SDB

Luc Weber
Observatoire de Genève

24 octobre 2012

Table des matières

Chapitre 1

Introduction

SDB regroupe un ensemble de logiciels permettant gérer une base de données de type DBM (Unix). Ce type de base gère l'accès a un champ de type caractère que l'on accède par une clé, elle aussi de type caractère.

Le principal outil est le serveur *XSDBManager*. C'est l'interlocuteur de tout les clients utilisant la base de données

XSDBManager permet de créer un outil de type moniteur d'état. Une de ses utilisations possible est de permettre à un serveur d'instrument de fonctionner en simulateur, c'est à dire pouvoir retourner à son client des données instrumentales lorsque l'instrument n'est pas connecté. Dans ce cas, celles–ci sont recherchées dans la base de données et c'est l'utilisateur, au travers de *XSDBManager* qui peut agir sur la réponse de instrument.

Les logiciels *SDB* sont :

xsdbmanager	manager de base de données sous OpenWindows
libsdb.a	librairie d'accès au manager
sdbset	commande Unix d'assignation de couple (clé-contenu)
xsdbset	utilitaire interactif sous OpenWindows pour assigner la base
sdbread	commande Unix de lecture d'un champ
sdbshow	commande Unix d'extraction du contenu de la base

Tous ces logiciels communiquent entre eux avec le protocole de communication GOP. Ce protocole utilise actuellement des sockets de type Unix (communication sur la même machine) ou Inet (entre deux machines). Le mode de travail par défaut de l'ensemble de ces logiciels se fait sur les socket Unix avec une socket nommée "socket.sdb" pour "sdbset" "sdbread" "sdbshow" et "xsdbset" et une socket nommée "socket.*init_file*" pour les manager "*XSDBManager*". Les options permettent de changer le type de communication et le nom du socket.

Le principe de fonctionnement est le suivant : la base de données est gérée par un programme de type "manager" (*XSDBManager*). Si on l'initialise avec un fichier d'initialisation ASCII, il crée une nouvelle base de donnée. Si on l'initialise avec une base de données, il fonctionne avec cette base de données.

Après cette initialisation, *XSDBManager* se met en attente de clients désirant accéder la base. Il répond aux ordres suivant :

1. **read** <**clé**> **:** lecture d'une valeur ; on donne la clé et on récupère son contenu.

2. **rsim** <**clé**> **:** lecture d'une valeur et des données associées (cas spécial) ; on donne la clé et on récupère son contenu ainsi que les contenu des variables <clé>_flag, <clé>_error et <clé>_time.
3. **write** <**clé**> <**contenu**> **:** écriture d'une valeur ; on donne une clé (qui peut être nouvelle) et un contenu. La base est mise à jour.
4. **stat :** lecture de la base complète ; on reçoit l'ensemble des couples (le séparateur est l'espace) séparés par des *linefeed.*
5. **timer** <**timer_command**> <**timer_value**> **:** enregistrement d'un timer. Au bout du temps définis par <timer_value>, *XSDBManager* envoie la commande <timer_command> au client qui l'a envoyé. Cela permet de faire de *XSDBManager* un client, pouvant ainsi d'une manière asynchrone demander des demandes de rafraîchissement (si ce type de commande est valide sur le client, qui est un serveur d'instrument par exemple).
6. **setstruct** <**No**> <**params**> **... :** définis les caractéristiques de la structure dans le mode de transfert par structure.
7. **wrstruct** <**No**> <**struct**> **... :** écrit la structure dans le mode de transfert par structure.

Remarque : la librairie *LibSDB* contient les fonctions interfaces pour les accès à *XSDBManager.*

L'accès à la base est sans priorité, les clients sont pris dans l'ordre des demandes.

1.1 Le fichier d'initialisation

Ce fichier est de type ASCII. Il contient les définitions permettant de créer un panneau de commande.
Un panneau de commande a une organisation verticale pour les variables et horizontale pour les variables associées.

Le fichier d'initialisation permet de :
– définir une variable (<NAME>)
– définir une variable associée (<TAB><NAME>)
– définir les emplacement des colonnes des variables associées (%MARGIN :<marge_1> :<...>)
– changer de colonne et lui donner un titre (%COLUMN :<name>)
– faire des sections et leur donner un titre (%SECTION :<name>)
– de changer l'interligne (%YGAP : :<nb_pixel>)
– de changer l'intercolonne (%XGAP :<nb_pixel>)
– definir la largeur des labels (%LABEL :<nb_caractères>)
– definir la largeur des contenu (type PROMPT) (%CONTENT :<nb_caractères>)
– arréter la construction d'objet, mais en continuant à les mettre dans la base (%END)
– reprendre la construction d'objet (%RESTART)
– traitement conditionnel, execute si défini (%IFDEF <str>)
– traitement conditionnel, execute si non définis (%IFNDEF <str>)
– fin de traitement conditionnel (%ENDIF)
– début de zone readonly (%READONLY)
– fin de zone readonly (%WRITEENABLE)

Apres la description d'une variable ou d'une variable associée, on définis le type d'objet associé :

```
# PROMPT[:<label>|-[:<defaut>|-[:<nbcar>|-[:suffix|-]]]]
# SLIDER[:<label>|-[:<defaut>|-[:<long>|-[:<min>|-[:<max>|-[:V|H[:R|-
        [:<ntics>|-]]]]]]]
# GAUGE[:<label>|-[:<defaut>|-[:<long>|-[:<min>|-[:<max>|-[:V|H[:R|-
        [:<ntics>|-]]]]]]]:
# NUMERIC[:<label>|-[:<defaut>|-[:<long>|-[:<min>|-[:<max>|-]]]]]
# CHOICE:<label>|-:<defaut>|-:<arg1>:[...:[<argn>]][:END[:V|H]]
# NCHOICE:<label>|-:<defaut>|-:<arg1>:[...:[<argn>]][:END[:V|H]]
# CHMENU:<label>|-:<defaut>|-:<arg1>:[...:[<argn>]][:END[:V|H]]
# CHECK:<label>|-:<defaut>|-:<arg1>[...:[<argn>]][:END[:V|H]]
# LIST:<label>|-:<fichier>|-[:<haut>|-[:<larg>|-]]
```

avec

```
#  <label>   : texte de l'objet.
#                  S'il vaut "-", il n'y a pas de texte
#                  S'il vaut "$", le texte est <NAME> ou <SUB> selon le cas
#  <defaut>  : valeur d'initialisation
#  <nbcar>   : nombre de caracteres du champ
#  <min>     : valeur limite
#  <max>     : valeur limite
#  <argn>    : argument d'un choix
#  <fichier> : fichier pour la liste scrollable
#  <haut>    : nb de lignes de la fenetre scrollable
#  <larg>    : nb de caractere de la fenetre scrollable
#  <ntics>   : nb de tics sur le slider ou la gauge (defaut: 0==pas de tic)
#  "V"       : layout Vertical
#  "H"       : layout Horizontal (defaut)
#  "R"       : Affiche les limites sur le slider ou la gauge
```

La description complète est actuellement faite dans le fichier /usr/local/lib/skeleton.sdb.
Exemple :

```
%READONLY
LED:CHOICE:Diode pre flash:Off   :Off   :On    :
>Commande: LED /ON /OFF
>spectro: Diode pre flash (LED)
>Off  == 0 == teint
>On   == 1 == allume
        state:CHOICE:-:UNCAL:OK:MOVING:DEFECT:UNCAL:
>état réel de élément
%IFDEF SIMUL
%WRITEENABLE
        flag:CHECK:-:0
>si la boite est cochée, le simulateur retourne une erreur avec comme
>argument la valeur à droite de la boite
        error:CHOICE:-:0:0:1
>code d'erreur
        time:SLIDER:-:0:50:0:60
>timeout pour l'attente sur cette fonction (en secondes)
%ENDIF
```

Remarque : Les lignes commancant par un "$>$" sont les commentaires associés a la variable. Pour les afficher, il suffit de pointer le chmap sur le panneau et appuyer la touche HELP.

Chapitre 2

xsdbmanager

XSDBManager est un manager de base de données possédant un panneau de contrôle. Il peut gérer plusieurs base de données à la fois et afficher leurs contenus sous différentes formes (widgets) sous OpenWindows.
A chaque accès d'un client, la base et l'affichage est mis à jour.
XSDBManager lit pour chaque base de données et dans tout les cas un fichier d'initialisation ASCII décrivant le contenu du panneau et facultativement un couple de fichier dbm qui permet l'initialisation des champs selon leurs valeurs lors de la session précédente. Si le fichier dbm n'est pas donné les valeurs d'initialisation sont prises dans le fichier d'intialisation.
L'extension du fichier d'initialisation ASCII est "`.init`". Elles sont "`.dir`" et "`.pag`" pour le couple de fichier dbm. Lors de l'appel à *XSDBManager*, on ne précise pas les extensions. La description complète est actuellement faite dans le fichier /usr/local/lib/skeleton.sdb.

Les widgets se mettent naturellement en colonnes, lorsque la hauteur maximum est atteinte, une nouvelle colonne est crée.

Le fichier d'initialisation contient des lignes de definitions, de help (>), de commande(%) et des ligne de commentaires (#) ;

Les commandes permettent de modifier l'apparence du panneau de contrôle.
XSDBManager initialise des sockets de type Internet et de type Unix. On peut changer les valeurs par défaut :

Syntaxe :

```
xsdbmanager [<opt_base_1>] -f <init_file> | -d <dbm_file>  [<opt_base_2> ...
```

```
Options connexion et base de donnée:
        -f <init_file>
                base de donnée ASCII pour l'initialisation. Pas de défaut
                Les options pour cette base doivent avoir été données avant.
```

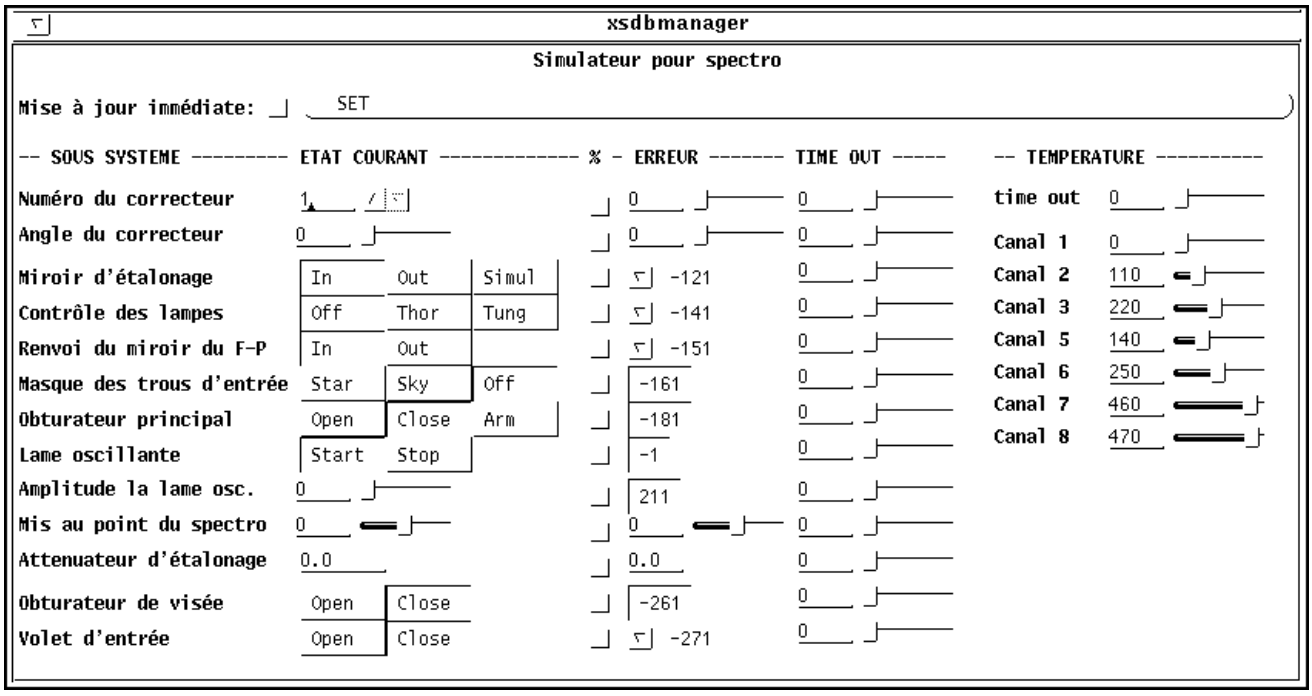
```
                Le panneau a ce nom et la base de données nommée
                (<init_file>.dir <init_file>.pag) est reconstruite
        -d <dbm_file>
                base de donnée dbm pour l'initialisation. Pas de défaut
                Les options pour cette base doivent avoir été données avant.
                Le fichier ascii <dbm_file> et pris pour le layout et la
                base dbm doit exister (<dbm_file>.dir <dbm_file>.pag). Elle
                est prise pour initialiser les champs.
                Le panneau a ce nom.
        -p <internet_socket_port_number>
                pour connexion socket Internet. (Entre machines)
                Défaut: 6182 pour la première base puis incremental.
        -n <unix_socket_name>
                pour connexion socket Unix. (Sur la même machine)
                Défaut: "<init_file>" ou "<dbm_file>".
                Fabrique la socket: "socket.<unix_socket_name>."

Options standard:
        -c <Unix_command>
                Crée un pipe sur <Unix_command> et y envoie le log des accès
                en écriture à la base de donnée
        -i <key>
                définition d'une clé pour l'exécution conditionnelle (IFDEF)
        -D
                mode Debug ON (dans fenêtre log interne).
        -m <max_pixel>
                taille maximum d'un panneau selon l'axe vertical donné en pixel.
                Une fois cette dimension atteinte, un nouvelle colonne est crée.
        -u
                initialise le mode de mise a jour automatique.
        -h
                Ce message de help.

Format du fichier d'initialisation:
        voir fichier /usr/local/lib/skeleton.sdb
```

Attention : les clients accèdent toujours la base de données et non le contenu des champs affichés. Il est donc nécessaire de valider l'ensemble des champs au moyen du bouton "SET" pour que la base soit mise à jour, le mode update automatique permet de mettre la base à jour lors de chaque modifications.

Exemple panneau avec communication sur socket Unix ("socket.spectro") de la base "spectro" :

```
unix> xsdbmanager -f spectro.init
```

On obtient :



Chapitre 3

La librarie libsdb.a

Cette librarie est utilisée par les clients qui veulent se connecter sur un manager (*XSDBManager*). Elle contient les fonction de connexion, c'est à dire :

3.1 Connexion par socket Internet

```
#include <gop.h>

sdb_open_internet(sdb_connect)
        struct gop_connect *sdb_connect;
```

Connexion par socket Unix :

```
#include <gop.h>

sdb_open_unix(sdb_connect)
        struct gop_connect *sdb_connect;
```

3.2 Les fonctions d'écriture

```
#include <gop.h>

sdb_write        (sdb_connect, key, svalue)
sdb_write_int    (sdb_connect, key, ivalue)
sdb_write_long   (sdb_connect, key, lvalue)
sdb_write_llong  (sdb_connect, key, llvalue)
sdb_write_float  (sdb_connect, key, fvalue)
sdb_write_double (sdb_connect, key, dvalue)

avec:
        struct gop_connect *sdb_connect;
        char           *key;
        char           *svalue;
        int             ivalue;
        long            lvalue;
        long long       llvalue;
        float           fvalue;
        double          dvalue;
```

3.3 Les fonctions de lecture

```
#include <gop.h>

sdb_read        (sdb_connect, key, svalue, size_of_svalue)
sdb_read_int    (sdb_connect, key, ivalue)
sdb_read_long   (sdb_connect, key, lvalue)
sdb_read_llong  (sdb_connect, key, llvalue)
sdb_read_float  (sdb_connect, key, fvalue)
sdb_read_double (sdb_connect, key, dvalue)

avec:
        struct gop_connect *sdb_connect;
        char           *key;
        char           *svalue;
        int             size_of_svalue;
        int            *ivalue;
        long           *lvalue;
        long long      *llvalue;
        float          *fvalue;
        double         *dvalue;
```

3.4 La fonction de status

```
#include <gop.h>

sdb_stat(sdb_connect, str, size_of_str)

avec:
        struct gop_connect *sdb_connect;
        char           *key;
        char           *str;
        int             size_of_str;
```

3.5 Les fonction de simulations

Pour accélérer les accès à la base, la serie de routines suivante retourne la valeur d'une variable et de ses variables associées. Ces variables sont au nombre de trois et elles sont préfixées par le nom de la variable. Par exemple, pour la variable "`SHUTTER`" on a "`SHUTTER_flag`" qui indique si la requête en lecture retourne la variable (==0) ou la variable associée "`SHUTTER_error`", "`SHUTTER_error`" la variable contenant la valeur de simulation, et "`SHUTTER_time`" contient la durée du time–out nécessaire à l'accomplissement de la commande.
Donc, les fonction de simulations sont :

```
#include <gop.h>

sdb_read_simul        (sdb_connect, key, flag, svalue, time)
sdb_read_int_simul    (sdb_connect, key, flag, ivalue, time)
sdb_read_long_simul   (sdb_connect, key, flag, lvalue, time)
sdb_read_llong_simul  (sdb_connect, key, flag, llvalue, time)
sdb_read_float_simul  (sdb_connect, key, flag, fvalue, time)
sdb_read_double_simul (sdb_connect, key, flag, dvalue, time)

avec:
        struct gop_connect *sdb_connect;
        char           *key;
        char           *svalue;
        int            *ivalue;
        long           *lvalue;
        long long      *llvalue;
        float          *fvalue;
        double         *dvalue;
        int             time
```

3.6 La fonction timer

Un client de *XSDBManager*, s'il est déjà de type serveur pour d'autres application (ex : serveur d'instrument) peut également fonctionner comme serveur de *XSDBManager*. Cela permet notament d'avoir un mode de rafraichissement demandé par *XSDBManager* mais sous la direction du serveur d'instrument.

Dans cette situation, le serveur d'instrument demande à *XSDBManager* de lui envoyer une commande (choisie par le serveur d'instrument) après un certain laps de temps. Cette commande sera traitée par le serveur d'instrument sans aucune priorité vis–à–vis des ses autres clients. Ansi le serveur d'instrument peut choisir la vitesse de rafraichissement en fonction de son occupation. De plus cette technique permet d'avoir à tout moment une seule demande de rafraissement en attente.

Pour executer ce type de controle, il est nécessaire d'avoir un second cannal de communication entre le client et *XSDBManager*. Son initialisation est identique au premier canal à cette différence :

– Pour les connexions internet, le numéro de port du canal timer est le numéro de port du canal de base plus un.
– Pour les connexions unix, le nom du socket se termine par "_timer".

Son appel est le suivant :

```
#include <gop.h>

sdb_set_timer(sdb_connect, cmd, value)

avec:
        struct gop_connect *sdb_connect;
        char           *cmd;
        int             value;
```

Exemple :

```
     struct gop_connect *connect_sdb;
     struct gop_connect *connect_sdb_timer;
     ...
     memcpy(connect_sdb_timer, connect_sdb, sizeof(struct gop_connect));
     gop_set_port(connect_sdb_timer, gop_get_port(connect_sdb) + 1);
     sdb_open_internet(connect_sdb);
     sdb_open_internet(connect_sdb_timer);
```

3.7 Ecriture de structure

Pour accélerer la communication, on peut passer des structures. Ces structure contiennent uniquement les données mais pas les clés. De plus *XSDBManager* a la capacité de formater les données binaire. Chaque donnée a un flag associé qui indique si la donnée est valide. Si elle l'est, la donnée est formatée par le format "normal"associé à cette valeur, si elle est invalide, le format d'erreur associé à cette donnée est utilisé.
ATTENTION, le binaire du serveur et du client doivent être compatible. *XSDBManager* modifie uniquement les données qui ont changé depuis le dernier envoi de structure.
Comme les clés ne sont pas dans la structure, il faut les préciser grâce à un appel permettant d'enregistrer la description d'une structure. On peut enregistrer plusieurs structure, chacune ayant un identificateur unique (valeur numérique entière plus grande que zéro).
Chaque donnée a un type, celui ci permet de récupérer la donnée dans la structure et de la formatter de manière adéquate. En plus des type "char", "long", "long long" "float" et "double", on peut également transformer des "long" ou "long long" donnés en millisecondes ou en microsecondes en "heure–minutes-secondes" ou en "degré–minutes-secondes". On peut également transformer des "float" ou "double" donnéus en Radian ou en

Degré en "heure–minutes-secondes" ou en "degré–minutes–secondes" et finalement on peut également formater les 8, 16 ou 32 bits du bas d'un "long" sous une représentation binaire. Les type sont les suivant (définis dans sdb.h) :

```
#define   SDB_TYPE_CHAR              1
#define   SDB_TYPE_LONG              2
#define   SDB_TYPE_LLONG             3
#define   SDB_TYPE_FLOAT             4
#define   SDB_TYPE_DOUBLE            5
#define   SDB_TYPE_L_MICRO_S_HMS     6
#define   SDB_TYPE_L_MILLI_S_HMS     7
#define   SDB_TYPE_L_MICRO_S_DMS     8
#define   SDB_TYPE_L_MILLI_S_DMS     9
#define   SDB_TYPE_LL_MICRO_S_HMS   10
#define   SDB_TYPE_LL_MILLI_S_HMS   11
#define   SDB_TYPE_LL_MICRO_S_DMS   12
#define   SDB_TYPE_LL_MILLI_S_DMS   13
#define   SDB_TYPE_F_RAD_HMS        14
#define   SDB_TYPE_F_RAD_DMS        15
#define   SDB_TYPE_F_DEG_HMS        16
#define   SDB_TYPE_F_DEG_DMS        17
#define   SDB_TYPE_D_RAD_HMS        18
#define   SDB_TYPE_D_RAD_DMS        19
#define   SDB_TYPE_D_DEG_HMS        20
#define   SDB_TYPE_D_DEG_DMS        21
#define   SDB_TYPE_L_BIN_BYTE       22
#define   SDB_TYPE_L_BIN_SHORT      23
#define   SDB_TYPE_L_BIN_LONG     24
```

Lors de l'enregistrement, on precise :
- l'identificateur
- le nombre d'element dans la structure
- la taille en bytes de la structure
- la liste des clés
- la liste des types
- la liste des offsets des données dans la structure
- la liste des formats
- la liste des formats d'erreur

Son appel est le suivant :

```
#include <gop.h>
#include <sdb.h>

int
sdb_set_struct(struct gop_connect *sdb_connect,
               int     id,
               int     nb_elem,
               int     size,
               char  **key,
               int    *type,
               int    *offset,
               char  **fmt,
               char  **fmt_err)
```

L'ecriture de la structure se fait avec :

```
#include <gop.h>
#include <sdb.h>

int
sdb_write_struct(struct gop_connect *sdb_connect,
                 int     key,
                 char   *struct,
                 int     size)
```

Exemple :

```
#define   TEST_STRUCT   1

long            sdb_test_nb_elem = 16;

struct sdb_test_content {

   long            invalid_v_long;
   long            v_long;
   long long       invalid_v_llong;
   long long       v_llong;
   long            invalid_v_float;
   float           v_float;
   long long       invalid_v_double;
   double          v_double;

}               test;
```

```
char           *sdb_test_key[] = {
   "V_LONG",
   "V_LLONG",
   "V_FLOAT",
   "V_DOUBLE",
};
```

```
int             sdb_test_type[] = {
   SDB_TYPE_LONG,
   SDB_TYPE_LL_MICRO_S_HMS,
   SDB_TYPE_FLOAT,
   SDB_TYPE_D_DEG_HMS
};
```

```
char            *sdb_test_fmt[] = {
   "%s %d",
   "%s %4dh %.2dm %05.2fs",
   "%s %f",
   "%s %c%3dd %.2dm %05.2fs",
};
```

```
char            *sdb_test_fmt_err[] = {
   "%s **",
   "%s **",
   "%s **",
   "%s **",
};
```

```
int              sdb_test_offset[] = {
   (long) &test.invalid_v_long - (long) &test,
   (long) &test.invalid_v_llong - (long) &test,
   (long) &test.invalid_v_float - (long) &test,
   (long) &test.invalid_v_double - (long) &test,
};
```

```
   if (sdb_set_struct(&connect_sdb, TEST_STRUCT, sdb_test_nb_elem,
            sizeof(struct sdb_test_content),
            sdb_test_key, sdb_test_type, sdb_test_offset,
            sdb_test_fmt, sdb_test_fmt_err) != GOP_OK)
      manage_gop_error("sdb_set_struct");

   if (sdb_write_struct(&connect_sdb, TEST_STRUCT,
       (char *) &test, sizeof(struct sdb_test_content)) != GOP_OK)
      manage_gop_error("sdb_write_struct");
```

3.8 Loging et debugging

Tout les messages en provenance de la librarie sont envoyé sur stderr. L'utilisateur peut enregistrer une fonction personalisée qui sera subtilisée à la fonction d'écriture. (Voir documentation logbook). Cette fonction demande comme argument une fonction qui recevra comme argument la chaîne de caractère formant le message (déjà terminée par un NEWLINE) :

```
void
lbk_registration_for_warning(fct)

avec:
        void            (*fct) ();

exemple):

{...
        lbk_registration_for_warning(print_on_logbook);
...}

print_on_logbook(str)
     char           *str;
{
     if (log_book_status != -1) {
          log_book_status = lbk_write_log_book(str);
     } else {
          print_on_stdout(str);
     }
}
```

Le debugging de la librarie est activé ou désativé par l'appel a la fonction

```
sdb_set_flag_debug(TRUE);    /* pour activer */
sdb_set_flag_debug(FALSE);   /* pour desactiver */
```

Chapitre 4

sdbshow

SDBShow est une commande qui permet recuperer sur `stdout` le contenu de la base.

Syntaxe :

```
xsdbshow [<options>]
```

Ses options sont les suivantes :

```
Options connexion:
                Connexion par défaut sur socket Unix (Sur la même machine).
                Nom: "socket.sdb".

        -u -:<unix_socket_name>:<gop_mode>
                pour connexion socket Unix. (Sur la même machine)
                Défaut: -:sdb:0.
                Fabrique la socket: "socket.<unix_socket_name>."
        -i <socket_port_number>:<destination_host_name>:<gop_mode>
                pour connexion socket Internet. (Entre machines)
                Défaut: 6182:localhost:0.
        -h
                Ce message de help.
```

Chapitre 5

sdbset

SDBSet est une commande qui permet de mettre à jour la base. On donne sur la ligne de commande (après les options) un ensemble de couples <clé>–<contenu> que l'on veut mettre à jour.

Syntaxe :

```
sdbset [<options>] <clé> <contenu> [<clé> <contenu> ...]
```

Les options sont les suivantes :

```
Options connexion:
                Connexion par défaut sur socket Unix (Sur la même machine).
                Nom: "socket.sdb".

        -u -:<unix_socket_name>:<gop_mode>
                pour connexion socket Unix. (Sur la même machine)
                Défaut: -:sdb:0.
                Fabrique la socket: "socket.<unix_socket_name>."
        -i <socket_port_number>:<destination_host_name>:<gop_mode>
                pour connexion socket Internet. (Entre machines)
                Défaut: 6182:localhost:0.
        -h
                Ce message de help.
```

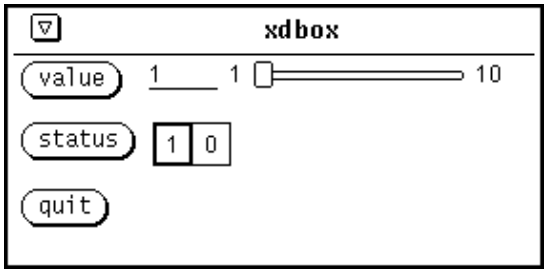
5.1 sdbset & XDBox

Avec *XDBox* on peut se fabriquer des panneaux OpenWindows contenant d'autres objets que des *Prompts*. Si par exemple on désire une fenêtre contenant un *Slider* avec des valeurs comprises entre 1 et 10 attribué à la clé

nommée "value" et un *Choice* avec des valeurs "1" et "0" attribué à la clé nommée "status". On lancera :

```
unix> xdbox -n  value:-:"sdbset":ECHO:SLIDER:-:1:100:1:10 \
               status:-:"sdbset":ECHO:CHOICE:-:0:1:0      \
                 quit:-:-:-:EXIT
```

Pour obtenir :



Chaque champ peut alors être mis à jour en cliquant sur le bouton correspondant.

Chapitre 6

xsdbset

XSDBSet est un utilitaire qui permet de mettre à jour la base de manière interactive. On lui donne de manière optionnelle en entrée un fichier contenant une liste de clé existant dans le fichier d'initialisation de *XSDBManager* et/ou une série de clés. Il fabrique un panneau de contrôle sous OpenWindows avec une ligne d'entrée (de type PROMPT) pour chaque clé. *XSDBSet* se connecte sur *XSDBManager* et modifie une clé chaque fois qu'un champ est validé d'un "*return*". La base peut être relue et les champs mis à jour grâce au bouton READ.

Syntaxe :

```
xsdbset [<options>] [<clé> ...]
```

Les champs sont positionnés en colonnes, lorsque la hauteur maximum est atteinte, une nouvelle colonne est crée.

Si des noms de paragraphes sont donnés (commencent par un "%"), alors une nouvelle colonne est crées et le nom du paragraphe y est écrit.

Les champ donnés après le paragraphe "%END" ne sont pas affichés.

La taille des champs et leurs espacements peuvent être changés par options.

Les options de *XSDBSet* sont :

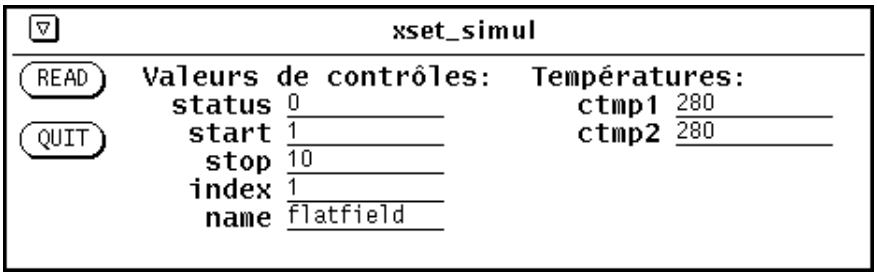
```
connexion:
                Connexion par défaut sur socket Unix (Sur la même machine).
                Nom: "socket.sdb".

        -u -:<unix_socket_name>:<gop_mode>
                pour connexion socket Unix. (Sur la même machine)
                Défaut: -:sdb:0.
```

```
                Fabrique la socket: "socket.<unix_socket_name>."
        -i <socket_port_number>:<destination_host_name>:<gop_mode>
                pour connexion socket Internet. (Entre machines)
                Défaut: 6182:localhost:0.

Options base de donnée:
        -f <init_file>
                base de donnée ASCII pour l'initialisation. Pas de défaut
                Le panneau a ce nom.

Options layout:
        -m <max_pixel>
                taille maximum du panneau selon l'axe vertical donné en pixel.
                Une fois cette dimension atteinte, un nouvelle colonne est crée.
        -x <item_x_gap>
                Espace entre les colonnes en pixels. Défaut: 5.
        -y <item_y_gap>
                Espace entre les lignes en pixels. Défaut: 0.
        -l <label_width>
                taille des labels en pixels. Défaut: 70.
        -c <content_width>
                taille des labels en nb de caractères. Défaut: 12.
        -h
                Ce message de help.
Format du fichier d'initialisation:
        #<commentaire>
        %<titre de colonne>
        <key> ...
        ...
```

Exemple : Avec le fichier et la commande :

```
unix> cat camera.set
# fichier d'initialisation pour la caméra CCD de guidage
%Valeurs de contrôles:
status
start
stop
index
name
%Températures:
ctmp1
ctmp2
unix> xsdbset -i 5555 -f camera.set
```

On obtient :



Chapitre 7

sdbread

SDBRead est une commande qui permet lire une clé dans la base. On donne sur la ligne de commande (après les options) le nom de la clé et et on récupere son contenu sur `stdout`

Syntaxe :

```
sdbread [<options>] <clé> ...
```

Les options sont les suivantes :

```
Options connexion:
                Connexion par défaut sur socket Unix (Sur la même machine).
                Nom: "socket.sdb".

        -u -:<unix_socket_name>:<gop_mode>
                pour connexion socket Unix. (Sur la même machine)
                Défaut: -:sdb:0.
                Fabrique la socket: "socket.<unix_socket_name>."
        -i <socket_port_number>:<destination_host_name>:<gop_mode>
                pour connexion socket Internet. (Entre machines)
                Défaut: 6182:localhost:0.
        -h
                Ce message de help.
```

Exemple :

```
unix> setenv STATUS ‘sdbread -u -:camera status‘
```