“libaff.a”
Guide de l’utilisateur Manuel de Référence

Luc Weber

Observatoire de Genève

23 octobre 1996

Table des matières

Chapitre 1

Tutorial

1.1 Utilisation

La librarie `libaff.a` permet le contrôle de l'afficheur (`xaff` sous OpenWindows) depuis un programme client. Le client peut être un programme écrit soit en Fortran soit en C. Les appels sont similaires, à la différence qu'en C, le status est retourné par la fonction alors qu'en Fortran le status est retourné dans le dernier argument supplémentaire.

Attention les chaînes de caractères écrite en Fortran doivent impérativement être terminée par un caractère nul (compatibilité avec le C). Exemple:

```
name = 'Wolfgang' // char(0)
```

Cette librarie est construite sur le protocole de communication GOP `Geneva Observatory Protocol`.

Les programmes clients se compilent en C avec la commande:

```
cc prog.c -o prog -I/share/include [options] -laff -lgop -ltpudummy
```

ou en Fortran avec

```
f77 prog.f -o prog [options] -laff -lgop -ltpudummy
```

1.2 Connection – Déconnection

La connection à `xaff` se fait avec la fonction `aff_client()`. Cette fonction alloue une structure pour le cannal de communication et initialise la connection. La déconnection est automatique, et `xaff` survit au client. Si le client qui désire tuer l'afficheur le fait avec la fonction `aff_close_connection`. Les exemples suivant montrent une connection suivit d'une déconnection en C et en Fortran.

```
#include <gop.h>
int    actual_status=0;

main()
{
   struct gop_connect *connect_aff;
   char   option[20];
   int    delay=1;
   int    repeat=10;
   ...
   if (aff_client(&connect_aff, actual_status, option, delay, repeat, verbose)
                  != GOP_OK){...}
   actual_status = 1;
   ...
   if (aff_close_connection(connect_aff) != GOP_OK){...}
}
```

```
      program xxxx

      integer    connect_aff
      integer    actual_status     /0/
      common     / status_common / actual_status
      character*20   option[20]
      integer    delay             /1/
      integer    repeat            /10/
      integer    status
      ...
      call aff_client(connect_aff, actual_status, option, delay,
     .                repeat, verbose, status)
      if(status.ne.0) ...
      actual_status = 1
      ...
      call aff_close_connection(connect_aff, status)
      if(status.ne.0) ...
      ...
      end
```

La fonction `aff_client` exécute la commande système `aff` suivie des options donnée par la chaîne de caractère `option`.
Comme le démarage de `xaff` peut prendre du temps, le client attend que la socket soit créée. La présence de la socket est testée au maximum `repeat` fois avec un intervalle de temps donne en secondes entière de `delay` [seconde]. Le niveau de verbosité du protocole de communication est initialisé avec la variable verbose: 0, 1, 2 et 3 à 9 (voir libgop.a).
Le flag `actual_status` indique par `1` que le client se croit connecté et par `0` qu'il n'est pas connecté. Ce flag global de determiner la tactique de connection si la commande `aff_client` est lancée plusieurs fois dans le même programme.

1.3 Chargement d'image

Avec la fonction `aff_load_file()` qui permet de charger des images de type `BDF` ou `FITS`.
Exemple:

```
program xxxx

integer    connect_aff
...

call aff_client(connect_aff, actual_status, option, delay, repeat, status)
if(status.ne.0) ...

call aff_load_file(connect_aff, '/images/spiral.bdf'//char(0), status)
....
end
```

1.4 Ecriture d'image

L'écriture se fait avec la fonction `aff_patch()` qui modifie juste la partie commune entre l'image courante et l'image (ou portion d'image) à écrire. On précise: offset (en pixel) du pixel de destination sur l'afficheur (dy et dy), offset sur le tableau de départ en pixel (x et y), la taille du tableau de départ à transférer (ncol et nlin), le pas en coordonnées utilisateur du tableau de départ, le tableau de départ (mem) et taille réel de ce tableau (nx).
Si le pas du tableau de destination est plus élevé que le pas sur l'afficheur, l'image est ventillée.
Exemple ou l'on transfert 20∗30 pixel du tableau `mem` à la position <50;80> de l'afficheur:

```
      program xxxx

      integer    connect_aff

      real       mem(200,200), xstp, yst
      integer    dx, dy, x, y, ncol, nlin, nx
      ...
      dx = 50
      dy = 80
      x  = 0
      y  = 0
      ncol = 20
      nlin = 30
      nx   = 200
      xstp = 1.
      ystp = 1.
      call aff_patch(connect_aff, dx, dy, x, y, ncol, nlin,
     &               xstp, ystp, mem, nx, status)
      if(status.ne.0) ...
      ...
      end
```

1.5 Lecture d'image

La lecture se fait avec la fonction `aff_extract()` qui lit la totalité (ou une partie) de l'image de l'afficheur. Les arguments sont les mêmes que ceux de la fonction `aff_patch()` à la seul différence que le pas des deux interlocuteurs n'entre pas en jeux (transfert de pixel). Exemple:

```
program xxxx

integer    connect_aff

real       mem(200,200)
integer    dx, dy, x, y, ncol, nlin, nx
...
call aff_extract(connect_aff, dx, dy, x, y, ncol, nlin, mem, nx, status)
if(status.ne.0) ...
...
end
```

1.6 Coordonnée utilisateur

`xaff` affiche les coordonnées utilisateur selon le déplacement du curseur. Ces coordonnées sont calculées par rapport à la coordonnée du pixel bas–gauche et au pas selon X et selon Y. Ces coordonnées et le pas peuvent être modifiés avec la fonction `aff_set_coord()` et lue avec la fonction `aff_get_size()` . Exemple:

```
#include <gop.h>
int    actual_status=0;

main()
{
   struct gop_connect *connect_aff;
   int                ncol, nlin;
   float              x, y, sx, sy;

   if (aff_client(&connect_aff, actual_status, option, delay, repeat)
              != GOP_OK){...}
   ...
   if (aff_set_coord(connect_aff, x, y, sx, sy) != GOP_OK)
      {...}
   ...
   if (aff_get_size(connect_aff, ncol, nlin, x, y, sx, sy) != GOP_OK)
      {...}
   ...
}
```

1.7 Curseur et centrage

`aff_read_cursor()` permet le lire la position du curseur au moment d'un click sur celui–ci, ainsi que la valeur du pixel sous le curseur et le numéro du bouton de souris selectionné (`1=gauche, 2=milieu, 3=droite`)

```
main()
{
   struct gop_connect *connect_aff;
   int    x, y, ret;
   float  xw, yw, value;
   ...
   if (aff_read_cursor(connect_aff, &x, &y, &xw, &yw, &value, &ret) != GOP_OK)
      {...}
   printf("coord pixel <%d;%d>, coord utilisateur <%f;%f>\n",x, y, xw, yw);
   printf("valeur pixel = %f,   souris bouton %d\n",value, ret);
   ...
}
```

Le curseur peut être placé sur l'image avec `aff_set_cursor()` selon des coordonnées utilisateur. Si la coordonnées est hors fenêtre, mais dans l'image, l'image est recentrée. Exemple:

```
main()
{
   struct gop_connect *connect_aff;
   float  xw, yw;
   ...
   if (aff_set_cursor(connect_aff, xw, yw) != GOP_OK)
      {...}
   ...
}
```

Si le curseur ne doit pas être posé mais que la coordonnée doit être visible, la fonction `aff_image_center()` centrera l'image si nécessaire. Exemple:

```
program xxxx

integer    connect_aff
real       xw, yw

...
call aff_image_center(connect_aff, xw, yw, status)
if(status.ne.0) ...
...
end
```

1.8 Zoom

Le zoom d'une position donnée en coordonnées utilisateur se fait avec `aff_set_zoom()`. La taille de la fenêtre zoom (en pixel écran) ainsi que sa position se donnent avec `aff_set_size_zoom()` et `aff_position_zoom()`. Si le facteur de zoom est donné plus petit ou égal à zéro, alors le facteur de zoom du panneau de contrôle est pris. Exemple:

```
main()
{
   struct gop_connect *connect_aff;
   float  xw, yw;
   int    xpos, ypos, ncol, nlin, fact;
   ...
   if(aff_position_zoom(connect_aff, xpos, ypos) != GOP_OK)
       {...}
   if(aff_set_size_zoom(connect_aff, ncol, nlin) != GOP_OK)
       {...}
   if(aff_set_zoom(connect_aff, xw, yw, fact) != GOP_OK)
       {...}
   ...
}
```

1.9 Position et taille de l'image

La fenêtre image se positionne avec la fonction `aff_position_image()`. Exemple:

```
program xxxx

integer    connect_aff
integer    x, y

...
call aff_position_image(connect_aff, ncol, nlin, status)
if(status.ne.0) ...
...
end
```

On initialise une image en lui donnant sa taille avec la fonction `aff_set_size_image`. Si une image est déjà présente, elle est effacée. Cette fonction permet également de donner la taille de la fenêtre contenant l'image.
Si les arguments concernant la taille de l'image sont posé à zéro, alors la taille de l'image est posée à 416∗578 (taille ccd du Chili). S'ils sont négatif la taille n'est pas changée, l'image est conservée et seul la taille de la fenêtre est modifiée. Exemple:

```
main()
{
   struct gop_connect *connect_aff;
   int    hscroll, vscroll;
   ...
   if(aff_set_size_image(connect_aff, -1, -1, hscroll, vscroll) != GOP_OK)
       {...}
   ...
}
```

1.10 Ouverture et fermeture

La fonction `aff_iconify()` met `xaff` sous forme d'icone. La fonction `aff_deiconify()` ouvre l'icone. La fonction `aff_hide_image()` dépunaise l'image. La fonction `aff_show_image()` faire réaparaître l'image dépunaisée. Exemple

```
main()
{
   struct gop_connect *connect_aff;
   int    hscroll, vscroll;
   ...
   if(aff_iconify(connect_aff) != GOP_OK)
       {...}
   if(aff_deiconify(connect_aff) != GOP_OK)
       {...}
   if(aff_hide_image(connect_aff) != GOP_OK)
       {...}
   if(aff_show_image(connect_aff) != GOP_OK)
       {...}
   ...
}
```

1.11 Effacement et raffraîchissement

L'image s'efface avec la fonction `aff_clear_window()` et se raffraîchit avec `aff_redisplay()`. Le rafraîchissement permet de recalculer l'image après un changement de cuts ou d'effacer les symboles. Exemple:

```
      program xxxx

      integer    connect_aff
      ...
      call aff_clear_window(connect_aff, status)
      if(status.ne.0) ...
      ...
      call aff_redisplay(connect_aff, status)
      if(status.ne.0) ...
      ...
      end
```

1.12 LUT, ITT et cuts

On selectionne une LUT ou une ITT par leur numéro d'ordre avec les fonctions `aff_load_lut()` et `aff_load_itt()`. La mise à jour des cuts se fait avec la fonction `aff_set_cuts()`. (il faut exécuter la fonction `aff_redisplay()` pour recalculer l'image.
La lecture de la lut courante se fait avec `aff_extract_lut()`. Exemple:

```
main()
{
   struct gop_connect *connect_aff;
   int    no;
   int    red[128], green[128], blue[128];
   float  lcuts, hcuts;
   ...
   if(aff_load_itt(connect_aff, no) != GOP_OK)
       {...}
   if(aff_load_lut(connect_aff, no) != GOP_OK)
       {...}
   if(aff_set_cuts(connect_aff, lcuts, hcuts) != GOP_OK)
       {...}
   if(aff_redisplay(connect_aff) != GOP_OK)
       {...}
   ...
   if(aff_extract_lut(connect_aff, &size, red, green, blue, &lcuts, &hcuts)
       != GOP_OK) {...}
   ...
}
```

1.13 Symbole

On pose un symbole à une coordonnée utilisateur avec la fonction `aff_write_symbol_pixel() pour la taille des symboles données en pixels et aff_write_symbol_world() pour la taille des symboles donnés en coordonnées utilisateur. On précise: le type (1=croix centrée, 2=carré centré, 3=carré plein centré, 4=cercle centré, 5=rectangle centré, 6=carré positioné selon le coin bas--gauche, 7=rectangle positioné selon le coin bas--gauche), la taille selon X et Y, l'epaisseur du trait (en pixel écran) et la couleur (0=blanc, 1=noir, 2=traitillé).`

`On pose un texte à une coordonnée utilisateur avec la fonction aff_write_text(). On précise: le centrage (1=NE, 2=SE, 3=SW, 4=NW, 5=N, 6=E, 7=S, 8=W), la couleur (0=blanc, 1=noir), la taille du symbole associé (pour eviter le recouvrement du symbole s'il existe), son type et naturellement le texte. Exemple`

```
main()
{
   struct gop_connect *connect_aff;
   float  x, y;
   int    type, size[2] ,color, width, centrage;
   float  world_size[2];
   char   *texte;
   ...
   if(aff_write_symbol_pixel(connect_aff, x, y, type, size, width, color) != GOP_OK)
       {...}
   ...
   if(aff_write_symbol_world(connect_aff, x, y, type, world_size,
                             width, color) != GOP_OK)
       {...}
   ...
   if(aff_write_text(connect_aff, x, y, centrage, color, size, type, texte)
!= GOP_OK)
       {...}
   ...
}
```

On peut également poser un vecteur de points avec la fonction aff_put_multi_points().
Exemple:

```
main()
{
   struct gop_connect *connect_aff;
   float  x[SIZE_MAX], y[SIZE_MAX];
   int    color;
   ...
   if(aff_put_multi_points(connect_aff, x, y, size, color) != GOP_OK)
       {...}
   ...
}
```

1.14 Traitement des erreurs

Un status de retour différent de zéro indique une erreur. Le numéro d'une erreur de transmission est lut avec la fonction aff_get_error_number(), le message avec aff_get_error_string().
On peut également tester si l'erreur est due à une interruption de la communication (broken pipe) avec la fonction aff_is_broken_pipe(). Exemple:

```
program xxxx

integer       connect_aff
integer       pipe, no, ilen
character*80  message
...
call aff_read_cursor(connect_aff, ...)
if(status.ne.0) then
  call aff_is_broken_pipe(pipe)
  if(pipe)then
     write(*,*) deconnection avec l'afficheur
     actual_status = 0
     goto 8888
  else
     call aff_get_error_number(no)
     call aff_get_error_string(message, ilen)
     write(*,'(i,a)') no, message(1:ilen)
     goto 9999
  endif
endif
...
end
```

1.15 Verbosité de GOP

Le passage des messages entre le client et xaff peut être visualisé avec la fonction aff_set_verbose() en donnant un niveau de verbosité: 0, 1, 2 et 3 à 9 (voir libgop.a).
Exemple:

```
program xxxx

integer       connect_aff
...
call aff_set_verbose(connect_aff, 2)
...
end
```