TECHNICAL REPORT
IGE-296

LCM – GUIDE DU PROGRAMMEUR

A. Hébert

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
October 10, 2010

RÉSUMÉ

Les objets LCM sont des structures de données, implémentées en C, et possédant les caractéristiques de tables associatives (tables de hachage ou dictionnaires) ou de listes hétérogènes. Leur stockage est soit en mémoire, soit persistant. Ces objets possèdent une API permettant de les utiliser depuis les langages Python, Java, C et Fortran. La persistance de ces objets est gérée par l'API XSM *qui utilise des fichiers à accès direct. Des utilitaires de copie et de sérialisation sont également fournis.*

Les objets LCM ont été conçus de façon à permettre

- *un accès optimisé à l'information lorsqu'ils sont utilisés depuis des logiciels écrits en Fortran ou en C*
- *un accès complet à l'information (lecture et écriture) depuis Python et Java.*

Ce document fournit au programmeur un descriptif détaillé de chacune des interfaces Python, Java, Fortran et C composant le système LCM.

Contents

Contents . . . ii
1 Introduction . . . 1
1.1 Les tables associatives . . . 2
1.2 Les listes . . . 2
1.3 Les blocs d'information élémentaires . . . 2
2 API Python de l'objet LCM . . . 3
2.1 Les variables d'attribut . . . 3
2.2 lcm.new() . . . 3
2.3 lcm.xsm() . . . 4
2.4 lcm.file() . . . 4
2.5 lcm.impor() . . . 5
2.6 o.keys() . . . 5
2.7 o.lib() . . . 5
2.8 o.rep() . . . 5
2.9 o.lis() . . . 6
2.10 o.copy() . . . 6
2.11 o.expor() . . . 6
2.12 o.Close() . . . 6
3 API Java de l'objet LCM . . . 8
3.1 Création, ouverture, fermeture et validation d'objets Jlcm . . . 8
3.1.1 Jlcm() . . . 8
3.1.2 o.close() . . . 8
3.1.3 o.open() . . . 8
3.1.4 o.val() . . . 9
3.2 Interrogation des objets Jlcm . . . 9
3.2.1 o.getName() . . . 9
3.2.2 o.getType() . . . 9
3.2.3 o.getLength() . . . 10
3.2.4 o.getDirectory() . . . 10
3.2.5 o.getAccess() . . . 10
3.2.6 o.isEmpty() . . . 10
3.2.7 o.isNew() . . . 11
3.2.8 o.length() . . . 11
3.2.9 o.type() . . . 11
3.2.10 o.keys() . . . 12
3.3 Gestion des blocs d'information élémentaires, des tables associatives et des listes . . . 12
3.3.1 o.put() . . . 12
3.3.2 o.get() . . . 12
3.3.3 o.rep() . . . 13
3.3.4 o.lis() . . . 13
3.4 Utilitaires . . . 13
3.4.1 o.lib() . . . 13
3.4.2 o.dump() . . . 13
3.4.3 o.copy() . . . 14
3.4.4 o.expor() . . . 14
3.4.5 procédure de sérialisation . . . 14
4 API Fortran de l'objet LCM . . . 16
4.1 Ouverture, fermeture et validation d'objets LCM . . . 16
4.1.1 LCMOP . . . 16

4.1.2 LCMCL . . . 16
4.1.3 LCMVAL . . . 17
4.2 Interrogation des objets LCM . . . 17
4.2.1 LCMLEN . . . 17
4.2.2 LCMINF . . . 18
4.2.3 LCMNXT . . . 18
4.2.4 LCMLEL . . . 19
4.3 Gestion des blocs d'information élémentaires . . . 19
4.3.1 LCMGET . . . 19
4.3.2 LCMPUT . . . 20
4.3.3 LCMIOF . . . 20
4.3.4 LCMPOF . . . 21
4.3.5 LCMDEL . . . 22
4.3.6 LCMGDL . . . 22
4.3.7 LCMPDL . . . 23
4.3.8 LCMGSL . . . 23
4.3.9 LCMPSL . . . 24
4.4 Gestion des tables associatives et des listes . . . 25
4.4.1 LCMDID . . . 25
4.4.2 LCMLID . . . 25
4.4.3 LCMLIL . . . 26
4.4.4 LCMDIL . . . 27
4.4.5 LCMGID . . . 28
4.4.6 LCMGIL . . . 28
4.4.7 LCMSIX . . . 29
4.5 Utilitaires . . . 29
4.5.1 LCMLIB . . . 29
4.5.2 LCMEQU . . . 30
4.5.3 LCMEXP . . . 30
4.6 Traitement des vecteurs de variables caractères . . . 30
4.6.1 LCMGCD . . . 31
4.6.2 LCMPCD . . . 31
4.6.3 LCMGCL . . . 32
4.6.4 LCMPCL . . . 32
4.7 Gestion des exceptions et arrêt . . . 33
4.7.1 XABORT . . . 33
4.8 Allocation dynamique de la mémoire en FORTRAN-77 . . . 33
4.8.1 SETARA . . . 33
4.8.2 RLSARA . . . 34
4.8.3 Exemple d'allocation mémoire en FORTRAN-77 . . . 34
5 API C de l'objet LCM . . . 37
5.1 Ouverture, fermeture et validation d'objets LCM . . . 37
5.1.1 LCMOP_C . . . 37
5.1.2 LCMCL_C . . . 37
5.1.3 LCMVAL_C . . . 38
5.2 Interrogation des objets LCM . . . 38
5.2.1 LCMLEN_C . . . 39
5.2.2 LCMINF_C . . . 39
5.2.3 LCMNXT_C . . . 40
5.2.4 LCMLEL_C . . . 40
5.3 Gestion des blocs d'information élémentaires . . . 41
5.3.1 LCMGET_C . . . 41

5.3.2 LCMPUT_C . . . 42
5.3.3 LCMGPD_C . . . 42
5.3.4 LCMPPD_C . . . 43
5.3.5 LCMDEL_C . . . 44
5.3.6 LCMGDL_C . . . 44
5.3.7 LCMPDL_C . . . 45
5.3.8 LCMGPL_C . . . 46
5.3.9 LCMPPL_C . . . 46
5.4 Gestion des tables associatives et des listes . . . 47
5.4.1 LCMDID_C . . . 47
5.4.2 LCMLID_C . . . 48
5.4.3 LCMLIL_C . . . 48
5.4.4 LCMDIL_C . . . 49
5.4.5 LCMGID_C . . . 50
5.4.6 LCMGIL_C . . . 50
5.4.7 LCMSIX_C . . . 51
5.5 Utilitaires . . . 51
5.5.1 LCMLIB_C . . . 51
5.5.2 LCMEQU_C . . . 52
5.5.3 LCMEXP_C . . . 52
5.6 Traitement des vecteurs de variables caractères . . . 53
5.6.1 LCMGCD_C . . . 53
5.6.2 LCMPCD_C . . . 53
5.6.3 LCMGCL_C . . . 54
5.6.4 LCMPCL_C . . . 55
5.7 Allocation dynamique des blocs d'information élémentaires . . . 56
5.7.1 SETARA_C . . . 56
5.7.2 RLSARA_C . . . 56
References . . . 57

1 Introduction

Les objet LCM sont des structures récursives contenant des blocs d'information, des tables associatives et des listes. L'API (*Application Programming Interface*) LCM est le résultat d'une généralisation des logiciels LCM et XSM introduits pour la première fois à la référence [1]. Chacune de ces composantes est accessible aussi bien depuis le langage de développement que du langage de commandes Python [2] ou Java grace à l'organisation schématisée à la Figure 1.

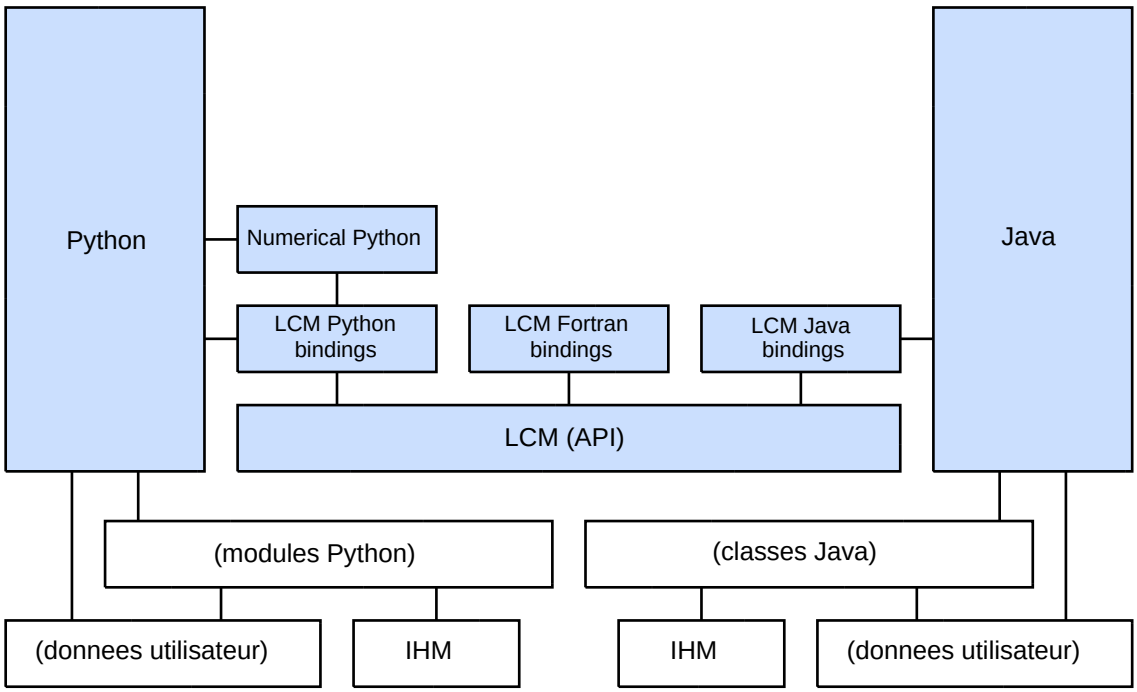

Figure 1: Utilisation de LCM dans une architecture logicielle d'intégration.

Les tables associatives sont semblables aux dictionnaires Python et se manipulent comme tel dans le jeu de données Python. Chaque élément d'une table associative est associé à une chaîne de caractères. Les listes sont semblables aux listes Python et sont donc un ensemble ordonné d'éléments. Chaque élément de liste est repéré par un indice et contient un bloc d'information. Quand aux blocs d'information, ils sont, soit des chaînes de caractères, soit des tableaux Numerical Python. [3] Un objet LCM peut physiquement être, soit en mémoire, soit sur fichier XSM.

Le système LCM est structuré en plusieurs parties:

- L'API LCM comporte les fonctions C décrites à la section Section 5. Ces fonctions ne font appel à *aucun* API externe et sont donc indépendantes de l'API Python ou Java. Elles ont été optimisées pour accélérer le traitement de l'information par les codes de calcul. L'API LCM gère de plus la persistance des objets via l'API XSM.

  L'API LCM prévoit le stockage d'une table associative ou d'une liste en mémoire (in core) ou sur un fichier à accès direct au format XSM. Dans les deux cas, les API sont les mêmes, ce qui signifie qu'un opérateur de calcul n'a pas besoin de savoir quel mode de stockage est utilisé. Cette fonctionnalité est essentielle pour stocker les très grosses tables associatives.

- Les *LCM Python bindings*, décrits à la section Section 2, permettent à Python d'utiliser l'API LCM de façon transparente. Les tables associatives et les listes sont représentées sous la forme de dictionnaires Python et de listes Python. Les blocs d'information en variables entières et flottantes sont automatiquement transformées en array *Numerical Python*. Notons finalement qu'une méthode a été ajoutée pour gérer l'utilisation des fichiers.
- Les *LCM Java bindings* sont les wrapper permettant d'appeler les fonctions de l'API LCM depuis un programme informatique écrit en Java. Les spécifications des appels Java sont décrites à la section Section 3.
- Les *LCM Fortran bindings* sont les wrapper permettant d'appeler les fonctions de l'API LCM depuis un code de calcul écrit en Fortran. Les spécifications des appels Fortran sont décrites à la section Section 4.

1.1 Les tables associatives

Une table associative est équivalente à un dictionnaire Python. Chaque élément d'une table est une association entre une chaîne de 12 caractères et un bloc d'information (valeur scalaire ou vecteur d'un type donné). Des tables associatives peuvent contenir des listes ou d'autres tables associatives et ainsi former une structure arborescente. Un élément contenant une table associative fille est aussi appelé *répertoire scalaire*.

1.2 Les listes

Une liste est un ensemble ordonné d'éléments de types hétérogènes. Chaque élément est accédé par un indice entier et contient un bloc d'information. Les listes peuvent contenir des valeurs scalaires ou des blocs d'information élémentaire, tel que décrits à la section suivante. Des listes peuvent également contenir des listes filles ou d'autres tables associatives.

1.3 Les blocs d'information élémentaires

Un bloc d'information élémentaire constitue un ensemble de valeurs dont l'opérateur a besoin pour réaliser le calcul. Contrairement aux tables ou aux listes qui ne permettent que de "ranger", le bloc d'information élémentaire est la donnée utile au calcul. Un bloc d'information est, soit des chaînes de caractères, soit des tableaux numériques (array) à une dimension (assimilables à des array Numerical Python). Les blocs d'information élémentaires appartiennent à l'un des types suivants:

Int array Un item de la table associative peut correspondre à un array d'entiers de 32 bits (type `l` de Numerical Python ou type `int[]` de Java).

Float32 array Un item de la table associative peut correspondre à un array de réels de 32 bits (type `f` de Numerical Python ou type `float[]` de Java).

Character array Un item de la table associative peut correspondre à un array de caractères (array de type `Char` de Python ou type `String` de Java).

Float64 array Un item de la table associative peut correspondre à un array de réels de 64 bits (type `d` de Numerical Python ou type `double[]` de Java).

Logical32 array Un item de la table associative peut correspondre à un array d'entiers de 32 bits (type `i` de Numerical Python contenant 1/0 pour désigner vrai/faux ou type `boolean[]` de Java).

Complex32 array Un item de la table associative peut correspondre à un array de variables complexes de 64 bits (type `F` de Numerical Python ou type `Complex[]` de Visual Numerics en Java).

2 API Python de l'objet LCM

Le module `lcm`, accessible depuis Python, est importé par la commande

```
import lcm
```

Il possède quatre constructeurs: `lcm.new()` pour les objets en mémoire, `lcm.xsm()` pour les objets persistants, `lcm.file()` pour les objets de type fichier et `lcm.impor()` pour créer un objet à partir de l'information sérialisée contenue dans un fichier. L'objet `o` ainsi créé possède plusieurs méthodes: `o.keys()`, `o.lib()`, `o.rep()`, `o.lis()`, `o.copy()` et `o.Close()`.

2.1 Les variables d'attribut

Un objet PyLCM `o` contient cinq variables d'attribut pouvant être accédées directement en lecture:

`o._name` Nom Python de l'objet PyLCM contenant la racine

`o._directory` Nom du répertoire actif. =`'/'` pour le répertoire racine. Cette variable d'attribut n'est pas définie pour les listes et pour les fichiers crées par `lcm.file()`.

`o._long` $= -1$: table associative; ≥ 1: liste de longueur `o._long`.

`o._access` Mode d'accès de l'objet. = 0: objet fermé (i.e., non accessible); = 1: objet en mode de modification; = 2: objet en mode de lecture (read-only). *Cet attribut est le seul qui puisse être modifié directement.*

`o._type` Type de l'objet. = 1: objet LCM en mémoire (semblable à un dictionnaire Python); = 2: objet LCM persistant (de type fichier XSM); = 3: fichier séquentiel binaire; = 4: fichier séquentiel ASCII; = 5: fichier à accès direct.

2.2 lcm.new()

Cette méthode permet de créer un objet PyLCM constitué d'une table associative de type LCM (semblable à un dictionnaire Python). Cet objet est contenu en mémoire et permet un accès rapide grâce à un algorithme de hachage. Au terme de cet appel, la variable d'attribut `o._access` est égale à 1.

```
o=lcm.new([name])
```

paramètre d'entrée:		
`name`	*string*	nom de l'objet PyLCM qui sera créé. Ce nom est limité à 12 caractères. Par défaut, un nom est généré automatiquement à partir de l'adresse de l'objet PyLCM.

paramètre de sortie:		
`o`	*LCM*	objet PyLCM créé.

2.3 lcm.xsm()

Cette méthode permet de créer un objet PyLCM persistant constitué d'un fichier XSM (fichier à accès direct contenant une table associative archivée). Cet objet occupe très peu d'espace mémoire et peut être utilisé pour stocker de très gros objets dont la dimension maximum n'est limitée que par l'espace disque disponible. En général, on peut toujours remplacer un objet PyLCM "mémoire" par un objet PyLCM persistant (au prix d'une certaine augmentation du temps CPU). Un objet PyLCM persistant peut également être utilisé comme médium d'archivage d'un objet PyLCM "mémoire". Au terme de cet appel, la variable d'attribut `o._access` est égale à 1 ou 2.

```
o=lcm.xsm([name],[iact])
```

paramètres d'entrée:		
`name`	*string*	nom du fichier XSM qui sera ouvert. Ce nom est limité à 12 caractères. Par défaut, un nom est généré automatiquement à partir de l'adresse de l'objet PyLCM.
`iact`	*int*	=0 pour créer et ouvrir un nouveau fichier XSM (défaut); =1 pour ouvrir un fichier XSM existant en mode de modification; =2 pour ouvrir un fichier XSM existant en mode read-only.

paramètre de sortie:		
`o`	*LCM*	objet PyLCM persistant.

2.4 lcm.file()

Cette méthode permet de créer un objet PyLCM contenant un fichier. Cette méthode est utile pour récupérer un fichier fabriqué par un opérateur Fortran ou pour transmettre un fichier à un opérateur Fortran. Au terme de cet appel, la variable d'attribut `o._access` est égale à 1 ou 2. L'objet PyLCM ainsi créé n'a pas les attributs `_directory` et `_long`. Il ne connaît pas les méthodes `keys()`, `lib()`, `rep()`, `lis()` et `copy()`.

```
o=lcm.file([name],[iact],[itype],[lrda])
```

paramètres d'entrée:		
`name`	*string*	nom du fichier qui sera ouvert. Ce nom est limité à 12 caractères. Par défaut, un nom est généré automatiquement à partir de l'adresse de l'objet PyLCM.
`iact`	*int*	=0 pour créer et ouvrir un nouveau fichier (défaut); =1 pour ouvrir un fichier existant en mode de modification; =2 pour ouvrir un fichier existant en mode. read-only.
`itype`	*int*	type du fichier. Seules les valeurs suivantes sont permises: =1 séquentiel binaire; =2 séquentiel ASCII (par défaut); =3 fichier binaire à accès direct.
`lrda`	*int*	nombre de mots dans un enregistrement à accès direct (utilisé uniquement si `itype`= 3). Par défaut, `lrda`= 128.

paramètre de sortie:		
`o`	*LCM*	objet PyLCM contenant un fichier.

2.5 lcm.impor()

Cette méthode permet de créer un objet PyLCM mémoire ou persistant à partir de l'information sérialisée contenue dans un fichier séquentiel. Au terme de cet appel, la variable d'attribut `o._access` est égale à 1.

```
o=lcm.impor(name,[medium])
```

paramètres d'entrée:		
`name`	*string*	nom du fichier séquentiel contenant l'information sérialisée. Le nom de ce fichier doit débuter par le caractère `"_"`.
`medium`	*int*	= 0 pour créer un objet en mémoire (défaut); = 1 pour créer un fichier XSM.

paramètre de sortie:		
`o`	*LCM*	objet PyLCM mémoire ou persistant.

2.6 o.keys()

Cette méthode permet de créer une liste Python contenant le nom des clés de la table associative (mémoire ou fichier XSM). Cette méthode n'est pas disponible pour les objets FILE.

```
o2=o.keys()
```

paramètre de sortie:		
`o2`	*list*	liste Python contenant les clés de la table associative.

2.7 o.lib()

Cette méthode permet d'imprimer le nom des clés de la table associative (mémoire ou fichier XSM) ou de l'information sur une liste (longueurs, types, signature). Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

2.8 o.rep()

Cette méthode permet de créer une table associative fille dans la table associative (dictionnaire) ou la liste `o`. Cette méthode n'est pas disponible pour les objets FILE.

```
[o2=]o.rep({key|iset})
```

paramètres d'entrée:		
`key`	*string*	si `o` est une table; chaîne obligatoire correspondant à la clé de la table associative fille
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve la table associative fille.

paramètre de sortie:		
`o2`	*LCM*	table associative fille.

2.9 o.lis()

Cette méthode permet de créer une liste fille imbriquée dans la table associative (dictionnaire) ou la liste `o`. Le premier élément de la liste fille est situé à l'index [0]. Cette méthode n'est pas disponible pour les objets FILE.

```
[o2=]o.lis({key|iset},ilong)
```

paramètres d'entrée:		
`key`	*string*	si `o` est une table; chaîne obligatoire correspondant à la clé de la liste fille.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve la liste fille.
`ilong`	*int*	entier positif (obligatoire) qui donne la longueur de la liste fille.

paramètre de sortie:		
`o2`	*lcmlist*	liste fille.

2.10 o.copy()

Cette méthode permet de faire une copie profonde (deep copy) d'une table associative (mémoire ou fichier XSM) dans une autre. Cette méthode n'est pas disponible pour les objets FILE.

```
o2=o.copy([name],[medium])
```

paramètres d'entrée:		
`name`	*string*	chaîne facultative correspondant au nom de la table associative créée. Par défaut, un nom est généré automatiquement.
`medium`	*int*	=1 pour créer un objet en mémoire (défaut); =2 pour créer un fichier XSM.

paramètre de sortie:		
`o2`	*LCM*	nouvelle table associative issue de la copie.

2.11 o.expor()

Cette méthode permet de sérialiser un objet PyLCM et de créer un fichier séquentiel contenant cette information. Cette méthode n'est pas disponible pour les objets FILE.

```
o.expor([name])
```

paramètre d'entrée:		
`name`	*string*	chaîne facultative correspondant au nom du fichier séquentiel qui sera créé. Le nom de ce fichier doit débuter par le caractère `"_"`. Par défaut, le nom est la concaténation du caractère `"_"` avec le nom de l'objet PyLCM (`o._name`).

2.12 o.Close()

Cette méthode permet de fermer un objet `o` de type PyLCM. Après fermeture avec `iact`= 1, l'objet PyLCM existe toujours mais ne peut plus être utilisé (il est inaccessible) et sa variable d'attribut `_access` est mise à 0.

Lorsque le compteur de référence d'un objet ouvert `o` tombe à zéro (après `del o` par exemple), la méthode `Close()` est automatiquement appelée

- avec `iact=2` si l'objet est ouvert en mode de modification ou si l'objet est en mémoire. L'objet est alors détruit.
- avec `iact=1` sinon.

Si l'objet est en mémoire (de type dictionnaire) et est déjà fermé, il est détruit lorsque son compteur de référence tombe à zéro. Lorsqu'ils sont utilisés depuis Python, les objets PyLCM possèdent une structure de tête C qui leur permettent de se comporter comme des objets Python à part entière (par exemple, cette structure comporte le compteur de référence). Cette structure de tête est détruite lorsque le compteur de référence de l'objet PyLCM tombe à zéro.

Les objets PyLCM de type "table associative" ne peuvent être fermés que si leur variable d'attribut `_directory` vaut `'/'`, c'est à dire qu'ils pointent vers un répertoire racine.

`o.Close([iact])`

paramètre d'entrée:		
`iact`	*int*	type de fermeture: =1: fermeture sans destruction de l'objet PyLCM (i.e., l'objet existe toujours mais n'est plus accessible); =2: fermeture avec destruction de l'objet PyLCM (défaut).

3 API Java de l'objet LCM

Les fonctionnalités LCM sont disponibles depuis un programme écrit en Java en utilisant les bindings Java écrits à partir des API *Java Native Interface* (JNI). Tous les appels LCM sont disponibles a partir de méthodes appartenant à une classe `Jlcm`.

3.1 Création, ouverture, fermeture et validation d'objets Jlcm

3.1.1 Jlcm()

Ce constructeur permet de créer un nouvel objet Jlcm, encapsulant un objet LCM ou un fichier. Cet objet Jlcm est ouvert en mode de modification. Si un fichier XSM de nom `name` existe, le nouvel objet Jlcm pointera vers ce fichier ouvert en mode `read-only`.

Ce constructeur peut être utilisé pour initialiser l'objet Jlcm créé à partir du contenu d'un fichier séquentiel ASCII contenant un objet Jlcm sérialisé. Dans ce cas, le nom du fichier doit obligatoirement débuter par le caractère "_" suivi du nom de l'objet Jlcm qui sera créé. Cet objet Jlcm est ouvert en mode de modification.

```
o = new Jlcm(type,name,[lrda]);
```

paramètres d'entrées:		
`type`	*String*	type de l'objet Jlcm qui sera créé. =`LCM` objet LCM en mémoire; =`XSM` objet persistant de type XSM; =`BINARY` fichier séquentiel binaire; =`ASCII`; fichier séquentiel ASCII; =`DA`; fichier à accès direct; =`LCM_IMP` objet LCM en mémoire construit à partir du fichier "_"+`name` contenant un objet Jlcm sérialisé; =`XSM_INP` objet persistant de type XSM construit à partir du fichier "_"+`name` contenant un objet Jlcm sérialisé.
`name`	*String*	nom de l'objet Jlcm qui sera créé. Ce nom est limité à 12 caractères.
`lrda`	*int*	nombre de mots dans un enregistrement à accès direct (utilisé uniquement si `type`=`DA`). Par défaut, `lrda`= 128.

paramètre de sortie:		
`o`	*Jlcm*	objet Jlcm créé.

3.1.2 o.close()

Méthode permettant de fermer un objet Jlcm.

```
o.close(type);
```

paramètre d'entrée:		
`type`	*String*	type de fermeture. =`KEEP` fermeture sans destruction de l'information interne; =`DESTROY` fermeture avec destruction de l'information interne.

3.1.3 o.open()

Méthode permettant de ré-ouvrir un objet Jlcm fermé par `o.close()` (avec l'option `KEEP`).

```
o.open(type);
```

paramètre d'entrée:		
`type`	*String*	type d'ouverture. =`READ/WRITE` ouverture en mode de modification; =`READ-ONLY` ouverture en mode de lecture seule.

3.1.4 o.val()

Méthode permettant de vérifier la cohérence interne et l'intégrité d'un objet Jlcm de type LCM. Si l'objet est cohérent, aucune action n'est effectuée; sinon, une exception est levée. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
o.val([key]);
```

paramètre d'entrée:		
`key`	*String*	nom de la table associative fille ou de la liste fille qui sera validée. Par défaut, tout l'objet référencé par `o` est validé.

3.2 Interrogation des objets Jlcm

Les objets Jlcm sont auto-descriptifs. Il est donc possible de les interroger pour connaître leur caractéristiques.

3.2.1 o.getName()

Méthode retournant le nom d'un objet Jlcm.

```
name = o.getName();
```

paramètre de sortie:		
`name`	*String*	nom de l'objet `o`

3.2.2 o.getType()

Méthode retournant le type d'un objet Jlcm.

```
itype = o.getType();
```

paramètre de sortie:		
`itype`	*int*	type de l'objet `o`. = 1: objet LCM en mémoire; = 2: objet LCM persistant (de type fichier XSM); = 3: fichier séquentiel binaire; = 4: fichier séquentiel ASCII; = 5: fichier à accès direct.

3.2.3 o.getLength()

Méthode retournant la longueur d'un objet Jlcm de type liste. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
ilong = o.getLength();
```

paramètre de sortie:		
`ilong`	*int*	longueur de la liste `o`. = −1: table associative; ≥ 1: liste de longueur `ilong`.

3.2.4 o.getDirectory()

Méthode retournant le nom du répertoire directement accessible d'un objet Jlcm. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
hdir = o.getDirectory();
```

paramètre de sortie:		
`hdir`	*String*	nom du répertoire directement accessible de l'objet `o`. =`"/"` pour le répertoire racine.

3.2.5 o.getAccess()

Méthode retournant le code d'accès de l'objet `o`.

```
iact = o.getAccess();
```

paramètre de sortie:		
`iact`	*int*	code d'accès de l'objet `o`. = 0: objet fermé; = 1: objet ouvert en mode de modification; = 2: objet ouvert en mode read-only.

3.2.6 o.isEmpty()

Méthode retournant une variable `boolean` egale à `true` si le répertoire directement accessible est vide. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
empty = o.isEmpty();
```

paramètre de sortie:		
`empty`	*boolean*	variable egale à `true` si le répertoire directement accessible est vide.

3.2.7 o.isNew()

Méthode retournant une variable `boolean` egale à `true` si l'objet `o` vient d'être créé.

```
new = o.isNew();
```

paramètre de sortie:		
`new`	*boolean*	variable egale à `true` si l'objet `o` vient d'être créé.

3.2.8 o.length()

Méthode retournant la longueur d'un bloc d'information élémentaire (array).

```
ilong = o.length({key|iset});
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé de l'objet Jlcm fille.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve l'objet Jlcm fille.

paramètre de sortie:		
`ilong`	*int*	longueur du bloc d'information élémentaire (array). $=-1$ pour une table associative fille; $=N$ pour une liste fille contenant N composantes; =0 si le bloc n'existe pas.

3.2.9 o.type()

Méthode retournant le type d'un bloc d'information.

```
itype = o.type({key|iset});
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé de l'objet Jlcm fille.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve l'objet Jlcm fille.

paramètre de sortie:		
`itype`	*int*	type de l'information. =0 table associative; =1 entier; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =10 liste; =99 indéfini (99 est retourné si le bloc n'existe pas).

3.2.10 o.keys()

Méthode retournant un objet de type *Enumeration* permettant d'itérer un objet Jlcm contenant une table associative.

```
enum = o.keys();
```

paramètre de sortie:		
`enum`	*Enumeration*	objet *Enumeration* correspondant à l'objet `o`.

On peut itérer un objet Jlcm contenant une table associative en procédant comme dans l'exemple suivant:

```
Enumeration ee = my_lcm.keys();
while(ee.hasMoreElements()) {
  String key = (String)ee.nextElement();
  System.out.println("element=" + key);
}
```

3.3 Gestion des blocs d'information élémentaires, des tables associatives et des listes

3.3.1 o.put()

Méthode permettant de stocker un bloc d'information élémentaire dans un objet Jlcm.

```
o.put({key|iset},data);
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé du bloc d'information élémentaire.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve le bloc d'information élémentaire.
`data`	*Object*	bloc d'information élémentaire (array).

3.3.2 o.get()

Méthode permettant de récupérer un objet Jlcm fille (table associative ou liste), ou un bloc d'information élémentaire.

```
data = o.get({key|iset});
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé de l'objet Jlcm fille ou du bloc d'information élémentaire.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve l'objet Jlcm fille ou le bloc d'information élémentaire.

paramètre de sortie:		
`data`	*Object*	objet Jlcm fille (table associative ou liste) *ou* bloc d'information élémentaire (array).

3.3.3 o.rep()

Méthode permettant de créer un objet Jlcm fille encapsulant une table associative.

```
o2 = o.rep({key|iset});
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé de l'objet Jlcm fille.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve l'objet Jlcm fille.

paramètre de sortie:		
`o2`	*Jlcm*	objet Jlcm fille contenant une table associative.

3.3.4 o.lis()

Méthode permettant de créer un objet Jlcm fille encapsulant une liste.

```
o2 = o.lis({key|iset},ilong);
```

paramètres d'entrée:		
`key`	*String*	si `o` est une table; chaîne obligatoire correspondant à la clé de l'objet Jlcm fille.
`iset`	*int*	si `o` est une liste; index dans la liste `o` où on trouve l'objet Jlcm fille.
`ilong`	*int*	entier positif qui donne la longueur de la liste fille.

paramètre de sortie:		
`o2`	*Jlcm*	objet Jlcm fille contenant une liste.

3.4 Utilitaires

3.4.1 o.lib()

Cette méthode permet d'imprimer le nom des clés de la table associative (mémoire ou fichier XSM) ou de l'information sur une liste (longueurs, types, signature). Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

3.4.2 o.dump()

Cette méthode permet d'imprimer le contenu ASCII de la table associative (mémoire ou fichier XSM) ou de la liste. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

3.4.3 o.copy()

Cette méthode permet de faire une copie profonde (deep copy) d'une table associative (mémoire ou fichier XSM) dans une autre. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
o2 = o.copy(key,[medium]);
```

paramètres d'entrée:		
`key`	*String*	chaîne correspondant au nom de la table associative créée.
`medium`	*int*	=1 pour créer un objet en mémoire (défaut); =2 pour créer un fichier XSM.

paramètre de sortie:		
`o2`	*Jlcm*	nouvelle table associative issue de la copie. La table `o2` est créé par la méthode `copy`.

3.4.4 o.expor()

Cette méthode permet de sérialiser un objet Jlcm et de créer un fichier séquentiel contenant cette information. Cette méthode n'est pas disponible pour les objets Jlcm de type fichier.

```
o.expor([name]);
```

paramètre d'entrée:		
`name`	*String*	chaîne facultative correspondant au nom du fichier séquentiel qui sera créé. Le nom de ce fichier doit débuter par le caractère `"_"`. Par défaut, le nom est la concaténation du caractère `"_"` avec le nom de l'objet Jlcm (`o.getName()`).

3.4.5 procédure de sérialisation

Cette procédure permet d'utiliser la méthode `expor()` pour sérialiser un objet Jlcm et transmettre son contenu sur un objet `stream` de Java. Cette opération est illustrée dans l'exemple suivant:

```
try{
  FileOutputStream fo = new FileOutputStream("serialized_object");
  ObjectOutputStream so = new ObjectOutputStream(fo);
  so.writeObject(my_lcm);
  so.flush();
```

```
} catch (Exception ex) {
  System.out.println(ex);
  System.exit(1);
}
```

où `my_lcm` est une variable de type `Jlcm` et où `so` est la variable de type `stream`. Le stream Java est copié dans le fichier nommé `serialized_object`.

De même, l'opération inverse de dé-sérialisation est illustrée dans l'exemple suivant:

```
try{
  FileInputStream fi = new FileInputStream("serialized_object");
  ObjectInputStream si = new ObjectInputStream(fi);
  my_lcm = (Jlcm)si.readObject();
} catch (Exception ex) {
  System.out.println(ex);
  System.exit(1);
}
```

4 API Fortran de l'objet LCM

4.1 Ouverture, fermeture et validation d'objets LCM

4.1.1 LCMOP

Appel utilisé pour ouvrir un objet LCM (mémoire ou fichier XSM) et fournir son adresse si l'objet est créé. Dans le cadre d'un code, c'est l'interface avec Python qui est responsable de faire appel à `LCMOP`. L'utilisation de `LCMOP` est généralement restreinte à l'utilisation de tables associatives temporaires à l'intérieur d'un opérateur.

```
CALL LCMOP(IPLIST,NAMP,IMP,MEDIUM,IMPX)
```

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de l'objet LCM si `IMP=1` ou `IMP=2`. Cette adresse est celle de la table associative racine de l'objet.
`NAMP`	*CHARACTER*12*	nom de la table associative si `IMP=0`.
`IMP`	*INTEGER*	=0 pour créer une nouvelle table associative ; =1 pour modifier un objet LCM existant; =2 pour accéder à un objet LCM en mode **read-only**.
`MEDIUM`	*INTEGER*	=1 utilise un objet LCM en mémoire; =2 utilise un fichier XSM pour stocker la table associative.
`IMPX`	*INTEGER*	paramètre d'impression. Egal à zéro pour supprimer toute trace d'impression.

paramètres de sortie:		
`IPLIST`	*INTEGER*	adresse d'un objet LCM si `IMP=0`.
`NAMP`	*CHARACTER*12*	nom de la table associative si `IMP=1` ou `IMP=2`.

4.1.2 LCMCL

Commande utilisée pour fermer un objet LCM (mémoire ou fichier XSM). Dans le cadre d'un code, c'est l'interface avec Python qui est responsable de faire appel à `LCMCL`. L'utilisation de `LCMCL` est généralement restreinte à la fermeture de tables associatives temporaires à l'intérieur d'un opérateur.

On ne doit jamais utiliser la commande `LCMCL` *sur un objet LCM crée par la fonction* `lcm.new()` *ou* `lcm.xsm()` *de Python.*

Un objet LCM ne peut être fermé que si `IPLIST` pointe vers son répertoire racine.

```
CALL LCMCL(IPLIST,IACT)
```

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de l'objet LCM (adresse du répertoire racine de l'objet).
`IACT`	*INTEGER*	=1 ferme la table associative sans la détruire; =2 la détruit.

paramètre de sortie:		
IPLIST	*INTEGER*	IPLIST=0 indique que la table associative est détruite ou que le fichier XSM est fermé. Un objet LCM en mémoire garde le même IPLIST durant son existence. Un fichier XSM est représenté par un IPLIST différent chaque fois qu'il est réouvert.

4.1.3 LCMVAL

Commande utilisée pour valider un simple bloc, une table associative ou une liste avec ses blocs (de façon récursive). Si IPLIST se réfère à un fichier XSM, aucune action n'est faite. La validation consiste à vérifier les connections entre les éléments de l'objet LCM, vérifier si chaque élément de la table associative ou de la liste est défini et détecter éventuellement les destructions mémoire. Si une erreur est détectée, le message suivant s'inscrit:

```
LCMVAL: BLOCK xxx OF THE TABLE yyy HAS BEEN OVERWRITTEN.
```

L'appel se fait comme suit:

```
CALL LCMVAL(IPLIST,NAMP)
```

paramètres d'entrées:		
IPLIST	*INTEGER*	adresse de la table associative ou de la liste.
NAMP	*CHARACTER*12*	nom du bloc à valider dans la table associative. Si NAMP=' ', tous les blocs de la table associative sont validés de façon récursive.

4.2 Interrogation des objets LCM

Les structures d'information LCM sont auto-descriptives. Il est donc possible de les interroger pour connaître leur caractéristiques.

		type d'interrogation	
		structure mère	bloc d'information
mère	table associative	LCMINF LCMNXT	LCMLEN
	liste	LCMINF	LCMLEL

4.2.1 LCMLEN

Commande utilisée pour récupérer la longueur et le type d'un bloc d'information stocké dans une table associative (mémoire ou fichier XSM). La valeur de la longueur est le nombre d'éléments dans une liste ou le nombre de mots nécessaires pour stocker l'information dans un bloc élémentaire. Par exemple, la longueur pour stocker un tableau de DOUBLE PRECISION est deux fois supérieure à sa dimension.

```
CALL LCMLEN(IPLIST,NAMP,ILONG,ITYLCM)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du bloc.

paramètres de sortie:		
ILONG	*INTEGER*	longueur du bloc en mots. $=-1$ pour une table associative fille; $=N$ pour une liste fille contenant N composantes; =0 si le bloc n'existe pas.
ITYLCM	*INTEGER*	type de l'information. =0 table associative; =1 entier; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =10 liste; =99 indéfini (99 est retourné si le bloc n'existe pas).

4.2.2 LCMINF

Commande utilisée pour récupérer des informations générales sur une table associative ou une liste (mémoire ou fichier XSM).

```
CALL LCMINF(IPLIST,NAMLCM,NAMMY,EMPTY,ILONG,LCM)
```

paramètre d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative ou de la liste.

paramètres de sortie:		
NAMLCM	*CHARACTER*12*	nom de l'objet Python contenant la table associative (nom de l'attribut `_name` de l'objet Python).
NAMMY	*CHARACTER*12*	nom de la table associative pointée par IPLIST (nom de l'attribut `_directory` de l'objet Python). =`'/'` si la table associative est la racine de l'objet; =`' '` si la table associative est un élément de liste.
EMPTY	*LOGICAL*	variable logique positionnée à `.true.` si la table associative ne contient aucun élément; positionnée à `.false.` sinon.
ILONG	*INTEGER*	$=-1$: IPLIST est une table associative; >0: nombre de composantes dans la liste IPLIST
LCM	*LOGICAL*	variable logique positionnée à `.true.` si l'information est stockée dans une table associative en mémoire ou positionnée à `.false.` si l'information est stockée dans un fichier XSM.

4.2.3 LCMNXT

Commande utilisée pour trouver le nom du bloc suivant dans une table associative (mémoire ou fichier XSM). L'utilisation de LCMNXT est interdite si la table associative ne contient aucun élément.

```
CALL LCMNXT(IPLIST,NAMP)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom d'un bloc existant. `NAMP=' '` peut être utilisé pour obtenir le nom du premier bloc.

paramètre de sortie:		
NAMP	*CHARACTER*12*	nom du bloc suivant. Un appel à XABORT est effectué si la table associative est vide.

4.2.4 LCMLEL

Commande utilisée pour récupérer la longueur et le type du bloc stocké dans une liste (mémoire ou fichier XSM). La valeur de la longueur est le nombre de mots nécessaires pour stocker l'information. Par exemple, la longueur pour stocker un tableau de DOUBLE PRECISION est deux fois supérieure à sa dimension.

```
CALL LCMLEL(IPLIST,ISET,ILONG,ITYLCM)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la liste.
ISET	*INTEGER*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 1.

paramètres de sortie:		
ILONG	*INTEGER*	longueur en mots du bloc. =0 si le bloc n'existe pas.
ITYLCM	*INTEGER*	type de l'information. =1 integer; =2 single precision real; =3 character*4 data; =4 double precision real; =5 logical; =6 complex number; =10 list; =99 indéfini (99 est retourné si le bloc n'existe pas).

4.3 Gestion des blocs d'information élémentaires

La gestion des blocs d'information élémentaires est réalisée avec copie de l'information (LCMPUT, LCMGET, LCMPDL ou LCMGDL) ou sans copie (LCMPOF, LCMIOF, LCMPSL ou LCMGSL). Dans ce dernier cas, on utilise un déplacement par rapport à une adresse de référence prise égale, par convention, à l'adresse du *blank COMMON*.

		type d'opération	
		put	get
mère	table associative	LCMPUT LCMPOF	LCMGET LCMIOF
	liste	LCMPDL LCMPSL	LCMGDL LCMGSL

4.3.1 LCMGET

Commande utilisée pour récupérer un bloc d'information dans une table associative (mémoire ou fichier XSM) et le copier en mémoire.

```
CALL LCMGET(IPLIST,NAMP,DATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du bloc à récupérer. Un appel à XABORT est déclenché si le bloc n'existe pas.

paramètre de sortie:		
DATA	*	vecteur de dimension $\geq$ ILONG dans lequel le bloc a été copié.

La procédure LCMGET peut être utilisée pour récupérer des chaînes de caractères écrites dans un bloc. Il est également possible d'utiliser le sous-programme LCMGCD présenté à la section Section 4.6.1. Dans l'exemple suivant, un bloc de longueur 5 nommé NAMP stocké dans une table associative pointée par IPLIST est récupéré en utilisant LCMGET et copié dans la variable de type character*20 nommée HNAME en utilisant un WRITE interne:

```
CHARACTER NAMP*12,HNAME*20
INTEGER IDATA(5)
IPLIST=...
NAMP=...
CALL LCMGET(IPLIST,NAMP,IDATA)
WRITE(HNAME,'(5A4)') (IDATA(I),I=1,5)
```

4.3.2 LCMPUT

Commande utilisée pour stocker un bloc d'information dans une table associative (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la table associative. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode read-only.

```
CALL LCMPUT(IPLIST,NAMP,ILONG,ITYLCM,DATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du bloc.
ILONG	*INTEGER*	longueur du bloc en mots. Si un mot contient N valeurs en double précision, ILONG doit avoir la valeur $2 \times N$.
ITYLCM	*INTEGER*	type de l'information. =1 integer; =2 single precision real; =3 character*4 data; =4 double precision real; =5 logical; =6 complex number; =99 indéfini.
DATA	*	vecteur de dimension $\geq$ ILONG duquel le bloc est copié. Les ILONG premiers éléments de DATA doivent être initialisés avant l'appel à LCMPUT.

La procédure LCMPUT peut être utilisée pour stocker un bloc de chaînes de caractères. Il est également possible d'utiliser le sous-programme LCMPCD présenté à la section Section 4.6.2. Dans l'exemple suivant, une variable de type character*20 nommée HNAME va être copiée dans un tableau d'entiers de longueur 5 nommé IDATA en utilisant un READ interne et stockée dans le bloc NAMP de la table associative pointée par IPLIST en utilisant LCMPUT:

```
CHARACTER NAMP*12,HNAME*20
INTEGER IDATA(5)
IPLIST=...
NAMP=...
READ(HNAME,'(5A4)') (IDATA(I),I=1,5)
CALL LCMPUT(IPLIST,NAMP,5,3,IDATA)
```

4.3.3 LCMIOF

Commande utilisée pour récupérer l'adresse `SETARA` (voir section Section 4.8.1) d'un bloc d'information stocké dans une table associative (mémoire ou fichier XSM) *sans faire de copie* de l'information. L'utilisation de cette commande doit se faire en respectant les règles suivantes:

- Si l'information est modifiée après l'appel à `LCMIOF`, un appel à `LCMPOF` doit ultérieurement être effectué pour prendre en compte les modifications.
- Le bloc `BASE(IOFSET)` ne doit jamais être libéré par une primitive de désallocation mémoire (`RLSARA`, `deallocate`, etc.).
- La variable `IOFSET` ne doit jamais être copiée dans une autre variable.

Le non respect de ces règles peut mener à des anomalies d'exécution (core dump, segmentation fault, etc) sans possibilité de lever une exception.

La commande `LCMIOF` représente une capacité avancée du logiciel `LCM` et elle doit être utilisée uniquement dans les situations où la réduction des ressources mémoire est primordiale. Une adresse est retournée dans le vecteur `BASE` stocké dans un common de la forme `COMMON BASE(1)`. Un appel à `LCMIOF` ne cause aucune modification de la table associative. L'information utile est par conséquent atteinte depuis `BASE(IOFSET)` à `BASE(IOFSET+ILONG-1)`.

```
CALL LCMIOF(IPLIST,NAMP,IOFSET)
```

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de la table associative.
`NAMP`	*CHARACTER*12*	nom du bloc à récupérer. Un appel à `XABORT` est réalisé si le bloc n'existe pas.

paramètre de sortie:		
`IOFSET`	*INTEGER*	adresse `SETARA` de l'information.

4.3.4 LCMPOF

Commande utilisée pour stocker un bloc d'information dans une table associative (mémoire ou fichier XSM) *sans faire de copie* de l'information. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée sur un objet LCM ouvert en mode `read-only`.

Si l'élément `NAMP` *existe déjà, son pointeur* `SETARA` *est remplacé par le nouveau et l'information pointée par l'ancien est désallouée.*

Cette fonction représente une capacité avancée du logiciel `LCM` et elle doit être utilisée uniquement dans les situations où la réduction des ressources mémoire est primordiale. Le bloc mémoire contenant l'information récupérée par `LCMPOF` devrait avoir été préalablement alloué par un appel à `SETARA` de la forme `CALL SETARA(BASE,ILONG,IOFSET)` où `BASE` est un vecteur stocké dans un common de la forme `COMMON BASE(1)`.

```
CALL LCMPOF(IPLIST,NAMP,ILONG,ITYLCM,IOFSET)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du bloc.
ILONG	*INTEGER*	longueur du bloc en mots.
ITYLCM	*INTEGER*	type de l'information. =1 integer; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =99 undefined.
IOFSET	*INTEGER*	adresse `SETARA` de l'information. Les éléments de `BASE(IOFSET)` à `BASE(IOFSET+ILONG-1)` doivent avoir été intialisés avant l'appel à `LCMPOF`.

paramètre de sortie:		
IOFSET	*INTEGER*	`IOFSET=0` indique que l'information précédemment pointée par `IOFSET` est maintenant gérée par `LCM`.

4.3.5 LCMDEL

Commande utilisée pour détruire un bloc d'information, une table associative fille ou une liste fille stocké par une table associative en mémoire. La fonctionnalité de `LCMDEL` n'est pas utilisable pour les fichiers XSM.

```
CALL LCMDEL(IPLIST,NAMP)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du bloc à détruire.

4.3.6 LCMGDL

Commande utilisée pour récupérer un bloc d'information dans une liste (mémoire ou fichier XSM) et la copier en mémoire.

```
CALL LCMGDL(IPLIST,ISET,DATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la liste.
ISET	*INTEGER*	indice du bloc dans la liste. Un appel à `XABORT` est effectué si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 1.

paramètre de sortie:		
DATA	*	vecteur de dimension $\geq$ `ILONG` dans lequel le bloc a été copié.

La procédure `LCMGDL` peut être utilisée pour récupérer des chaînes de caractères écrites dans un bloc. Il est également possible d'utiliser le sous-programme `LCMGCL` présenté à la section Section 4.6.3. Dans l'exemple suivant, un bloc de longueur 5 situé en `ISET`–ème position dans une liste pointée par `IPLIST` est récupéré en utilisant `LCMGDL` et copié dans la variable de type `character*20` nommée `HNAME` en utilisant un `WRITE` interne:

```
CHARACTER HNAME*20
INTEGER IDATA(5)
IPLIST=...
ISET=...
CALL LCMGDL(IPLIST,ISET,IDATA)
WRITE(HNAME,'(5A4)') (IDATA(I),I=1,5)
```

4.3.7 LCMPDL

Commande utilisée pour stocker un bloc d'information dans une liste (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la `ISET`–ème position de la liste. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
CALL LCMPDL(IPLIST,ISET,ILONG,ITYLCM,DATA)
```

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de la liste.
`ISET`	*INTEGER*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 1.
`ILONG`	*INTEGER*	longueur du bloc en mots. Si un mot contient N valeurs en double précision, `ILONG` doit avoir la valeur $2 \times N$.
`ITYLCM`	*INTEGER*	type de l'information. =1 integer; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =99 undefined.
`DATA`	*	vecteur de dimension $\geq$ `ILONG` duquel le bloc est copié. Les `ILONG` premiers éléments de `DATA` doivent être initialisés avant l'appel à `LCMPDL`.

La procédure `LCMPDL` peut être utilisée pour stocker un bloc de chaînes de caractères. Il est également possible d'utiliser le sous-programme `LCMPCL` présenté à la section Section 4.6.4. Dans l'exemple suivant, une variable de type `character*20` nommée `HNAME` va être copiée dans un tableau d'entiers de longueur 5 situé en `ISET`–ème position dans la liste en utilisant `LCMPDL`:

```
CHARACTER HNAME*20
INTEGER IDATA(5)
IPLIST=...
ISET=...
READ(HNAME,'(5A4)') (IDATA(I),I=1,5)
CALL LCMPDL(IPLIST,ISET,5,3,IDATA)
```

4.3.8 LCMGSL

Commande utilisée pour récupérer l'adresse `SETARA` d'un bloc d'information stocké dans une liste (mémoire ou fichier XSM) *sans faire de copie* de l'information. L'utilisation de cette commande doit se faire en respectant les règles suivantes:

- Si l'information est modifiée après l'appel à `LCMGSL`, un appel à `LCMPSL` doit ultérieurement être effectué pour prendre en compte les modifications.

- Le bloc `BASE(IOFSET)` ne doit jamais être libéré par une primitive de désallocation mémoire (`RLSARA`, `deallocate`, etc.).
- La variable `IOFSET` ne doit jamais être copiée dans une autre variable.

Le non respect de ces règles peut mener à des anomalies d'exécution (core dump, segmentation fault, etc) sans possibilité de lever une exception.

Cette fonction représente une capacité avancée du logiciel `LCM` et elle doit être utilisée uniquement dans les situations où la réduction des ressources mémoire est primordiale. Une adresse est retournée dans le vecteur `BASE` stocké dans un common de la forme `COMMON BASE(1)`. Un appel à `LCMGSL` ne cause aucune modification de la liste. L'information utile est par conséquent atteinte depuis `BASE(IOFSET)` à `BASE(IOFSET+ILONG-1)`.

`CALL LCMGSL(IPLIST,ISET,IOFSET)`

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de la liste.
`ISET`	*INTEGER*	indice du bloc dans la liste. Un appel à `XABORT` est réalisé si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 1.

paramètre de sortie:		
`IOFSET`	*INTEGER*	adresse `SETARA` de l'information.

4.3.9 LCMPSL

Commande utilisée pour stocker un bloc d'information dans une liste (mémoire ou fichier XSM) *sans faire de copie* de l'information. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

Si le `ISET`*–ème élément existe déjà, son pointeur* `SETARA` *est remplacé par le nouveau et l'information pointée par l'ancien est désallouée.*

Cette fonction représente une capacité avancée du logiciel `LCM` et elle doit être utilisée uniquement dans les situations où la réduction des ressources mémoire est primordiale. Le bloc mémoire contenant l'information récupérée par `LCMPSL` devrait avoir été préalablement alloué par un appel à `SETARA` de la forme `CALL SETARA(BASE,ILONG,IOFSET)` où `BASE` est un vecteur stocké dans un common de la forme `COMMON BASE(1)`.

`CALL LCMPSL(IPLIST,ISET,ILONG,ITYLCM,IOFSET)`

paramètres d'entrée:		
`IPLIST`	*INTEGER*	adresse de la liste.
`ISET`	*INTEGER*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 1.
`ILONG`	*INTEGER*	longueur du bloc en mots.
`ITYLCM`	*INTEGER*	type de l'information. =1 integer; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =99 undefined.
`IOFSET`	*INTEGER*	adresse `SETARA` de l'information. Les éléments de `BASE(IOFSET)` à `BASE(IOFSET+ILONG-1)` doivent avoir été intialisés avant l'appel à `LCMPSL`.

paramètre de sortie:		
IOFSET	*INTEGER*	IOFSET=0 indique que l'information précédemment pointée par IOFSET est maintenant gérée par LCM.

4.4 Gestion des tables associatives et des listes

Ces sous-programmes permettent de créér (LCMSIX, LCMDID, LCMDIL, LCMLID, LCMLIL) et d'accéder (LCMSIX, LCMGID, LCMGIL) les tables associatives filles et les listes filles. L'utilisation de ces sous-programmes est résumée dans le tableau suivant:

		fille	
		table associative	liste
mère	table associative	LCMDID LCMGID	LCMLID LCMGID
	liste	LCMDIL LCMGIL	LCMLIL LCMGIL

4.4.1 LCMDID

Commande utilisée pour créer la structure arborescente d'une tables associatives fille (mémoire ou fichier XSM) incluse dans une table associative mère. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

La table associative fille est créée si elle n'existe pas déjà. Sinon, elle est seulement accédée. Dans ce dernier cas, il est préférable d'utiliser le sous-programme LCMGID qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.

```
JPLIST=LCMDID(IPLIST,NAMP)
```

paramètres d'entrées:		
IPLIST	*INTEGER*	adresse de la table associative mère.
NAMP	*CHARACTER*12*	nom de la table associative fille.

paramètre de sortie:		
JPLIST	*INTEGER(*)*	adresse de la table associative fille.

4.4.2 LCMLID

Commande utilisée pour créer une liste fille dans un table associative mère (mémoire ou fichier XSM). Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

Dans l'exemple suivant, la liste LIST contenant 5 composantes est créée et un bloc d'information est stocké dans chaque composante avec LCMPDL:

```
...
JPLIST=LCMLID(IPLIST,'LIST',5)
DO 10 I=1,5
   CALL LCMPDL(JPLIST,I,...
```

```
      ...
10 CONTINUE
```

La capacité des listes est implémentée à travers des appels au sous-programme `LCMLID`. Cet appel permet les possibilités suivantes:

- la liste est créée si elle n'existe pas déjà.
- la liste est seulement accédée si elle existe déjà et si sa longueur est égale à la précédente. Dans ce dernier cas, il est préférable d'utiliser le sous-programme `LCMGID` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.
- la liste est étendue (composantes ajoutées) si elle existe déjà et si sa longueur est supérieure à la précédente.

`JPLIST=LCMLID(IPLIST,NAMP,ILONG)`

paramètres d'entrées:		
`IPLIST`	*INTEGER*	adresse de la table associative mère.
`NAMP`	*CHARACTER*12*	nom de la liste fille.
`ILONG`	*INTEGER*	nombre de composantes de la liste fille.

paramètre de sortie:		
`JPLIST`	*INTEGER*	adresse de la liste nommée `NAMP`.

4.4.3 LCMLIL

Commande utilisée pour créer une liste incluse dans une autre (mémoire ou fichier XSM). Si l'objet LCM est ouvert en mode `read-only`, un déplacement dans une liste inexistante ne peut être effectué.

Dans l'exemple suivant, une liste d'adresse `JPLIST` est incluse dans l'élément 77 de la liste mère d'adresse `IPLIST`. La liste `JPLIST` contenant 5 composantes est créée et un bloc d'information est stocké dans chaque composante avec `LCMPDL`:

```
   ...
   JPLIST=LCMLIL(IPLIST,77,5)
   DO 10 I=1,5
      CALL LCMPDL(JPLIST,I,...
      ...
10 CONTINUE
```

La capacité des listes est implémentée à travers des appels au sous-programme `LCMLIL`. Cet appel permet les possibilités suivantes:

- la liste est créée si elle n'existe pas déjà.
- la liste est seulement accédée si elle existe déjà et si sa longueur est égale à la précédente. Dans ce dernier cas, il est préférable d'utiliser le sous-programme `LCMGIL` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.

- la liste est étendue (composantes ajoutées) si elle existe déjà et si sa longueur est supérieure à la précédente.

```
JPLIST=LCMLIL(IPLIST,ISET,ILONG)
```

paramètres d'entrées:		
IPLIST	*INTEGER*	adresse de la liste mère.
ISET	*INTEGER*	position dans la liste mère de la liste incluse. Le premier élément de la liste mère est situé à l'index 1.
ILONG	*INTEGER*	nombre de composantes de la liste.

paramètre de sortie:		
JPLIST	*INTEGER*	adresse de la liste incluse.

4.4.4 LCMDIL

Commande utilisée pour accéder à la structure arborescente d'une table associative fille (mémoire ou fichier XSM) contenue dans une liste mère. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

La table associative fille est créée si elle n'existe pas déjà. Sinon, elle est seulement accédée. Dans ce dernier cas, il est préférable d'utiliser le sous-programme `LCMGIL` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.

Il est souvent intéressant de remplacer un ensemble de N tables associatives distinctes par une liste de N tables associatives, comme indiqué dans Figure 2.

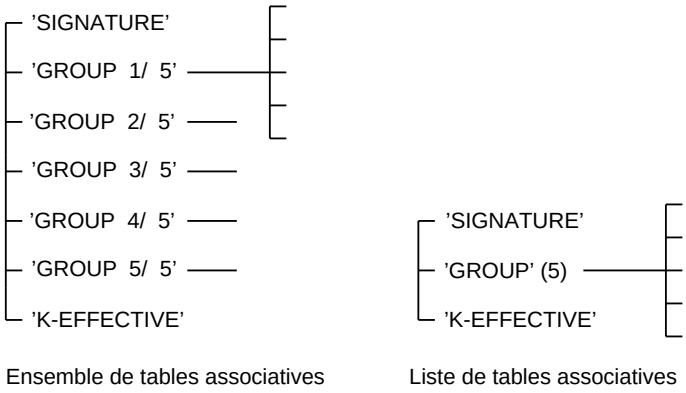

Figure 2: Exemple d'une liste de tables associatives.

Dans l'exemple de Figure 2, on devrait remplacer un ensemble de 5 tables associatives distinctes, chacune créée par `LCMDID`

```
CHARACTER HDIR*12
...
DO 10 I=1,5
```

```
      WRITE(HDIR,'(5HGROUP,I3,4H/  5)') I
      JPLIST=LCMDID(IPLIST,HDIR)
         CALL LCMPUT(JPLIST,...
         ...
   10 CONTINUE
```

par une liste de 5 tables associatives:

```
      ...
      JPLIST=LCMLID(IPLIST,'GROUP',5)
      DO 10 I=1,5
         KPLIST=LCMDIL(JPLIST,I)
         CALL LCMPUT(KPLIST,...
         ...
   10 CONTINUE
```

La capacité d'inclusion de tables associatives dans une liste est implémentée à travers des appels au sous-programme LCMDIL:

```
JPLIST=LCMDIL(IPLIST,ISET)
```

paramètres d'entrées:		
IPLIST	*INTEGER*	adresse de la liste mère.
ISET	*INTEGER*	position dans la liste mère de la table associative fille. Le premier élément de la liste mère est situé à l'index 1.

paramètre de sortie:		
JPLIST	*INTEGER*	adresse de la table associative fille.

4.4.5 LCMGID

Commande utilisée pour récupérer l'adresse d'une table associative ou d'une liste stockée dans une table associative mère.

```
JPLIST=LCMGID(IPLIST,NAMP)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative mère.
NAMP	*CHARACTER*12*	nom de la table associative fille ou de la liste fille à récupérer. Un appel à XABORT est réalisé si cette information n'existe pas.

paramètre de sortie:		
JPLIST	*INTEGER*	adresse de la table associative fille ou de la liste fille.

4.4.6 LCMGIL

Commande utilisée pour récupérer l'adresse d'une table associative ou d'une liste stockée dans une liste mère.

```
JPLIST=LCMGIL(IPLIST,ISET)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la liste mère.
ISET	*INTEGER*	indice de la table associative fille ou de la liste fille à récupérer. Un appel à XABORT est réalisé si cette information n'existe pas. Le premier élément de la liste mère est situé à l'index 1.

paramètre de sortie:		
JPLIST	*INTEGER*	adresse de la table associative fille ou de la liste fille.

4.4.7 LCMSIX

Commande utilisée pour se déplacer dans la structure arborescente d'une table associative ou pour changer de répertoire actif, sans se préoccuper de conserver les adresses des différents niveaux hiérarchiques. Si l'objet LCM est ouvert en mode `read-only`, un déplacement dans une table associative inexistante ne peut être effectué.

```
CALL LCMSIX(IPLIST,NAMP,IACT)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative avant exécution de LCMSIX.
NAMP	*CHARACTER*12*	nom de la table associative fille si IACT=1. Ce paramètre n'est pas utilisé si IACT=0 ou IACT=2.
IACT	*INTEGER*	type du déplacement. =0 revient à la racine de l'objet LCM; =1 va à la table associative fille (la créer si elle n'existe pas déjà); =2 revient à la table associative mère.

paramètre de sortie:		
IPLIST	*INTEGER*	adresse de la table associative après exécution de LCMSIX.

4.5 Utilitaires

4.5.1 LCMLIB

Commande utilisée pour imprimer le contenu d'une table associative ou d'une liste (mémoire ou fichier XSM).

```
CALL LCMLIB(IPLIST)
```

paramètre d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative ou de la liste.

4.5.2 LCMEQU

Commande utilisée pour copier l'information contenue dans une table associative (pointé(e) par IPLIS1) et ses filles vers la table associative pointé(e) par IPLIS2. IPLIS1 et IPLIS2 peuvent être stockés en mémoire ou sur fichier XSM. Notez que la seconde table associative (ou fichier XSM) est modifiée mais pas créée par LCMEQU.

```
CALL LCMEQU(IPLIS1,IPLIS2)
```

paramètre d'entrée:		
IPLIS1	*INTEGER*	adresse de la table associative ou de la liste existante (accédée en mode read-only).

paramètre de sortie:		
IPLIS2	*INTEGER*	adresse de la table associative ou de la liste qui sera modifiée par LCMEQU.

4.5.3 LCMEXP

Commande utilisée pour exporter (importer) le contenu d'une table associative (mémoire ou fichier XSM) vers un (d'un) fichier séquentiel binaire ou ascii en utilisant la méthode des contours. L'exportation commence du répertoire actif. Il s'agit d'un algorithme de sérialisation.

```
CALL LCMEXP(IPLIST,IMPX,NUNIT,IMODE,IDIR)
```

paramètres d'entrées:		
IPLIST	*INTEGER*	adresse de la table associative ou de la liste qui doit être exportée (ou importée).
IMPX	*INTEGER*	paramètre d'impression (positionné à zéro pour aucune impression).
NUNIT	*INTEGER*	numéro d'unité du fichier séquentiel où l'exportation s'effectue (ou d'où l'importation est réalisée).
IMODE	*INTEGER*	=1 pour fichier séquentiel BINAIRE; =2 pour fichier séquentiel ASCII.
IDIR	*INTEGER*	=1 pour exporter; =2 pour importer.

4.6 Traitement des vecteurs de variables caractères

Les sous-programmes suivants ont été écrits en FORTRAN-77 à partir des API LCM précédentes afin de faciliter le traitement des vecteurs de variables caractères.

		type d'opération	
		put	get
mère	table associative	LCMPCD	LCMGCD
	liste	LCMPCL	LCMGCL

4.6.1 LCMGCD

Commande utilisée pour récupérer un vecteur de variables caractères dans une table associative (mémoire ou fichier XSM) et le copier en mémoire.

```
CALL LCMGCD(IPLIST,NAMP,HDATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du vecteur de variables caractères à récupérer. Un appel à XABORT est déclenché si le bloc n'existe pas.

paramètre de sortie:		
HDATA	*CHARACTER*(*)(*)*	vecteur de dimension $\geq$ ILONG dans lequel le vecteur de variables caractères a été copié.

4.6.2 LCMPCD

Commande utilisée pour stocker un vecteur de variables caractères dans une table associative (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la table associative. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode read-only.

```
CALL LCMPCD(IPLIST,NAMP,ILONG,HDATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la table associative.
NAMP	*CHARACTER*12*	nom du vecteur de variables caractères.
ILONG	*INTEGER*	dimension du vecteur de variables caractères.
HDATA	*CHARACTER*(*)(*)*	vecteur de dimension $\geq$ ILONG duquel le vecteur de variables caractères est copié.

Exemple d'utilisation:

```
      PARAMETER (ILONG=5)
      CHARACTER*16 HDATA1(ILONG),HDATA2(ILONG)
*
      CALL LCMOP(IPLIST,'mon_dict',0,1,2)
*
* STOCKAGE DE L'INFORMATION.
      HDATA1(1)='string1'
      HDATA1(2)='  string2'
      HDATA1(3)='    string3'
      HDATA1(4)='      string4'
      HDATA1(5)='        string5'
      CALL LCMPCD(IPLIST,'node1',ILONG,HDATA1)
 *
```

```
* RECUPERATION DE L'INFORMATION.
      CALL LCMGCD(IPLIST,'node1',HDATA2)
      DO 10 I=1,ILONG
      PRINT *,'I=',I,' RECOVER HDATA2 -->',HDATA2(I),'<--'
   10 CONTINUE
*
      CALL LCMCL(IPLIST,2)
```

4.6.3 LCMGCL

Commande utilisée pour récupérer un vecteur de variables caractères dans une liste (mémoire ou fichier XSM) et le copier en mémoire.

```
CALL LCMGCL(IPLIST,NAMP,HDATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la liste.
ISET	*INTEGER*	indice du vecteur de variables caractères dans la liste. Un appel à XABORT est déclenché si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 1.

paramètre de sortie:		
HDATA	*CHARACTER*(*)(*)*	vecteur de dimension $\geq$ ILONG dans lequel le vecteur de variables caractères a été copié.

4.6.4 LCMPCL

Commande utilisée pour stocker un vecteur de variables caractères dans une liste (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la liste. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
CALL LCMPCL(IPLIST,ISET,ILONG,HDATA)
```

paramètres d'entrée:		
IPLIST	*INTEGER*	adresse de la liste.
ISET	*INTEGER*	indice du vecteur de variables caractères dans la liste. Le premier élément de la liste est situé à l'index 1.
ILONG	*INTEGER*	dimension du vecteur de variables caractères.
HDATA	*CHARACTER*(*)(*)*	vecteur de dimension $\geq$ ILONG duquel le vecteur de variables caractères est copié.

Exemple d'utilisation:

```
PARAMETER (ILONG=5)
CHARACTER*16 HDATA1(ILONG),HDATA2(ILONG)
```

```
*
      CALL LCMOP(IPLIST,'mon_dict',0,1,2)
*
* CREATION DE LA LISTE IMBRIQUEE.
      JPLIST=LCMLID(IPLIST,'node2',77)
*
* STOCKAGE DE L'INFORMATION.
      HDATA1(1)='string1'
      HDATA1(2)='  string2'
      HDATA1(3)='    string3'
      HDATA1(4)='      string4'
      HDATA1(5)='        string5'
      CALL LCMPCL(JPLIST,1,ILONG,HDATA1)
*
* RECUPERATION DE L'INFORMATION.
      CALL LCMGCL(JPLIST,1,HDATA2)
      DO 10 I=1,ILONG
      PRINT *,'I=',I,' RECOVER HDATA2 -->',HDATA2(I),'<--'
   10 CONTINUE
*
      CALL LCMCL(IPLIST,2)
```

4.7 Gestion des exceptions et arrêt

4.7.1 XABORT

Commande utilisée pour réaliser un *arrêt propre* depuis un opérateur de calcul (si le module d'extension a été construit selon les instructions de la référence [4]) ou depuis une procédure Python. Cette procédure lance un appel à la fonction `PYABORT()` de l'API LCM qui lève une exception Python.

Cette procédure est utile pour gérer des situations anormales qui peuvent se produire dans une application. Il est recommandé d'inclure le nom de la subroutine ou fonction dans le message `XABORT`. Par exemple, si une erreur est détectée dans la subroutine `SUB001`, on devrait écrire:

```
      CALL XABORT('SUB001: EXECUTION FAILURE.')
```

```
CALL XABORT(HSMG)
```

paramètre d'entrée:		
`HSMG`	*CHARACTER*(*)*	message décrivant les conditions de l'arrêt.

4.8 Allocation dynamique de la mémoire en FORTRAN-77

Cette fonction permet aux applications FORTRAN-77 d'allouer et libérer dynamiquement une zone mémoire en appelant des fonctions de gestion mémoire (**malloc** sur les systèmes UNIX). Ces possibilités sont requises dans une application modulaire pour ajuster la mémoire à la taille du problème en cours de résolution.

4.8.1 SETARA

Commande utilisée pour allouer une zone mémoire. Elle retourne l'adresse dans un vecteur `BASE` stocké dans un common de la forme `COMMON BASE(1)`. Si le système ne peut allouer `ILONG` mots, le programme s'arrête.

```
CALL SETARA(BASE,ILONG,IOFDUM)
```

paramètres d'entrée:		
`BASE`	*INTEGER*	l'adresse de `BASE(1)` est utilisée omme origine par `SETARA` pour donner `IOFDUM`.
`ILONG`	*INTEGER*	longueur en mots de la zone mémoire. Pour allouer une zone mémoire afin de stocker N éléments en double précision, `ILONG` devrait être positionné à $2 \times N$.

paramètre de sortie:		
`IOFDUM`	*INTEGER*	adresse de la zone mémoire dans le vecteur `BASE`. La zone allouée est par conséquent comprise entre `BASE(IOFDUM) et BASE(IOFDUM+ILONG-1)`.

Notez que `BASE(1)` ne devrait jamais être déclaré en vecteur de double précision car l'adresse `IOFDUM` est calculée en considérant que `BASE` ne contient que des mots simples.

4.8.2 RLSARA

Commande utilisée pour libérer une zone mémoire préalablement allouée par le sous-programme Fortran `SETARA` ou la fonction C `SETARA_C`. `RLSARA` utilise des fonctions de gestion mémoire (**free** sur les systèmes UNIX). Si le système ne peut libérer la zone mémoire, le programme s'arrête.

```
CALL RLSARA(BASE(IOFDUM))
```

paramètre d'entrée:		
`BASE(IOFDUM)`	*	zone mémoire à libérer.

4.8.3 Exemple d'allocation mémoire en FORTRAN-77

Dans le premier exemple, une zone mémoire est allouée dynamiquement, un élément `'RECORD1'` est créé et copié dans une table associative en utilisant la procédure `LCMPUT`. La procédure `SUB` est appelée pour faciliter la création de l'élément:

```
COMMON BASE(1)
.
.
.
ILONG=30
CALL SETARA(BASE,ILONG,IOFDUM)
CALL SUB(BASE(IOFDUM),ILONG)
CALL LCMPUT(IPLIST,'RECORD1',ILONG,2,BASE(IOFDUM))
CALL RLSARA(BASE(IOFDUM))
RETURN
```

```
      END
      .
      .
      .
      SUBROUTINE SUB(PHI,ILONG)
      DIMENSION PHI(ILONG)
      DO 10 I=1,ILONG
   10 PHI(I)=REAL(I)
      RETURN
      END
```

Dans le second exemple, une zone mémoire est allouée dynamiquement, un élément `'RECORD1'` est créé et stocké dans une table associative (sans la copier) en utilisant la procédure `LCMPOF`:

```
      COMMON BASE(1)
      .
      .
      .
      ILONG=30
      CALL SETARA(BASE,ILONG,IOFDUM)
      CALL SUB(BASE(IOFDUM),ILONG)
      CALL LCMPOF(IPLIST,'RECORD1',ILONG,2,IOFDUM)
      RETURN
      END
      .
      .
      .
      SUBROUTINE SUB(PHI,ILONG)
      DIMENSION PHI(ILONG)
      DO 10 I=1,ILONG
   10 PHI(I)=REAL(I)
      RETURN
      END
```

Dans le troisième exemple, on récupère un bloc stocké dans un table associative en utilisant la fonction `LCMIOF`, sans copier ni allouer de mémoire. Une procédure `SUB` est appelée pour faciliter l'utilisation de l'information récupérée:

```
      COMMON BASE(1)
      LOGICAL EMPTY,LCM
      CHARACTER*12 NAMLCM,NAMMY
      .
      .
      .
      CALL LCMINF(IPLIST,NAMLCM,NAMMY,EMPTY,LCM)
      CALL LCMLEN(IPLIST,'RECORD1',ILONG,ITYLCM)
      CALL LCMIOF(IPLIST,'RECORD1',IOFDUM)
      CALL SUB(BASE(IOFDUM),ILONG)
      RETURN
      END
      .
      .
      .
```

```
      SUBROUTINE SUB(PHI,ILONG)
      DIMENSION PHI(ILONG)
      PRINT *,'VECTOR PHI CONTENT=',(PHI(I),I=1,ILONG)
      RETURN
      END
```

5 API C de l'objet LCM

Ces fonctions constituent la base logicielle sur laquelle les *bindings* Python et Fortran ont été écrits. Elles sont appelables directement depuis C et depuis les langages informatiques qui permettent l'interopérabilité avec C.

La definition des structures `lcm` est accédée par un include de la forme

```
#include "lcm.h"
```

5.1 Ouverture, fermeture et validation d'objets LCM

5.1.1 LCMOP_C

Appel utilisé pour ouvrir un objet LCM (mémoire ou fichier XSM) et fournir son adresse si l'objet est créé. Dans le cadre d'un code, c'est l'interface avec Python qui est responsable de faire appel à `LCMOP_C`. L'utilisation de `LCMOP_C` est généralement restreinte à l'utilisation de tables associatives temporaires à l'intérieur d'un opérateur.

```
LCMOP_C(iplist,namp,imp,medium,impx);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de l'objet LCM si `imp=1` ou `imp=2`. Cette adresse est celle de la table associative racine de l'objet.
`namp`	*char**	nom de la table associative si `imp=0`.
`imp`	*long*	=0 pour créer une nouvelle table associative ; =1 pour modifier un objet LCM existant; =2 pour accéder à un objet LCM en mode **read-only**.
`medium`	*long*	=1 utilise un objet LCM en mémoire; =2 utilise un fichier XSM pour stocker la table associative.
`impx`	*long*	paramètre d'impression. Egal à zéro pour supprimer toute trace d'impression.

paramètres de sortie:		
`iplist`	*lcm***	adresse d'un objet LCM si `imp=0`.
`namp`	*char**	nom de la table associative si `imp=1` ou `imp=2`.
valeur de la fonction:		
void		

5.1.2 LCMCL_C

Fonction utilisée pour fermer un objet LCM (mémoire ou fichier XSM). Dans le cadre d'un code, c'est l'interface avec Python qui est responsable de faire appel à `LCMCL_C`. L'utilisation de `LCMCL_C` est généralement restreinte à la fermeture de tables associatives temporaires à l'intérieur d'un opérateur.

On ne doit jamais utiliser la fonction `LCMCL_C` *sur un objet LCM crée par la fonction* `lcm.new()` *ou* `lcm.xsm()` *de Python.*

Un objet LCM ne peut être fermé que si `iplist` pointe vers son répertoire racine.

```
LCMCL_C(iplist,iact);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de l'objet LCM (adresse du répertoire racine de l'objet).
`iact`	*long*	=1 ferme la table associative sans la détruire; =2 la détruit.

paramètre de sortie:		
`iplist`	*lcm***	`iplist=0` indique que la table associative est détruite ou que le fichier XSM est fermé. Un objet LCM en mémoire garde le même `iplist` durant son existence. Un fichier XSM est représenté par un `iplist` différent chaque fois qu'il est réouvert.
valeur de la fonction:		
void		

5.1.3 LCMVAL_C

Fonction utilisée pour valider un simple bloc ou l'ensemble des blocs contenus dans une table associative ou une liste. Si `iplist` se réfère à un fichier XSM, aucune action n'est faite. La validation consiste à vérifier les connections entre les éléments de l'objet LCM, vérifier si chaque élément de la table associative ou de la liste est défini et éventuellement les destructions mémoire. Si une erreur est détectée, le message suivant s'inscrit:

```
LCMVAL_C: BLOCK xxx OF THE TABLE yyy HAS BEEN OVERWRITTEN.
```

L'appel se fait comme suit:

```
LCMVAL_C(iplist,namp);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la table associative ou de la liste.
`namp`	*char**	nom du bloc à valider dans la table associative. Si `namp=' '`, tous les blocs de la table associative sont validés.

valeur de la fonction:	
void	

5.2 Interrogation des objets LCM

Les structures d'information LCM sont auto-descriptives. Il est donc possible de les interroger pour connaître leur caractéristiques.

		type d'interrogation	
		structure mère	bloc d'information
mère	table associative	`LCMINF_C` `LCMNXT_C`	`LCMLEN_C`
	liste	`LCMINF_C`	`LCMLEL_C`

5.2.1 LCMLEN_C

Fonction utilisée pour récupérer la longueur et le type d'un bloc d'information stocké dans une table associative (mémoire ou fichier XSM). La valeur de la longueur est le nombre d'éléments dans une liste ou le nombre de mots nécessaires pour stocker l'information dans un bloc élémentaire. Par exemple, la longueur pour stocker un tableau de `double` est deux fois supérieure à sa dimension.

```
LCMLEN_C(iplist,namp,ilong,itylcm);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc.

paramètres de sortie:		
`ilong`	*long**	longueur du bloc en mots. $=-1$ pour une table associative fille; $=N$ pour une liste fille contenant N composantes; =0 si le bloc n'existe pas.
`itylcm`	*long**	type de l'information. =0 table associative; =1 entier; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =10 liste; =99 indéfini (99 est retourné si le bloc ou répertoire n'existe pas).
valeur de la fonction:		
void		

5.2.2 LCMINF_C

Fonction utilisée pour récupérer des informations générales sur une table associative ou une liste (mémoire ou fichier XSM).

```
LCMINF_C(iplist,namlcm,nammy,empty,ilong,lcm);
```

paramètre d'entrée:		
`iplist`	*lcm***	adresse de la table associative ou de la liste.

paramètres de sortie:		
namlcm	*char**	nom Python de l'objet contenant la table associative (nom de l'attribut _name de l'objet Python).
nammy	*char**	nom de la table associative pointée par iplist (nom de l'attribut _directory de l'objet Python). ='/' si la table associative est la racine de l'objet; =' ' si la table associative est un élément de liste.
empty	*long**	variable logique positionnée à 1 si la table associative ne contient aucun élément; positionnée à 0 sinon.
ilong	*long**	$= -1$: iplist est une table associative; > 0: nombre de composantes dans la liste iplist.
lcm	*long**	variable int positionnée à 1 si l'information est stockée dans une table associative en mémoire ou positionnée à 0 si l'information est stockée dans un fichier XSM.
valeur de la fonction:		
void		

5.2.3 LCMNXT_C

Fonction utilisée pour trouver le nom du bloc suivant dans une table associative (mémoire ou fichier XSM). L'utilisation de LCMNXT_C est interdite si la table associative ne contient aucun élément.

```
LCMNXT_C(iplist,namp);
```

paramètres d'entrée:		
iplist	*lcm***	adresse de la table associative.
namp	*char**	nom d'un bloc existant. namp=' ' peut être utilisé pour obtenir le nom du premier bloc.

paramètre de sortie:		
namp	*char**	nom du bloc suivant. Un appel à XABORT est effectué si la table associative est vide.
valeur de la fonction:		
void		

5.2.4 LCMLEL_C

Fonction utilisée pour récupérer la longueur et le type du bloc stocké dans une liste (mémoire ou fichier XSM). La valeur de la longueur est le nombre de mots nécessaires pour stocker l'information. Par exemple, la longueur pour stocker un tableau de double est deux fois supérieure à sa dimension.

```
LCMLEL_C(iplist,iset,ilong,itylcm);
```

paramètres d'entrée:		
iplist	*lcm***	adresse de la liste.
iset	*long*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 0.

paramètres de sortie:		
`ilong`	*long**	longueur en mots du bloc. =0 si le bloc n'existe pas.
`itylcm`	*long**	type de l'information. =1 int; =2 float; =3 `char[4]` data; =4 double; =5 int (0 ou 1); =6 complex; =10 list; =99 indéfini (99 est retourné si le bloc n'existe pas).
valeur de la fonction:		
void		

5.3 Gestion des blocs d'information élémentaires

La gestion des blocs d'information élémentaires est réalisée avec copie de l'information (`LCMPUT_C`, `LCMGET_C`, `LCMPDL_C` ou `LCMGDL_C`) ou sans copie (`LCMPPD_C`, `LCMGPD_C`, `LCMPPL_C` ou `LCMGPL_C`).

		type d'opération	
		put	get
mère	table associative	`LCMPUT_C` `LCMPPD_C`	`LCMGET_C` `LCMGPD_C`
	liste	`LCMPDL_C` `LCMPPL_C`	`LCMGDL_C` `LCMGPL_C`

5.3.1 LCMGET_C

Fonction utilisée pour récupérer un bloc d'information dans une table associative (mémoire ou fichier XSM) et le copier en mémoire.

```
LCMGET_C(iplist,namp,data);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc à récupérer. Un appel à `XABORT` est déclenché si le bloc n'existe pas.

paramètre de sortie:		
`data`	*long**	vecteur de dimension $\geq$ `ilong` dans lequel le bloc a été copié.
valeur de la fonction:		
void		

La fonction `LCMGET_C` peut être utilisée pour récupérer des chaînes de caractères écrites dans un bloc. Dans l'exemple suivant, un bloc de longueur 5 nommé `namp` stocké dans une table associative pointée par `iplist` est récupéré en utilisant `LCMGET_C` et copié dans la variable de type `char[20]` nommée `hname` en utilisant un `sprintf`:

```
char *namp="...", hname[21];
int idata[5],i;
lcm *iplist;
iplist=... ;
LCMGET_C(&iplist,namp,idata);
for(i=0;i<5;i++) {
```

```
  (void)sprintf(hname+i1*4,"%4d",idata[i]);
}
hname[20] = '\0'
```

5.3.2 LCMPUT_C

Fonction utilisée pour stocker un bloc d'information dans une table associative (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la table associative. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
LCMPUT_C(iplist,namp,ilong,itylcm,data);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc.
`ilong`	*long*	longueur du bloc en mots. Si un mot contient N valeurs en double précision, `ilong` doit avoir la valeur $2 \times N$.
`itylcm`	*long*	type de l'information. =1 integer; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =99 indéfini.
`data`	*long**	vecteur de dimension $\geq$ `ilong` duquel le bloc est copié. Les éléments de `iofset[0]` à `iofset[ilong-1]` doivent être initialisés avant l'appel à `LCMPUT_C`.

valeur de la fonction:	
void	

La fonction `LCMPUT_C` peut être utilisée pour stocker un bloc de chaînes de caractères. Dans l'exemple suivant, une variable de type `char[20]` nommée `hname` va être copiée dans un tableau d'entiers de longueur 5 nommé `idata` en utilisant un `sscanf` et stocké dans le bloc `namp` dans la table associative pointée par `iplist` en utilisant `LCMPUT_C`:

```
char *namp="...", hname[20];
int idata[5],i,it=3,il=5;
lcm *iplist;
iplist=... ;
for(i=0;i<5;i++) {
  (void)sscanf(hname+i1*4,"%4d",idata+i);
}
LCMPUT_C(&iplist,namp,il,it,idata);
```

5.3.3 LCMGPD_C

Fonction utilisée pour récupérer l'adresse d'un bloc d'information stocké dans une table associative (mémoire ou fichier XSM) *sans faire de copie* de l'information. L'utilisation de cette commande doit se faire en respectant les règles suivantes:

- Si l'information est modifiée après l'appel à `LCMGPD_C`, un appel à `LCMPPD_C` doit ultérieurement être effectué pour prendre en compte les modifications.
- Le bloc `*iofset` ne doit jamais être libéré par une primitive de désallocation mémoire (`RLSARA_C`, `free`, etc.).
- L'adresse `*iofset` ne doit jamais être copiée dans une autre variable.

Le non respect de ces règles peut mener à des anomalies d'exécution (core dump, segmentation fault, etc) sans possibilité de lever une exception.

Un appel à `LCMGPD_C` ne cause aucune modification de la table associative. L'information utile est par conséquent atteinte depuis `*iofset[0]` à `*iofset[ilong-1]` où `*iofset` est l'adresse retournée par la fonction `LCMGPD_C`.

```
LCMGPD_C(iplist,namp,iofset);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc à récupérer. Un appel à `XABORT` est réalisé si le bloc n'existe pas.

paramètre de sortie:		
`iofset`	*long***	adresse de l'information.
valeur de la fonction:		
void		

5.3.4 LCMPPD_C

Fonction utilisée pour stocker un bloc d'information dans une table associative (mémoire ou fichier XSM) *sans faire de copie* de l'information. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée sur un objet LCM ouvert en mode `read-only`.

Si l'élément `namp` *existe déjà, son pointeur est remplacé par le nouveau et l'information pointée par l'ancien est désallouée.*

Le bloc mémoire contenant l'information récupérée par `LCMPPD_C` devrait avoir été préalablement alloué par un appel de la forme `iofset = SETARA_C(ilong)`.

```
LCMPPD_C(iplist,namp,ilong,itylcm,iofset);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc.
`ilong`	*long*	longueur du bloc en mots.
`itylcm`	*long*	type de l'information. =1 int; =2 float; =3 `char[4]` data; =4 double; =5 int (0 ou 1); =6 complex; =99 indéfini.
`iofset`	*long**	adresse de l'information. Les éléments de `iofset[0]` à `iofset[ilong-1]` doivent avoir été intialisés avant l'appel à `LCMPPD_C`.

valeur de la fonction:	
void	

Le bloc d'information d'adresse `iofset` sera libéré par un `RLSARA_C` au moment où la table associative sera fermée. Il existe des situations où ce bloc est partagé par un logiciel environnant, de telle sorte que l'on ne doive pas libérer le bloc. Dans ce cas, il suffit de faire suivre l'appel à `LCMPPD_C` par un appel à la fonction `refpush` de la forme:

```
refpush(iplist,iofset);
```

5.3.5 LCMDEL_C

Fonction utilisée pour détruire un bloc d'information, une table associative fille ou une liste fille stocké dans une table associative en mémoire. La fonctionnalité de `LCMDEL_C` n'est pas utilisable pour les fichiers XSM.

```
LCMDEL_C(iplist,namp);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du bloc à détruire.

valeur de la fonction:	
void	

5.3.6 LCMGDL_C

Fonction utilisée pour récupérer un bloc d'information dans une liste (mémoire ou fichier XSM) et la copier en mémoire.

```
LCMGDL_C(iplist,iset,data);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la liste.
`iset`	*long*	indice du bloc dans la liste. Un appel à `XABORT` est effectué si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 0.

paramètre de sortie:		
`data`	int*	vecteur de dimension $\geq$ `ilong` dans lequel le bloc a été copié.
valeur de la fonction:		
void		

La fonction `LCMGDL_C` peut être utilisée pour récupérer des chaînes de caractères écrites dans un bloc. Dans l'exemple suivant, un bloc de longueur 5 situé en `iset`–ème position dans une liste pointée par

`iplist` est récupéré en utilisant `LCMGDL_C` et copié dans la variable de type `char[20]` nommée `hname` en utilisant un `sprintf`:

```
char *namp="...", hname[21];
int iset,idata[5],i;
lcm *iplist;
iplist=... ;
iset=...;
LCMGDL_C(&iplist,iset,idata);
for(i=0;i<5;i++) {
  (void)sprintf(hname+i1*4,"%4d",idata[i]);
}
hname[20] = '\0'
```

5.3.7 LCMPDL_C

Fonction utilisée pour stocker un bloc d'information dans une liste (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la `iset`–ème position de la liste. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
LCMPDL_C(iplist,iset,ilong,itylcm,data);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la liste.
`iset`	*long*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 0.
`ilong`	*long*	longueur du bloc en mots. Si un mot contient N valeurs en double précision, `ilong` doit avoir la valeur $2 \times N$.
`itylcm`	*long*	type de l'information. =1 integer; =2 single precision real; =3 `character*4` data; =4 double precision real; =5 logical; =6 complex number; =99 undefined.
`data`	*long**	vecteur de dimension $\geq$ `ilong` duquel le bloc est copié. Les éléments de `iofset[0]` à `iofset[ilong-1]` doivent doivent être initialisés avant l'appel à `LCMPDL_C`.

valeur de la fonction:	
void	

La fonction `LCMPDL_C` peut être utilisée pour stocker un bloc de chaînes de caractères. Dans l'exemple suivant, une variable de type `char[20]` nommée `hname` va être copiée dans un tableau d'entiers de longueur 5 situé en `iset`–ème position dans la liste en utilisant `LCMPDL_C`:

```
char *namp="...", hname[20];
int iset,idata[5],i,it=3,il=5;
lcm *iplist;
iplist=... ;
iset=...;
for(i=0;i<5;i++) {
  (void)sscanf(hname+i1*4,"%4d",idata+i);
```

```
}
LCMPDL_C(&iplist,iset,il,it,idata);
```

5.3.8 LCMGPL_C

Fonction utilisée pour récupérer l'adresse d'un bloc d'information stocké dans une liste (mémoire ou fichier XSM) *sans faire de copie* de l'information. L'utilisation de cette commande doit se faire en respectant les règles suivantes:

- Si l'information est modifiée après l'appel à `LCMGPL_C`, un appel à `LCMPPL_C` doit ultérieurement être effectué pour prendre en compte les modifications.
- Le bloc `*iofset` ne doit jamais être libéré par une primitive de désallocation mémoire (`RLSARA_C`, `free`, etc.).
- L'adresse `*iofset` ne doit jamais être copiée dans une autre variable.

Le non respect de ces règles peut mener à des anomalies d'exécution (core dump, segmentation fault, etc) sans possibilité de lever une exception.

Un appel à `LCMGPL_C` ne cause aucune modification de la liste. L'information utile est par conséquent atteinte depuis `*iofset[0]` à `*iofset[ilong-1]` où `*iofset` est l'adresse retournée par la fonction `LCMGPL_C`.

```
LCMGPL_C(iplist,iset,iofset);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la liste.
`iset`	*long*	indice du bloc dans la liste. Un appel à `XABORT` est réalisé si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 0.

paramètre de sortie:		
`iofset`	*long***	adresse de l'information.
valeur de la fonction:		
void		

5.3.9 LCMPPL_C

Fonction utilisée pour stocker un bloc d'information dans une liste (mémoire ou fichier XSM) *sans faire de copie* de l'information. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée sur un objet LCM ouvert en mode `read-only`.

Si le `iset`*–ème élément existe déjà, son pointeur est remplacé par le nouveau et l'information pointée par l'ancien est désallouée.*

Le bloc mémoire contenant l'information récupérée par `LCMPPL_C` devrait avoir été préalablement alloué par un appel de la forme `iofset = SETARA_C(ilong)`.

```
LCMPPL_C(iplist,iset,ilong,itylcm,iofset);
```

paramètres d'entrée:		
iplist	*lcm***	adresse de la liste.
iset	*long*	indice du bloc dans la liste. Le premier élément de la liste est situé à l'index 0.
ilong	*long*	longueur du bloc en mots.
itylcm	*long*	type de l'information. =1 int; =2 float; =3 char[4] data; =4 double; =5 int (0 ou 1); =6 complex; =99 indéfini.
iofset	*long**	adresse de l'information. Les éléments de iofset[0] à iofset[ilong-1] doivent avoir été intialisés avant l'appel à LCMPPL_C.

valeur de la fonction:	
void	

Le bloc d'information d'adresse iofset sera libéré par un RLSARA_C au moment où la liste sera fermée. Il existe des situations où ce bloc est partagé par un logiciel environnant, de telle sorte que l'on ne doive pas libérer le bloc. Dans ce cas, il suffit de faire suivre l'appel à LCMPPL_C par un appel à la fonction refpush de la forme:

```
refpush(iplist,iofset);
```

5.4 Gestion des tables associatives et des listes

Ces fonctions permettent de créér (LCMSIX_C, LCMDID_C, LCMDIL_C, LCMLID_C, LCMLIL_C) et d'accéder (LCMSIX_C, LCMGID_C, LCMGIL_C) les tables associatives filles et les listes filles. L'utilisation de ces fonctions est résumée dans le tableau suivant:

		fille	
		table associative	liste
mère	table associative	LCMDID_C LCMGID_C	LCMLID_C LCMGID_C
	liste	LCMDIL_C LCMGIL_C	LCMLIL_C LCMGIL_C

5.4.1 LCMDID_C

Fonction utilisée pour créer la structure arborescente d'une tables associatives fille (mémoire ou fichier XSM) incluse dans une table associative mère. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode read-only.

La table associative fille est créée si elle n'existe pas déjà. Sinon, elle est seulement accédée. Dans ce dernier cas, il est préférable d'utiliser la fonction LCMGID_C qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode read-only.

```
LCMDID_C(iplist,namp);
```

paramètres d'entrées:		
iplist	*lcm***	adresse d'une table associative mère.
namp	*char**	nom de la table associative fille.

valeur de la fonction:	
*lcm**	adresse de la table associative fille.

5.4.2 LCMLID_C

Fonction utilisée pour créer ou accéder à une liste dans un table associative (mémoire ou fichier XSM). Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

Dans l'exemple suivant, la liste `LIST` contenant 5 composantes est créée et un bloc d'information est stocké dans chaque composante avec `LCMPPL_C`:

```
lcm *iplist,*jplist;
int n=5, i ;
...
jplist=LCMLID_C(&iplist,"LIST",n);
for(i=0;i<5;i++) {
  LCMPPL_C(&jplist,i,...
}
```

La capacité des listes est implémentée à travers des appels à la fonction `LCMLID_C`. Cet appel permet les possibilités suivantes:

- la liste est créée si elle n'existe pas déjà.
- la liste est seulement accédée si elle existe déjà et si sa longueur est égale à la précédente. Dans ce dernier cas, il est préférable d'utiliser le sous-programme `LCMGID_C` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.
- la liste est étendue (composantes ajoutées) si elle existe déjà et si sa longueur est supérieure à la précédente.

```
LCMLID_C(iplist,namp,ilong);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la table associative mère.
`namp`	*char**	nom de la liste.
`ilong`	*long*	nombre de composantes de la liste.

valeur de la fonction:	
*lcm**	adresse de la liste nommée `namp`.

5.4.3 LCMLIL_C

Fonction utilisée pour créer ou accéder à une liste dans une autre (mémoire ou fichier XSM). Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

Dans l'exemple suivant, une liste d'adresse `jplist` est incluse dans l'élément 77 de la liste mère d'adresse `iplist`. La liste `jplist` contenant 5 composantes est créée et un bloc d'information est stocké dans chaque composante avec `LCMPPL_C`:

```
PyObject *iplist,*jplist;
int n=5, i, iset=77 ;
...
jplist=LCMLIL_C(&iplist,iset,n);
for(i=0;i<5;i++) {
  LCMPPL_C(&jplist,i,...
}
```

La capacité des listes est implémentée à travers des appels à la fonction `LCMLIL_C`. Cet appel nécessite les possibilités suivantes:

- la liste est créée si elle n'existe pas déjà.
- la liste est seulement accédée si elle existe déjà et si sa longueur est égale à la précédente. Dans ce dernier cas, il est préférable d'utiliser le sous-programme `LCMGIL_C` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.
- la liste est étendue (composantes ajoutées) si elle existe déjà et si sa longueur est supérieure à la précédente.

`LCMLIL_C(iplist,iset,ilong);`

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la liste mère.
`iset`	*long*	position dans la liste mère de la liste incluse. Le premier élément de la liste mère est situé à l'index 0.
`ilong`	*long*	nombre de composantes de la liste.

valeur de la fonction:	
*lcm**	adresse de la liste incluse.

5.4.4 LCMDIL_C

Fonction utilisée pour créer ou accéder à la structure arborescente d'une table associative (mémoire ou fichier XSM) contenue dans une liste. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

La table associative fille est créée si elle n'existe pas déjà. Sinon, elle est seulement accédée. Dans ce dernier cas, il est préférable d'utiliser la fonction `LCMGIL_C` qui permet un accès plus rapide et qui est compatible avec les objets LCM en mode `read-only`.

Il est souvent intéressant de remplacer un ensemble de N tables associatives distinctes par une liste de N tables associatives, comme indiqué dans Figure 2.

Dans l'exemple de Figure 2, on devrait remplacer un ensemble de 5 tables associatives distinctes créés par `LCMDID_C`

```
char HDIR[13]
PyObject *iplist,*kplist ;
int i;
HDIR[12] = '\0';
```

```
for(i=0;i<5;i++) {
  (void)sprintf(HDIR,"GROUP%3d/  5",i+1);
  kplist=LCMSIX_C(&iplist,HDIR);
    LCMPPD_C(&kplist,...);
    ...;
}
```

par une liste de 5 tables associatives:

```
PyObject *iplist,*jplist,*kplist;
long n=5 ;
jplist=LCMLID_C(&iplist,'GROUP',n);
for(i=0;i<5;i++) {
 kplist=LCMDIL_C(&jplist,i);
 LCMPPD_C(&kplist,...);
}
```

La capacité d'inclusion de tables associatives dans une liste est implémentée à travers des appels à la fonction `LCMDIL_C`:

```
LCMDIL_C(iplist,iset);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la liste mère.
`iset`	*long*	position dans la liste mère de la table associative fille. Le premier élément de la liste mère est situé à l'index 0.

valeur de la fonction:	
*lcm**	adresse de la table associative fille.

5.4.5 LCMGID_C

Fonction utilisée pour récupérer l'adresse d'une table associative ou d'une liste stockée dans une table associative mère.

```
LCMGID_C(iplist,namp);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse d'une table associative mère.
`namp`	*char**	nom de la table associative fille.

valeur de la fonction:	
*lcm**	adresse de la table associative fille ou de la liste.

5.4.6 LCMGIL_C

Fonction utilisée pour récupérer l'adresse d'une table associative ou d'une liste stockée dans une liste mère.

```
LCMGIL_C(iplist,iset);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la liste mère.
`iset`	*long*	position dans la liste mère de la table associative fille. Le premier élément de la liste mère est situé à l'index 0.

valeur de la fonction:	
*lcm**	adresse de la table associative fille ou de la liste.

5.4.7 LCMSIX_C

Fonction utilisée pour se déplacer dans la structure arborescente d'une table associative ou pour changer de répertoire actif, sans se préoccuper de conserver les adresses des différents niveaux hiérarchiques. Si l'objet LCM est ouvert en mode `read-only`, un déplacement dans une table associative inexistante ne peut être effectué.

```
LCMSIX_C(iplist,namp,iact);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative avant l'exécution de `LCMSIX_C`.
`namp`	*char***	nom de la table associative fille si `iact=1`. Ce paramètre n'est pas utilisé si `iact=0` ou `iact=2`.
`iact`	*long*	type du déplacement. =0 revient à la racine de l'objet LCM; =1 va à la table associative fille (la créer si elle n'existe pas déjà); =2 revient à la table associative mère.

paramètre de sortie:		
`iplist`	lcm**	adresse de la table associative après exécution de `LCMSIX_C`.
valeur de la fonction:		
void		

5.5 Utilitaires

5.5.1 LCMLIB_C

Commande utilisée pour imprimer le contenu d'une table associative ou d'une liste (mémoire ou fichier XSM).

```
LCMLIB_C(iplist);
```

paramètre d'entrée:		
`iplist`	*lcm***	adresse de la table associative ou de la liste.

valeur de la fonction:	
void	

5.5.2 LCMEQU_C

Fonction utilisée pour copier l'information contenue dans une table associative (pointé(e) par `iplis1`) et ses filles vers la table associative pointé(e) par `iplis2`. `iplis1` et `iplis2` peuvent être stockés en mémoire ou sur fichier XSM. Notez que la seconde table associative (ou fichier XSM) est modifiée mais pas créée par `LCMEQU_C`.

```
LCMEQU_C(iplis1,iplis2);
```

paramètre d'entrée:		
`iplis1`	*lcm***	adresse de la table associative ou de la liste existante (accédée en mode `read-only`).

paramètre de sortie:		
`iplis2`	*lcm***	adresse de la table associative ou de la liste qui sera modifiée par `LCMEQU`.
valeur de la fonction:		
void		

5.5.3 LCMEXP_C

Fonction utilisée pour exporter (importer) le contenu d'une table associative (mémoire ou fichier XSM) vers un (d'un) fichier séquentiel binaire ou ascii en utilisant la méthode des contours. L'exportation commence du répertoire actif. Il s'agit d'un algorithme de sérialisation.

```
LCMEXP_C(iplist,impx,file,imode,idir);
```

paramètres d'entrées:		
`iplist`	*lcm***	adresse de la table associative ou de la liste qui doit être exportée (ou importée).
`impx`	*long*	paramètre d'impression (positionné à zéro pour aucune impression).
`file`	*FILE**	fichier séquentiel C où l'exportation s'effectue (ou d'où l'importation est réalisée).
`imode`	*long*	=1 pour fichier séquentiel BINAIRE; =2 pour fichier séquentiel ASCII.
`idir`	*long*	=1 pour exporter; =2 pour importer.

valeur de la fonction:	
void	

5.6 Traitement des vecteurs de variables caractères

Les fonctions suivantes ont été écrites en C à partir des API LCM précédentes afin de faciliter le traitement des vecteurs de variables caractères (strings C). Contrairement à la version Fortran, cette version écrite en C permet l'utilisation de composantes de longueur variables.

		type d'opération	
		put	get
mère	table associative	`LCMPCD_C`	`LCMGCD_C`
	liste	`LCMPCL_C`	`LCMGCL_C`

5.6.1 LCMGCD_C

Fonction utilisée pour récupérer un vecteur de variables caractères dans une table associative (mémoire ou fichier XSM) et le copier en mémoire.

```
LCMGCD_C(iplist,namp,hdata);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du vecteur de variables caractères à récupérer. Un appel à `XABORT` est déclenché si le bloc n'existe pas.

paramètre de sortie:		
`hdata`	*char***	vecteur de dimension $\geq$ `ilong` dans lequel le vecteur de variables caractères a été copié. L'espace mémoire permettant de copier chaque variable caractère est alloué par `LCMGCD_C`.
valeur de la fonction:		
void		

5.6.2 LCMPCD_C

Commande utilisée pour stocker un vecteur de variables caractères dans une table associative (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la table associative. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
LCMPCD_C(iplist,namp,ilong,hdata);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la table associative.
`namp`	*char**	nom du vecteur de variables caractères.
`ilong`	*long*	dimension du vecteur de variables caractères.
`hdata`	*char***	vecteur de dimension $\geq$ `ilong` duquel le vecteur de variables caractères est copié.

valeur de la fonction:	
void	

Exemple d'utilisation:

```
lcm *iplist;
long i;
long ilong = 5;
char *hdata1[ilong],*hdata2[ilong];

hdata1[0] = "string1";
hdata1[1] = "  string2";
hdata1[2] = "    string3";
hdata1[3] = "      string4";
hdata1[4] = "        string5";
for (i=0;i<ilong;i++) {
  printf("i=%d string='%s' size=%d\n",i,hdata1[i],strlen(hdata1[i]));
}

LCMOP_C(&iplist,"mon_dict",0,1,2);

/* Stockage de l'information */
LCMPCD_C(&iplist,"node1",ilong,hdata1);

/* Recuperation de l'information */
LCMGCD_C(&iplist,"node1",hdata2);

for (i=0;i<ilong;i++) {
  printf("in table i=%d string='%s' size=%d\n",i,hdata2[i],strlen(hdata2[i]));
}
for (i=0;i<ilong;i++) free(hdata2[i]);
LCMCL_C(&iplist,2);
```

5.6.3 LCMGCL_C

Fonction utilisée pour récupérer un vecteur de variables caractères dans une liste (mémoire ou fichier XSM) et le copier en mémoire.

```
LCMGCL_C(iplist,namp,hdata);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la liste.
`iset`	*long*	indice du vecteur de variables caractères dans la liste. Un appel à `XABORT` est déclenché si le bloc n'existe pas. Le premier élément de la liste est situé à l'index 0.

paramètre de sortie:		
`hdata`	*char***	vecteur de dimension $\geq$ `ilong` dans lequel le vecteur de variables caractères a été copié. L'espace mémoire permettant de copier chaque variable caractère est alloué par `LCMGCL_C`.
valeur de la fonction:		
void		

5.6.4 LCMPCL_C

Fonction utilisée pour stocker un vecteur de variables caractères dans une liste (mémoire ou fichier XSM). L'information est copiée de la mémoire vers la liste. Si le bloc existe déjà, il est remplacé; sinon, il est créé. Cette opération ne peut être effectuée dans un objet LCM ouvert en mode `read-only`.

```
LCMPCL_C(iplist,iset,ilong,hdata);
```

paramètres d'entrée:		
`iplist`	*lcm***	adresse de la liste.
`iset`	*long*	indice du vecteur de variables caractères dans la liste. Le premier élément de la liste est situé à l'index 0.
`ilong`	*long*	dimension du vecteur de variables caractères.
`hdata`	*char***	vecteur de dimension $\geq$ `ilong` duquel le vecteur de variables caractères est copié.

valeur de la fonction:	
void	

Exemple d'utilisation:

```
lcm *iplist, *jplist;
long i;
long ilong = 5;
char *hdata1[ilong],*hdata2[ilong];

hdata1[0] = "string1";
hdata1[1] = "  string2";
hdata1[2] = "    string3";
hdata1[3] = "      string4";
hdata1[4] = "        string5";
for (i=0;i<ilong;i++) {
  printf("i=%d string='%s' size=%d\n",i,hdata1[i],strlen(hdata1[i]));
}

LCMOP_C(&iplist,"mon_dict",0,1,2);

/* Creation de la liste imbriquee */
jplist = LCMLID_C(&iplist,"node2",77);

/* Stockage de l'information */
LCMPCL_C(&jplist,4,ilong,hdata1);
```

```
/* Recuperation de l'information */
LCMGCL_C(&jplist,4,hdata2);

for (i=0;i<ilong;i++) {
  printf("in list i=%d string='%s' size=%d\n",i,hdata2[i],strlen(hdata2[i]));
}
for (i=0;i<ilong;i++) free(hdata2[i]);

LCMCL_C(&iplist,2);
```

5.7 Allocation dynamique des blocs d'information élémentaires

5.7.1 SETARA_C

Fonction utilisée pour allouer une zone mémoire destinée à contenir un bloc d'information élémentaire. Elle retourne l'adresse à partir de laquelle on pourra stocker un bloc d'information de longueur `ilong` mots (de type *long*). La fonction `SETARA_C` *doit être utilisée* pour stocker les blocs d'information qui seront par la suite incorporés à l'objet LCM par les fonctions `LCMPPD_C` ou `LCMPPL_C`. Si `LCMPPD_C` ou `LCMPPL_C` ne sont pas utilisés, la libération ultérieure de cette zone mémoire devra impérativement être effectuée par le sous-programme `RLSARA` en Fortran ou par la fonction `RLSARA_C` en C. Si le système ne peut allouer `ilong` mots, le programme s'arrête.

```
SETARA_C(ilong);
```

paramètre d'entrée:		
`ilong`	*long*	longueur en mots de la zone mémoire.

valeur de la fonction:	
*long**	adresse de l'information.

5.7.2 RLSARA_C

Fonction utilisée pour libérer une zone mémoire préalablement allouée par le sous-programme Fortran `SETARA` ou la fonction C `SETARA_C`. `RLSARA_C` utilise la fonction **free** sur les systèmes UNIX. Si le système ne peut libérer la zone mémoire, le programme s'arrête.

```
RLSARA_C(iofset);
```

paramètre d'entrée:		
`iofset`	*long**	adresse de la zone mémoire à libérer.
valeur de la fonction:		
void		

References

[1] A. Hébert and R. Roy, "A Programmer's Guide for the GAN Generalized Driver, FORTRAN-77 version", Report IGE-158, Institut de Génie Nucléaire, École Polytechnique de Montréal, December 1994.

[2] G. van Rossum, "Python/C API Reference Manual",Corporation for National Research Initiatives, Reston, Va, http://www.python.org, July 1999.

[3] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin and T. Oliphant, "Numerical Python", Lawrence Livermore National Laboratory, Report UCRL–MA–128569, 1999.

[4] A. Hébert et C. Carémoli, "DESCARTES: Description de l'interface Python de gestion des structures de données et de commande des modules de calcul", CEA-DRN-SERMA/LENR/RT/00-2842, EDF-R&D-HI-76/00/007A, Novembre 2000.