The REBOL Documentation Project

-- FR - Documentation REBOL - Articles Techniques --

Articles Techniques

Interfacer Rebol et les bibliothèques dynamiques

Bouba

Première publication : 8 juillet 2005, et mis en ligne le vendredi 8 juillet 2005

Résumé :

Cet article a pour but de faire quelques rappels et de construire un exemple d'interface entre **Rebol** et les bibliothèques dynamiques.

Historique du document

Date	Version	Commentaires	Auteur	Email
mardi 30 juin 2005	0.0.1	Version initiale	J.C.A Miranda [aka Bouba]	jca—miranda—gmail—com
mardi 05 juillet 2005	0.0.2	Premier jet de corrections	J.C.A Miranda [aka Bouba]	jca—miranda—gmail—com

Introduction

Cet article ne se veut absolument pas exhaustif sur le sujet, et prend comme parti pris :

- que vous êtes familier avec le C,
- que vous avez déjà lu la documentation disponible en ligne sur le site de **Rebol Technologies**.

Pour illustrer le propos (en tout cas en partie), nous développerons un lecteur de fichier audio sans prétention dans la deuxième partie de l'article. Avis aux amateurs qui veulent s'essayer à en faire un produit fini, le code est fourni absolument libre de droit. :)

Les exemples ont été compilés et testés avec succès sous Linux (Rebol/View 1.2.47).

Règles et rappels

Commençons la partie un peu rébarbative par quelques trucs et astuces qui peuvent toujours servir. Si vous en connaissez d'autres, n'hésitez pas à les envoyer, ils seront les bienvenus. :)

Le type integer!

Ce paragraphe pour souligner une remarque importante dans la documentation de Rebol Technologies. Dans le paragraphe concernant les pointeurs C (paragraphe 4.4), il est dit :

```
"For return values, use long instead of void*"
```

Le type **integer!** peut donc être substitué à un pointeur lors de la création de vos interfaces.

Cette propriété s'avère très utile dans le cas où :

- une interface directe de votre fonction n'est pas possible (par exemple, si la routine que vous définissez vous retourne une structure dont la taille ne vous est connu qu'au retour de l'appel).
- le pointeur qui est retourné référence une structure opaque (un exemple-type : le pointeur sur la structure définissant le morceau a jouer dans l'exemple que nous développerons plus loin).

Le type decimal!

Le type **decimal!** a la particularité d'être converti en flottant au format IEEE. En utilisant une **struct!** ont peut écrire facilement une fonction de conversion :

```
to-ieee: func [
    "Conversion en float standard IEEE d'un decimal! Rebol"
    arg [decimal!]
    /double "Conversion en type double"
][
    third make struct! compose/deep [
        f [(either double ['double]['float])]
    ] reduce [arg]
]
```

Pensez-y si vous devez échanger des données binaires contenant des valeurs flottantes.

Le type struct !

Comme indiqué dans la documentation, le type **struct!** permet de définir une passerelle entre les struct C et **Rebol**. Comme dans l'exemple qui suit :

```
Code C :
struct {
  int  x;
  int  y;
  char point_desc[20];
}Point;
```

qui donne :

```
Code Rebol :
point: make struct! [
    x          [integer!]
    y          [integer!]
    point-desc [string!]
] none ; initialisation par défaut de la structure
```

Jusqu'ici, rien d'inconnu !

Mais, le type **struct!** a d'autres utilisations qui peuvent s'avérer très intéressantes. Essayons d'en voir certaines.

Définition de pointeurs

Imaginons la fonction C suivante :

```
void dummy_func(int x , int * y) {
  *y = x;
}
```

Comment traduire le pointeur sur entier *y* en **Rebol** ?

Si nous prions pour que Saint Rebol nous fasse le travail en lui donnant comme prototype :

```
dummy-func: make routine! [
    x [integer!]
    y [integer!]
] dummy-lib "dummy_func"
```

Voilà ce qui se passe lorsque nous tentons d'utiliser notre fonction révolutionnaire (Tracing sous Linux) :

```
>> dummy-lib: load/library %dummy_lib.so
>> dummy-func: make routine! [
[          x [integer!]
[          y [integer!]
[      ] dummy-lib "dummy_func"
>>
>> a: make integer! none
== 0
>> dummy-func 3 a
```
zsh : segmentation fault rebol/view47 —secure allow

A priori, loin du résultat que nous espérions. :(

En fait, **Rebol** a considéré que l'entier passé en tant que paramètre *y* était le pointeur (Cf plus haut). Malheureusement, nous n'avons pas alloué l'espace nécessaire au stockage d'un entier à l'adresse 0 (ce qui aurait de toute façon été un peu difficile !).

Heureusement, le type **struct!** peut nous aider à résoudre ce problème.

Une struct C n'étant qu'une interprétation d'une zone mémoire, nous pouvons imaginer que la représentation suivante peut résoudre notre problème :

```
int-ptr: make struct! [value [integer!]] none
```

Redéfinissons donc notre **routine! Rebol** avec cette modification.

```
dummy-func: make routine! compose/deep/only [
    x [integer!]
    y [struct! (first int-ptr)]
] dummy-lib "dummy_func"
```

L'utilisation de compose/deep/only nous permet ici d'éviter de réécrire la structure *int-ptr*. En faisant appel à *first int-ptr*, nous récupérons la spécification de la structure.

Une bonne habitude dès lors que vous faites appel à des structures récurrentes dans votre code.

Et maintenant, essayons donc notre nouvelle fonction inutile :

```
>> int-ptr: make struct! [value [integer!]] none ; initialisation à NULL.
>>
>> dummy-lib: load/library %dummy_lib.so
>> dummy-func: make routine! compose/deep/only [
[      x [integer!]
[      y [struct! (first int-ptr)]
[  ] dummy-lib "dummy_func"
>>
>> a: make struct! int-ptr none
>>
>> dummy-func 4 a
>>
>> a/value
== 4
```

Cela fonctionne ! Nous voilà donc avec un moyen simple de définir des pointeurs tout en allouant l'espace nécessaire au stockage de la valeur. :)

La fonction get-mem ? de Ladislav Mecir

Vous trouverez les fonctions de Ladislav sur le site : RIT - peek_and_poke.r

Cette fonction est un petit bijou, merci à lui ! :)) Vous vous demandez ce qu'elle permet ?

Regardons donc son code pour tenter de comprendre son utilité :

```
get-mem?: function [
    {get the byte from a memory address}
    address [integer!]
    /nts {a null-terminated string}
    /part {a binary with a specified length}
    length [integer!]
] [m] [
    address: make struct! [i [integer!]] reduce [address]
    if nts [
        m: make struct! [s [string!]] none
        change third m third address
        return m/s
    ]
    if part [
        m: head insert/dup copy [] [. [char!]] length
```

```
        m: make struct! compose/deep [bin [struct! (reduce [m])]] none
        change third m third address
        return to string! third m/bin
    ]
    m: make struct! [c [struct! [chr [char!]]]] none
    change third m third address
    m/c/chr
]
```

Personne ne s'est perdu ? L'opération magique de cette fonction est la ligne

```
change third m third address
```

qui permet de changer l'initialisation de la **struct!** pour que l'adresse pointée soit celle que nous avons fourni en paramètre. Il est ainsi possible d'analyser le contenu de la memoire à cette adresse, à vos risques et perils comme le dit Ladislav lui-même. :)

Voyons maintenant son fonctionnement par l'exemple. Considérons te type et la fonction C suivante :

```
typedef struct type_t{
      int len;
      char * arr;
}Type_t;

Type_t * allocate_byte_array(void);
```

Comment obtenir le contenu de la zone mémoire contenant le tableau alloué par la fonction C alors que l'on ne connait pas sa taille à l'avance ? La fonction *get-mem ?* peut nous y aider.

Considérons dans un premier temps le tableau comme un **integer!** sans nous intéresser à la zone pointée.

```
type-t: make struct! [
    len [integer!]
    arr [integer!]
] none
```

nous pouvons donc définir notre **routine!** en utilisant cette structure :

```
allocate-byte-array: make routine! compose/deep/only [
    return: [struct! (first type-t)]
] my-lib "allocate_byte_array"
```

Et maintenant, utilisons la fonction de Ladislav pour récupérer notre tableau :

```
>> result: allocate-byte-array
>>
>>
>> probe result
make struct! [
```

```
    len [integer!]
    arr [integer!]
] [256 139123696]
>>
>>
>> print to-binary get-mem?/part result/arr result/len
#{
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F
606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F
808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F
A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
}
>>
```

Voici la fonction utilisée pour faire le test. Elle fournit certes une table de taille fixe pour éviter un code inutilement complexe, mais elle illustre tout de même le propos.

Après tout, si vous n'aviez que le prototype, auriez-vous fait la supposition que la table faisait 256 octets ?

```
#include

typedef struct type_t{
      int len;
      char * arr;
}Type_t;


Type_t * allocate_byte_array(void) {
  Type_t * ret;
  int i;

  if (NULL == (ret = malloc(sizeof(Type_t)))) {
    exit(-1);
  }

  ret->len = 256;

  if (NULL == (ret->arr = malloc(ret->len))) {
    exit(-1);
  }

  for (i = 0 ; i len ; i++) {
```

```
    ret->arr[i] = i;
  }

  return ret;
}
```

En toute rigueur, il faudrait définir une fonction pour libérer la mémoire allouée.

Comme il ne s'agit pas du propos de cet exemple, elle est volontairement omise.

Le type binary !

Oui, je sais, il n'est pas référencé dans la documentation fournie par **RT**, mais pourtant on peut l'utiliser dans une interface avec le C ! A quoi peut-il bien servir vous demandez-vous ?

Prenons le cas d'une fonction que ceux qui connaissent openGL ont déjà dû voir :

```
GLAPI void GLAPIENTRY glColor3bv( const GLbyte *v );
```

Ne prêtez pas attention aux macros GLAPI et GLAPIENTRY, elles ne nous intéressent pas dans le cadre de cet exemple. :)

Cette fonction prend comme paramètre un tableau de 3 octets *v* représentant une couleur RGB. Bien sûr, on pourrait penser à utiliser une **struct!** ce qui donnerait ceci :

```
v: make struct! [r [char!] g [char!] b [char!]] none
```

notre routine :

```
gl-color-3bv: make routine! compose/deep/only [
    v [struct! (v)]
] libgl "glColor3bv"
```

et un exemple d'utilisation :

```
c-red: make struct! v reduce [to-char red/1 to-char red/2 to-char red/3]
gl-color-3bv c-red
```

ce qui somme toute conviendrait parfaitement !

Oui, mais sachant que dans le cas d'openGL, il existe la même fonction gérant une couleur à 4 composantes (en ajoutant le canal alpha), on s'apercoit bien vite que l'on multiplie les definitions de **struct!**, pas très léger pour du Rebol !

De plus, il serait agréable de pouvoir utiliser uniquement des **tuple!** dans notre script pour gérer des couleurs, après tout, ils sont là pour ça ! :)

Le type **binary!** va pouvoir nous aider à simplifier tout ça. En effet, le type **tuple!** a une propriété intéressante losqu'on le convertit en **binary!** :

```
>> to-binary 1.2.3.4
== #{01020304}
```

On dirait un tableau d'octet ! Vous voyez ou je veux en venir ? :o) Réécrivons donc notre fonction :

```
gl-color-3bv: make routine! [
    v [binary!]
] libgl "glColor3bv"
```

Et notre exemple devient :

```
gl-color-3bv to-binary red
```

A vous de choisir ce qui vous semble le mieux ! Personnellement, je préfère la deuxième définition, mais à chacun ses goûts. :o)

A noter que le type **tuple!** n'est pas le seul ayant cette propriété, les **block!** d'entiers également. Par contre, les éléments d'un **block!** n'étant pas limités à un octet comme les **tuple!**, seul l'octet de poids faible est gardé lors de la conversion en binaire. Amusez-vous bien ! :)

Entracte

Voilà pour ce qui est de faire le tour des trucs et astuces. Si vous en avez d'autres, n'hésitez pas, ils pourront sûrement servir à d'autres.

Un exemple d'utilisation : un player audio.

Bien, maintenant que tout le monde a enfin senti les effets de son aspirine, nous allons pouvoir continuer de manière plus légère cette excursion dans le monde des **struct!** et des **routine!**.

Pour cela, nous allons construire un lecteur audio sans prétention basé sur la bibliothèque SDL_mixer. Celle-ci est portée sur les principales plate-forme que supporte **Rebol**, donc tout le monde devrait pouvoir jeter son WinAmp, Xmms et autres RythmBox pour utiliser le fruit de son travail, ou presque ! :o)

Un très rapide tour d'horizon de SDL

SDL : qu'est ce que c'est ?

SDL est une bibliothèque multimédia multi-plateforme qui permet l'accès aux claviers, souris, joystick, périphériques audio et vidéo (2D/3D). Vous trouverez plus d'informations sur SDL et ses petits sur le site www.libsdl.org

De quoi avons-nous besoin ?

Pour les besoins de notre application, nous aurons besoin des bibliothèques suivantes :

- SDL
- SDL_mixer
- Smpeg
- Ogg/Vorbis

Les deux dernières ne seront pas chargés par notre script mais sont indispensables si l'on veut lire respectivement des MP3 et des fichiers Ogg. Leur chargement est géré par SDL_mixer.

Pour les utilisateurs de Windows, l'ensemble des bibliothèques compilées est disponible sur le site PyGame (oui je sais, Python c'est maaaal ! Mais là, le serpent nous aide ! :op).

Les specifications de notre player

Le player restera volontairement simple, il doit permettre :

- d'initialiser l'audio,
- d'obtenir la configuration obtenue,
- de lire un fichier audio,
- d'arrêter la lecture,
- de faire une pause/reprendre la lecture,
- de contrôler le volume,
- de quitter "proprement".

Création du wrapper Rebol

Etape essentielle à notre travail, il faut jeter un coup d'oeil au(x) fichier(s) d'include. Je ne saurais trop conseiller de regarder également les documentations associées, histoire que je ne me sente pas trop seul ! :)

Dans notre exemple, deux fichiers d'include sont à analyser :

- SDL.h qui regroupe les fonctions communes à tout applicatif utilisant SDL,
- SDL_mixer.h qui regroupe les fonctions audios qui nous interessent.

SDL.h

Dans ce fichier, nous trouvons les fonctions nécessaires à l'intialisation et à la fermeture du système SDL :

```
/* This function loads the SDL dynamically linked library and initializes
 * the subsystems specified by 'flags' (and those satisfying dependencies)
 * Unless the SDL_INIT_NOPARACHUTE flag is set, it will install cleanup
 * signal handlers for some commonly ignored fatal signals (like SIGSEGV)
 */
extern DECLSPEC int SDLCALL SDL_Init(Uint32 flags);
```

```
/* This function cleans up all initialized subsystems and unloads the
 * dynamically linked library.  You should call it upon all exit conditions.
 */
extern DECLSPEC void SDLCALL SDL_Quit(void);
```

Les macros DECLSPEC et SDLCALL ne doivent pas vous inquiéter, elles servent uniquement à assurer la portabilité du code. Si vous voulez plus de détails sur ces macros, jetez un oeil au fichier begin_code.h qui se trouvent parmi les include SDL.

Dans ce fichier, sont également déclarés un certain nombre de "constantes" qui servent à calculer le paramètre *flags* de SDL_init, une seule nous intéresse dans notre cas :

```
#define SDL_INIT_AUDIO 0x00000010
```

Nous pouvons d'ores et déjà définir les bases de notre wrapper :

```
sdl: context [
    libsdl: compose [main (load/library %/usr/lib/libSDL.so)]

    c-sdl: context [
        ;; constants
        INIT_AUDIO: to-integer #{00000010}
        protect 'SDL_INIT_AUDIO
;; functions
init: make routine! [flags [int] return: [int]] libsdl/main "SDL_Init"
quit: make routine! [] libsdl/main "SDL_Quit"
get-error: make routine! [return: [string!]] libsdl/main "SDL_GetError"
]
INIT_AUDIO: get in c-sdl 'INIT_AUDIO

init: func [flags [integer! block!] /local value] [
    value: 0
    foreach flag compose [(flags)] [value: or~ value flag]
    if 0  c-sdl/init value [
        make error! "SDL initialization problem!"
    ]
]
quit: does [
    c-sdl/quit
    foreach [name lib] libsdl [free lib]
]

get-error: does [make error! c-sdl/get-error]
]
```

SDL_mixer.h

Pour l'instant, notre wrapper bien qu'utilisable ne fait pas grand chose, passons à l'include qui nous permettra de faire un peu de musique.

Nous n'utiliserons pas les capacités de mixage de la librairie SDL_mixer, donc concentrons-nous sur les types et les routines liées à la gestion d'un morceau de musique.

Commençons par les constantes et les types :

```
/* Good default values for a PC soundcard */
#define MIX_DEFAULT_FREQUENCY 22050
#if SDL_BYTEORDER == SDL_LIL_ENDIAN
#define MIX_DEFAULT_FORMAT    AUDIO_S16LSB
#else
#define MIX_DEFAULT_FORMAT    AUDIO_S16MSB
#endif
#define MIX_DEFAULT_CHANNELS  2
#define MIX_MAX_VOLUME                128     /* Volume of a chunk */
```

Ce groupe de "constantes" définit des valeurs par défaut pour la fréquence, le format et le nombre de canaux à utiliser lors de l'ouverture de l'audio.

Comme nous pouvons le voir, MIX_DEFAULT_FORMAT est une redéfinition, on trouve les valeurs utilisées dans l'include SDL_audio.h :

```
#define AUDIO_S16LSB  0x8010  /* Signed 16-bit samples */
#define AUDIO_S16MSB  0x9010  /* As above, but big-endian byte order */
```

On s'aperçoit que le format est dépendant du "byte order" de la machine. Il nous faudra en tenir compte lors de notre construction de l'interface Rebol pour maintenir la portabilité.

```
typedef enum {
  MUS_NONE,
  MUS_CMD,
  MUS_WAV,
  MUS_MOD,
  MUS_MID,
  MUS_OGG,
  MUS_MP3
} Mix_MusicType;
```

Ce type énuméré défini le type du morceau joué, il nous servira à fournir cette information dans l'interface. Aucune valeur initiale n'est spécifiée, donc MUS_NONE vaut 0.

```
/* The internal format for a music chunk interpreted via mikmod */
typedef struct _Mix_Music Mix_Music;
```

Ce type sert à stocker les informations d'un morceau, la structure *_Mix_Music* nous est inconnue. Nous pourrons donc utiliser un **integer!** pour représenter les données de ce type.

Passons aux fonctions nécessaires à notre player.

```
extern DECLSPEC int SDLCALL Mix_OpenAudio(int frequency, Uint16 format, int channels,
                                                    int chunksize);
extern DECLSPEC int SDLCALL Mix_QuerySpec(int *frequency,Uint16 *format,int *channels);

extern DECLSPEC Mix_Music * SDLCALL Mix_LoadMUS(const char *file);

extern DECLSPEC void SDLCALL Mix_FreeMusic(Mix_Music *music);

extern DECLSPEC Mix_MusicType SDLCALL Mix_GetMusicType(const Mix_Music *music);

extern DECLSPEC int SDLCALL Mix_PlayMusic(Mix_Music *music, int loops);

extern DECLSPEC int SDLCALL Mix_VolumeMusic(int volume);

extern DECLSPEC void SDLCALL Mix_PauseMusic(void);
extern DECLSPEC void SDLCALL Mix_ResumeMusic(void);
extern DECLSPEC int SDLCALL Mix_PausedMusic(void);

extern DECLSPEC int SDLCALL Mix_PlayingMusic(void);

extern DECLSPEC void SDLCALL Mix_CloseAudio(void);
```

Que font-elles ?

Mix_OpenAudio : *Cette fonction permet l'ouverture du système audio par le mixer.*

Mix_QuerySpec : *Cette fonction permet de connaître les paramètres avec lesquels le système audio a effectivement été initialisé. Ces valeurs peuvent être différentes des paramètres que vous avez demandé lors de l'ouverture en fonction des capacités de votre système.*

Mix_LoadMUS : *Cette fonction permet de charger le fichier sonore à jouer, elle nous retourne un pointeur sur la structure opaque Mix_Music.*

Mix_FreeMusic : *Cette fonction nous permet de demander à SDL_mixer de libérer la mémoire allouée pour le morceau, indispensable si vous ne voulez pas que votre mémoire soit envahie par les résidus de votre derniere séance de détente musicale.*

Mix_GetMusicType : *Cette fonction nous permet de déterminer le type du fichier sonore manipulé au moment de son appel. S'agit-il d'un MP3, d'un WAV standard ?*

Mix_PlayMusic : *Cette fonction, un peu indispensable pour notre player, permet de jouer le morceau préalablement chargé.*

Mix_VolumeMusic : *Cette fonction nous sera indispensable pour éviter de réveiller les voisins, en nous permettant de contrôler le volume.*

Mix_PauseMusic/ Mix_ResumeMusic/ Mix_PausedMusic : *Cet ensemble de fonctions nous permettra d'interrompre/de reprendre/de savoir si nous avons interrompu la lecture de notre fichier.*

Mix_PlayingMusic : *Cette fonction nous permet de savoir si nous sommes en cours de lecture d'un fichier et nous permettra de détecter la fin d'un morceau.*

Mix_CloseAudio : *Et enfin, cette fonction nous permettra de quitter proprement notre application en évitant de polluer le système audio de notre machine.*

Bien ! En nous souvenant de ce qui a été dit dans la partie précédente (oui, je sais c'est dur !), on peut sans trop de difficultés enrichir notre contexte SDL :

```
int-ptr: make struct! [value [integer!]] none

sdl: context [
    libsdl: compose [
        main  (load/library %/usr/lib/libSDL.so)
        mixer (load/library %/usr/lib/libSDL_mixer.so)
    ]

    c-sdl: context [
        ;; constants
        INIT_AUDIO: to-integer #{00000010}
        protect 'SDL_INIT_AUDIO

        ;; functions
        init: make routine! [flags [int] return: [int]] libsdl/main "SDL_Init"

        quit: make routine! [] libsdl/main "SDL_Quit"

        get-error: make routine! [return: [string!]] libsdl/main "SDL_GetError"
    ]

    INIT_AUDIO: get in c-sdl 'INIT_AUDIO

    init: func [flags [integer! block!] /local value] [
        value: 0
        foreach flag compose [(flags)] [value: or~ value flag]
        if 0  c-sdl/init value [
            get-error
        ]
    ]

    quit: does [
        c-sdl/quit
        foreach [name lib] libsdl [free lib]
    ]

    get-error: does [make error! c-sdl/get-error]

    mixer: context [

        c-mixer: context [
```

```
            MIX_DEFAULT_FREQUENCY: 22050

            AUDIO_U8: to-integer 16#{0008}    ;; Unsigned 8-bit samples
            AUDIO_S8: to-integer 16#{8008}    ;; Signed 8-bit samples

            set [AUDIO_U16SYS AUDIO_S16SYS] switch get-modes system/ports/system 'endian
compose/deep [
                little [[(to-integer 16#{0010}) (to-integer 16#{8010})]]
                big    [[(to-integer 16#{1010}) (to-integer 16#{9010})]]
            ]

            MIX_DEFAULT_FORMAT: AUDIO_S16SYS

            MIX_DEFAULT_CHANNELS: 2

            MIX_MAX_VOLUME: 128

            Music-type: [
                0 NONE
                1 CMD
                2 WAV
                3 MOD
                4 MIDI
                5 OGG
                6 MP3
            ]

            open-audio: make routine! [
                frequency  [integer!]
                format     [integer!]
                channels   [integer!]
                chunk-size [integer!]
                return:    [integer!]
            ] libsdl/mixer "Mix_OpenAudio"

            query-spec: make routine! compose/deep/only [
                frequency [struct! (first int-ptr)]
                format    [struct! (first int-ptr)]
                channels  [struct! (first int-ptr)]
                return: [integer!]
            ] libsdl/mixer "Mix_QuerySpec"

            load-music: make routine! [
                file    [string!]
                return: [integer!]
            ] libsdl/mixer "Mix_LoadMUS"

            free-music: make routine! [music [integer!]] libsdl/mixer "Mix_FreeMusic"
```

```
        get-music-type: make routine! [
            music [integer!]
            return: [integer!]
        ] libsdl/mixer "Mix_GetMusicType"

        play-music: make routine! [
            music [integer!]
            loops [integer!]
            return: [integer!]
        ] libsdl/mixer "Mix_PlayMusic"

        halt-music: make routine! [return: [integer!]] libsdl/mixer "Mix_HaltMusic"

        volume-music: make routine! [
            volume [integer!]
            return: [integer!]
        ] libsdl/mixer "Mix_VolumeMusic"

        pause-music: make routine! [] libsdl/mixer "Mix_PauseMusic"

        resume-music: make routine! [] libsdl/mixer "Mix_ResumeMusic"

        paused-music: make routine! [return: [integer!]] libsdl/mixer "Mix_PausedMusic"

        playing-music: make routine! [return: [integer!]] libsdl/mixer "Mix_PlayingMusic"

        close-audio: make routine! [] libsdl/mixer "Mix_CloseAudio"

    ]

    MIX_DEFAULT_FREQUENCY: get in c-mixer 'MIX_DEFAULT_FREQUENCY
    MIX_DEFAULT_FORMAT:    get in c-mixer 'MIX_DEFAULT_FORMAT
    MIX_DEFAULT_CHANNELS:  get in c-mixer 'MIX_DEFAULT_CHANNELS
    MIX_MAX_VOLUME:        get in c-mixer 'MIX_MAX_VOLUME

    open-audio: func [frequency format channels chunk-size] [
        if 0  c-mixer/open-audio frequency format channels chunk-size [
            get-error
        ]
    ]

    query-spec: has [frequency format channels] [
        frequency: make struct! int-ptr none
        format: make struct! int-ptr none
        channels: make struct! int-ptr none

        c-mixer/query-spec frequency format channels
        reduce [frequency/value format/value channels/value]
    ]
```

```
        load-music: func [file [file!]] [
            c-mixer/load-music to-local-file file
        ]

        get-music-type: func [music [integer!]] [
            select c-mixer/Music-type c-mixer/get-music-type music
        ]

        play-music:           get in c-mixer 'play-music

        halt-music:           get in c-mixer 'halt-music

        free-music:           get in c-mixer 'free-music

        volume-music:         get in c-mixer 'volume-music

        pause-music:          get in c-mixer 'pause-music

        resume-music:         get in c-mixer 'resume-music

        paused-music:         get in c-mixer 'paused-music

        playing-music:        get in c-mixer 'playing-music

        close-audio:          get in c-mixer 'close-audio

    ]

]
```

Bien, maintenant, nous pouvons construire notre application en utilisant notre bébé. :)

```
music-fmt-filt: ["*.mp3" "*.ogg" "*.wav" "*.mod" "*.mid"]

mms-quit: does [
        sdl/mixer/close-audio
        sdl/quit
]

free-music: does [
    sdl/mixer/free-music song
    song: make none! none
]

init: does [
    sdl/init sdl/INIT_AUDIO
    sdl/mixer/open-audio 44100 sdl/mixer/MIX_DEFAULT_FORMAT sdl/mixer/MIX_DEFAULT_CHANNELS
4096

    song: make none! none
```

```
    song?: false
volume-ctl/data: 128
show volume-ctl

status/text: "Stopped" show status
main/effect: [gradient 1x1 255.255.255 160.160.160]
]


mstyles: stylize/master [
mbtn: btn 40 ivory
sep:  box 280x2 black
]

main: layout [
styles mstyles
h1 "audio player" black rate 0:0:1 feel [
    engage: func [f a e] [
        if all [a = 'time song? 0 = sdl/mixer/playing-music] [
            song?: false
            status/text: "Stopped" show status
        ]
    ]
]
sep
across
mbtn "select" [
    if song [free-music]
    if song: request-file/filter music-fmt-filt [
        song: sdl/mixer/load-music first song
        type/text: mold sdl/mixer/get-music-type song
        show type
    ]
]
mbtn "play"  [
    if song [
        sdl/mixer/play-music song 0
        status/text: "Playing" show status
        song?: true
    ]
]
mbtn "pause" [
    either 0 = sdl/mixer/paused-music [
        sdl/mixer/pause-music
        status/text: "Paused" show status
    ][
        sdl/mixer/resume-music
        status/text: "Playing" show status
    ]
]
```

```
 mbtn "stop"  [
     sdl/mixer/halt-music song?: false
     status/text: "Stopped" show status
 ]
 mbtn "halt"  [free-music mms-quit quit]
 mbtn "info"  [
     mixer-info: sdl/mixer/query-spec
     view/new info-win: layout [
         across
         label black 80 "frequency" info 80 mold first  mixer-info return
         label black 80 "format"    info 80 mold second mixer-info return
         label black 80 "channels"  info 80 mold third  mixer-info return
         btn 170 ivory "Close" [unview]
     ]
     info-win/effect: [gradient 1x1 255.255.255 160.160.160]
     show info-win
 ]

 return sep return

 label black "volume"
 volume-ctl: scroller 150x15 [volume: to-integer multiply volume-ctl/data 128
sdl/mixer/volume-music volume]

 return sep return

 type: info 130x25 status: info 130x25
 ]

 init
 view main
```

A vous de jouer maintenant ! :)

Conclusion

Nous voilà arrivé au bout. J'espère que cette lecture vous aura servi et vous permettra d'enrichir **Rebol** et d'en faire profiter tous vos amis codeurs de par le monde, en attendant l'interface plugin promise par Carl ! :)

Amusez-vous bien !