Un standard pour une representation d'objets de type liste dans le domaine du traitement d'image

Philippe Garnesson, Gérard Giraudon

▶ **To cite this version:**

Philippe Garnesson, Gérard Giraudon. Un standard pour une representation d'objets de type liste dans le domaine du traitement d'image. [Rapport de recherche] RT-0105, INRIA. 1988, pp.65. inria-00070061

HAL Id: inria-00070061
https://inria.hal.science/inria-00070061

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

N° 105

Programme 6

UN STANDARD POUR UNE REPRESENTATION D'OBJETS DE TYPE LISTE DANS LE DOMAINE DU TRAITEMENT D'IMAGE

Philippe GARNESSON
Gérard GIRAUDON

Décembre 1988

RT_0105

UN STANDARD POUR UNE REPRESENTATION D'OBJETS DE TYPE LISTE DANS LE DOMAINE DU TRAITEMENT D'IMAGE

A STANDARD FOR A LIST-TYPE OBJECT REPRESENTATION IN COMPUTER VISION

MANUEL D'UTILISATION

USER'S GUIDE

VERSION III

Philippe Garnesson Gérard Giraudon

INRIA Sophia Antipolis
Route des Lucioles
06560 Valbonne

Décembre 1988

Résumé

La vision par ordinateur nécessite de manipuler de nombreux attributs d'image, tels que chaînes, segments ou régions.
Ce rapport montre qu'il est possible de définir une structure de représentation commune à ces différents types d'objets. Cette unification, nous a conduit à développer un standard de programmation qui facilite les échanges entre la mémoire centrale et la mémoire secondaire. Cette représentation en mémoire secondaire se veut indépendante de l'utilisation qui est faite en mémoire centrale des différents objets.

Une première version de ce standard a donné lieu en Mars 1987 à l'édition du rapport technique INRIA numéro 82. Ce standard ayant été largement adopté, de nouveaux besoins sont apparus et avec eux la version II. Cette version n'a pas été diffusée, mais a permis de définir les bases de la version III qui se veut plus *professionnelle*.

La version III reprend les fonctionnalités de la version I et offre à l'utilisateur une nouvelle gamme de fonctions.
Cela signifie que le passage de la version I à la version III nécessite simplement de recompiler les sources.
Quant à la nouvelle gamme de fonctions, *(les fonctions généralisées)*, elle permet de développer des applications communes à toutes les structures d'objets représentées par ce standard. On peut citer comme exemple un outil capable de visualiser n'importe quel type de fichier de ce standard.

Abstract

Computer vision requires the manipulation of numerous image structures such as chaines, segments or regions. This report shows that it is possible to define a common representation for these different types of objects. This unification led us to develop a programming standard which facilitates IO between central and peripheral memory. The representation in peripheral memory was designed to be independent of the usage made of the different objects in central memory.

A first version of this standard was described in INRIA technical report number 82 in March 1987. Since the adoption of this standard by a large number of users, new needs have appeared and with them version II. This version was never made public, but served as the basis for the more professional version III.

Version III includes all functions of version I and offers the user a new series of capabilities. This implies that updating from version I to version III requires only source code recompilation. The new functions, called "generalized functions," permit the development of applications common to all of the structures represented by the standard. An example would be a tool capable of displaying any of the standard's file types.

Table des matières

1 Présentation

1.1 Pourquoi ce standard

Ce standard a été développé pour répondre à des besoins qui appartiennent au domaine du traitement d'images. En fait, il peut également s'adresser à d'autres domaines.

L'objectif auquel on souhaite répondre est:

Disposer d'un gestionnaire de fichier suffisamment souple pour manipuler les différents types d'informations de notre domaine.

Les intérêts de ce gestionnaire sont multiples:

- Disposer d'un outil fiable et optimisé.
- Pouvoir facilement stocker et relire l'information.
- Etre capable de développer des applications communes à tous les fichiers de ce standard.
  Dans le domaine du traitement d'image on peut citer l'exemple d'un outil de visualisation.

1.2 Les problèmes de sa mise en oeuvre

Le problème qui se pose vient du fait que ces différents types d'informations ne rentrent pas dans un "cadre fixe". On peut citer trois raisons essentielles:

- Ces informations sont de natures différentes. On veut manipuler des régions, des contours, des chaînes, des segments des figures géométriques, et cette liste n'est pas exhaustive.
- Le codage de l'information peut être variable. Certaines applications ont besoin de manipuler des entiers, d'autres applications des réels.
- Il doit y avoir indépendance entre la représentation sur disque et en mémoire centrale. En effet suivant le type de traitement à effectuer, l'organisation des données en mémoire centrale est différente.

1.3 Les solutions retenues

1.3.1 Une structure de représentation commune

L'étude des différents objets qui nous intéressent montre qu'on est capable de dégager deux principes qui permettent de caractériser leur organisation.

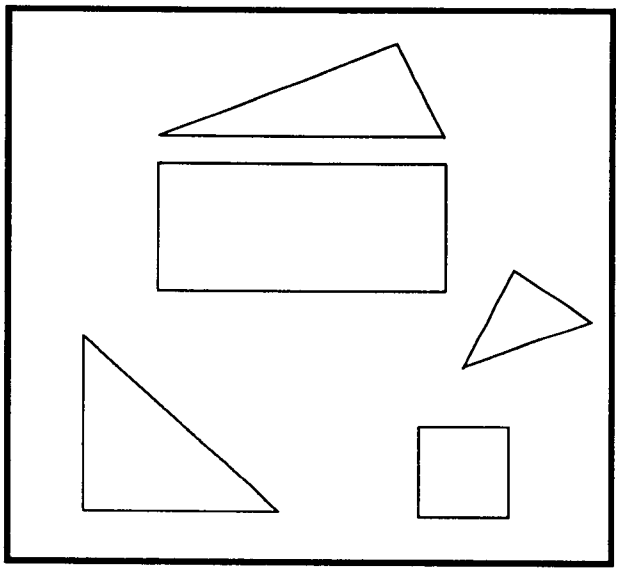
1. L'information qu'on veut manipuler est en fait un ensemble (ou une liste) d'objets de même nature. Les objets sont soient des segments, soient des chaînes, soient des régions ...
   Ces objets sont appelés **éléments.**

2. Chacun des **eléments** est composé lui même d'un ensemble (ou d'une liste) d'un autre type d'objets appelés **maillons.** Ces maillons sont souvent dans le domaine du traitement de l'image des "pixels". Néanmoins, la seule chose qu'impose le standard, c'est l'existence de ces **maillons.**

Ces deux remarques vont nous permettre d'unifier tous les objets dans une structure de représentation commune.

Nous allons développer ces notions sur un exemple précis.

Dans cet exemple on souhaite représenter les contours des objets de l'image suivante:



Dans cette exemple les **éléments** *sont les contours de ces différents objets. On a ici cinq* **éléments**: *un carré, un rectangle et trois triangles.*
On peut choisir comme **maillons** *les sommets de chacun de ces contours. Le carré à donc 4* **maillons**, *les triangles 3 et le rectangle 4.*
Cet exemple répond donc bien aux contraintes concernant l'organisation imposée par le standard.

1.3.2 Une gestion dynamique de l'information

L'originalité de ce standard provient du fait que la définition de ce qu'est un **élément** et ce qu'est un **maillon** est faite "dynamiquement" par l'utilisateur. Le standard adapte son

comportement en fonction de cette description.
La description de l'information ce fait en trois étapes:

- On doit décrire ce qu'est un **maillon.**
- On doit décrire ce qu'est un **élément.**
- On peut également décrire des informations qui sont relatives à l'ensemble des éléments.
  Cette information est appelée **entête.**

Dans l'exemple précédent, ces informations pourraient être:

- *Pour un* **maillon.**
  - *Ses coordonnées*
  - *Sa couleur.*
  - *Un secteur angulaire.*
- *Pour un* **élément.**
  - *Sa nature, est-ce un triangle, un rectangle ?*
  - *Sa surface.*
  - *Le nombre de maillons.*
- *Pour l'***entête.**
  - *Le nombre d'***éléments** *(ici il y en a 5).*
  - *La dimension de l'image.*
  - *Quel est l'***élément** *dont la surface est la plus grande ?*

En fait, certaines informations sont imposées par le standard.

- Pour les maillons, rien n'est imposé.
- Pour les éléments:
  - Chaque élément doit avoir un numéro, ce numéro pourra servir de clef d'accès.
  - A chaque élément est associé un nombre de maillons.
- Pour l'entête:
  - Le nombre total d'éléments dans le fichier.
  - Un nom logique, dans l'exemple on pourrait par exemple choisir "exemple 1"

  D'un point de vue pratique, l'utilisateur doit construire trois **descripteurs** pour caractériser les informations **variables.** Un descripteur pour l'entête, un descripteur pour les éléments et un descripteur pour les maillons.

  Un **descripteur** permet de décrire les différents champs élémentaires qui caractérisent l'information, chacun de ces champs élementaires étant appelé **zone.**
  Pour chaque zone, l'utilisateur indique trois types d'informations :

- Le type de codage qu'il veut utiliser (entier, flottant etc.).
- Un commentaire qui est une étiquette logique représentant la zone.
- Un facteur de répétition.

Par la suite on utilisera l'abréviation **DZV** pour désigner un **Descripteur** de **Zones Variables.**
Pour un fichier donné, on distingue 3 DZV, celui correspondant à l'entête, celui correspondant aux éléments et celui correspondant aux maillons.

Par la suite, on désignera également le standard sous ce nom.

1.4 Les fonctionnalités du standard

La manipulation de l'information se fait par l'intermédiare de primitives écrites en langage C.
On distingue trois mode d'accès : la création, la lecture et la modification. Deux autres fonctionnalités également importantes du standard sont la détection d'erreur et la portabilité.

1.4.1 L'écriture

- Ecriture séquentielle des fichiers. Les fichiers sont alors *pipables* au sens Unix. Par la suite on englobera également dans la notion de *pipe* la notion de redirection des entrées et sorties standards.

1.4.2 La lecture

- Lecture séquentielle des fichiers. Les fichiers sont alors *pipables* au sens Unix.
- Accès direct. On peut faire de l'accès direct au niveau de l'élément et au niveau du maillon. Cet accès direct est réalisé sans la mise en oeuvre d'un fichier "d'index". On évite ainsi les problèmes de cohérence.
- Il existe un mode de lecture qui fonctionne quelque soit le fichier. On peut ainsi développer des applications communes à tous les fichiers du DZV.

1.4.3 La modification

Les modes lecture et écriture peuvent être utilisés simultanément. Néanmoins, il est à noter que ce mode d'accès est coûteux en temps et donc à utiliser avec modération.

- On peut ajouter des éléments à un fichier existant.
- On peut supprimer un élément.
- Il est interdit de modifier le nombre de maillons d'un élément. On peut néanmoins contourner ce problème en le détruisant puis en le créant à nouveau avec un nombre de maillons différents.

1.4.4 Détection d'erreur

La librairie comporte un module qui contrôle la détection d'erreur. On distingue trois types d'erreurs:

1. La validité des paramètres.
2. Un enchaînement incohérent d'appels à la librairie.
3. La détection de fichier incohérent (en cas de fichier non fermé par exemple)

En cas d'erreur, une fonction est automatiquement déclenchée et affiche un message explicitant l'erreur. Cette fonction peut être redéfinie par l'utilisateur.

1.4.5 Portabilité des fichiers du standard

Pour coder les informations élémentaires (entier, flottant etc.), on utilise la même codification que la machine hôte. Cela signifie que les fichiers sont dépendants du type de cette machine et peuvent donc poser des problèmes de transfert vers une autre machine.
Néanmoins, pour la plupart des machines seul le codage des flottants est différent. Pour ne pas être dépendant de cette contrainte, on a écrit un module de conversion de flottant intégré au standard. Ce module est automatiquement appelé si nécessaire.

2 Description des primitives

2.1 Mécanismes généraux, l'enchaînement des primitives

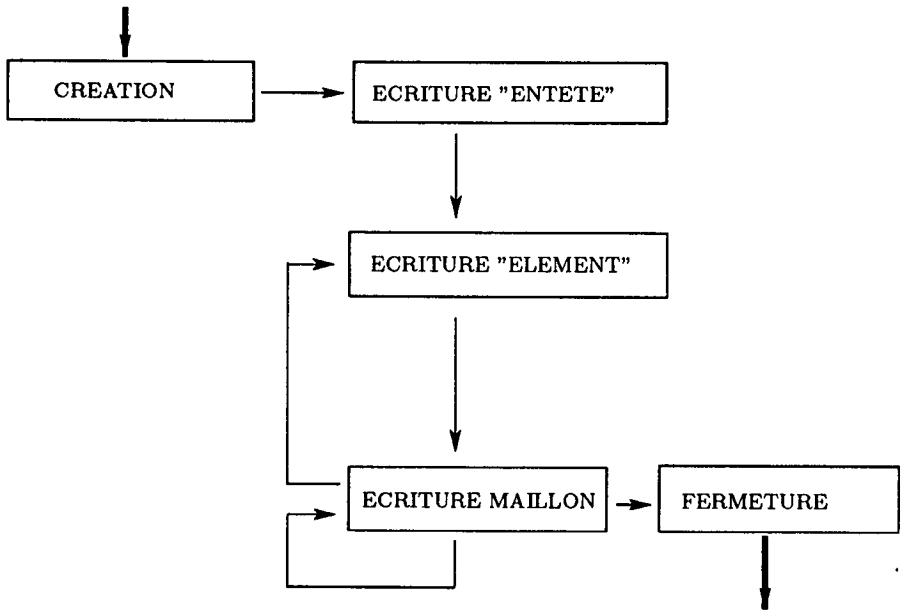
Ce paragraphe a pour objectif de mettre en évidence l'enchaînement normal de l'appel des primitives du standard.
On se contente de présenter les notions les plus importantes. La compréhension de ces notions facilitera l'utilisation de l'annexe concernant les exemples.

2.1.1 Création d'un fichier

On distingue 5 étapes:

1. La création: l'utilisateur décrit son fichier (création des DZV) et demande sa création.
2. Ecriture de l'entête: l'utilisateur doit demander l'écriture des zones variables concernant l'entête du fichier.
3. Ecriture de l'élément: l'utilisateur doit demander l'écriture des zones concernant l'élément (son numéro, le nombre de maillons qui lui est associé ...).
4. Ecriture du maillon: l'utilisateur en fonction du nombre "n" de maillons de l'élément, doit demander à écrire les "n" maillons.
5. La fermeture: elle est obligatoire.

Schéma récapitulatif:

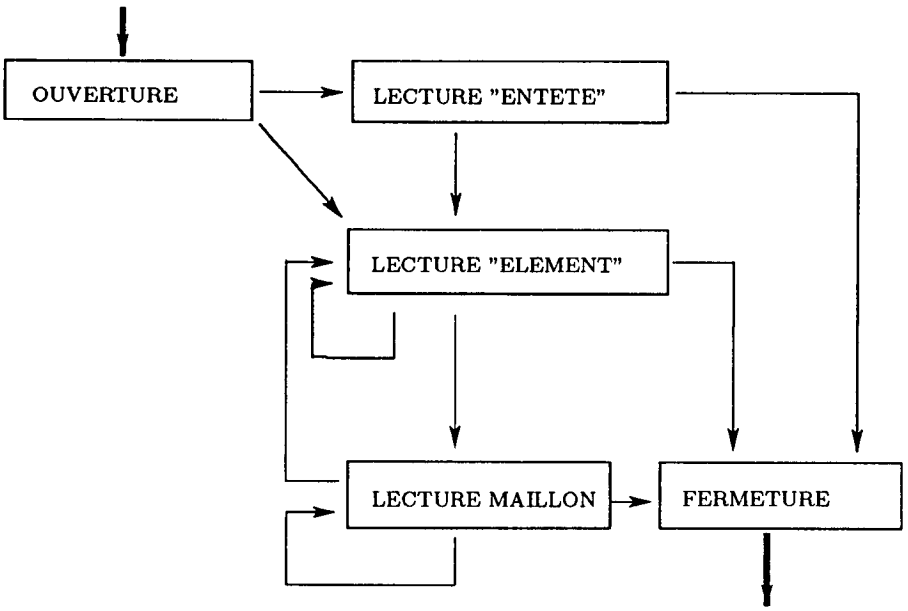


2.1.2 Lecture d'un fichier

On distingue 5 étapes:

1. L'ouverture: l'utilisateur demande l'ouverture de son fichier, il peut récupérer la description qui en a été faite lors de la création.
2. Lecture de l'entête: l'utilisateur peut demander la lecture des zones variables concernant le fichier, c'est à dire l'ensemble des éléments. Cette étape est facultative.
3. Lecture de l'élément: l'utilisateur doit demander la lecture des zones concernant un élément (son numéro, le nombre de maillons qui lui est associé ...).
4. Lecture du maillon: l'utilisateur en fonction du nombre "n" de maillons de l'elément, peut demander la lecture des "n" maillons. Cette étape est facultative.
5. La fermeture: elle est obligatoire.

Schéma récapitulatif:

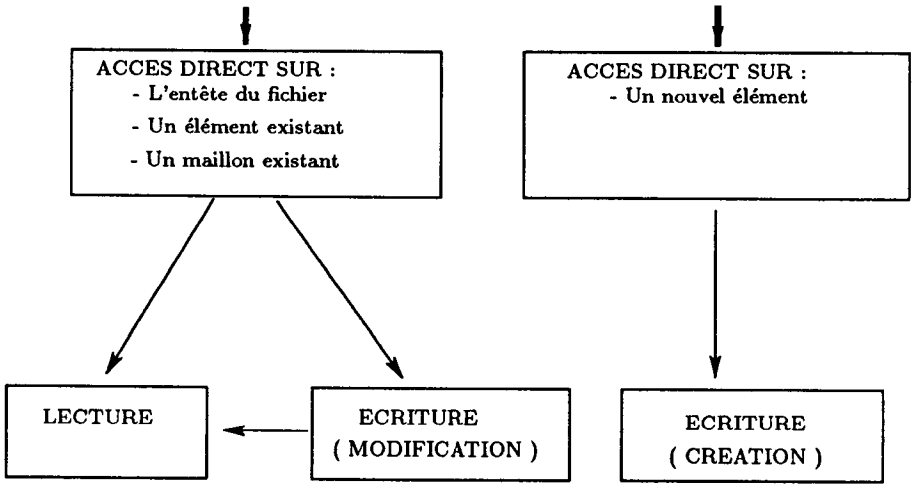


2.2 L'accès direct

Avant de faire une lecture ou une écriture, on peut demander à faire un accès direct. Cet accès s'applique soit sur les zones variables concernant le fichier, soit sur un élément (en fonction de son numéro), soit sur un maillon (en fonction de sa position dans la liste des maillons).

Quant on demande un accès sur un objet existant on peut alors demander sa lecture ou sa modification, quant l'objet est inexistant on peut demander sa création.
Une modification ne peut être faite qu'après un accès direct. Après une modification on ne peut pas demander une écriture à moins de refaire un accès direct.

Schéma récapitulatif:



2.3 Ce qu'il est indispensable de savoir

1. La librairie DZV peut manipuler simultanément un certain nombre de fichiers. Pour ce, la plupart des fonctions prennent pour premier argument un identificateur de fichier. Cet identificateur est un entier (au sens C) et sa valeur est retournée soit par la fonction d'ouverture soit par la fonction de création de fichier.
2. Les valeurs de retour des fonctions:
   Les fonctions dont la valeur de retour est significative sont soulignées. Quand on a un double soulignement, cela signifie que la valeur est l'adresse d'un objet alloué par la fonction.
3. Les arguments des fonctions:
   Quand on indique le type d'un argument, on indique le type tel qu'il est déclaré dans la fonction et non pas tel que doit le déclarer l'utilisateur.
   Les arguments de sortie sont soulignés. Quand on a un double soulignement, cela signifie que l'argument est l'adresse d'un objet alloué par la fonction.
4. En cas de problèmes, on peut trouver en annexe des exemples qui illustrent l'utilisation de chacune des fonctions du standard.

2.4 Le nom des fonctions : une notation à connaître

Tous les noms des fonctions du standard sont construits sur le même modèle :

la catégorie _ l'action [le type d'action] _ [le domaine d'application]

2.4.1 La catégorie

On distingue deux catégories de fonctions:

1. Les fonctions *généralisées* : ce sont des fonctions qui sont indépendantes du type du fichier utilisé. On peut citer par exemple l'ouverture d'un fichier. Ces fonctions ont un nombre de paramètres fixes, excepté pour la fonction G_set_dzv. Toutes ses fonctions sont préfixées par les lettres **G_**.
2. Les fonctions spécialisées : Les fonctions de cette catégorie sont complètement dépendantes du type du fichier utilisé. Elles ont un nombre de paramètres qui dépend des DZV du fichier.
   Toutes ses fonctions sont préfixées par les lettres **S_**.

2.4.2 L'action

On distingue deux types d'actions :

1. Les actions en mémoire centrale : Ces actions se contentent de lire ou modifier des informations en mémoire centrale. Elles n'ont aucun effet direct sur le fichier. La liste de ces actions est : get, cmp, set et copy.
2. Les actions modifiant le fichier. Ces actions ont pour but de lire ou écrire de l'information sur un fichier. Elles ont pour effet secondaire de modifier le **pointeur** sur le fichier.
   Cela signifie qu'en cas de création ou lecture séquentielle l'utilisateur n'a pas à se soucier de ce pointeur. En revanche, en cas **d'accès direct** ou de **modification** l'utilisateur doit modifier ce pointeur en utilisant une des fonctions **G_seek_fic**, **G_seek_elt** ou **G_seek_mai**.
   La liste de ces actions est : open, create, close, read, write, fread, fwrite et seek.

2.4.3 Le type d'action

Ce champ n'est présent que pour l'action **get**. Il spécifie le type de l'action.
La liste des types d'actions est : nb, max, min, first, last, version, error_type, error_mes, context et fic_name.

2.4.4 Le domaine d'application

On distingue trois domaines d'applications :

1. Les informations variables concernant : l'entête, les éléments et les maillons. Les notations utilisées pour chacun de ces groupes sont **fic**, **elt** et **mai**.
2. Les DZV :
   On utilise comme notation **dzv**.
3. Les autres : certaines fonctions ne rentrent dans aucune de ces classes, pour ces dernières on ne précise pas le domaine d'application.

2.5 Les fonctions généralisées

2.5.1 Création d'un descripteur de zones variable

Cette fonction permet de créer un objet de type DZV. Un DZV est un objet qui décrit les informations que l'utilisateur veut associer soit à l'entête du fichier, soit aux éléments ou soit aux maillons.

```
DZV *G_set_dzv(  nb_triplet,
                 ... , nb_i, format_i, commentaire_i, ...)
```

- **int nb_triplet;**
  Nombre de zones dans le DZV, c'est aussi le nombre de triplets d'arguments qui suivent.
- **int nb_i;**
  Facteur de répétition concernant la zone i du DZV.
  Ce facteur a de fortes implications sur l'utilisation des fonctions spécialisées. Ce facteur est souvent utilisé pour manipuler des pseudo chaînes de caractères.
- **char *format_i;**
  Le format permet de préciser quel est le type utilisé pour coder l'information. Ce type est un des types élémentaires du langage C, la codification utilisée est la suivante :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char"
"%d"	pour un "double"	"%l"	pour un "long"

  Si on indique **2, "%i"** alors l'utilisateur devra passer comme argument aux fonctions de lecture ou d'écriture un tableau de 2 entiers.

- **char *comment_i;**
  Signification de la zone i. Si on veut pouvoir exploiter par la suite cet argument il est nécessaire de donner une chaîne de caractères qui contienne autant de mots que le spécifie nb_i, chacun séparé par un blanc. Ces mots doivent tous être différents.

Exemple : Voir annexe A.1.

2.5.2 Comparaison de deux DZV

Cette fonction permet de comparer deux objets de type DZV. Elle retourne **0** s'ils sont différents, **1** sinon. Deux DZV sont différents s'ils ont été créés avec des arguments différents. Les arguments concernant les commentaires ne sont pas significatifs. Au cas où cette fonction ne donne pas suffisamment d'information, on peut utiliser la fonction **G_get_dzv**.

int G_cmp_dzv(dzv1, dzv2)

- **DZV *dzv1;**
  Premier argument de la comparaison.
- **DZV *dzv2;**
  Deuxième argument de la comparaison.

Exemple : Voir annexe A.6.

2.5.3 Utiliser le contenu d'un DZV

Quand on ouvre un fichier, on récupère des objets de type DZV. L'utilisateur n'a pas directement accès aux champs de ces objets. S'il veut pouvoir les manipuler, il doit appeler la fonction **G_get_dzv**.
Attention, cette fonction se charge d'allouer les tableaux **nb, format, commentaire.**

int G_get_dzv(dzv, nb_triplet, nb, format, commentaire)

- **DZV *dzv;**
  Adresse du DZV.
- **int *nb_triplet;**
  On récupère le nombre de triplet du DZV. Les tableaux nb, format et commentaire vont permettre d'accéder à ces triplets.
- **int **nb;**
  (*nb)[i] est un entier qui représente le facteur de répétition de la zone i. Attention, pour utiliser la zone 1 on référence nb[1].
  Si l'utilisateur ne veut pas prendre connaissance de cette information, il peut passer comme argument la constante NULL.
- **char ***format;**
  (*format)[i] est une chaîne de caractères qui représente le format de la zone i. On récupère le format qui a été spécifié lors de la création du DZV. Ce format est une chaîne de caractères de la forme :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char"
"%d"	pour un "double"	"%l"	pour un "long"

  Attention, pour utiliser la zone 1 on référence format[1].
  Si l'utilisateur ne veut pas prendre connaissance de cette information, il peut passer comme argument la constante NULL.
- **int ***commentaire;**
  (*commentaire)[i] est une chaîne de caractères qui représente le commentaire de la zone i. Attention, pour utiliser la zone 1 on référence commentaire[1].
  Si l'utilisateur ne veut pas prendre connaissance de cette information, il peut passer comme argument la constante NULL.

Exemple : Voir annexe A.6.

2.5.4 Création d'un fichier

Cette fonction permet de créer un fichier. Si un fichier de même nom existe, ce dernier est écrasé.
Elle retourne un entier qui permettra d'identifier le fichier.

int G_create(nom, type, nb_elt, dzv_fic, dzv_elt, dzv_mai)

- **char *nom;**
  Nom du fichier. Un nom de fichier défini par ">" indique que le fichier doit être écrit sur le *"stdout"*. Dans ce cas, l'accès direct sur le fichier est interdit.
- **char *type;**
  Aucun contrôle n'est fait sur cet argument. Son objectif est de permettre à l'utilisateur d'identifier rapidement le type du fichier. La chaîne de caractère est limitée à 32 caractères, y compris le délimiteur de fin.
- **int nb_elt**
  L'argument *nb_elt* peut être différent du nombre exact des éléments qu'on va écrire dans le fichier.
  Il a pour but d'optimiser la gestion de la table permettant les accès directs.
  Néanmoins, c'est ce nombre qu'on recupèrera dans la primitive G_open() en cas d'utilisation d'un pipe. Dans les autres cas, le nombre réel d'éléments dans le fichier est automatiquement mis à jour.
- **DZV *dzv_fic, *dzv_elt, *dzv_mai**
  Ces arguments représentent les DZV relatifs à l'entête, aux éléments et aux maillons. Ces DZV doivent avoir été préalablement initialisés par la fonction G_set_dzv ou G_open.

Exemple : Voir annexe A.1.

2.5.5 Ouverture d'un fichier

Cette fonction permet d'ouvrir un fichier existant. Elle retourne un entier qui permettra d'identifier le fichier par la suite. Cette fonction provoque une erreur si le fichier n'est pas cohérent (un fichier peut être incohérent s'il n'a jamais été fermé avec la fonction G_close).

int G_open(nom, type, nb_elt, dzv_fic, dzv_elt, dzv_mai)

- **char *nom;**
  Nom du fichier à ouvrir.
  Un nom de fichier défini par "<" indique que le fichier doit être lu sur le *"stdin"*. Dans ce cas, l'accès direct sur le fichier est interdit.
- **char **type;**
  Renvoie le type du fichier qui a été donné lors de la création. Attention, il faut fournir comme argument non pas l'adresse d'un tableau, mais l'adresse d'un pointeur. L'allocation est faite par la fonction d'ouverture.
- **int *nb_elt;**
  Permet de récupérer le nombre d'éléments du fichier.
  Dans un cas très particulier, il se peut que ce nombre ne soit pas exact. Ce cas correspond à la lecture en mode "pipe" d'un fichier lui même créé avec un "pipe". Dans ce contexte très particulier, on récupère la valeur qui a été indiquée lors de la création du fichier.
- **DZV **dzv_fic, **dzv_elt, **dzv_mai;**
  Renvoient les DZV relatifs à l'entête, aux éléments et aux maillons. Ces DZV doivent avoir été préalablement initialisés par la fonction G_set_dzv ou G_open. Attention, l'allocation est faite par la fonction d'ouverture.

Exemple : Voir annexe A.2.

2.5.6 Lecture du fichier

Les fonctions suivantes permettent de demander la lecture d'une partie du fichier. Elles ont pour effet d'initialiser une zone mémoire qui va pouvoir être exploitée par d'autres fonctions (G_get_???, G_copy_??? ou G_print_???).
Pour indiquer que tous les éléments ont été lus, ou que tous les maillons d'un élément ont été lus, **G_read_elt** et respectivement **G_read_mai** renvoient 1 quand la lecture a été possible et 0 sinon.

L'appel de ces fonctions est facultatif. Dans ce cas elles sont en fait automatiquement appelées. On préserve ainsi la possibilité d'exploiter le fichier en mode "pipe".

```
int   G_read_fic( fd )
int   G_read_elt( fd, numero, nb_maillons )
int   G_read_mai( fd )
```

- **int fd;**
  Identificateur du fichier.
- **int *numero;**
  On récupère le numéro de l'élément courant.
- **int *nb_maillons;**
  On récupère le nombre de maillons de l'élément courant.

Exemples : Voir annexe A.10, A.12.

2.5.7 Récupérer une valeur associée à un commentaire

Ces fonctions permettent de récupérer une ou plusieurs valeurs qui ont été mémorisées par le standard suite à un appel à G_read_fic , G_read_elt ou G_read_mai.
Pour sélectionner la ou les valeurs à lire, on doit indiquer le ou les commentaires qui ont été associés à ces valeurs lors de la création des DZV. On peut avoir connaissance de ces commentaires par l'intermédiaire de la fonction G_get_dzv.
Le fonctionnement de ces fonctions est similaire à la fonction *scanf* du langage C. Cela signifie qu'on doit indiquer un format dans lequel on veut les valeurs, et une liste d'adresses qui vont recevoir ces valeurs. Le format indiqué est indépendant du codage du fichier, il y a conversion automatique si nécessaire.
Cette fonction renvoie **1** si elle réussit à lire les valeurs demandées, et **0** sinon.

```
int  G_get_fic( fd, commentaire, format, ... vi ... )
int  G_get_elt( fd, commentaire, format, ... vi ... )
int  G_get_mai( fd, commentaire, format, ... vi ... )
```

- **int fd;**

- **char *commentaire;**
  Liste des commentaires associés aux valeurs qu'on veut lire. Les commentaires sont séparés par des espaces.
- **char *format;**
  Pour chacune des valeurs on doit indiquer le format élémentaire dans lequel on veut l'utiliser. Le format est une chaîne de caractères composée d'une liste de formats élémentaires.
  Les formats élémentaires possibles sont :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char"
"%d"	pour un "double"	"%l"	pour un "long"
"%g"	pour une "string"		

  L'utilisation de "%g" suppose que le champ correspondant du DZV a été déclaré de type "%c" avec un facteur de répétition qui caractérise la longueur de la chaîne.
  Exemple de formats : "%d" ou "%i%f%f".
- **??? *vi;**
  Pour chacune des valeurs on indique l'adresse de la variable qui doit recevoir le résultat.

Exemple : Voir annexe A.7, A.14.

2.5.8 Modifier une valeur associée à un commentaire

Ces fonctions permettent de modifier une ou plusieurs valeurs qui ont été mémorisées par le standard suite à un appel à G_read_??? ou G_copy_???.
Pour sélectionner la ou les valeurs à modifier, on doit indiquer le ou les commentaires qui ont été associés à ces valeurs lors de la création des DZV.
Le fonctionnement de ces fonctions est similaire à la fonction *printf* du langage C. Cela signifie qu'on doit indiquer un format dans lequel on veut utiliser les valeurs dans le programme, et une liste d'adresses correspondant aux variables du programme. Le format indiqué est indépendant du codage du fichier, il y a conversion automatique si nécessaire.
Cette fonction renvoie **1** si elle réussit a modifier les valeurs demandées, et **0** sinon. Attention, les modifications sont faites en mémoire centrale, pour être prise en compte, on doit utiliser les fonctions G_write_??? .

```
int  G_set_fic( fd, commentaire, format, ... vi ... )
int  G_set_elt( fd, commentaire, format, ... vi ... )
int  G_set_mai( fd, commentaire, format, ... vi ... )
```

- **int fd;**
  Identificateur du fichier.
- **char *commentaire;**
  Liste des commentaires associés aux valeurs qu'on veut affecter. Les commentaires sont séparés par des espaces.
- **char *format;**
  Pour chacune des valeurs on doit indiquer le format élémentaire dans lequel on veut l'utiliser. Le format est une chaîne de caractères composée d'une liste de formats élémentaires.
  Les formats élémentaires possibles sont :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char"
"%d"	pour un "double"	"%l"	pour un "long"
"%g"	pour une "string"		

  L'utilisation de "%g" suppose que le champ correspondant du DZV a été déclaré de type "%c" avec un facteur de répétition qui caractérise la longueur de la chaîne.
  Exemple de formats : "%d" ou "%i%f%f".
- **??? *vi;**
  Pour chacune des valeurs on indique l'adresse de la variable qui doit être prise pour nouvelle valeur.

Exemple : Voir annexe A.8, A.15.

2.5.9 Transfert d'information entre fichiers

Les fonctions suivantes permettent de transférer des valeurs d'un fichier vers un autre. Les deux fichiers doivent avoir exactement les mêmes DZV.
Ces fonctions travaillent en mémoire centrale. Pour modifier effectivement le fichier il faut ensuite utiliser G_write_fic, G_write_elt ou G_write_mai.

```
int   G_copy_fic( source, destination )
int   G_copy_elt( source, destination )
int   G_copy_mai( source, destination )
```

- **int source;**
  Identificateur du fichier source.
- **int destination;**
  Identificateur du fichier destination.

Exemple : Voir annexe A.10.

2.5.10 Ecriture

Les fonctions suivantes permettent de transférer une zone mémoire vers le fichier disque.

```
int   G_write_fic( fd )
int   G_write_elt( fd, numero, nb_maillons )
int   G_write_mai( fd )
```

- **int fd;**
  Identificateur du fichier.
- **int numero;**
  On indique le numéro de l'élément.
- **int nb_maillons;**
  On indique le nombre de maillons de l'élément courant.

Exemple : Voir annexe A.10.

2.5.11 Acquisition du numéro de version

Cette fonction permet de connaître la version qui a permis de créer le fichier. Cette fonction renvoie un résultat valide pour tous les fichiers créés depuis la version III. Quand le résultat de la fonction est valide, elle renvoie **1** et **0** sinon.

```
int   G_get_version( fd, version, release )
```

- **int fd;**
  Identificateur du fichier.
- **int *version;**
  Numéro de la version.
- **int *release;**
  Numéro de la release.

Exemple : Voir annexe A.11.

2.5.12 Acquisition d'information sur les numéros des éléments

On dispose de cinq fonctions qui permettent d'obtenir des informations sur le numéro des éléments du fichier.
Le résultat de ces fonctions est un entier dont la valeur est retournée par la fonction. Elles renvoient **-1** si le fichier ne contient pas d'élément.
Dans le cas particulier de l'ouverture ou création d'un fichier "en mode pipe", ces fonctions rendent un résultat relatif aux seuls éléments qui ont été lus.

1. Retourne le plus grand numéro d'élément du fichier fp.

```
int        G_get_max_elt( fd )
```

2. Retourne le plus petit numéro d'élément du fichier fp.

```
int        G_get_min_elt( fd )
```

3. Retourne le nombre d'élément du fichier.

```
int        G_get_nb_elt( fd )
```

4. Retourne le premier élément du fichier fp.

```
int        G_get_first_elt( fd )
```

5. Retourne le dernier élément du fichier fp.

```
int        G_get_last_elt( fd )
```

Exemple : Voir annexe A.11.

2.5.13 Les erreurs

L'utilisation de la librairie DZV peut engendrer un certain nombre d'erreurs.

- Les erreurs systèmes : on essaie d'ouvrir un fichier inexistant.
- Les erreurs d'utilisation : mauvais enchainement de commandes, etc.

De nombreuses erreurs sont détectées par le gestionnaire DZV. En cas de détection d'erreur, une fonction est appelée qui a pour but de localiser l'erreur et d'en expliquer la cause.
Pour la localisation, on affiche le nom du fichier, et les cinq dernières primitives appelées.
Pour expliquer l'erreur, on affiche un message et un numéro qui renvoie à une table dont l'objectif est de faciliter la compréhension de l'erreur. En cas d'erreur système, on fait en fait appel à la fonction < perror > du langage C.

Exemple de message possible en cours d'exécution:

```
*** ERREUR DZV *** /usr/garnesso/fichier ***
    contexte        : G_write_mai, G_read_elt , G_open.
    explication     : G_write_??? : impossible en ouverture , utiliser G_seek_elt..
    reference       : voir erreur 8.
```

La fonction qui est appelée par le système est définie ainsi :

```
dzv_fonction_erreur( fd, num )
        int    fd, num;
        {
        fprintf( stderr, "*** ERREUR DZV *** %s ***\n", G_get_fic_name( fd ) );
        fprintf( stderr, " contexte : %s\n", G_get_context( fd, 5));
        if (num == 1)
              perror( " explication ");
        else
              fprintf( stderr, " explication : %s\n",G_get_error_mes( fd ));
        fprintf( stderr, " type : %s\n", G_get_error_type( fd ));
        fprintf( stderr, " reference : voir erreur %d\n", num );
        exit( -num );
        }
```

Cette fonction peut être redéfinie par l'utilisateur. Ceci peut par exemple permettre de récuperer l'erreur au cas où on ouvre un fichier qui n'existe pas.

Pour remplacer la fonction "dzv_fonction_erreur" par une autre fonction "f", il suffit d'appeler la fonction **G_set_error_handler**

Voici la description de chacune des fonctions citées précédemment.

1. Retourne un pointeur sur une chaîne de caractères qui représente les n derniers appels relatifs à un fichier.

```
char    *G_get_context( fd, n )
```

int **n;**
Spécifie le nombre d'appel maximum qu'on veut récupérer.

2. Retourne un pointeur sur une chaîne de caractères qui représente le nom du fichier.

```
char    *G_get_fic_name( fd )
```

3. Retourne un pointeur sur une chaîne de caractères qui explique la dernière erreur qui c'est produite.

```
char    *G_get_error_mes( fd )
```

4. Retourne un pointeur sur une chaîne de caractères qui explique le type de l'erreur qui c'est produite. Le type retourné peut être "**fatal**" ou "**warning**". Dans le cas ou l'erreur est de type "**fatal**", par exemple un identificateur de fichier invalide, il est souhaitable d'arrêter le programme.

```
char    *G_get_error_type( fd )
```

5. Prend en paramètre une nouvelle fonction de traitement d'erreur ou la constante **NULL** qui indique que l'on souhaite remettre en place la fonction d'erreur par défaut. Cette fonction est commune à tous les fichiers.

```
int G_set_error_handler( fonction_erreur )
```

- **int** **fonction_erreur()**
  Nouvelle fonction de récupération d'erreur. En cas d'erreur cette fonction est appelée avec 2 paramètres qui sont respectivement le numéro du fichier et le numéro de l'erreur.

2.5.14 Affichage du contenu d'un fichier

Les fonctions suivantes permettent d'afficher le contenu d'un fichier en spécifiant un format de sortie.
Les valeurs affichées correspondent à la dernière opération de lecture ou modification.

```
int   G_print_fic( fd, sortie, format )
int   G_print_elt( fd, sortie, format )
int   G_print_mai( fd, sortie, format )
```

- **int fd;**
  Identificateur du fichier.
- **FILE sortie;**
  Spécifie le fichier sur lequel va l'affichage. Eventuellement **stderr** ou **stdout**.
- **char *format;**
  Quatre formats sont possibles:
  * **"tableur"** : On affiche les valeurs séparées par des tabulations. Ce format a pour but de pouvoir utiliser les valeurs du fichier depuis un tableur.
  * **"complet"** : On affiche les valeurs, les commentaires et le type des objets.
  * **"reduit"** : On affiche les valeurs et les commentaires.
  * **"header"** : On se contente d'afficher les commentaires et le type des objets, pas les valeurs associées.

Exemple : Voir annexe A.12.

2.5.15 Accès direct

Les fonctions suivantes permettent de se positionner dans le fichier.
Elles provoquent une erreur au cas ou le fichier est en mode *"pipe"*. Les fonctions **G_seek_elt** et **G_seek_mai** renvoient **1** si l'élément existe et **0** sinon. **G_seek_mai** provoque une erreur si le numéro du maillon est incorrect.
Si le numéro d'élément indiqué n'existe pas, on se trouve positionné en fin de fichier. On peut alors ajouter un nouvel élément. Un moyen sûr de se positionner en fin de fichier est :
G_seek_elt(fd, G_get_max_elt(fd) + 1);

```
int   G_seek_fic( fd )
int   G_seek_elt( fd, num_element )
int   G_seek_mai( fd, num_element, num_maillon )
```

- **int fd;**
  Identificateur du fichier.
- **int num_element;**
  Numéro de l'entier sur lequel on veut se positionner.
- **int num_maillon;**
  Numéro du maillon sur lequel on veut se positionner, les maillons sont numérotés à partir de 1.

Exemple : Voir annexe A.13.

2.5.16 Destruction d'un élément

Il est important de noter que détruire un élément d'un fichier ne signifie pas le supprimer physiquement du fichier mais seulement logiquement (lui et ses maillons). Cela implique que la taille des fichiers ne diminue pas, mais une fois qu'un élément est détruit, on ne peut plus y accéder.
Remarque :
On peut fort bien une fois qu'un élément a été détruit en créer un nouveau de même numéro. Cette fonction fait appel à la primitive <G_seek_elt> pour se positionner sur l'élément "num" du fichier "fp" puis l'invalide. Après l'appel de cette primitive " le pointeur du fichier " est positionné sur l'élément suivant dans le fichier.

int G_del_elt(fd, num)

- **int fd;**
  Identificateur du fichier.
- **int num;**
  Numéro de l'élément à détruire.

Exemple : Voir annexe A.9.

2.5.17 Fermeture du fichier

Cette fonction doit absolument être appelée en cas de création de fichier ou en cas de modification d'un fichier existant.

Dans le cas contraire la prochaine tentative d'ouverture du fichier déclenchera une erreur.

int G_close(fd)

- **int fd;**
  Identificateur du fichier.

Exemple : Voir annexe A.1.

2.6 Les fonctions spécialisées

2.6.1 Ecriture

Les fonctions suivantes permettent de demander une écriture sur le fichier.
Pour ce, on passe à cette fonction une liste de variables qui dépend des DZV.

```
int   S_write_fic( fd, ... vi ...)
int   S_write_elt( fd, numero, nb_maillons, ... vi ... )
int   S_write_mai( fd, ... vi ...)
```

- **int fd;**
  Identificateur du fichier.
- **int numero;**
  On indique le numéro de l'élément courant.
- **int nb_maillons;**
  On indique le nombre de maillons de l'élément courant.
- **??? *vi;**
  Le nombre de vi est égal au nombre de triplets indiqué dans le DZV. Chaque vi doit être l'adresse d'une zone dont la taille est au moins égale à celle indiquée dans le DZV. Cela signifie que c'est à l'utilisateur d'allouer la taille nécessaire. On peut éprouver le besoin de ne pas vouloir affecter certaines zones, dans ce cas il faut passer la constante **NULL** et non pas une adresse.

Exemple : Voir annexe A.2.

2.6.2 Lecture du fichier

Les fonctions suivantes permettent de demander la lecture du fichier. Elles ont pour effet de lire et récupérer les valeurs lues en fonction des DZV.
Il faut donc passer à cette fonction une liste de variables qui dépend des DZV.
Le format des variables manipulées en mémoire centrale doit être le même que celui indiqué dans les DZV.
Pour indiquer que tous les éléments ont été lus, ou que tous les maillons d'un élément ont été lus, **S_read_elt** et respectivement **S_read_mai** renvoient **1** quand la lecture a été possible, et **0** sinon.

L'appel de ces fonctions est facultatif si on ne veut pas exploiter les valeurs associées. Dans ce cas elles sont en fait automatiquement appelées. On préserve ainsi la possibilité d'exploiter le fichier en mode *"pipe"*.

```
int   S_read_fic( fd, ... vi ...)
int   S_read_elt( fd, numero, nb_maillons, ... vi ... )
int   S_read_mai( fd, ... vi ...)
```

- **int** **fd;**
  Identificateur du fichier.
- **int** ***numero;**
  On récupère le numéro de l'élément courant.
- **int** ***nb_maillons;**
  On récupère le nombre de maillons de l'élément courant.
- **???** ***vi;**
  Le nombre de vi est égal au nombre de triplets indiqué dans le DZV. Chaque vi doit être l'adresse d'une zone dont la taille est au moins égale à celle indiquée dans le DZV. Cela signifie que c'est à l'utilisateur d'allouer la taille nécessaire. On peut éprouver le besoin de ne pas récupérer certaines zones, dans ce cas il faut passer la constante **NULL** et non pas une adresse.

Exemple : Voir annexe A.3.

2.6.3 Ecriture sur un fichier DZV en spécifiant un format

Les fonctions suivantes permettent de demander une écriture sur le fichier. Ces fonctions ont pour effet de modifier le fichier. Contrairement aux fonctions S_write_???, les variables manipulées en mémoire centrale peuvent avoir un type différent de celui indiquer dans les DZV, il suffit de le spécifier dans le format.

```
int  S_fwrite_fic( fd, format, ... vi ...)
int  S_fwrite_elt( fd, numero, nb_maillons, format, ... vi ... )
int  S_fwrite_mai( fd, format, ... vi ...)
```

- **int fd;**
  Identificateur du fichier.
- **int numero;**
  On indique le numéro de l'élément courant.
- **int nb_maillons;**
  On indique le nombre de maillons de l'élément courant.
- **char *format;**
  Pour chacune des valeurs on doit indiquer le format élémentaire dans lequel on veut l'utiliser. Le format est une chaîne de caractères composée d'une liste de formats élémentaires.
  Les formats élémentaires possibles sont :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char",
"%d"	pour un "double"	"%l"	pour un "long".

  Exemple de formats : "%d" ou "%i%f%f".
- **??? *vi;**
  Le nombre de vi est égal au nombre de triplets indiqué dans le DZV. Chaque vi doit être l'adresse d'une zone allouée par l'utilisateur. La taille nécessaire est fonction du facteur de répétition indiqué dans les DZV et du format que l'utilisateur spécifie.
  On peut éprouver le besoin de ne pas affecter certaines zones, dans ce cas il faut passer la constante **NULL** et non pas une adresse.

Exemple : Voir annexe A.4.

2.6.4 Lecture d'un fichier en spécifiant un format

Les fonctions suivantes permettent de demander la lecture du fichier. Elles ont pour effet de lire et récupérer les valeurs lues en fonction des DZV et d'un format.
Il faut donc passer à cette fonction une liste de variables qui dépend des DZV.
Contrairement aux fonctions S_fread_??? , les variables manipulées en mémoire centrale peuvent avoir un type différent de celui indiqué dans les DZV.
Pour indiquer que tous les éléments ont été lus, ou que tous les maillons d'un élément ont été lus, **S_fread_elt** et respectivement **S_fread_mai** renvoient **1** quand la lecture a été possible et **0** sinon.
L'appel de ces fonctions est facultatif si on ne veut pas exploiter les valeurs associées. Dans ce cas elles sont en fait automatiquement appelée. On préserve ainsi la possibilité d'exploiter le fichier en mode *"pipe"*.

```
int  S_fread_fic( fd, format, ... vi ...)
int  S_fread_elt( fd, numero, nb_maillons, format,... vi ... )
int  S_fread_mai( fd, format, ... vi ...)
```

- **int fd;**
  Identificateur du fichier.
- **int *numero;**
  On récupère le numéro de l'élément courant.
- **int *nb_maillons;**
  On récupère le nombre de maillons de l'élément courant.
- **char *format;**
  Pour chacune des valeurs on doit indiquer le format élémentaire dans lequel on veut l'utiliser. Le format est une chaîne de caractères composée d'une liste de formats élémentaires.
  Les formats élémentaires possibles sont :

"%i"	pour un "int"	"%s"	pour un "short"
"%f"	pour un "float"	"%c"	pour un "char",
"%d"	pour un "double"	"%l"	pour un "long".

  Exemple de formats : "%d" ou "%i%f%f".
- **??? *vi;**
  Le nombre de vi est égal au nombre de triplets indiqué dans le DZV. Chaque vi doit être l'adresse d'une zone allouée par l'utilisateur. La taille nécessaire est fonction du facteur de répétition indiqué dans les DZV et du format que l'utilisateur spécifie. On peut éprouver le besoin de ne pas récupérer certaines zones, dans ce cas il faut passer la constante **NULL** et non pas une adresse.

Exemple : Voir annexe A.5.

3 Conclusion

Nous proposons dans ce rapport une standardisation pour la représentation de toutes les primitives extraites par les algorithmes de segmentation d'image. Les grandes classes de primitives sont les chaînes de contour, les segments de droite et les régions. La caractéristique principale de ces classes est la représentation sous forme de listes d'éléments. L'utilisation de ce standard permet de simplifier la programmation et résoud le problème de la portabilité des données.

Après une brève présentation des caractéristiques du standard, on donne dans le chapitre II et III tout ce qu'il est utile de connaître pour pouvoir l'utiliser sans problèmes. Les nombreux exemples donnés devraient compléter efficacement cette description.

Ce logiciel présente trois particularités très intéressantes.

- La première particularité, c'est de pouvoir s'adapter à de nombreuses primitives de type "liste".
- La deuxième particularité, c'est la possibilité d'avoir aussi bien l'approche accès séquentiel, que l'approche accès direct.
- La troisième particularité concerne les fonctions généralisées qui permettent de développer des applications communes.

Ce manuel d'utilisation permet de répondre à toutes les questions que peut se poser un utilisateur du standard:

- Quels sont les possibilités du standard
- Comment l'utiliser
- Comment le maintenir

Note : les sources en langage C des programmes de ce standard sont disponibles. La demande doit en être faite à G. Giraudon à l'adresse indiquée.

A Exemples

L'objectif de cette annexe est de permettre un apprentissage rapide à l'utilisation des DZV. Il doit être complété par la lecture du paragraphe concernant la descriptions des primitives.

L'exemple que nous avons choisi correspond à l'exemple indiquer dans le chapitre de présentation.
Pour caractériser totalement l'information, il reste à fixer le type des objets en tant qu'objet C. On suppose que les choix suivants correspondent à nos besoins :

- Pour l'entête:
  * Les dimensions de l'image : deux entiers.
  * Le numéro de l'élément dont la surface est la plus grande : un entier.
- Pour l'élément:
  * Sa nature : une chaîne de 20 caractères.
  * Sa surface: un flottant.
- Pour le maillon:
  * Ses coordonnées : deux entiers.
  * Sa couleur : un short.
  * Le secteur angulaire: un double.

A.1 Création d'un DZV

Avant de créer le fichier, il faut créer trois structures qui ont pour but de décrire respectivement :

- L'entête.
- Les éléments.
- Les maillons.

Programme correspondant à la création des 3 DZV :

```
#include <dzv.h>

DZV *dzv_dentete, *dzv_element, *dzv_maillon;

dzv_dentete = G_set_dzv ( 2, /*** nombre de triplets qui vont suivre ***/
                           2, "%i" , "dimx dimy",
                           1, "%i" , "max");
dzv_element = G_set_dzv ( 2,
                           20, "%c", "nature",
                           1, "%f", "surface",
dzv_maillon = G_set_dzv ( 3,
                           2, "%i" , "x y",
                           1, "%s", "couleur",
                           1, "%d", "angle");
```

Ces 3 instructions ont permis de décrire les informations de notre fichier. Il est à noter qu'à cette étape, on n'a pas encore créé de fichier. On s'est contenté de décrire le fichier en donnant en plus une étiquette logique à chacun des champs. Par exemple, la couleur associée à un point, a pour nom logique **couleur**. Ce nom pourra par la suite être réutilisé pour relire le fichier. Dans la description qu'on a faite, on a groupé certains éléments. On peut donner comme exemple les **2 "int"** des maillons pour représenter les coordonnées des pixels. Ce "regroupement" est important et aura une influence par la suite.

Pour plus d'information concernant la fonction **G_set_dzv** on peut consulter le chapitre "Description des primitives".

A.2 La création du fichier (en utilisant S_write_???)

On suppose que les types utilisés sont identiques en mémoire centrale et sur disque.

```
#include <dzv.h>

DZV     *dzv_entete, *dzv_element, *dzv_maillon;
int     fd, max, numero, nombre_de_pixels;
int     dimension[2];
int     coordonnees[2];
float   surface;
char    nature[20];
double  angle;
short   couleur;

dzv_dentete = G_set_dzv ( 2, 2,"%i","dimx dimy", 1, "%i" ,"max");
dzv_element = G_set_dzv ( 2, 20, "%c", "nature", 1, "%f", "surface",
dzv_maillon = G_set_dzv ( 3, 2, "%i" ,"x y", 1, "%s", "couleur", 1, "%d", "angle");

/************** CREATION DU FICHIER *****************/

fd = G_create(    "titi",
                  "exemple 1",
                  100,
                  dzv_entete,
                  dzv_element,
                  dzv_maillon);

/****************** INITIALISATION DE L'ENTETE ******************/

dimension[0] = 256; dimension[1] = 256; max = 2;
S_write_fic( fd, dimension, &max );

/****************** INITIALISATION D'UN ELEMENT *****************/

numero = 1; nombre_de_pixels = 3;surface = 50.0;
strcpy( nature, "triangle");
S_write_elt( fd, numero, nombre_de_pixels, nature, &surface);

/****************** INITIALISATION DE SES MAILLONS **************/

coordonnees[0]= 10; coordonnees[1]=20; couleur=255; angle = 45.0;
S_write_mai( fd, coordonnees, &couleur, &angle);
```

```
coordonnees[0]= 10; coordonnees[1]=10; couleur=0; angle = 90.0;
S_write_mai( fd, coordonnees, &couleur, &angle);

coordonnees[0]= 20; coordonnees[1]=10; couleur=0; angle = 45.0;
S_write_mai( fd, coordonnees, &couleur, &angle);

/****************** ON FAIT DE MEME POUR TOUT LES ELEMENTS DU FICHIER **/

                                    ...

/****************** FERMETURE DU FICHIER ( OBLIGATOIRE ) ********/

G_close( fd );
```

A.3 Lecture du fichier (en utilisant S_read_???)

On suppose que les types utilisés sont identiques en mémoire centrale et sur disque.

```
#include <dzv.h>

DZV     *dzv_entete, *dzv_element, *dzv_maillon;
int     fd, max, numero, nombre_de_pixels;
int     dimension[2];
int     coordonnees[2];
float   surface;
char    nature[20];
double  angle;
short   couleur;

/************** OUVERTURE DU FICHIER *****************/

fd = G_open(  "titi",
              &type,
              &nb_element,
              &dzv_entete,
              &dzv_element,
              &dzv_maillon);

/****************** INITIALISATION DE L'ENTETE ******************/

S_read_fic( fd, dimension, &max );

/********************** LECTURE DES ELEMENTS **********************/

while (S_read_elt( fd, &numero, &nombre_de_pixels, nature, &surface ))
              {
              /********* LECTURE DES MAILLONS ***************************/
              while (S_read_mai( fd, coordonnees, &couleur, &angle))
                            {
                            }
              }
G_close( fd );
```

Il est à noter que les boucles de contrôle **while** s'expliquent facilement quand on sait que le résultat des fonctions **S_read_elt** et **S_read_mai** est 1 si la lecture a été possible, 0 sinon.

A.4 La création du fichier (en utilisant S_fwrite_???)

Dans cet exemple on suppose que l'application travaille sur des variables de types différents que ceux du fichier. On spécifie donc un format.

```
#include <dzv.h>

DZV     *dzv_entete, *dzv_element, *dzv_maillon;
int     fd, max, numero, nombre_de_pixels;
short   dimension[2];
float   coordonnees[2];
short   surface;
char    nature[20];
float   angle;
short   couleur;

dzv_dentete = G_set_dzv ( 2, 2,"%i","dimx dimy", 1, "%i" ,"max");
dzv_element = G_set_dzv ( 2, 20, "%c", "nature", 1, "%f", "surface",
dzv_maillon = G_set_dzv ( 3, 2, "%i" ,"x y", 1, "%s", "couleur", 1, "%d", "angle");

/************** CREATION DU FICHIER *****************/

fd = G_create(    "titi",
                  "exemple 1",
                  100,
                  dzv_entete,
                  dzv_element,
                  dzv_maillon);

/****************** INITIALISATION DE L'ENTETE ********************/

dimension[0] = 256; dimension[1] = 256; max = 2;
S_fwrite_fic( fd, "%s%i", dimension, &max );

/****************** INITIALISATION D'UN ELEMENT ******************/

numero = 1; nombre_de_pixels = 3;surface = 50.0;
strcpy( nature, "triangle");
S_fwrite_elt( fd, numero, nombre_de_pixels, "

/****************** INITIALISATION DE SES MAILLONS **************/

coordonnees[0]= 10; coordonnees[1]=20; couleur=255; angle = 45.0;
S_fwrite_mai( fd, "%f%s%f",coordonnees, &couleur, &angle);
```

```
coordonnees[0]= 10; coordonnees[1]=10; couleur=0; angle = 90.0;
S_fwrite_mai( fd, "%f%s%f",coordonnees, &couleur, &angle);

coordonnees[0]= 20; coordonnees[1]=10; couleur=0; angle = 45.0;
S_fwrite_mai( fd, "%f%s%f",coordonnees, &couleur, &angle);

/****************** ON FAIT DE MEME POUR TOUT LES ELEMENTS DU FICHIER **/

                                    ...

/****************** FERMETURE DU FICHIER ( OBLIGATOIRE ) ********/

G_close( fd );
```

A.5 Lecture du fichier (en utilisant S_fread_???)

Dans cet exemple on suppose que l'application travaille sur des variables de types différents que ceux du fichier. On spécifie donc un format.

```
#include <dzv.h>

DZV     *dzv_entete, *dzv_element, *dzv_maillon;
int     fd, max, numero, nombre_de_pixels;
short   dimension[2];
float   coordonnees[2];
short   surface;
char    nature[20];
float   angle;
short   couleur;

/************** OUVERTURE DU FICHIER ****************/

fd = G_open(  "titi",
              &type,
              &nb_element,
              &dzv_entete,
              &dzv_element,
              &dzv_maillon);

/***************** INITIALISATION DE L'ENTETE ******************/

S_fread_fic( fd, "%s%i",dimension, &max );

/********************* LECTURE DES ELEMENTS *********************/

while (S_fread_elt( fd, &numero, &nombre_de_pixels,"
              {
              /********* LECTURE DES MAILLONS *************************/
              while (S_fread_mai( fd, coordonnees, "%f%s%f", &couleur, &angle))
                            {
                            }
              }
G_close( fd );
```

Il est à noter que les boucles de contrôle **while** s'expliquent facilement quand on sait que le résultat des fonctions **S_fread_elt** et **S_fread_mai** est 1 si la lecture a été possible, 0 sinon.

A.6 La fonction G_cmp_dzv

La fonction G_cmp_dzv pourrait être définie de la manière suivante :

```
int G_cmp_dzv( dzv1, dzv2 )
    DZV   dzv1, dzv2;
    {
    int     nb_triplet1, nb_triplet2, i;
    int     *nb1, *nb2;
    char    **format1, **format2;
    char    **commentaire1, **commentaire2;

    G_get_dzv( dzv1, &nb_triplet1, &nb1, &format1, &commentaire1 );
    G_get_dzv( dzv2, &nb_triplet2, &nb2, &format2, &commentaire2 );

    if ( nb_triplet1 != nb_triplet2) return( 0 );

    for (i=1; i <= nb_triplet1; i++)
            {
            if (nb1[i] != nb2[i] ) return( 0 );
            if (strcmp(format1[i],format2[i])) return( 0 );
            }
    return( 1 );
    }
```

A.7 Afficher les dimensions associées à un fichier

On suppose qu'on a réservé dans l'entête du fichier 2 zones dont les commentaires sont "dimx" et "dimy". On veut afficher les valeurs correspondantes.

```
main()
    {
    DZV   *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd, nb_elt;
    char  *type;
    float dimx, dimy;

    fd = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);
    G_read_fic( fd );

    if ( ! G_get_fic( fd, "dimx dimy", "%f%f", &dimx, &dimy ))
            fprintf( stderr, "dimension inconnu\n");
    else
            fprintf( stdout, "dimx=%f dimy=%f\n", dimx, dimy);
    G_close( fd );
    }
```

Ce programme fonctionne quelque soit le fichier. On a uniquement utilisé des fonctions généralisées. On n'a pas eu besoin de tenir compte du DZV de l'entête du fichier.

A.8 Modifier les dimensions associées à un fichier

On suppose qu'on a réservé dans l'entête du fichier 2 zones dont les commentaires sont "dimx" et "dimy". On veut affecter à ces valeurs 256 et 256.

```
main()
    {
    DZV   *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd, nb_elt;
    char  *type;
    float dimx, dimy;

    fd = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);
    G_read_fic( fd );

    dimx = dimy = 256;
    if ( ! G_set_fic( fd, "dimx dimy", "%f%f", &dimx, &dimy ))
          fprintf( stderr, "dimension inconnu\n");
    G_write_fic( fd );
    G_close( fd );
    }
```

Ce programme fonctionne quelque soit le fichier. On a uniquement utilisé des fonctions généralisées. On n'a pas eu besoin de tenir compte du DZV de l'entête du fichier. C'est la fonction G_set_fic qui se charge de trouver leur position et de convertir éventuellement les flottants.

A.9 Supprimer dans un fichier les éléments qui ont plus de 100 maillons

On veut à partir d'un fichier garder uniquement les éléments qui ont plus de 100 maillons.
La méthode qui suit n'est pas souhaitable pour deux raisons :

- Au cas où l'exécution est arrêtée avant la fermeture, le fichier deviendra "incohérent" et donc inutilisable.
- On accède au fichier en faisant simultanéement de la lecture et de l'écriture, on risque donc d'avoir un programme lent.

Il est donc préférable d'utiliser la méthode de l'exemple du paragraphe suivant.

```
main()
    {
    DZV   *dzv_entete, *dzv_element, *dzv_maillon;
    char  *type;
    int   fd1, numero, nb_maillons, nb_elt;

    fd1 = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);

    while ( G_read_elt( fd1, &numero, &nb_maillons ))
          {
          if (nb_maillon > 100)
               {
               G_del_elt( fd1, numero);
               }
          }
    G_close( fd1 );
    }
```

A.10 Dupliquer un fichier en gardant les éléments qui ont plus de 100 maillons

On veut à partir d'un fichier garder uniquement les éléments qui ont plus de 100 maillons, le résultat est stocker dans un nouveau fichier. Il est intéressant de comparer cette technique à l'exemple précédent.

```
main()
    {
    DZV *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd1, fd2, nb_elt;
    char  *type;
    int   numero, nb_maillons, nb_elt;

    fd1 = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);
    fd2 = G_create( "/tmp/titi.copy", type, nb_elt, dzv_entete, dzv_element, dzv_maillon);

    G_read_fic( fd1 );
    G_copy_fic( fd1, fd2 );
    G_write_fic( fd2 );

    while ( G_read_elt( fd1, &numero, &nb_maillons ))
        {
        if (nb_maillon > 100)
            {
            G_copy_elt( fd1, fd2 );
            G_write_elt( fd2, numero, nb_maillon );
            while (G_read_mai( fd1 ))
                {
                G_copy_mai( fd1, fd2 );
                G_write_mai( fd2 );
                }
        }
    G_close( fd1 );
    G_close( fd2 );
    }
```

A.11 Acquisition d'information sur les numéros des éléments

Le programme qui suit affiche des informations concernant les éléments du fichier.

```
main()
    {
    DZV *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd, nb_elt, version, release;
    char  *type;

    fd = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete,&dzv_element, &dzv_maillon);

    if (G_get_version( fd, &version, &release))
        fprintf( stdout,"/tmp/titi est un fichier version %d release %d", version, release);
    fprintf( stdout, "le plus petit %d\n", G_get_min_elt(fd));
    fprintf( stdout, "le plus grand %d\n", G_get_max_elt(fd));
    fprintf( stdout, "le premier %d\n", G_get_first_elt(fd));
    fprintf( stdout, "le dernier %d\n", G_get_last_elt(fd));
    fprintf( stdout, "le nombre d'element dans le fichier %d\n", G_get_nb_elt(fd));
    G_close( fd );
    }
```

A.12 Un programme qui transforme un fichier DZV en ASCII

Le programme qui suit, est une version simplifiée de l'utilitaire dzv_pr.

```
main()
    {
    DZV *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd, nb_elt, num, nb;
    char  *type;

    fd = G_open( "toto", &type, &nb_elt, &dzv_fic, &dzv_element, &dzv_maillon);
    G_read_fic( fd );
    G_print_fic( fd, stdout, "complet");
    while (G_read_elt( fd, &num, &nb ))
        {
        G_print_elt( fd, stdout, "reduit");
        while (G_read_mai( fd ))
            {
            G_print_mai( fd, stdout, "tableur");
            }
        }
    G_close( fd );
    }
```

A.13 Accès direct

Le programme qui suit imprime les caractéristiques de l'élément 10.

```
main()
    {
    DZV *dzv_entete, *dzv_element, *dzv_maillon;
    int   fd, nb_elt, num, nb;
    char  *type;

    fd = G_open( "toto", &type, &nb_elt, &dzv_fic, &dzv_element, &dzv_maillon);

    if (! G_seek_elt( fd, 10 ))
          fprintf( stderr, "l'element 10 n'existe pas\n");
    else
          {
          G_read_elt( fd, &num, &nb);
          G_print_elt( fd, stdout, "reduit");
          }
    G_close( fd );
    }
```

L'utilisation des fonctions **G_seek_elt** et **G_seek_mai** est similaire.

A.14 Selectionner dans un fichier les éléments dont la surface est supérieure à 2000

On veut, à partir d'un fichier, sélectionner les éléments qui ont une surface supérieure à 2000. On suppose qu'un des champs variables des éléments a pour commentaire associé "surface".

```
main()
    {
    DZV   *dzv_entete, *dzv_element, *dzv_maillon;
    char  *type;
    int   fd1, numero, nb_maillons, nb_elt, valeur;

    fd1 = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);

    while ( G_read_elt( fd1, &numero, &nb_maillons ))
        {
        if (! G_get_elt( fd1, "surface", "%i", &valeur))
            fprintf( stdout, "commentaire incorrect\n");
        if (valeur > 2000)
            {
            fprintf( stdout, "Element %d selectionne\n", numero);
            }
        }
    G_close( fd1 );
    }
```

De façon symilaire, on peut utiliser **G_get_mai** précédé par **G_read_mai**.

A.15 Modifier dans un fichier la valeur de la surface de certains éléments

Cet exemple ne doit pas être abordé avant la compréhension de l'exemple précédent. On veut, à partir d'un fichier, sélectionner les éléments qui ont une surface supérieure à 2000 et leur affecter la valeur 1000. On suppose qu'un des champs variables des éléments a pour commentaire associé "surface".

```
main()
    {
    DZV   *dzv_entete, *dzv_element, *dzv_maillon;
    char  *type;
    int   fd1, numero, nb_maillons, nb_elt, valeur;

    fd1 = G_open( "/tmp/titi", &type, &nb_elt, &dzv_entete, &dzv_element, &dzv_maillon);

    while ( G_read_elt( fd1, &numero, &nb_maillons ))
          {
          if (! G_get_elt( fd1, "surface", "%i", &valeur))
               fprintf( stdout, "commentaire incorrect\n");
          if (valeur > 2000)
               {
               valeur = 1000;
               G_set_elt( fd1, "surface", "%i", &valeur);
               G_seek_elt( fd1, numero);
               G_write_elt( fd1, numero, nb_maillons);
               }
          }
    G_close( fd1 );
    }
```

De façon symilaire, on peut utiliser **G_set_mai**.

B Les différentes versions

B.1 Ce qui est incompatible

1. Les fichiers DZV générées par la version I sont incompatibles avec les versions II et III.
2. Pour les programmes de la version II, il est nécessaire de modifier la fonction **re_com**.

B.2 Ce qu'il est souhaitable de modifier

Avertissement :

> Aucune des modifications qui suivent n'est obligatoire.
> Les fonctions de la version I qui n'ont pas d'homologues dans la version III, continuent à exister.
> Pour les autres, le fichier dzv.h contient les "**#define**" nécessaires.

- **de_bug** : à supprimer.
- **pr_fic** : à supprimer ou à remplacer par **G_print_fic**. (attention les paramètres sont différents)
- **ra_fic** : Utiliser de préférence **D_seek_fic** ou **D_seek_elt** ou **D_seek_mai** suivant le contexte.
- **cr_fic** : Utiliser de préférence **D_create** suivi de **S_write_fic**.
- **op_fic** : Utiliser de préférence **D_open** suivi de **S_read_fic**.
- **re_dzv** : à supprimer ou à remplacer par **G_get_dzv**. (attention les paramètres sont différents)
- Pour les autres fonctions adopter la nouvelle notation.

Tableau récapitulatif des trois versions.

Version I	Version II	Version III
dzv-protection	x	x
x	cr_fic_base	G_create
cr_fic	cr_fic	x
x	op_fic_base	G_open
op_fic	op_fic	x
x	get_fic get_elt get_mai	G_read_fic G_read_elt G_read_mai
x	re_com	G_get_fic G_get_elt G_get_mai
x	x	G_set_fic G_set_elt G_set_mai
x	x	G_copy_fic G_copy_elt G_copy_mai
x	x	G_write_fic G_write_elt G_write_mai
x	x	G_seek_fic
ra_fic(fd,x,0)	ra_fic(fd,x,0)	G_seek_elt(fd, x)
ra_fic(fd,x,y)	ra_fic(fd,x,y)	G_seek_mai(fd, x, y)
cl_fic	cl_fic	G_close
x	get_max_elt	G_get_max_elt
x	get_min_elt	G_get_min_elt
x	get_nb_elt	G_get_nb_elt
x	get_first_elt	G_get_first_elt
x	get_last_elt	G_get_last_elt
de_bug	de_bug	x
x	x	G_get_context
x	x	G_get_fic_name
x	x	G_get_error_mes
x	x	G_get_error_type
x	x	G_set_error_handler
x	x	G_get_version
cr_dzv	cr_dzv	G_set_dzv
re_dzv	re_dzv	x
x	x	G_get_dzv
cp_dzv	cp_dzv	G_cmp_dzv
x	del_elt	G_del_elt
x	x	G_print_fic G_print_elt G_print_mai
pr_fic	pr_fic	x
wr_fic wr_elt wr_mai	wr_fic wr_elt wr_mai	S_write_fic S_write_elt S_write_mai
x	x	S_fwrite_fic S_fwrite_elt S_fwrite_mai
re_fic re_elt re_mai	re_fic re_elt re_mai	S_read_fic S_read_elt S_read_mai
x	x	S_fread_fic S_fread_elt S_fread_mai

C Les utilitaires

La librairie des DZV permet de développer un certain nombre d'utilitaires.

C.1 Visualisation du type du fichier

Un utilitaire **dzv_par** permet de visualiser l'entête d'un fichier DZV. Cette utilitaire utilise les fonctions G_get_???_elt et G_print_???

C.2 Visualisation de la totalité du contenu d'un fichier

Affichage du contenu du fichier. On peut éventuellement sélectionner un élément ou un maillon.
Cet utilitaire utilise les fonctions G_print_???.

C.3 "Compactage" d'un fichier DZV

Permet de réduire la taille d'un fichier dont certains éléments ont été détruits en utilisant la fonction G_del_elt.

C.4 D'autres utilitaires

Les utilitaires qui suivent ont été développés mais ne sont pas fournis avec la librairie de base étant donné leurs spécificités.

- Conversion d'un fichier format DZV en fichier format Inrimage.
- Filtrage suivant des caractéristiques numériques associées aux éléments.
- Visualisation graphique des maillons des fichiers DZV sous environnement X11.
- Calculs statistiques relatifs à une des zones du fichier.

D Maintenance du logiciel

D.1 Organisation du fichier

L'organisation qui a été retenue est basée sur des impératifs qui proviennent de la volonté de faire cohabiter la possibilté d'accès directs et séquentiels (éventuellement avec un "pipe" ou une redirection).

D.1.1 L'accès direct

L'accès direct est réalisé en utilisant un seul fichier. Pour permettre ce mode d'accès, on gère une table qui contient l'adresse relative des éléments par rapport au début du fichier.
On ne peut connaître à l'ouverture la place qui sera nécessaire à cette table, cette remarque implique que cette table soit stockée à la fin du fichier. Cette table est donc gérée en mémoire centrale tant que le fichier n'est pas fermé.
A la fermeture du fichier, on écrit cette table en fin de fichier.

D.1.2 L'accès séquentiel

Pour permettre au fichier du standard d'être "pipable" l'accès à l'information doit pouvoir être fait séquentiellement.
Cette remarque impose l'organisation suivante pour le standard :
On trouve au début des informations générales. Ensuite sont stockés les éléments et les maillons dans l'ordre dans lequel ils ont été donnés par l'utilisateur. En fin de fichier on trouve la table qui permet les accès directs.

D.2 Installation

D.2.1 Portabilité du logiciel

- Ce logiciel est écrit en langage C, il utilise depuis la version II les fonctions standards de la librairie C.
  Son portage hors du monde UNIX ne devrait donc pas poser de problème.

- La conversion automatique des flottants (voir paragraphe suivant) peut néanmoins être limitée.

- Contrairement à la version I, le logiciel n'est plus dépendant de l'ordre d'empilement des paramètres dans la pile, (on utilise la bibliothèque "varargs").
- Restrictions:
  * Les objets C de type pointeur doivent tous être de même taille.

  * Les variables booléennes utilisées sont supposées être la valeur 0 pour faux et différent de 0 pour vrai.
  * Il est nécessaire d'avoir à sa disposition une librairie d'entrée sortie comportant les fonctions "fopen", "fclose", "fread", "fwrite", "fseek" et "ftell".
- Les primitives de cette version peuvent être "linker" avec la version 15.2 de Le_Lisp développé par l'INRIA.

D.2.2 Les constantes

Certaines constantes du standard peuvent facilement être modifiées. Elles sont définies dans le fichier **des_fic.h**. Ces constantes sont:

- SUP_ELT :
  C'est la dimension supplémentaire qui est accordée à la table d'accès direct en cas de débordement de celle ci. A chaque fois que le logiciel détecte un débordement il fait un realloc( ). Cette valeur à donc intérêt à être la plus grande possible à condition qu'on ait suffisamment de place en mémoire centrale ...
- MAX_FICHIERS :
  C'est le nombre de fichiers ouverts simultanément par le standard dans un programme. Il est fixé à 30 et ne devrait donc pas être sujet à modification ...
- MAX_MEMORISE :
  C'est la taille maximum de l'historique associé à un fichier. Cet historique est utilisé par la fonction **G_get_context**.

D.2.3 Module de conversion des flottants

Actuellement le standard reconnaît quatre type de flottants:

- Les flottants format GOULD.
- Les flottants format IEEE.
- Les flottants format PERKIN
- Les flottants format CONVEX.

Le module de base qui gère la conversion de ces flottants se trouve dans le fichier **cnvflt.c**.
Une variable gère le type de la machine hôte. Cette variable est initialisée par une compilation conditionnelle. (Voir le fichier **common.c**).

D.2.4 Installation de la librairie

Après s'être arrangé pour que la variable concernant la machine hôte soit définie (voir paragraphe précédent), il faut compiler la librairie **dzv.a**. Un Makefile facilite cette opération.

L'utilisateur doit avoir accès à cette librairie ainsi qu'au fichier **dzv.h**.

D.2.5 Utilisation

Tous les fichiers utilisant les primitives du standard doivent inclure le fichier **dzv.h**. On doit ensuite linker le résultat avec la librairie **dzv.a**.

E Table des messages d'erreurs

1	erreur détectée par le système d'exploitation ...
2	"paramètre de DZV incorrect" Il se peut qu'on ait essayé de créer un DZV en indiquant un nombre négatif de zones ou plus probablement que l'un des types de zones soit incorrect. Les types possibles sont : "%i", "%f", "%s", "%c", "%l", "%d".
3	"Commentaire incorrect" Chaque zone d'un DZV est caractérisée par un facteur de répétition, un type de zone et un commentaire. Ce commentaire est obligatoire et ne doit pas dépasser 256 caractères.
4	"Liste de paramètres incorrecte" Le nombre de triplet n'est pas cohérent
5	"type de fichier incorrect (30 caractères maxi)" Le type du fichier indiqué doit être une chaîne de caractères terminée par le caractère NULL.
6	"Fichier protégé en écriture" La protection du fichier au sens unix interdit une écriture.
7	"impossible en création" Alors que la dernière opération à consisté à créer un fichier ou à y ajouter un nouvel élément, on essaie de faire une lecture. Ce n'est pas possible à moins d'utiliser un accès direct.
8	"impossible en ouverture" Alors que la dernière opération a consisté à ouvrir un fichier ou à y lire un élément, on essaie de faire une écriture. Ce n'est pas possible à moins d'utiliser un accès direct.
9	"opération impossible avec un pipe ..." En mode "*pipe*" on ne peut faire que de l'accès séquentiel.
10	"organisation du fichier incoherente ..." Trois possibilités : - le fichier qu'on ouvre n'est pas un fichier créé par le standard. - le fichier n'a pas été refermé. - Peut être ce fichier correspond il à la version I ?

11	"paramètre numéro de maillon incorrect." On essaie de faire un accès direct sur un maillon qui n'existe pas, ou bien le numéro de l'élément correspond à un nouvel élément mais le numéro de maillon était différent de 0.
12	"nombre incorrect d'accès aux maillons d'un elt ." Quand on ajoute un élément, on est obligé d' écrire autant de maillons que spécifié lors de l'écriture de ce nouvel élément.
13	"tentative d'écriture de deux éléments de meme clef" On ne peut écrire deux éléments de même clef. Pour modifier un élément, il faut faire précéder la primitive d'écriture par un **G_seek_elt**.
14	"requète incompatible avec le dernier seek" On demande à lire ou écrire un maillon alors que l'on vient de demander un accès direct sur un élément.
15	"tentative de modification du nbre de maillons d'un elt" On essaie de modifier le nombres de maillons d'un élément existant.
16	"tentative de destruction d'élt inexistant" On demande à détruire un élément qui n'existe pas.
17	"G_write_fic obligatoire" En cas de création de fichier, il est nécessaire d'indiquer les informations variables relatifs au fichier.
18	"Fichiers incompatibles" Le transfert d'information ne peut se faire que pour des fichiers ayant les mêmes DZV.
19	"Accès à l'entête interdit" Pour modifier l'entête il faut au préalable utilser G_seek_fic.

20	"clef d'accès au fichier incorrecte" Le premier paramètre (la clef d'accès au fichier) n'est pas valide. Le fichier a t'il été correctement ouvert ou créé ?
21	"cette primitive doit être utilisé après G_read_???" Cette fonction doit être obligatoirement précédée par une lecture physique du fichier.
22	"Accès à l'entête obligatoire" L'écriture de l'entête du fichier est obligatoire en cas de création.
23	"format incorrect" On doit avoir autant de commentaires que de formats.
24	"format incorrect" Les formats doivent commencer par %. Il doit y avoir autant de format que de zones à lire.
25	"accès a un fichier non créé par DZV" Le fichier existe mais ne correspond pas a un fichier DZV. Peut être ce fichier correspond il à la version I ?
29	"format incorrect" Les formats possibles sont ["complet" "reduit" "tableur" "header"]
30	"Nombre de fichiers maximum atteint ..." On essaie d'utiliser simultanéement plus de fichiers que ne le permet le langage C.
31	"Echec d'une allocation dynamique ..." Cette erreur arrive quand la taille de la mémoire virtuelle n'est pas suffisante. Ceci peut être provoquer par un numéro d'élément trop grand.
32	"Format de flottant non reconnu !!!" Le fichier qu'on essaie de lire a été créé par une librairie DZV qui n'est pas cohérente avec celle utilisée par l'application.
33	"Fichier de sortie incorrect " Le paramètre vaut NULL.
34	"numéro d'erreur non répertorié" Cette erreur ne doit pas s'afficher. Si c'est le cas c'est un bug du logiciel...

F Indexe

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique