`asexec`	Script de lancement de l'exemple.
`asterix/include/`	Fichiers header de `lib_exec_aster` et `lib_asterix` (définition des types, variables globales interface des fonctions fournies).
`asterix/lib/`	Bibliothèques partageables compressées avec `gzip`.
`bin/asexecB`	Exécutable de l'exemple créé par la commande `make`.
`gunzip`	Utilitaire de décompression du format `gzip (.gz)`.
`obj/`	Répertoire des fichiers objets.
`src/asexec.c`	Programme principal de l'exemple.
`zz.pret`	Fichier profil d'étude asterix d'exemple.

Exemple d'utilisation de `lib_asterix`, dans le répertoire `ex_lib_asterix/` :

Nom de fichier	Description
`A_LIRE.txt`	Description des étapes pour compiler l'exemple.
`Makefile`	Fichier de `make` de l'exemple.
`asjob`	Script de lancement d'asjob.
`asterix/bin/asjobB.gz`	Exécutable du suivi des jobs compressé.
`asterix/include/`	Fichiers header de `lib_exec_aster` et `lib_asterix` (définition des types, variables globales interface des fonctions fournies).
`asterix/lib/`	Bibliothèques partageables compressées avec `gzip`.
`asterix/rdb/`	Fichier de ressources X pour asjob et `lib_asterix`.
`bin/exempleB`	Exécutable de l'exemple créé par la commande `make`.
`obj/`	Répertoire des fichiers objets.
`exemple`	Script de lancement de l'exemple.
`src/exemple.fm`	Fichier XFaceMaker de l'interface.
`src/exemple.c`	Fichier C généré par XFaceMaker à partir du fichier `exemple.fm`
`src/exemple_pp.c`	Programme principal de l'interface.
`xfm3.5/include/`	Fichiers header pour l'utilisation de XFM.
`xfm3.5/lib/libFm_c.a`	Bibliothèque XFM pour l'édition des liens de l'application.

2.1 Sur Sparc Sun sous SOLARIS 2.x

Répertoire de livraison : /aster/interface/lib/sun_solaris

Les bibliothèques partageables ont pour extension `.so`.

2.2 Sur HP 9000 sous HP-UX 10.x

Répertoire de livraison : `/aster/interface/lib/hp`

Les bibliothèques partageables ont pour extension `.sl`.

2.3 Sur SGI sous IRIX 6.5

Répertoire de livraison : `/aster/interface/lib/sgi`

Les bibliothèques partageables ont pour extension `.so`.

3 Anomalies, évolutions

Toutes les anomalies constatées et les évolutions souhaitées doivent suivre le circuit du *Code_Aster* (anomalie logicielle, anomalie documentation, évolution logicielle) via l'interface asterix.

4 Utilisation des bibliothèques

Les bibliothèques sont sous forme de librairies partageables UNIX. Elles et se trouvent dans le répertoire `rep_asterix/lib/`.

Elles ne sont livrées qu'en version partageable pour permettre une évolution rapide du mode de soumission du *Code_Aster* sans nécessiter d'intervention sur les applications l'utilisant.

4.1 Utilisation de `lib_exec_aster`

On utilisera `lib_exec_aster` sans `lib_asterix` pour être indépendant de l'environnement graphique X. Dans ce cas là, il faut lui adjoindre les utilitaires sans comportement graphique : `lib_util_exec_aster`.

L'édition des liens de l'application doit donc avoir une référence du type :

```
-l_util_exec_aster -l_exec_aster
```

4.2 Utilisation de `lib_asterix`

Pour tenir compte de l'évolution de la bibliothèque `lib_asterix`, et éventuellement des ressources X qui lui sont associées, il est recommandé de mettre à jour les ressources de l'application de la manière suivante (exemple sur SUN) :

```
sed -e '1,$ s/ZZZZB*/nomApplication*/' lib_asterix.rdb  >/tmp/ressources.rdb
xrdb -merge /tmp/ressources.rdb
rm /tmp/ressources.rdb
```

La bibliothèque lib_asterix a été développée avec le générateur d'interface XFaceMaker. Comme c'est une bibliothèque partageable, elle n'intègre pas toutes les ressources nécessaires en statique. Pour pouvoir effectuer l'édition des liens, l'application utilisant lib_asterix doit également faire référence à la bibliothèque libFm_c.a qui est également livrée. lib_asterix fait référence à lib_exec_aster, et pour avoir le comportement graphique de certains utilitaires de lib_exec_aster, il faut faire référence à lib_util_exec_aster_X.

L'édition des liens de l'application doit donc avoir une référence du type :
```
-lFm_c -l_util_exec_aster_X -l_exec_aster -l_asterix .
```

4.3 Sous Sun SOLARIS

Dans le script de lancement de l'interface il faut enrichir la variable LD_LIBRARY_PATH.

Exemple en C-shell :
```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:rep_asterix/lib
rep_asterix/interfaceB
```

Pour la compilation des fichiers incluant le fichier lib_exec_aster.h ou lib_asterix.h, il faut transmettre au compilateur la directive -D_SOLARIS.

4.4 Sous HP-UX

Dans le script de lancement de l'interface il faut enrichir la variable SHLIB_PATH.

Exemple en C-shell :
```
setenv SHLIB_PATH ${SHLIB_PATH}:rep_asterix/lib
rep_asterix/interfaceB
```

Pour la compilation des fichiers incluant le fichier lib_exec_aster.h ou lib_asterix.h, il faut transmettre au compilateur la directive -D_HP.

4.5 Sur SGI

Dans le script de lancement de l'interface il faut enrichir la variable LD_LIBRARYN32_PATH.

Exemple en C-shell :
```
setenv LD_LIBRARYN32_PATH ${LD_LIBRARYN32_PATH}:rep_asterix/lib
rep_asterix/interfaceB
```

Pour la compilation des fichiers incluant le fichier lib_exec_aster.h ou lib_asterix.h, il faut transmettre au compilateur la directive -D_SGI.

5 Les Variables globales

5.1 lib_exec_aster

lib_exec_aster utilise un certain nombre de variables globales. Certaines variables sont intéressantes pour les applications autres qu'asterix. Celles-ci vont être présentées. Cependant la plupart servent à la gestion interne des outils d'asterix, donc il faudra s'assurer qu'il n'y a pas conflit avec les variables globales des applications qui veulent utiliser les outils décrits dans ce document (à partir du fichier lib_exec_aster.h).

Nom	Type	Fonction
HOSTNAME	char *	Nom de la machine d'où est lancée l'interface
LOGINUSER	char *	User Unix qui a lancé l'interface
TMPDIR_INTERFACE	char *	Répertoire de travail
FLASH_STATION	char *	Nom du répertoire sur station pour le suivi des jobs sur la machine d'exécution
HOMEHOST	char *	$HOME de LOGINUSER sur HOSTNAME
MACHINE_EXEC	char *	Machine d'exécution du *Code_Aster*
OIAUser	char *	User sur la machine d'exécution du *Code_Aster*
RSH	char *	Commande pour exécuter une commande remote à partir de la machine HOSTNAME (rsh ou remsh)

5.2 lib_asterix

lib_asterix utilise également un certain nombre de variables globales. Elles sont déclarées dans le fichier lib_asterix.h.

Nom	Type	Fonction
CMDxedit	char *	Commande de lancement de l'éditeur xedit jusqu'à la 2.2
CMDvi	char *	Commande de lancement de l'éditeur vi jusqu'à la 2.2
CMDsedit	char *	Commande de lancement de l'éditeur sedit jusqu'à la 2.2
CMDemacs	char *	Commande de lancement de l'éditeur emacs jusqu'à la 2.2
CMDtextedit	char *	Commande de lancement de l'éditeur textedit jusqu'à la 2.2

6 Les services de lib_exec_aster

6.1 Initialisation de la bibliothèque

Interface :

```
/*----------------------------------------------------------------------
   Initialistation de lib_exec_aster
         - traitement des arguments non X-Motif
----------------------------------------------------------------------
*/
void Init_lib_exec_aster
  (
   int    argc,   /*  -> nb d'argument de la ligne de commande   */
   char **argv,   /*  -> liste des arguments                     */
   String tmpdir  /*  -> Prefixe du repertoire pour les fichiers
                         temporaires                             */
);
```

Interprétation des arguments passés à l'interface pour détecter les paramètres suivants :

Paramètre	Fonction
-bavard	Permet de renvoyer l'écho des commandes Shell lancées à partir de la fonction sysint(). Cet écho est envoyé sur stdout dans un contexte non X et dans une fenêtre si c'est dans le contexte lib_asterix.
-noexec	Permet de ne pas exécuter les commandes Shell lancées à partir de la fonction sysint().
-host nom_machine	Permet d'utiliser un autre nom de machine que le nom renvoyé par la commande uname (sert quand le nom de la machine n'est pas le même que celui déclaré pour l'authentification EDF)
-rep_flash *rep*	Répertoire dans lequel les fichiers de message sont écrits sur la machine locale (cf. [§7.5.4] pour un description de ces fichiers).
-rep_flash_exec rep	Répertoire dans lequel les fichiers de message sont écrits sur la machine d'exécution (cf. [] pour un description de ces fichiers).
-rep_tmp tmpdir	Répertoire de travail sur la machine locale.
les paramètres ASURE	Cf. [§6.4.1].

Cette fonction d'initialisation réalise la création du répertoire de travail dont le radical est tmpdir et auquel on rajoute _LOGINUSER. Attention si l'on veut lancer plusieurs sessions simultanées de la même application avec le même user, il faut donner un radical unique (basé sur la date ou le numéro de processus par exemple) pour qu'il n'y ait pas confusion sur les noms des fichiers de travail. Le nom complet de ce répertoire de travail est contenu dans la variable TMPDIR_INTERFACE. L'analyse du fichier $HOME/.rhosts est réalisée dans cette phase.

6.2 Quitter l'application

Pour libérer proprement les ressources allouées par lib_exec_aster, notamment les fichiers et le répertoire de travail, la fonction Quitter_lib_exec_aster() est disponible.

Interface :

```
/*-----------------------------------------------------------------------
   Sortie de lib_exec_aster avec suppression des fichiers temporaires.
-----------------------------------------------------------------------
*/
void Quitter_lib_exec_aster
  (
   bool SupprRep  /*  -> Suppression du repertoire temporaire */
  );
```

6.3 Fichier local / Fichier remote

L'utilisation de la bsf implique que l'utilisateur peut choisir un fichier local (même user et même machine) ou distant (user ou machine différent). Pour tenir compte des organisations avec un disque serveur de fichiers pour les données utilisateurs, la référence à une machine et un user local est masquée. On a alors la convention suivante :

- ******* : user qui a lancé l'application,
- ******* : machine d'où a été lancée l'application.

Comme le traitement pour un fichier local et un fichier distant est différent, lib_exec_aster propose d'encapsuler la détection d'un fichier local par les fonctions suivantes :

Fonctions qui répondent VRAI si le fichier est considéré en local et FAUX sinon :

```
/*-----------------------------------------------------------------------
   Pour determiner si un fichier est local ou remote.
   Si le couple user/machine n'est pas detecte local, mais qu'une des 2
   valeurs est generique (*******) on renvoit la valeur reele a la place.
-----------------------------------------------------------------------
*/
bool Local
  (
   String usr,    /* <-> user    */
   String mac     /* <-> machine */
  );              /* <-  VRAI si local */


/*-----------------------------------------------------------------------
   Pour determiner si un fichier est local ou remote
   Si le couple user/machine n'est pas detecte local, mais qu'une des 2
   valeurs est generique (*******) on renvoit la valeur reele a la place.
-----------------------------------------------------------------------
*/
bool FichierLocalNFC
  (
   String fic  /*  -> Nom fichier sous forme user@machine:nom_fichier */
  );           /* <-  VRAI si fichier local */
```

Fonction pour tester si un fichier est considéré comme local par rapport à une autre référence que le user et la machine de lancement de l'application (pour tester notamment si un fichier est local dans le contexte d'un profil) :

```
/*-----------------------------------------------------------------------
   Pour determiner si un fichier est local par rapport a une reference.
-----------------------------------------------------------------------
*/
bool LocalRef
  (
   String usr,    /* <-> user                */
   String mac,    /* <-> machine             */
   String usrRef, /*  -> user de reference    */
   String macRef  /*  -> machine de reference */
  );              /* <-  VRAI si local a une reference  */
```

Fonction pour transformer le nom générique en nom réel :

```
/*-----------------------------------------------------------------------
   Si le user ou la machine ont le nom generique il est remplace
   par le nom reel.
-----------------------------------------------------------------------
*/
void NomLocal
  (
   String usr,    /* <-> user     */
   String mac     /* <-> machine  */
  );
```

Fonction pour transformer le nom réel en nom générique :

```
/*-----------------------------------------------------------------------
   Si le user ou la machine ont le nom reel il est remplace
   par le nom generique.
-----------------------------------------------------------------------
*/
void AliasLocal
  (
   String usr,    /* <-> user     */
   String mac     /* <-> machine  */
  );
```

6.4 rcp et rsh sécurisés via ASURE

`lib_exec_aster` tient compte des contraintes d'ASURE. Pour cela il faut lui passer des paramètres sur la ligne de commande et prévoir un fichier de déclaration de la topologie des machines qui sont définies dans le fichier `$HOME/.rhosts` . cf. [bib4] et [bib5].

6.4.1 Les paramètres ASURE pour `lib_exec_aster`

Paramètres ASURE pour `lib_asterix` :

Paramètre	Fonction
`-user_asure user`	Compte entrant sur la passerelle ASURE (extérieur -> EDF).
`-auth_asure auth`	Authentifiant pour le compte entrant ASURE (extérieur -> EDF) : mot de passe dynamique.
`-cf_asure cfa`	Nom du coupe feu (passerelle ASURE) `clasure.edf.fr` par défaut
`-display_asure disp`	display en relayage sur ASURE (n'est utilisé que dans le contexte `lib_asterix`).
`-user_asure_sortant user_s`	Compte sortant sur la passerelle ASURE (EDF -> extérieur).
`-dns_domain domaine`	Domaine DNS de l'utilisateur.

6.4.2 Fichier d'environnement ASURE pour lib_exec_aster

Si un de ces paramètres est détecté sur la ligne de commande, lib_exec_aster cherche le fichier $HOME/.topo_asure sur la machine d'exécution de l'application. Il indique si une machine est atteinte via ASURE ou en local. Format de ce fichier : machine asure | local

Exemple :

```
claster   asure
clcraya   asure
hawaii    local
tahiti    local
```

Les informations et le fichier de topologie des machines peuvent être préparés avec l'application asasure.

Si lib_exec_aster est utilisée avec un des paramètres concernant ASURE, pour tout rcp ou tout rsh de la bibliothèque, lib_exec_aster détermine si la machine impliquée est utilisée via ASURE (entrant ou sortant). Si c'est le cas la commande est transformée au format ASURE [bib4] [bib5] à partir des informations transmises.

Les autres applications peuvent bénéficier du même mécanisme à l'aide des fonctions décrites ci-dessous.

6.4.3 ASURE ENTRANT (extérieur -> EDF)

```
/*-------------------------------------------------------------------------
   Construction de la partie de la commande rsh identifiant le compte et
   la machine distante en tenant compte d'Asure. (ENTRANT)
-------------------------------------------------------------------------
*/
String carsh
  (
   String usr,  /*  -> user distant     */
   String mac   /*  -> machine distante */
  );            /* <-  identification distante avec ou sans Asure */
```

Si lib_exec_aster n'est pas lancée avec les paramètres ASURE ou si elle est lancée avec ASURE et que mac est locale, carsh revoie la chaîne : "mac -l usr". Si lib_exec_aster est lancée avec ASURE et que mac est accessible via ASURE, carsh renvoie la chaîne :

" cf_asure -l user_asure-auth_asure_code:usr@mac".

```
/*-------------------------------------------------------------------------
   Construction de la partie de la commande rcp identifiant le compte et
   la machine distante en tenant compte d'Asure. (ENTRANT)
-------------------------------------------------------------------------
*/
String carcp
  (
   String usr, /*  -> user distant     */
   String mac  /*  -> machine distante */
  );           /* <-  identification distante avec ou sans Asure */
```

Si lib_exec_aster n'est pas lancée avec les paramètres ASURE ou si elle est lancée avec ASURE et que mac est locale, carcp revoie la chaîne : "usr@mac". Si lib_exec_aster est lancée avec ASURE et que mac est accessible via ASURE, carcp renvoie la chaîne :

" user_asure-auth_asure@cf_asure_code:usr@mac".

6.4.4 ASURE SORTANT (EDF -> extérieur)

Pour des raisons pratiques (batch) le mot de passe ASURE EDF -> extérieur n'est pas un mot de passe dynamique (il doit servir la nuit et le week-end). Il doit être décrit dans le fichier `$HOME/.authAsure` sur la machine d'exécution. Il a le format suivant : `user_sortant_ASURE mot_de_passe_associe`. A chaque utilisation de ce fichier pour calculer le mot de passe reconnu par ASURE (avec MD5) ses droits Unix sont positionnés à `-rw-------` pour maintenir la confidentialité du mot de passe fixe qui doit obligatoirement être écrit en clair.

```
      /*-------------------------------------------------------------------------
         Construction de la partie de la commande rsh identifiant le compte et
         la machine distante en tenant compte d'Asure. (SORTANT)
      -------------------------------------------------------------------------
      */
      String carshS
        (
   String usr,  /*  -> user distant     */
   String mac,  /*  -> machine distante */
   String macA  /*  -> machine ASURE    */
  );            /* <-  identification distante avec ou sans Asure */
```

Si `lib_exec_aster` n'est pas lancée avec les paramètres ASURE ou si elle est lancée avec ASURE et que `mac` est locale, `carshS` revoie la chaîne : `"mac -l usr"`. Si `lib_exec_aster` est lancée avec ASURE et que `mac` est accessible via ASURE, `carshS` renvoie la chaîne :

```
      "cf_asure -l user_asure_sortant-`/aster/adm/tool/asmd5f
      user_asure_sortant`:usr@mac".
```

```
      /*-------------------------------------------------------------------------
         Construction de la partie de la commande rcp identifiant le compte et
         la machine distante en tenant compte d'Asure. (SORTANT)
      -------------------------------------------------------------------------
      */
      String carcpS
        (
         String usr, /*  -> user distant     */
   String mac,  /*  -> machine distante */
   String macA  /*  -> machine ASURE    */
        );           /* <-  identification distante avec ou sans Asure */
```

Si `lib_exec_aster` n'est pas lancée avec les paramètres ASURE ou si elle est lancée avec ASURE et que `mac` est locale, `carcpS` revoie la chaîne : `"usr@mac"`. Si `lib_exec_aster` est lancée avec ASURE et que `mac` est accessible via ASURE, `carcpS` renvoie la chaîne :

```
       "user_asure_sortant-`/aster/adm/tool/asmd5f user_asure_sortant`@cf_asure:usr@mac".
```

6.4.5 DISPLAY ASURE

La fonction `cadisp()` renvoie le `display` sur lequel est lancée `lib_exec_aster` pour les machines locales et le `display` ASURE pour les machines accessibles via ASURE. Cette information n'a de sens que lorsque l'on est dans un contexte X.

```
      String cadisp
        (
         String mac  /*  -> machine distante   */
        );           /* <-  display a utiliser */
```

6.5 Gestion de profil d'étude ou d'exécution asterix

6.5.1 Structure de données pour un profil d'étude ou d'exécution

```
typedef struct t_desc_fic TDescFic;
struct t_desc_fic {
   char type[16];     /* Type du fichier
*/
   char nom[1024];    /* nom du fichier
*/
   char user[32];     /* user du fichier
*/
   char machine[32];  /* machine du fichier
*/
   int  ul;           /* Unité Logique du fichier 0 s'il n'y en a pas besoin
*/
   bool donnee;       /* Flag donnée OUI/NON
*/
   bool resultat;     /* Flag résultat OUI/NON
*/
   bool aEditer;      /* Flag à éditer OUI/NON
*/
   TDescFic *suiv;    /* Pointeur sur le suivant
*/
   };

/* Pointeur sur un descripteur de fichier */
typedef TDescFic *PTDescFic;

/* Allocation d'un descripteur de fichier */
#define TDescFicAlloc (TDescFic *)malloc(sizeof(TDescFic))
```

Depuis la version 3.5 de `lib_asterix`, un nouveau Flag est géré pour prendre en compte des fichiers compressés avec l'utilitaire du domaine public `gzip` (GNU). Pour ne pas modifier la structure de donnée, ce nouvel état est détecté sur l'extension du nom de fichier. Si ce nom comporte l'extension '`.gz`', il est considéré comme compressé.

6.5.1.1 Création d'une liste chaînée profil à partir d'un fichier

Interface :

```
/*-------------------------------------------------------------------------
   Ouvrir un profil sur n'importe quelle machine pour le transformer
   en liste chainee.
-------------------------------------------------------------------------
*/
String OuvrirProfil
  (
   PTDescFic *rac, /* <-  Racine de la liste          */
   String mac,     /*  -> machine ou est le fichier   */
   String usr,     /*  -> user pour acceder au fichier */
   String rep,     /*  -> repertoire du fichier       */
   String fic      /*  -> nom du fichier              */
  );               /* <-  message d'erreur            */
```

Pour ouvrir un profil d'étude ou d'exécution sur n'importe quelle machine et le transformer en liste chaînée.

Cette fonction peut renvoyer un message d'erreur si elle détecte une anomalie. Dans le cas contraire elle renvoie une chaîne vide ('`\0`').

6.5.1.2 Sauvegarde d'une liste chaînée profil sur un fichier

Interface :

```
/*-----------------------------------------------------------------------
   Enregistrer un profil. Transforme une liste chainee
   en fichier sur n'importe quelle machine.
-----------------------------------------------------------------------
*/
String EnrProfil
  (
   PTDescFic *rac, /* -> Racine de la liste                          */
   String     nfc  /* -> Nom de fichier compose du profil
                       sous la forme user@machine:nom_de_fichier */
  );              /* <-  message d'erreur                           */
```

Cette fonction peut renvoyer un message d'erreur si elle détecte une anomalie. Dans le cas contraire elle renvoie une chaîne vide ('\0').

6.5.1.3 Gestion d'une liste chaînée profil

Allocation d'une structure de type descripteur et initialisation avec renvoi du pointeur :

```
/*-----------------------------------------------------------------------
   Alloue une structure de type descripteur de fichier, l'initialise
   avec les parametres passe, et renvoie un pointeur sur cette
   structure.
-----------------------------------------------------------------------
*/
PTDescFic ConsDescFic
  (
   String typ, /*  -> Type du fichier                    */
   String nom, /*  -> nom du fichier avec le path        */
   String usr, /*  -> user pour acceder au fichier       */
   String mac, /*  -> machine sur laquelle est le fichier */
   int  ul,    /*  -> Unite Logique Fortran du fichier   */
   bool fd,    /*  -> flag donnee                        */
   bool fr,    /*  -> flag resultat                      */
   bool fe     /*  -> flag editable                      */
  );           /* <-  Pointeur sur la nouvelle structure  */
```

Modification d'une structure descripteur à partir de son pointeur (seuls les paramètres non nuls surchargent) :

```
/*-----------------------------------------------------------------------
   Modife une structure de type descripteur de fichier,
   l'initialise avec les parametres passe. Ne surgarge qu'avec
   les parametres non nuls.
-----------------------------------------------------------------------
*/
void ModifDescFic
  (
   PTDescFic pt, /* -> Pointeur sur la structure     */
   String typ,   /* -> Type du fichier               */
   String nom,   /* -> nom du fichier                */
   String usr,   /* -> user pour accerder au fichier */
   String mac,   /* -> machine du fichier            */
   int  ul,      /* -> Unite Logique Fortran         */
   bool fd,      /* -> flag donnee                   */
   bool fr,      /* -> flag resultat                 */
   bool fe       /* -> flag editable                 */
  ) ;
```

Insertion en tête de liste d'une structure descripteur :

```
/*-----------------------------------------------------------------------
   Inserer un fichier dans la liste chainee d'un profil d'etude en tete.
-----------------------------------------------------------------------
*/
void InsDebPEtude
  (
   PTDescFic *rac,   /* -> Racine de la liste          */
   PTDescFic pdes    /* -> Structure a ajouter en tete */
  );
```

Insertion en fin de liste d'une structure descripteur :

```
/*-----------------------------------------------------------------------
   Inserer un fichier dans la liste chainee d'un profil d'etude en queue.
-----------------------------------------------------------------------
*/
void InsFinPEtude
  (
   PTDescFic *rac,   /* -> Racine de la liste           */
   PTDescFic  pdes   /* -> Structure a ajouter en queue */
  );
```

Insertion à une position donnée d'un descripteur (si la position est supérieure à la longueur de la liste insertion en queue, si la position est inférieure à 1 insertion en tête) :

```
/*-----------------------------------------------------------------------
   Inserer un fichier dans la liste chainee d'un profil d'etude
   a une position donnee.
   Si La position est supperieure a la longueur de la liste
   Insertion en queue.
   Si la position est inferieure a 1 insertion en tete.
-----------------------------------------------------------------------
*/
void InsPosPEtude
  (
   PTDescFic *rac,  /* -> Racine de la liste         */
   PTDescFic  pdes, /* -> Structure a ajouter en pos */
   int        pos   /* -> Position d'insertion       */
  );
```

Suppression d'un descripteur à une position donnée :

```
/*-----------------------------------------------------------------------
   Suppression d'un fichier dans un profil d'etude
-----------------------------------------------------------------------
*/
void SupPosPEtude
  (
   PTDescFic *rac,  /* -> Racine de la liste   */
   int        pos   /* -> Position d'insertion */
  );
```

Libération de la mémoire prise par une liste chaînée profil :

```
/*-----------------------------------------------------------------------
   Liberer la place memoire occupee par un profil d'etude (Liste chainee)
-----------------------------------------------------------------------
*/
void FreeProfil
  (
   PTDescFic *rac   /* -> Racine de la liste */
  );
```

6.5.2 Normalisation d'un profil

Avec la notion de fichier local générique, il est délicat de construire des commandes Shell qui seront exécutées sur une autre machine. Comme il ne faut pas non plus enlever les références aux noms génériques, la fonction DupNormaProfil va permettre de dupliquer un profil en normalisant les noms génériques. Après utilisation de ce profil temporaire pour la construction des commandes, il peut être libéré.

Interface :

```
/*-------------------------------------------------------------------------
   Cree une nouvelle structure profil en remplacant
   le user et la machine generique par LOGINUSER et HOSTNAME
-------------------------------------------------------------------------
*/
PTDescFic DupNormaProfil
  (
   PTDescFic *rac  /*  -> Racine de la liste             */
  );               /* <-  Racine sur le profil normalise */
```

DupNormaProfil prend comme référence LOGINUSER et HOSTNAME, ce qui ne convient pas pour un profil ouvert sur une autre machine. Il y a donc également DupNormaProfilRef pour effectuer la même opération par rapport à une autre référence.

```
/*-------------------------------------------------------------------------
   Cree une nouvelle structure profil en remplacant
   le user et la machine generique par usrRef et macRef
-------------------------------------------------------------------------
*/
PTDescFic DupNormaProfilRef
  (
   PTDescFic *rac,  /*  -> Racine de la liste   */
   String usrRef,   /*  -> Machine de reference */
   String macRef    /*  -> user de reference    */
  );                /* <-  Racine sur le profil normalise */
```

6.6 La fonction sysint()

La fonction sysint() est disponible avec lib_exec_aster et lib_asterix. Selon le contexte elle a un comportement différent. Dans le cadre d'une exécution non graphique, elle permet de centraliser les commandes systèmes exécutées, et de prendre en compte les paramètres (-bavard et -noexec).

Interface :

```
/*-------------------------------------------------------------------------
   Lance une commande shell.
-------------------------------------------------------------------------
*/
int sysint
  (
   String cmdstring /* -> commande a soummettre en Bourne Shell */
  );
```

6.7 Soumission d'une étude Aster

La soumission d'une exécution du *Code_Aster* se fait en général à partir d'un profil d'étude et d'un profil d'exécution. Selon les interfaces proposées, les profils doivent être sous forme d'une liste chaînée où chaque élément est une structure de description de fichier, ou bien désignés par un nom de fichier.

Toutes les fonctions décrites ici ont un comportement différent si elles sont appelées uniquement via `lib_exec_aster` ou `lib_asterix`. Si elles sont appelées via `lib_asterix`, il y a couplage avec l'application de suivi des job : asjob.

6.7.1 Création/soumission d'un script d'exécution

Cette fonction intègre la normalisation des profils (`DupNormaProfil`). Elle construit les commandes pour exécuter le *Code_Aster* en fonction des profils donnés. Si la soumission du job construit est demandée, il est envoyé sur la machine d'exécution (`machE`) par l'intermédiaire du user `userE`.

A partir de la version 3.2 de `lib_asterix` l'appel a beaucoup évolué. La machine d'exécution et le user d'exécution doivent êtres passés en paramètres. Sont également rajoutés les paramètres `ficunic`, `origine` (pour le traçage des utilisations du *Code_Aster*) et `suivi_int` (pour le suivi interactif de l'exécution). Pour éviter toute confusion, le nom de cette fonction a changé (`ConsJob() -> AsConsJob()`).
Interface :

```
/*----------------------------------------------------------------------
   Construction d'un job a soumettre a la machine d'execution pour lancer
Aster
----------------------------------------------------------------------
*/
String AsConsJob
  (
   PTDescFic *RacE,      /*  -> Racine de la liste profil d'Etude     */
   PTDescFic *RacX,      /*  -> Racine de la liste profil d'Execution */
   String     ver,       /*  -> Version Aster STA2 NEW2 ....          */
   int        tps,       /*  -> Temps en s pour le job                */
   int        tpg,       /*  -> Temps en s pour la gestion du job     */
   int        mem,       /*  -> Memoire pour le job en Mmots          */
   int        cpt,       /*  -> Compte d'imputation                   */
   String     classe,    /*  -> Classe job batch nbatch wbatch        */
   String     nomjobp,   /*  -> Nom du job                            */
   bool       aq,        /*  -> Sauvegarde aq des fichiers de donnees */
   bool       exec,      /*  -> Flag execution ou non                 */
   String     userE,     /*  -> user sur la machine d'execution       */
   String     machE,     /*  -> machine d'execution                   */
   String     ficunic,   /*  -> fichier script par defaut sous la forme
                            user@machine:repertoire/fic_script
                            ou "" si le fichier est deja defini
                            dans le profil d'etude                   */
   String     origine,   /*  -> application d'origine                 */
   bool       suivi_int  /*  -> suivi interactif de l'execution       */
  );                     /* <-  Message d'erreur                      */
```

Cette fonction peut renvoyer un message d'erreur si elle détecte une anomalie. Dans le cas contraire elle renvoie une chaîne vide (`'\0'`).

Si le pointeur sur le profil d'exécution est différent de `NULL` et qu'il contient un fichier de type `unic`, alors ce fichier est mis à jour. S'il n'existe pas, il est créé avec son nom par défaut : `REP_ETUDE/nom_etude.unic` et ajouté au profil d'étude.

S'il n'y a pas de profil d'étude et un profil d'exécution, le même test est effectué. S'il y a un fichier unic il est mis à jour, sinon il est créé avec son nom par défaut : REP_EXEC/nom_exec.unic et ajouté au profil d'exécution.

Si le flag exec est FAUX le fichier unic est remplacé ou créé mais n'est pas soumis à la machine d'exécution. Si le flag exec est VRAI, il est en plus soumis à la machine d'exécution, en liaison avec le suivi des jobs.

La sauvegarde AQ est la recopie du fichier de commande (comm), du fichier de messages (mess) et du fichier de résultats (resu) en ajoutant à leurs noms respectifs la date sous la forme : _jj-mm-aa_hh.mm.ss+.

Depuis la version 3.6, une nouvelle interface est ajoutée pour prendre en compte des comptes d'imputation alphanumériques (comme sur SSU) et pour retourner à l'utilisateur le numéro d'identification du job soumis.

```
/*----------------------------------------------------------------------
  Construction d'un job a soumettre a al machine d'execution pour lancer
Aster.
  Pour tenir compte du SSU le compte d'imputation est une chaine de car.
  au lieu d'un entier. Le num du job soumis est retourne.
----------------------------------------------------------------------
*/
String AsConsJobNum
  (
   PTDescFic *RacE,      /*  -> Racine de la liste profil d'Etude     */
   PTDescFic *RacX,      /*  -> Racine de la liste profil d'Execution */
   String     ver,       /*  -> Version Aster STA2 NEW2 ....          */
   int        tps,       /*  -> Temps en s pour le job                */
   int        tpg,       /*  -> Temps en s pour la gestion du job     */
   int        mem,       /*  -> Memoire pour le job en Mmots          */
   String     scpt,      /*  -> Compte d'imputation                   */
   String     classe,    /*  -> Classe job batch nbatch wbatch        */
   String     nomjobp,   /*  -> Nom du job                            */
   bool       aq,        /*  -> Sauvegarde aq des fichiers de donnees */
   bool       exec,      /*  -> Flag execution ou non                 */
   String     userE,     /*  -> user sur la machine d'execution       */
   String     machE,     /*  -> machine d'execution                   */
   String     ficunic,   /*  -> fichier unicos par defaut sous la forme
                            user@machine:repertoire/fic_unicos
                            ou "" si le fichier est deja defini
                            dans le profil d'etude                   */
   String     origine,   /*  -> application d'origine                 */
   bool       suivi_int, /*  -> suivi interactif de l'execution       */
   int        numId      /* <-  numero du job soumis                  */
  );                     /* <-  Message d'erreur                      */
```

6.7.2 Soumission d'un script d'exécution

Recopie un script d'exécution sur la machine d'exécution (machE) et le soumet au gestionnaire de batch par l'intermédiaire du user userE. Transmet également les informations au suivi des jobs.

A partir de la version 3.2 de lib_asterix, l'appel a beaucoup évolué. La machine d'exécution et le user d'exécution doivent êtres passés en paramètres. Est également rajouté le paramètre origine (pour le traçage des utilisations du *Code_Aster*). Pour éviter toute confusion le nom de cette fonction a changé (SubaFicUnic() -> AsSubaFicUnic()).

Interface :

```
/*-----------------------------------------------------------------------
   Recopie d'un script Unicos sur la machine d'execution (MACHINE_EXEC) et
   soumission avec qsub avec le user OIAUser. Transmets les informations au
   suivi des jobs.
-----------------------------------------------------------------------
*/
void AsSubaFicUnic
  (
   String err,    /* <-  message d'erreur               */
   String umac,   /*  -> machine ou est le script       */
   String uusr,   /*  -> user pour recopier le script   */
   String urep,   /*  -> repertoire ou est le script    */
   String ufic,   /*  -> nom du fichier script          */
   String nomjobp,/*  -> nom a donner au job            */
   String usrExec,/*  -> user pour executer le job      */
   String macExec,/*  -> machine d'execution            */
   String origine /*  -> application qui soummet le job */
  );
```

Depuis la version 3.6, une nouvelle interface est ajoutée pour prendre en compte des comptes d'imputation alphanumériques (comme sur SSU) et pour retourner à l'utilisateur le numéro d'identification du job soumis.

```
/*-----------------------------------------------------------------------
   Recopie d'un script sur la machine d'execution (MACHINE_EXEC) et
   soumission au gestionnaire de batch. Transmets les informations au
   suivi des jobs. Le num d'identification du job soumis est retourne.
-----------------------------------------------------------------------
*/
void AsSubaFicUnicNum
  (
   String err,    /* <-  message d'erreur                    */
   String umac,   /*  -> machine ou est le script           */
   String uusr,   /*  -> user pour recopier le script       */
   String urep,   /*  -> repertoire ou est le script        */
   String ufic,   /*  -> nom du fichier script              */
   String nomjobp,/*  -> nom a donner au job                */
   String usrExec,/*  -> user pour executer le job          */
   String macExec,/*  -> machine d'execution                */
   String origine,/*  -> application qui soumet le job      */
   String cpt,    /*  -> compte d'imputation
                       ("0" pour avoir le compte par defaut) */
   String fictail,/*  -> fichier de message pour le tail    */
   int    numNQS  /* <-  numero d'identification du job     */
  );
```

6.8 Soumission à partir d'un fichier profil

Cette fonction a été plus spécialement réalisée pour soumettre une exécution sans couplage avec le suivi des jobs, à partir d'un fichier de profil d'étude de type asterix. Le fichier de profil doit contenir une référence à un fichier de paramètres valides (version Aster, temps max, ...).

Interface :

```
/*----------------------------------------------------------------------
   Execution sans environnement X d'une etude Aster a partir d'un profil
   d'etude de type asterix designe par un nom de fichier. Si le nom de fichier
   donne est en relatif, le fichier est recherche a partir du repertoire courant.
   Le fichier profil doit contenir une reference a un fichier parametre.
----------------------------------------------------------------------*/
String AsExecFicProfil(
   String  nfic,     /*  -> nom du fichier profil de type asterix          */
   String  napp,     /*  -> nom de l'application qui soummet le job        */
   String  flocal,   /*  -> repertoire flasheur en local
                            facultatif (on peut mettre (String)0)          */
   String  fexec,    /*  -> repertoire flasheur sur machine d'execution
                            facultatif (on peut mettre (String)0)          */
   int    *njob      /* <-  numero du job soumis (si =0, il y a un probleme)*/
   );                /* <-  message d'erreur ou d'information a desallouer
                            avec free() si l'on veut recuperer la place memoire
                            = 0 s'il n'y a pas d'erreur detectee ni de message
                                a transmettre a l'utilisateur              */
```

6.9 Utilitaires en vrac

6.9.1 Version de `lib_exec_aster`

Pour connaître la version de la librairie utilisée.

Interface :

```
String Version_lib_exec_aster( void );
```

6.9.2 basename

Pour assurer la même fonctionnalité que la commande système Unix éponyme : enlever le chemin dans un nom de fichier (entre le dernier caractère '/' et la fin de la chaîne).

Interface :

```
/*----------------------------------------------------------------------
   Pour assurer la fonction de la commande Unix eponyme.
----------------------------------------------------------------------
*/
void basename
  (
   String path,  /*  -> chemin a analyser    */
   String base   /* <-  nom extrait          */
  );
```

6.9.3 dirname

Pour assurer la même fonctionnalité que la commande système Unix éponyme : extraire le chemin dans un nom de fichier (entre le début la chaîne et le dernier caractère '/').

Interface :

```
/*-----------------------------------------------------------------------
   Pour assurer la fonction de la commande Unix eponyme.
-----------------------------------------------------------------------
*/
void dirname
  (
   String path, /*  -> chemin a analyser    */
   String dir   /* <-  chemin extrait       */
  );
```

6.9.4 Fichier sans son extension : QuelBase

Renvoie un nom de fichier sans son extension avec son chemin complet (supprime les caractères entre le dernier '.' et la fin de la chaîne).

Interface :

```
/*-----------------------------------------------------------------------
   Renvoyer le nom d'un fichier sans son extension.
-----------------------------------------------------------------------
*/
String QuelBase
  (
   String fic  /*  -> nom de fichier    */
  );           /* <- nom sans extension */
```

6.9.5 Radical d'un fichier : radical

Renvoie un nom de fichier sans son extension et sans son chemin (extrait les caractères entre le dernier '/' et le dernier '.').

Interface :

```
/*-----------------------------------------------------------------------
   Renvoie le "radical" d'un nom de fichier =
   tous les caracteres compris entre le dernier '/', s'il
   existe, et le dernier '.',s'il existe.
   Tiens compte des noms complets : user@machine:fic
-----------------------------------------------------------------------
*/
String radical
  (
   String exp  /*  -> chaine dont il faut extraire un radical */
  );           /* <-  radical de la chaine                    */
```

6.9.6 Extension d'un fichier : QuelExt

Renvoie l'extension d'un fichier (extrait les caractères entre le dernier '.' et la fin de la chaîne).

Interface :

```
/*-----------------------------------------------------------------------
   Renvoyer l'extension d'un fichier.
-----------------------------------------------------------------------
*/
String QuelExt
  (
   String fic /*  -> nom du fichier */
  );          /* <-  extension      */
```

6.9.7 Composer un nom de fichier au format rcp : compNFC

Pour créer un nom de fichier composé au format `rcp` : `user@machine:nom_de_fichier`. Si les noms génériques pour le user et la machine sont utilisés, ils sont changés en noms réels.

Interface :

```
/*-----------------------------------------------------------------------
  Cree un nom de fichier compose sous la forme :
         user@machine:nom_du_fichier_avec_path
-----------------------------------------------------------------------
*/
void compNFC
  (
   String nfc,   /* <-  Nom du fichier compose */
   String usr,   /*  -> User                  */
   String mac,   /*  -> Machine               */
   String rep,   /*  -> Path                  */
   String fic    /*  -> Nom de Fichier        */
  );
```

6.9.8 Décomposer un nom de fichier au format rcp : decompNFC

Pour décomposer un nom de fichier au format `rcp` : `user@machine:nom_de_fichier`.

Interface :

```
/*-----------------------------------------------------------------------
   Decompose un nom de fichier compose de la forme :
        user@machine:nom_du_fichier_avec_path
-----------------------------------------------------------------------
*/
void decompNFC
  (
   String nfc,      /*  -> Nom du fichier compose   */
   String usr,      /* <-  User                     */
   String mac,      /* <-  Machine                  */
   String ficsel    /* <-  Nom de fichier complet   */
  );
```

6.9.9 Accoler un nom de répertoire et un nom de fichier : Ficsel

Pour composer un nom de fichier à partir de son chemin et d'un nom (en mettant un '/' et un seul entre les deux).

Interface :

```
/*-----------------------------------------------------------------------
   Donner un nom complet de fichier a partir d'un repertoire et
   d'un nom de fichier (pour mettre un / au milieu -et un seul-)
-----------------------------------------------------------------------
*/
String Ficsel
  (
   String rep,  /*  -> nom de repertoire              */
   String fic   /*  -> nom de fichier                 */
  );            /* <- nom de repertoire/nom de fichier */
```

6.9.10 'Normalisation' d'une chaîne de caractères : strlennor

Pour supprimer les caractères parasites et 'invisibles' que l'on peut récupérer dans un champ éditable Motif. Suppression des caractères blancs `' '`, tabulation `'\t'` et retour chariot `'\n'`. Renvoie également la longueur de la chaîne ainsi 'normalisée'.

Interface :

```
/*----------------------------------------------------------------------
   Renvoie le nb de caracteres d'une chaine sans compter :
    - les blancs,
    - les tabulations,
    - les fin de ligne,
   La chaine passee est renvoyee en enlevant les caracteres
   precedants.
   Permet de savoir si un champ est "apparement vide".
----------------------------------------------------------------------
*/
int strlennor
  (
   String mot /* <-> chaine a normaliser             */
  );          /* <-  longueur de la chaine normalisee */
```

6.9.11 Longueur d'une chaîne 'normalisée' : strlennor2

Renvoie la longueur qu'aurait la chaîne 'normalisée' avec les mêmes critères que pour `strlennor` mais sans modifier la chaîne. Sert à savoir si une chaîne contient des caractères 'signifiants'.

Interface :

```
/*----------------------------------------------------------------------
   Renvoie le nb de caracteres d'une chaine sans compter :
    - les blancs,
    - les tabulations,
    - les fin de ligne,
   La chaine passee n'est pas modifiee
----------------------------------------------------------------------
*/
int strlennor2
  (
   String mot /*  -> chaine a analyser               */
  );          /* <-  longueur de la chaine normalisee */
```

6.9.12 Contenu d'un fichier dans une chaîne : filestr

Pour mettre le contenu d'un fichier dans une chaîne de caractères. Le fichier doit être local. La chaîne est allouée dynamiquement, il est donc conseillé de la désallouer (par un `free()`) lorsque l'on en a plus besoin.

Interface :

```
/*----------------------------------------------------------------------
   Lire un fichier pour le mettre dans une chaine de caracteres
   (allouee pour la circonstance).
----------------------------------------------------------------------
*/
String filestr
  (
   String fed  /*  -> Nom du fichier a lire */
  );           /* <- chaine alouee par malloc contenant le fichier */
```

6.9.13 Copie de fichier ou répertoire entre machines

```
/*-----------------------------------------------------------------------
   Copie d'un fichier ou d'un repertoire quelles que soient la machine
   de depart et la machine de destination.
-----------------------------------------------------------------------
*/
String CopierConf
  (
   String Susr, /*  -> User du fichier source            */
   String Smac, /*  -> Machine du fichier source        */
   String srep, /*  -> Repretoire du fichier source     */
   String sfic, /*  -> Fichier source                   */
   String Dusr, /*  -> User du fichier destination      */
   String Dmac, /*  -> Machine du fichier destination   */
   String drep, /*  -> Repertoire du fichier destination */
   String dfic, /*  -> Fichier destination              */
   bool   conf  /*  -> Flag de confirmation ou non de l'ecrasement
                       d'un fichier existant            */
  );           /* <-  Mesage d'erreur                   */
```

7 Les services de `lib_asterix`

7.1 Initialisation de la bibliothèque

Interface :

```
/*-----------------------------------------------------------------------
   Initialistation de la lib_asterix et en particulier de la bsf
         - traitement des arguments non X-motif de lib_asterix
-----------------------------------------------------------------------
*/
void Init_lib_asterix
  (
   int    argc,   /*  -> nb d'argument de la ligne de commande   */
   char **argv,   /*  -> liste des arguments                     */
   String tmpdir, /*  -> Prefixe du repertoire pour les fichiers
                         temporaires                             */
   Widget topw    /*  -> ToplevelShell                           */
  );
```

Cette fonction fait appel à `Init_lib_exec_aster()`. Elle ajoute juste les initialisations spécifiques à l'environnement X-Motif comme le chargement des polices et pixmap nécessaires à la bsf.

Cette initialisation ne peut se faire que lorsque le `Widget toplevel` est connu.

7.2 Quitter l'application

Pour libérer proprement les ressources allouées par `lib_asterix`, notamment les fichiers et le répertoire de travail, la fonction `Quitter_lib_asterix()` est disponible.

Interface :

```
/*-----------------------------------------------------------------------
   Sortie de la lib_asterix avec suppression des fichiers temporaires.
-----------------------------------------------------------------------
*/
void Quitter_lib_asterix
  (
   bool SupprRep  /*  -> Suppression du repertoire temporaire */
  );
```

Cette fonction fait appel à `Quitter_lib_exec_aster()`.

7.3 Boîte de sélection de fichier : bsf

Il est possible d'avoir plusieurs bsf dans une application. En générale il y aura une bsf pour effectuer les sélections (en dialogue modal) et une ou plusieurs bsf non bloquantes (en dialogue non modal). Chaque bsf créée est indépendante et possède ses propres structures. Cependant le mécanisme de rémanence des répertoires utilisés est commun à toutes les bsf de l'application.

A partir de la version 3.0 de la `lib_asterix` la notion d'édition d'un fichier s'est étendue à l'exécution d'une commande Unix, définie par l'utilisateur, avec comme paramètre le fichier sélectionné. Le bouton Editer est identique mais il est surmonté d'un menu Commande... permettant de choisir une commande parmi celles définies. Un bouton Préférence... permet d'ajouter, modifier ou supprimer une commande (dans la limite de 20). Ces commandes sont conservées dans le fichier préférence `$HOME/.prefbsf`. A l'initialisation de la bsf ce fichier est lu pour mettre à jour les structures correspondantes.

Il est possible de faire référence aux commandes Unix de la bsf en créant des listes de `PushButton` donnant le nom des commandes. Ces listes peuvent êtres intégrées dans des `PullDownMenu` ou des `OptionMenu`. Ces listes sont mises à jour à chaque appel de la fenêtre de préférences de la bsf. (Exemple : choix des différents types d'éditeurs dans le panneau d'option d'asterix.)
A partir de la version 3.2 un nouveau mécanisme sur les répertoires rémanents est mis en place. Deux types de répertoires sont maintenant pris en compte dans la rémanence : les répertoires 'fixes' et les répertoires 'volatils'. Les répertoires fixes sont positionnés en tête du menu Répertoire ..., et ils sont en inverse vidéo (blanc sur fond noir). Ils ne changent pas de position quand on les invoque. Les répertoires volatils correspondent à la rémanence des versions précédentes. A chaque fois que l'on utilise un répertoire il est conservé. S'il existait déjà il passe en première position, sinon il est inséré en première position et les autres répertoires sont décalés vers le bas. S'il n'y a plus de place (15 + `$HOME`) le répertoire le plus ancien disparaît. Pour définir les répertoires fixes il faut utiliser la fenêtre de définition des préférences de la bsf (bouton Préférences ...). Le choix d'une machine fait apparaître la liste des répertoires associés. Par les mécanismes habituels (Ajouter, Modifier, Supprimer) on peut mettre la liste à jour. Pour que les modifications soient prises en compte il faut valider (avec le bouton du même nom) pour chaque machine. Il est possible de réarranger la liste avec les boutons 'Δ' et '∇' qui décalent la sélection d'une position. Le bouton Enregistrer sauvegarde dans le fichier `$HOME/.prefbsf` les modifications validées et la liste des répertoires de la machine courante si l'application ne gère pas ces informations. Les répertoires fixes et volatils peuvent êtres mélangés dans la liste, mais lorsqu'ils sont intégrés dans le menus Répertoire... de la bsf les répertoires fixes sont regroupés en tête.

Par défaut les commandes Unix et les répertoires rémanents sont conservés dans le fichier de préférence de la bsf (`$HOME/.prefbsf`) pour toutes les applications utilisant `lib_asterix`. Cependant `lib_asterix` permet à l'application de conserver ces informations dans son propre fichier de préférences à travers quelques utilitaires (cette solution est adoptée par asterix).

Pour le paramètrage de l'impression voir [bib1].

7.3.1 Création et appel d'une bsf en dialogue modal

Interfaces :

```
/*-----------------------------------------------------------------------
   Creation des differents widgets lies a une boite de selection
   de fichier modale.
   Initialisation pour la bsf mais egalement pour lib_asterix :
         - initialisation des curseurs
         - initialisation des polices de caracteres
         - creation du dialogue d'impression
         - creation du dialogue d'alerte
         - creation du dialogue d'information
-----------------------------------------------------------------------
*/
Widget CreateBSF
  (
   Widget appSh   /* -> ApplicationShell */
  );
```

Crée les structures de données pour une *bsf* et renvoie le Widget alloué.

```
/*-----------------------------------------------------------------------
   Affiche la bsf wBSF en dialogue modal avec initialisation de
   ses parametres et renvoie les memes parametres modifies.
-----------------------------------------------------------------------
*/
bool ShowBSFmodal
  (
   Widget wBSF,      /*  -> BSF a afficher en modal          */
   String tit,       /*  -> Titre du shell                   */
   String mac,       /* <-> Machine                          */
   String usr,       /* <-> User                             */
   String rep,       /* <-> Repertoire                       */
   String fic,       /* <-> Fichier                          */
   String nEdi,      /*  -> Nom de l'editeur                 */
   String nAct,      /*  -> Nom de l'action de la fsb        */
   String fil,       /* <-> Filtre sur les fichiers          */
   String aff,       /*  -> Mode d'affichage du ls           */
   String tsel       /*  -> Titre de la Selection            */
  );                 /* <-  VRAI si un fichier ou reptoire est selectionne
*/
```

`tit` : titre du Widget Shell de la bsf
`mac` : machine sélectionnée
`usr` : user sélectionné
`rep` : répertoire sélectionné (si vide = `$HOME`)
`fic` : fichier sélectionné
`nEdi` : nom de l'éditeur associé (parmi les commandes Unix définies dans le fichier `.prefbsf` si la commande n'est pas définie c'est la première commande de la liste qui est utilisée)
`nAct` : nom de l'action de la bsf (bouton OK)
`fil` : filtre Unix pour le `ls`, le `find` ou le `grep` (= expression régulière)
`aff` : mode d'affichage du `ls` (parmi `Date` -classé du plus récent au plus ancien- ou `Nom` -par ordre alphabétique-)
`tsel` : label du champ fichier sélectionné

7.3.2 Création et appel d'une bsf en dialogue non modal

Interfaces :

```
/*----------------------------------------------------------------------
   Creation des differents widgets lies a une boite de selection
   de fichier non modale (Autonome)
   Initialisation pour la bsf mais egalement pour lib_asterix :
         - initialisation des curseurs
         - initialisation des polices de caracteres
         - creation du dialogue d'impression
         - creation du dialogue d'alerte
         - creation du dialogue d'information
----------------------------------------------------------------------
*/
Widget CreateBSFA
  (
   Widget appSh   /* -> ApplicationShell */
  );

/*----------------------------------------------------------------------
   Affiche la bsf wBSF en dialogue non modal avec initialisation
   de ses parametres.
----------------------------------------------------------------------
*/
bool ShowBSFnm
  (
   Widget wBSF,      /*  -> BSF a afficher en modal          */
   String tit,       /*  -> Titre du shell                   */
   String mac,       /*  -> Machine                          */
   String usr,       /*  -> User                             */
   String rep,       /*  -> Repertoire                       */
   String fic,       /*  -> Fichier                          */
   String nEdi,      /*  -> Nom de l'editeur                 */
   String nAct,      /*  -> Nom de l'action de la fsb        */
   String fil,       /*  -> Filtre sur les fichiers          */
   String aff,       /*  -> Mode d'affichage du ls           */
   String tsel       /*  -> Titre de la Selection            */
  );
```

Il est à noter ici que, le dialogue n'étant pas modal, la dernière sélection effectuée par la bsf affichée ne peut être retournée. Ces informations sont quand même conservées dans les structures de la bsf et l'on peut les récupérer pour réafficher la bsf avec son dernier contexte grâce à la fonction `GetRemBSF`.

Interface :

```
/*----------------------------------------------------------------------
   Renvoie la remanence de la BSF dont le Widget Shell est wBSF.
----------------------------------------------------------------------
*/
void GetRemBSF
  (
   Widget wBSF, /*  -> Widget Shell de la BSF  */
   String usr,  /* <-  User selectionne        */
   String mac,  /* <-  Machine selectionnee    */
   String rep,  /* <-  Repertoire selectionne  */
   String fic,  /* <-  Fichier selectionne     */
   String fil   /* <-  Filtre                  */
  );
```

7.3.3 Commandes de lancement des éditeurs (jusqu'en 2.2)

Jusqu'à la version 2.2 les commandes de lancement des éditeurs sont par défaut :

```
char CMDxedit[1024]     = "xedit -fn Courier12 ";
char CMDvi[1024]        = "xterm -sb -e vi ";
char CMDemacs[1024]     = "emacs ";
char CMDtextedit[1024]  = "textedit ";
char CMDsedit[1024]     = "asedit ";
```

L'intialisation est effectuée dans `lib_asterix.h`, mais il est bien évidemment possible de changer les commandes à tout moment.

7.3.4 Les commandes Unix de la bsf et les listes de `PushButton` associées

Il n'est possible de définir que 20 commandes au maximum.

7.3.4.1 Gestion des listes de PB

Une application ne peut contenir que 24 listes de ce type. Pour chaque bsf créée il faut 2 listes : une pour la bsf principale et une pour la bsf de copie.

Comportement de cette liste : elle est mise à jour après chaque appel à la fenêtre des préférences de la bsf. Si elle est utilisée dans un contexte `OptionMenu`, l'`activateCallback` des `PushButton` de la liste met à jour les structures de l'`OptionMenu` en conséquence. Si l'on donne le `Widget-id` d'un `PushButton`, son `label` est mis à jour par l'`activateCallback` des `PushButton` de la liste. Après une mise à jour, pour chaque liste, si la commande précédemment sélectionnée existe encore dans la liste elle reste la commande sélectionnée, sinon c'est la première commande de la liste qui devient la commande courante.

7.3.4.2 Création d'une liste

```
/*-----------------------------------------------------------------------
   Creation d'une liste de commandes Unix liee au mecanisme de la BSF
   pour etre utilise dans l'application.
   Renvoie un identificateur pour referencer le structure cree.
-----------------------------------------------------------------------
*/
int InitPBprefBSF
  (
   Widget  wpm,     /* widget parent          */
   Widget  wom,     /* widget OM              */
   Widget  wedit
  );
```

7.3.4.3 Sélection d'une commande Unix d'un `OptionMenu` à partir de son nom

```
/*-----------------------------------------------------------------------
   ActivateCallback de la commande ncmd de la structure id de commandes Unix
   liees au mecanisme BSF.
-----------------------------------------------------------------------
*/
void activePBcmdUnix
  (
   int    id,     /* -> Identificateur de la liste de PB */
   Widget wom,    /* -> OptionMenu associe               */
   String ncmd    /* -> Nom de la commande Unix associee */
  );
```

7.3.4.4 Récupération du nom d'une commande Unix sélectionnée avec un `OptionMenu`

```
/*----------------------------------------------------------------------
   Renvoie le nom de la commande Unix selectionne dans l'OptionMenu d'une
   liste de PB associee au mecanisme des BSF.
----------------------------------------------------------------------
*/
String historiqueOMcmdUnix
  (
   int    id,   /*  -> identificateur de la liste des PB    */
   Widget wom   /*  -> OptionMenu concerne                   */
  );            /* <-  Nom de la commande Unix selectionnee */
```

7.3.4.5 Appel de la fenêtre des préférences de la bsf

```
/*----------------------------------------------------------------------
   Afficher, en dialogue modal, les preferences de la bsf.
   Renvoie VRAI si les modifications ont ete enregistrees dans
$HOME/.prefbsf
   et les modifications reportees dans les OptionsMenus et les PullDownMenu
   lies a ce mecanisme.
----------------------------------------------------------------------
*/
bool ShowPrefBSF(void);
```

7.3.5 Conservation des commandes Unix et des répertoires rémanents par l'application.

Les informations concernées doivent êtres mises dans un fichier à mots-clés de la forme :

```
'mot clé : valeur'
```

Si l'application ne possède pas ce type de fichier de préférences il est possible d'en créer un pour l'occasion.

7.3.5.1 Prévenir `lib_asterix`

Il faut indiquer à `lib_asterix` que l'on prend en charge la sauvegarde pour ne pas prendre en compte le fichier `$HOME/.prefbsf` à l'initialisation et surtout pour ne pas l'écraser lors de la sauvegarde. Cette déclaration doit être faite après l'initialisation de `lib_asterix` : `Init_lib_asterix()`.

```
/*----------------------------------------------------------------------
   Pour que l'application puisse declarer qu'elle gere les repertoires
   remanents et les commandes Unix dans son fichier de preferences.
----------------------------------------------------------------------
*/
void AsGestionPrefBSF( void );
```

7.3.5.2 Lecture des informations

```
/*----------------------------------------------------------------------
   Lecture des commandes Unix pour la bsf dans un fichier de preferences.
   Mise a jour des structures correspondantes.
   Appelee lorsque l'on rencontre le mot cle 'Definition des commandes Unix
:'.
   Le fichier est lu jusqu'au mot cle 'Fin definition des commandes Unix :'.
----------------------------------------------------------------------
*/
void AsLireCmdUnix
  (
   FILE *fl   /*  -> identificateur du fichier de preferences */
  );
/*----------------------------------------------------------------------
   Lecture des repertoires remanents pour la bsf dans un fichier
   de preferences.
   Mise a jour des structures correspondantes.
   Appelee lorsque l'on rencontre le mot cle 'Repertoires remanents :'.
   Le fichier est lu jusqu'au mot cle 'Fin repertoires remanents :'.
----------------------------------------------------------------------
*/
void AsLireRepRem
  (
   FILE *fl   /*  -> identificateur du fichier de preferences */
  );
```

7.3.5.3 Sauvegarde des informations

```
/*----------------------------------------------------------------------
   Ecriture des repertoires remanents de la bsf dans un fichier
   de preferences.
----------------------------------------------------------------------
*/
void AsEcrireCmdUnix
  (
   FILE *fl   /*  -> identificateur du fichier de preferences */
  );

/*----------------------------------------------------------------------
   Ecriture des repertoires remanents de la bsf dans un fichier
   de preferences.
----------------------------------------------------------------------
*/
void AsEcrireRepRem
  (
   FILE *fl   /*  -> identificateur du fichier de preferences */
  );
```

7.4 Exécuter une commande système interruptible : sysint()

Les outils d'asterix utilisent tous cette fonction à chaque fois qu'une commande système est potentiellement bloquante (ls en présence de montage NFS, rcp, rsh) ou une commande longue (find ou grep récursif). Pour l'utiliser en dehors de la bsf et du suivi des jobs il faut définir dans l'interface un bouton Interrompre.

7.4.1 Bouton Interrompre

Ce bouton n'apparaît que le temps de l'exécution de sysint(). Aucune action sur un autre bouton n'est possible tant que celui-ci est visible (les événements X sont consommés). Il peut y avoir plusieurs boutons Interrompre dans l'application. A un instant donné un seul bouton Interrompre est actif. En général lorsque l'on affiche une fenêtre ayant ce type de bouton, on déclare ce bouton courant (DefWint). Lorsque l'on ferme cette fenêtre il faut rendre ce bouton inactif pour qu'un autre bouton Interrompre prenne le relais (UndefWint).

L'événement d'interruption est détecté à partir d'un événement de type KeyRelease dans le Widget Interrompre.

7.4.1.1 Définir le bouton interrompre courant DefWint()

Peut se faire lors de la création du bouton pour la fenêtre principale de l'application ou lors du popupCallback pour une fenêtre qui n'est pas toujours affichée.

Interface :

```
/*-------------------------------------------------------------------------
   Definir le bouton interrompre courant.
-------------------------------------------------------------------------
*/
void DefWint
  (
   Widget wid    /* -> Widget du nouveau bouton Interrompre courant */
  );
```

7.4.1.2 Rendre inactif un bouton interrompre UndefWint()

A faire lorsque l'on masque une fenêtre contenant un bouton Interrompre rendu courant par DefWint (lors du popdownCallback).

Interface :

```
/*-------------------------------------------------------------------------
   Annuler le bouton interrompre courant et revnir au precedent.
-------------------------------------------------------------------------
*/
void UndefWint
  (
   Widget wid  /* -> bouton Interrompre a rendre inactif (depiler) */
  );
```

7.4.1.3 Exemple de déclaration de bouton Interrompre dans la fenêtre principale de l'application

Dans l'interface, à la création du bouton, il faut appeler les fonctions suivantes :

```
DefWint(Widget_du_bouton);/* Pour signaler que le bouton interrompre courant
*/
                                /* est celui-ci */
```

Exemple de définition d'un bouton Interrompre avec XFM :

```
xfmCreateCallback =
   function None DefWint(Widget);
   DefWint(self);
width = 102
height = 35
highlightThickness = 0
labelString = Interrompre
```

7.4.2 La fonction sysint()

Exécute une commande système en Bourne Shell avec possibilité de l'interrompre. Si le paramètre -bavard a été transmis à l'application, la commande exécutée est envoyée dans une fenêtre de texte avec ascenseur. Si aucun bouton Interrompre n'a été défini c'est la commande system() qui est utilisée. Sinon la commande est exécutée après un vfork. Tant que la commande n'est pas finie et que l'on a pas cliqué dans le bouton Interrompre, le processus père attend la terminaison du processus fils. Toutes les actions X sur les boutons autres que Interrompre sont consommées (pour ne pas avoir à gérer la cohérence des actions qui seraient "bufferisées").

Jusqu'à la version 2.3 l'interruption était détectée à l'aide de la variable globale interrompre mise à jour dans l'activateCallback du bouton. Celle-ci n'est plus utilisée, mais si elle existe, elle ne nuit pas au bon fonctionnement du mécanisme.

Cette fonction tient compte de deux arguments transmis à l'exécutable :

- -bavard qui permet de renvoyer l'écho dans un fenêtre des commandes Shell exécutées via la fonction sysint(),
- -noexec qui permet de ne pas exécuter les commandes Shell exécutées via la fonction sysint().

Interface :

```
/*-----------------------------------------------------------------------
   Lance une commande shell avec possibilite de l'interrompre.
   Pour cela a besoin de connaitre un Widget bouton d'arret
   d'urgence.
-----------------------------------------------------------------------
*/
int sysint
  (
   String cmdstring /* -> commande a soummettre en Bourne Shell */
  );
```

Renvoie 0 si la soumission de la commande s'est bien passée et 1 sinon. (Ce code de retour n'est pas le code de retour de la commande elle même comme pour la fonction system().)

7.4.3 Détection des erreurs : errshellRSH()

Pour détecter les erreurs d'une commande système soumise depuis un programme C il faut analyser les messages envoyés dans stderr. Lorsque, dans une commande système, on redirige stderr dans un fichier, la fonction errshellRSH() permet d'analyser ce fichier pour déterminer si la commande s'est mal passée. Si le fichier dans lequel on a redirigé stderr n'est pas vide la fonction renvoie VRAI et la première ligne du fichier. Cette fonction détecte en plus le message sur un mot de passe périmé sur le Cray depuis Unicos 8.0 (il n'y a pas encore d'équivalent sur claster). Si un mot de passe est périmé, la fonction propose de le changer (via un rlogin) si l'on n'utilise pas lib_asterix dans le cadre d'ASURE. Si l'on utilise ce mécanisme avec ASURE, il n'est pas possible d'effectuer directement un rlogin, il faut effectuer une connexion avec un authentifieur. Dans ce cas l'utilisateur est prévenu par un message dans une fenêtre popup.

Interface :

```
/*-----------------------------------------------------------------------
   Renvoie VRAI si le fichier tmperr est non vide et faux sinon.
   Si ce fichier est non vide sa premiere ligne est recopiee dans
   la variable err.
   pour tester si le mot de passe est perime
   et envoyer une connexion pour changer le mot de passe si l'on utilse pas
   ASURE.
-----------------------------------------------------------------------
*/
bool errshellRSH
  (
   String err,    /* <-  contenu de la premiere ligne du fichier tmperr   */
   String mac,    /*  -> machine destination du rsh                       */
   String usr,    /*  -> user destionation du rsh                         */
   String ficerr  /*  -> nom du fichier dans lequel stdout a ete redirige */
  );
```

Renvoie VRAI si ficerr est non vide et FAUX sinon.

7.5 Suivi des *jobs* sur la machine d'exécution

Cette solution pour voir évoluer l'exécution des jobs sur la machine d'exécution nécessite un répertoire sur station qui va contenir la liste des jobs gérés par le suivi et les fichiers de résultat qui sont associés (FLASH_STATION).

Jusqu'à la version 3.1 ce mécanisme était intégré dans l'application. Un job soumis par une autre application n'était pas pris en compte.

Depuis la version 3.2 c'est une application autonome (asjob) qui peut être utilisée par plusieurs applications simultanément. Pour conserver l'intérêt de cette fonctionnalité, il est souhaitable que toutes les applications utilisent le même répertoire flasheur sur station ($HOME/flash_Aster).

Pour une utilisation avec soumission de passages du *Code_Aster* seuls l'initialisation et l'affichage de la fenêtre de suivi de jobs sont à utiliser. Les fonctions de construction de scripts et de soumissions pour le *Code_Aster* intègrent l'utilisation du mécanisme de suivi. Pour les autres utilisations il faut, en plus, signaler à chaque fois qu'un nouveau job est à prendre en charge et renvoyer les informations attendues par le suivi des jobs dans des fichiers dans FLASH_STATION. La communication entre les applications qui soumettent des jobs et l'application de suivi est assuré par les mécanismes du serveur X (positionnement d'atomes et émission d'événements X).

7.5.1 Initialisation, création de la fenêtre de suivi

Avec la fonction suivante :

```
/*-----------------------------------------------------------------------
   Creation de la fenetre de suivi des jobs. et Initialisations
   inherentes a cette fonctionnalite, avec acces aux parametres de
connexion.
   Creation du repertoire flasheur sur station.
-----------------------------------------------------------------------
*/
bool AsCreateSuivJobTempo
  (
   String rep_flash_station, /*  -> Repertoire flasheur sur station
                                   avec chemin en absolu ou par rapport
                                   a $HOME.
*/
   bool   attente,           /*  -> Flag d'attente ou non de la fin de la
                                   connexion.
*/
   int    nbMaxEssaiConnex,  /*  -> nb max. d'essais de connexion.
*/
   int    nbMaxEssaiPhase,   /*  -> nb max. d'essais par phase.
*/
   int    delaiEssai         /*  -> delai d'attente (en s) entre chaque
                                   phase.
*/
  );                         /* <-  VRAI si la connexion est etablie.
*/
```

Cette interface donnant accès aux paramètres de la tentative de connexion est utilisable à partir de la version 3.6. Pour conserver la compatibilité, ou si l'on ne veut pas modifier les paramètres par défaut, l'ancienne interface est conservée. Les valeurs par défauts des nouveaux paramètres sont :

- nbMaxEssaiConnex=10,
- nbMaxEssaiPhase=5,
- delaiEssai=1.

```
/*-----------------------------------------------------------------------
   Creation de la fenetre de suivi des jobs et initialisations
   inherentes a cette fonctionnalite.
-----------------------------------------------------------------------
*/
void AsCreateSuivJob
  (
   String rep_flash_station, /*  -> Repertoire flasheur sur station
*/
   bool   attente            /*  -> Attente ou non de la fin de
                                   l'initialisation de asjob
*/
  );
```

L'application essaye de communiquer avec une application de type asjob ayant le même répertoire flasheur sur station et utilisant le même DISPLAY X. S'il n'y a pas d'application correspondant à ces critères, une application asjob est lancée ('`asjob -rep_flash rep_flash_station`'). Si l'application est utilisée avec les possibilités `rcp`/`rsh` d'ASURE les paramètres adéquats sont transmis à asjob.

Si le répertoire n'existe pas il est créé avec `mkdir` (donc sur un seul niveau). L'application asterix utilise le répertoire `$HOME/flash_Aster`. Pour conserver l'aspect universel de ce suivi de jobs sur le serveur Aster, il est conseillé d'utiliser ce répertoire pour toutes les applications faisant appel aux services d'asjob.

Pour ne pas allonger l'initialisation de l'application, il est possible de ne pas attendre la fin du lancement d'asjob et l'initialisation complète de la communication entre les deux applications en positionnant le paramètre `attente` à FAUX. Dans ce cas la communication n'est réellement établie que lorsque l'on soumet un job ou que l'on demande la fenêtre de suivi. Pour éviter d'avoir plusieurs applications asjob en même temps, un mécanisme 'anti-rebond' est mis en place. Il n'est pas possible de lancer, dans la même application, deux fois asjob à moins de 6 secondes d'intervalle.

7.5.2 Afficher la fenêtre de suivi

Si aucune application asjob avec les bonnes caractéristiques n'est détectée, elle est lancée. Sinon la fenêtre de suivi est affichée (dé-inconifiée).

Avec la fonction suivante :

```
/*----------------------------------------------------------------------
   Affichage de la fenetre de suivi des jobs.
----------------------------------------------------------------------
*/
void AsShowSuivJob( void );
```

7.5.3 Ajouter un job à gérer par le suivi

Si aucune application asjob avec les bonnes caractéristiques n'est détectée elle est lancée.

Cette fonctionnalité n'est à utiliser que lorsque l'on se sert du suivi pour des jobs non Aster ou non soumis avec la fonction `AsSubaFicUnic()`.

A partir de la version 3.2 de `lib_asterix` le paramètre `origine` est ajouté (pour le suivi des utilisations du *Code_Aster*). Pour éviter toute confusion le nom de cette fonction a changé (`AddJob() -> AsAddJob()`).

Interface :

```
/*----------------------------------------------------------------------
   Inserer un job en fin de la liste chainee des jobs.
   Incrementer le nb de job non finis.
----------------------------------------------------------------------
*/
String AsAddJob
  (
   int    num,    /*  -> num. NQS du job           */
   String nom,    /*  -> nom du job                */
   String usr,    /*  -> user d'execution          */
   String mac,    /*  -> machine d'execution       */
   String fic,    /*  -> nom de fichier pour le tail (suivi interactif du
job) */
   int    tpi,    /*  -> temps initial demande     */
   String date,   /*  -> date soumission   jj/mm/aa */
   String heure,  /*  -> heure soumission  hh:mm:ss */
   String origine /*  -> application d'origine     */
  );              /* <-  message d'erreur a la soumission */
```

Cette fonction peut renvoyer un message d'erreur si elle détecte une anomalie. Dans le cas contraire elle renvoie une chaîne vide (`'\0'`).

7.5.4 Les fichiers que le job doit envoyer sur le flasheur station

Uniquement lorsque l'on utilise le suivi pour des jobs non Aster. Le suivi des jobs détecte la fin d'un job soit en interrogeant le gestionnaire de batch sur la machine d'exécution, soit en détectant l'existence du fichier FLASH_STATION/nomjob.dn°id . Lorsque la machine d'exécution répond que le job ne tourne plus, le suivi des jobs va chercher, pour mettre à jour sa liste, les fichiers suivants :

- FLASH_STATION/nomjob.dn°id qui doit contenir le diagnostic du job,
- FLASH_STATION/nomjob.mn°id qui doit contenir les messages du job.

Pour suivre de manière interactive le déroulement du job, le script doit renvoyer, dès que cette information lui est connue, le répertoire de travail dans lequel arrive le fichier de message dans le fichier :

- FLASH_STATION/nomjob.tn°id qui doit contenir le nom du répertoire temporaire.

7.5.5 Sauvegarde des informations du Suivi des Jobs

Les informations du Suivi des Jobs sont stockées en mémoire centrale sous la forme d'une liste chaînée. Le contenu de la liste est sauvegardé dans le fichier FLASH_STATION/.flashjob lorsque l'on quitte l'application de suivi des jobs et à chaque ajout ou suppression de job dans la liste.

7.5.6 Fermer l'application asjob à partir d'une autre application.

Normalement quand on sort de l'application qui communique avec asjob, asjob reste actif. On peut néanmoins envoyer un message à asjob pour que l'application se termine et ainsi coupler la fin de son application avec la fin d'asjob.

Interface :

```
void AsQuitSuivJob( void );
```

7.6 Soumission d'une étude Aster

Les jobs créés par lib_asterix utilisent le suivi des jobs. Cela implique d'avoir initialisé le suivi des jobs avant la première soumission : AsCreateSuivJob() [§7.5].

Toutes les fonctions de soumission d'une exécution Aster de lib_exec_aster [§6.7] sont appelables dans le contexte lib_asterix. Elles ont juste un comportement différent (couplage avec asjob, bouton Interrompre, ...).

7.7 Fenêtre de confirmation

La fenêtre de confirmation Motif n'est pas idéale à utiliser pour deux raisons :

- il n'est pas possible de changer la couleur des boutons (ce sont des `Gadget`),
- le seul moyen de sélectionner un bouton comme étant l'action par défaut est de le faire à la création de la fenêtre (il faut donc créer le `Widget` à chaque utilisation).

Pour remédier à cela, la bibliothèque `lib_asterix` propose un `Widget` appelable de deux façons différentes selon que l'on veut que l'action positive ou négative soit l'action par défaut.

Interface avec le bouton Oui par défaut :

```
/*----------------------------------------------------------------------
   Afficher, en dialogue modal, une boite d'alerte avec le bouton Oui
   selectionne par defaut.
----------------------------------------------------------------------
*/
bool ShowAlerteOui
  (
   String titre,  /*  -> titre du shell                          */
   String msg     /*  -> message a afficher                      */
  );              /* <-  VRAI si confirmation avec le bouton Oui */
```

Interface avec le bouton Non par défaut :

```
/*----------------------------------------------------------------------
   Afficher, en dialogue modal, une boite d'alerte avec le bouton Non
   selectionne par defaut.
----------------------------------------------------------------------
*/
bool ShowAlerteNon
  (
   String titre,  /*  -> titre du shell                          */
   String msg     /*  -> message a afficher                      */
  );              /* <-  VRAI si confirmation avec le bouton Oui */
```

Ces fonctions renvoient `VRAI` si l'action choisie est Oui et `FAUX` si l'action choisie est Non.

7.8 Édition

Mise à disposition des utilitaires pour 'éditer' des fichiers sur n'importe quelle machine. Les commandes utilisées sont définie dans le fichier `$HOME/.prefbsf` ou dans le fichier de préférence de l'application. En général on ne récupère pas de message d'erreur lorsque cela se passe mal (quand la commande n'est pas trouvée par exemple) car la commande est lancée en '`background`'.

Si la commande n'est pas définie c'est la première commande de la liste qui est utilisée.

7.8.1 Éditer un fichier

Interface :

```
/*-----------------------------------------------------------------------
   Envoie la commande d'edition CMD... pour un editeur donne
   sur n'importe quelle machine.
   En gal pas de retour d'erreur car lancement en background.
-----------------------------------------------------------------------
*/
String Editer
  (
   String edi,  /*  -> Nom editeur parmi : sedit xedit vi emacs textedit */
   String mac,  /*  -> Machine du fichier a editer                       */
   String usr,  /*  -> User du fichier a editer                          */
   String rep,  /*  -> Repertoire du fichier a editer
                       Si vide c'est $HOME                               */
   String fic   /*  -> Nom du fichier a editer                           */
  );            /* <-  Message d'erreur                                  */
```

7.8.2 Editer tous les fichiers "éditables" des profils

Pour éditer tous les fichiers ayant le 'flag' E (éditable) dans un profil d'étude et un profil d'exécution (et de manière plus générale dans un ou deux profils). Une session éditeur est lancée sur les trois premières machines trouvées dans le(s) profil(s) (pour éviter la multiplication des fenêtres). Cette fonction n'a de sens qu'avec un éditeur multifichier (`sedit`, `emacs`, `vi`). Si l'on utilise un éditeur n'existant pas sur toutes les machines, la commande d'édition sur cette machine n'est pas prise en compte (il n'y a pas de message d'erreur car la commande est en '`background`').

Interface :

```
/*-----------------------------------------------------------------------
   Lance une session d'editeur pour tous les fichiers a editer
   d'un profil d'etude et d'un profil d'execution. Une session
   par machine avec au plus 3 sessions.
   Pas de retour d'erreur car lancement en background.
-----------------------------------------------------------------------
*/
void ToutEditer
  (
   PTDescFic *RacE, /* -> Racine de la liste        Etude  */
   PTDescFic *RacX, /* -> Racine de la liste        Exec   */
   String     edi   /* -> Nom Editeur utilise parmi :
                          sedit xedit vi emacs textedit    */
  );
```

7.9 Fenêtre de visualisation de texte

Mise à disposition d'une fenêtre affichant, en dialogue modal, un texte dans une fenêtre avec ascenseur et avec la possibilité d'imprimer le texte à partir de la boîte d'impression de la bsf.

Création de la fenêtre :

```
/*-------------------------------------------------------------------------
   Creation d'une fenetre pour visualiser du texte
-------------------------------------------------------------------------
*/
void CreateVisuInfo
  (
   Widget appShell         /*  -> Application Shell */
  );
```

Affichage de la fenêtre :

```
/*-------------------------------------------------------------------------
   Afficher, en dialogue modal, un texte dans une srcolled window avec
   possibilite d'impression.
-------------------------------------------------------------------------
*/
void ShowVisuTexte
  (
   String titre,   /*  -> Titre de la fenetre               */
   String Linf,    /*  -> texte du label au dessus du texte */
   String txt      /*  -> texte a afficher                  */
  );
```

7.10 Quelques utilitaires en vrac

7.10.1 Version de `lib_asterix`

Pour connaître la version de la librairie utilisée.

Interface :

```
String Version_lib_asterix( void );
```

7.10.2 Nom de l'item courant d'un option-menu : historiqueOM

Renvoie le nom du `Widget` de l'item courant d'un `option-menu`.

Interface :

```
/*-------------------------------------------------------------------------
   Renvoie le nom du dernier Widget clique dans un option-menu
-------------------------------------------------------------------------
*/
String historiqueOM
  (
   Widget wom  /*  -> Widget de l'option-menu                     */
  );           /* <-  Nom du Widget selectionne dans l'option-menu */
```

7.10.3 Choisir un item d'un option-menu : activeClick

Pour sélectionner un item dans un `option-menu` et activer l'action correspondante.

Interface :

```
/*-----------------------------------------------------------------------
   Mettre a jour un option-menu a l'item donne.
-----------------------------------------------------------------------
*/
void activeClick
  (
   Widget wom,   /* -> Widget de l'option-menu    */
   Widget witem  /* -> Widget de l'item a activer */
  );
```

7.10.4 Imprimer une chaîne de caractères : ImprString

Écrit la chaîne de caractères dans un fichier temporaire et affiche la boîte d'impression de fichier de la bsf.

Interface :

```
/*-----------------------------------------------------------------------
   Imprimer une chaine de caracteres en utilisant la fenetre de choix
   de la destination (imprimante) et du format de la BSF.
-----------------------------------------------------------------------
*/
void ImprString
  (
   String str    /*  -> chaine a imprimer     */
  );
```

7.10.5 MAJ d'une ressource XmNlabelString

Avec création et libération de la chaîne `XmString` indispensable à l'opération.

```
/*-----------------------------------------------------------------------
   Pour affecter proprement un labelString a partir d'un String
  (avec liberation de la memoire du XmString).
-----------------------------------------------------------------------
*/
void SetNlabelString
  (
   Widget w,   /*  -> Widget possedant une ressource labelString */
   String str  /*  -> Chaine a mettre dans le labelString        */
  );
```

7.10.6 Récupération d'une ressource XmNlabelString

```
/*-----------------------------------------------------------------------
   Pour recuperer proprement un labelString d'un Widget
  (avec liberation de la memoire du XmString).
-----------------------------------------------------------------------
*/
void GetNlabelString
  (
   Widget w,   /*  -> Widget possedant une ressource labelString */
   String str  /* <-> labelString recupere                       */
  );
```

7.10.7 Affichage du curseur montre dans une fenêtre

```
/*----------------------------------------------------------------------
   Affiche le curseur Montre (immediatement)
----------------------------------------------------------------------
*/
void tempoM
 (
  Widget  wid   /*  -> Widget auquel il faut appliquer le curseur Montre */
 );
```

7.10.8 Retour à un curseur normal

```
void UnDefCur
  (
   Widget wid  /* -> Identificateur de la fenetre */
  )
/*************************************************************/
/* Positionne le curseur par defaut de la fenetre            */
/*************************************************************/
```

8 Exemple d'utilisation

8.1 Exemple pour `lib_exec_aster`

Exemple d'utilisation de `lib_exec_aster`, avec une application qui soumet un profil d'étude à partir d'un nom de fichier donné sur la ligne de commande (`-pret`).

```
/*#############################################################################
#
  Exemple d'utilisation de lib_exec_aster pour soumettre une execution
  sur un serveur Aster a partir d'un fichier profil d'etude asterix.
#############################################################################
#*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <lib_exec_aster.h>

static void sortie(int status);
static void usage(char *nc);

main (int argc,char **argv)
{
int status=0;
char *nfp=0;
char *nc=0;
char *nomod=0;
char* err;
int numID;

printf("Version de la bibliothèque lib_exec_aster :%s\n\n",
         Version_lib_exec_aster());
```

```
/*-------------------------------------------------------------------------
--
   Initialisation lib_exec_aster
-------------------------------------------------------------------------
-*/
Init_lib_exec_aster(argc,argv,"/tmp/asexec");

/*-------------------------------------------------------------------------
--
   Traitement des arguments specifiques a l'application
-------------------------------------------------------------------------
-*/
nc=AsGetNomAppli();
while (*argv) {
   if (strcmp(*argv,"-pret") == 0) {
      argv++;
      if (*argv) nfp=strdup(*argv);
      else {
         fprintf(stderr,"err:%s: Il manque le nom du fichier profil
d'étude\n",
             nc);
         sortie(1);
         }
      }
   else if ((strcmp(*argv,"-help") == 0) || (strcmp(*argv,"-h") == 0) ||
            (strcmp(*argv,"-aide") == 0) ) {
      usage(nc);
      sortie(0);
      }
   argv++;
   }
if (! nfp) {
   fprintf(stderr,"err:%s: Il manque le nom du fichier profil
d'étude\n",nc);
   usage(nc);
   sortie(1);
   }
/*-------------------------------------------------------------------------
--
   Soumission execution
-------------------------------------------------------------------------
-*/
err=AsExecFicProfil(nfp,"EXEC_ASTER",(String)0,(String)0,&numID);
if (numID == 0) {
      if (err) {
         fprintf(stdout,"%s:err %s\n",nc,err);
         free(err);
         }
      else {
         fprintf(stdout,"%s:err ???\n",nc);
         }
      sortie(1);
      }
else {
   fprintf(stdout,"%s:%s\n",nc,err);free(err);
   fprintf(stdout,"%s:Num. Job:%d\n",nc,numID);
   fprintf(stdout,"%s:Flasheur local:%s\n",nc,AsGetFlashStation());
```

```
fprintf(stdout,"%s:Flasheur exec :%s\n",nc,AsGetFlashExec());
}
```

```
/*-----------------------------------------------------------------------------
--
  Sortie
-----------------------------------------------------------------------------
-*/
sortie(status);
}

/*-----------------------------------------------------------------------------
--
   Sortie de l'application, liberation des ressources
-----------------------------------------------------------------------------
-*/
static void sortie(int status)
{
Quitter_lib_exec_aster(VRAI);
exit(status);
}

/*-----------------------------------------------------------------------------
--
   Affichage des parametres reconnus par l'application
-----------------------------------------------------------------------------
-*/
static void usage(char *nc)
{
fprintf(stderr,"\nusage : %s -pret fichier_profil_d_etude [parametres
lib_exec_aster]\n",nc);
}
```

8.2 Exemple pour `lib_asterix`

Exemple de programme principal d'une application dont la partie interface est réalisée avec XFM. Utilisation de deux bsf (une modale et une non modale), de la fenêtre de visualisation d'un texte et du suivi des jobs.

```
/*############################################################################
#
  Exemple d'utilisation de lib_asterix
############################################################################
#*/

#include <stdio.h>
#include <Fm.h>

#include <lib_asterix.h>

Widget FmCreateexemple();

Widget wBSFmodale;   /* Widget de la BSF modale     */
Widget wBSFnm;       /* Widget de la BSF non modale */
```

```
Widget GetwBSF()
/*********************************************************/
/* Pour communiquer le Widget de la BSF modale créé dans     */
/* le main de l'interface de l'exemple                       */
/*********************************************************/
{
return(wBSFmodale);
}

Widget GetwBSFnm()
/*********************************************************/
/* Pour communiquer le Widget de la BSF non modale créé dans */
/*  le main de l'interface de l'exemple                      */
/*********************************************************/
{
return(wBSFnm);
}

void sortie(int status)
{
/* Envoyer un message a asjob pour terminer l'application */
AsQuitSuivJob();
Quitter_lib_asterix(VRAI);
exit(status);
}

main (argc,argv)
int argc;
char **argv;
{
Widget toplevel;
Widget appShell;

printf("Version de la bibliothèque lib_asterix
:%s\n",Version_lib_asterix());

/* Initilisation sans creation de widget */
toplevel = FmInitialize("zz","zz",NULL,0,&argc,argv);

/*-------------------------------------------------------------------------
--
   Initialisation lib_asterix
-------------------------------------------------------------------------
-*/
Init_lib_asterix(argc,argv,"/tmp/Exemple_lib_asterix",toplevel);

/*-------------------------------------------------------------------------
--
   Traitement des arguments specifiques a l'application
-------------------------------------------------------------------------
-*/
while (*argv) {
   if ((strcmp(*argv,"-help") == 0) || (strcmp(*argv,"-h") == 0) ||
(strcmp(*argv,"-aide") == 0) ) {
      sortie(0);
      }
   argv++;
```

```
}
```

```
/* Creation de l'application shell de l'IA */
appShell=FmCreateexemple("ex_lib_asterix",toplevel,NULL,0);

/* Creation du suivi de job */
AsCreateSuivJob("flash_Aster",FAUX);

/* Creation des BSF */
wBSFmodale=CreateBSF(appShell);
wBSFnm=CreateBSFA(appShell);

/* Creation d'une fenetre de visualisation de texte */
CreateVisuInfo(appShell);

/* Main Event Loop */
FmLoop();

}
```

9 Différences d'utilisation avec les versions antérieures

9.1 Avec la version 1.1

Pour pouvoir changer la librairie lib_asterix sans avoir besoin de recompiler l'application (ce qui est un des buts recherchés) deux points nécessitant des modifications par rapport à l'utilisation de la version 1.1 de lib_asterix ont été changés :

- la variable globale VERSION_LIB_ASTERIX n'existe plus, elle est remplacée par la fonction Version_lib_asterix(),
- il n'est plus nécessaire de précéder l'inclusion du fichier lib_asterix.h par #define INIT_DEF dans le fichier contenant le programme principal de l'application.

9.2 Avec la version 2.2

Extension de la notion d'édition à la notion d'exécution d'une commande Unix avec comme paramètre le fichier sélectionné. Ces commandes Unix peuvent être redéfinies par l'utilisateur et sont conservées dans le fichier $HOME/.prefbsf . De ce fait les variables globales CMD.... ne sont plus utilisées pour l'appel aux éditeurs.

9.3 Avec la version 3.0

Mise en conformité avec le système UNICOS 8.0 sur Cray.

9.4 Avec la version 3.1

La version 3.2 de lib_asterix est incompatible avec la précédente. L'installation de la nouvelle version de la bibliothèque dynamique ne suffit pas, il faut recompiler l'application et la modifier en faisant quelques choix.

Les interfaces des fonctions fournies par lib_asterix sont au format ANSI. Il faut compiler avec l'option -D_NO_PROTO_LIB_ASTERIX pour ne pas avoir la déclaration des paramètres.

Modification de l'appel à la construction et la soumission des jobs (ajout de paramètres). Pour éviter les erreurs, les fonctions dont le nombre d'arguments a changé ont changé de nom.

Liste des changements :

- ConsJob() -> AsConsJob()
- SubaFicUnic() -> AsSubaFicunic()
- AddJob() -> AsAddJob()
- CreateSuivJob() -> AsCreateSuivJob()
- ShowSuivJob() -> AsShowSuivJob()

Possibilité de suivi interactif des jobs.

Le suivi des jobs est une application autonome pouvant communiquer avec plusieurs applications. Il est conseillé de changer le répertoire FLASH_STATION pour utiliser celui d'asterix : $HOME/flash_Aster.

Possibilité de conserver les répertoires de la bsf dans un fichier de préférences. Ces répertoires, comme les commandes Unix de la bsf, peuvent être pris en compte par l'application ou par lib_asterix.

Plus de variable globale interrompre dans le mécanisme sysint().

Prise en compte des commandes sécurisées rcp/rsh à travers ASURE.

Les choix à faire sont :

- le choix d'une signature de l'application pour l'émission des jobs,
- le ralliement ou non à un répertoire flasheur sur station communs aux applications utilisant lib_asterix,
- la gestion des commandes Unix et des répertoires rémanents de la bsf est faite par l'application ou par lib_asterix (dans le fichier $HOME/.prefbsf),
- l'adaptation ou non des commandes rcp/rsh à ASURE.

9.5 Avec la version 3.2

Les interfaces de deux fonctions sont modifiées. Des paramètres IN sont transformés en IN-OUT.

Cela ne devrait pas poser de problème à l'utilisation car il y a le même nombre de paramètres avec le même type et ce sont des String.

Fonctions modifiées :

- Local()
- FichierLocalRef().

9.6 Depuis la version 4.0

Avant cette version il n'y avait qu'une seule bibliothèque (lib_asterix) pour assurer la soumission officielle des exécutions sur une machine centralisée Aster et distribuer les utilitaires graphiques développés pour asterix. Il fallait donc obligatoirement l'utiliser dans un contexte X-Motif. À partir de la version 4.0, lib_asterix a été découpée pour pouvoir soumettre des exécutions sans être obligatoirement dans un contexte graphique avec la création de la bibliothèque lib_exec_aster.

Ce découpage nécessite une édition de liens pour toutes les applications utilisant les versions précédentes de `lib_asterix`.

A partir de cette version il est dorénavant possible :

- De soumettre des exécutions Aster sans obligatoirement avoir un environnement X-Motif (on n'a plus besoin de l'application de suivi des jobs -asjob-).
- De changer les répertoires de destination des fichiers sur la machine locale et celle d'exécution ainsi que le répertoire de travail à partir des arguments sur la ligne de commande (ces paramètres sont donc modifiable par l'utilisateur ou l'administrateur système et ne sont plus réservés au développeur de l'application).

Autres ajouts :

- usage avec `-help`, `-h` ou `-aide` sur la ligne de commande
- possibilité d'arrêter asjob depuis son application,
- soumission d'une étude à partir d'un nom de fichier profil asterix (`AsExecFicProfil()`).

À ma connaissance cela concerne les applications suivantes :

- anthemix (si cela existe encore),
- mekelec,
- circus,
- cab (couplage BOSS Quatro-Aster).

10 Bibliographie

[1] A.M. DONORE, C.MASSERET - Accès au *Code_Aster* - Conditions générales d'accès au *Code_Aster* [U2.00.00].

[2] Interface graphique d'accès au *Code_Aster*. Configurations matérielle et logicielle requises. Document [U2.02.01].

[3] Installer l'interface graphique d'accès au *Code_Aster* sur sa station de travail. Document [U2.02.02].

[4] Y.DHERBECOURT - Extension d'ASURE aux services `rsh`/`rcp` : Spécification fonctionnelle HI-57/95/014.

[5] Installation et utilisation de rcp/rsh avec lib_asterix [U2.02.03] .