Libxmp 4.0 API documentation

Contents

Introduction

Libxmp is a module player library supporting many mainstream and obscure module formats including Protracker MOD, Scream Tracker III S3M and Impulse Tracker IT. Libxmp loads the module and renders the sound as linear PCM samples in a buffer at rate and format specified by the user, one frame at a time (standard modules usually play at 50 frames per second).

Possible applications for libxmp include stand-alone module players, module player plugins for other players, module information extractors, background music replayers for games and other applications, module-to-mp3 renderers, etc.

Concepts

- **Player context:** Most libxmp functions require a handle that identifies the module player context. Each context is independent and multiple contexts can be defined simultaneously.
- **Sequence:** Each group of positions in the order list that loops over itself, also known as “subsong”. Most modules have only one sequence, but some modules, especially modules used in games can have multiple sequences. “Hidden patterns” outside the main song are also listed as extra sequences, certain module authors such as Skaven commonly place extra patterns at the end of the module.

A simple example

This example loads a module, plays it at 44.1kHz and writes it to a raw sound file:

```
#include <stdio.h>
#include <stdlib.h>
#include <xmp.h>

int main(int argc, char **argv)
{
    xmp_context c;
    struct xmp_frame_info mi;
    FILE *f;

    /* The output raw file */
    f = fopen("out.raw", "wb");
    if (f == NULL) {
        fprintf(stderr, "can't open output file\n");
        exit(EXIT_FAILURE);
    }

    /* Create the player context */
    c = xmp_create_context();

    /* Load our module */
    if (xmp_load_module(c, argv[1]) != 0) {
        fprintf(stderr, "can't load module\n");
        exit(EXIT_FAILURE);
    }

    /* Play the module */
    xmp_start_player(c, 44100, 0);
    while (xmp_play_frame(c) == 0) {
        xmp_get_frame_info(c, &mi);

        if (mi.loop_count > 0)    /* exit before looping */
            break;
        fwrite(mi.buffer, mi.buffer_size, 1, f);  /* write audio data */
    }
    xmp_end_player(c);
    xmp_release_module(c);        /* unload module */
    xmp_free_context(c);          /* destroy the player context */

    fclose(f);

    exit(EXIT_SUCCESS);
}
```

A player context can load and play a single module at a time. Multiple contexts can be defined if needed.

Use xmp_test_module() to check if the file is a valid module and retrieve the module name and type. Use xmp_load_module() to load the module to memory. These two calls return 0 on success or <0 in case of error. Error codes are:

```
-XMP_ERROR_INTERNAL   /* Internal error */
-XMP_ERROR_FORMAT     /* Unsupported module format */
-XMP_ERROR_LOAD       /* Error loading file */
-XMP_ERROR_DEPACK     /* Error depacking file */
-XMP_ERROR_SYSTEM     /* System error */
```

If a system error occurs, the specific error is set in `errno`.

Parameters to xmp_start_player() are the sampling rate (up to 48kHz) and a bitmapped integer holding one or more of the following mixer flags:

```
XMP_MIX_8BIT          /* Mix to 8-bit instead of 16 */
XMP_MIX_UNSIGNED      /* Mix to unsigned samples */
XMP_MIX_MONO          /* Mix to mono instead of stereo */
XMP_MIX_NEAREST       /* Mix using nearest neighbor interpolation */
XMP_MIX_NOFILTER      /* Disable lowpass filter */
```

After xmp_start_player() is called, each call to xmp_play_frame() will render an audio frame. Call xmp_get_frame_info() to retrieve the buffer address and size. xmp_play_frame() returns 0 on success or -1 if replay should stop.

Use xmp_end_player(), xmp_release_module() and xmp_free_context() to release memory and end replay.

API reference

Version and player information

const char *xmp_version

A string containing the library version, such as “4.0.0”.

const unsigned int xmp_vercode

The library version encoded in a integer value. Bits 23-16 contain the major version number, bits 15-8 contain the minor version number, and bits 7-0 contain the release number.

char **xmp_get_format_list()

Query the list of supported module formats.

Returns: a NULL-terminated array of strings containing the names of all supported module formats.

Context creation

xmp_context xmp_create_context()

Create a new player context and return an opaque handle to be used in subsequent accesses to this context.

Returns:

the player context handle.

void xmp_free_context(xmp_context c)

Destroy a player context previously created using xmp_create_context().

Parameters:

c: the player context handle.

Module loading

int xmp_test_module(char *path, struct xmp_test_info *test_info)

Test if a file is a valid module.

Parameters:

path: pathname of the module to test.

test_info: a pointer to a structure used to retrieve the module title and format, if the file is a valid module. `struct xmp_test_info` is defined as:

```
struct xmp_test_info {
    char name[XMP_NAME_SIZE];      /* Module title */
    char type[XMP_NAME_SIZE];      /* Module format */
};
```

Returns:

0 if the file is a valid module, or a negative error code in case of error. Error codes can be `-XMP_ERROR_FORMAT` in case of an unrecognized file format, `-XMP_ERROR_DEPACK` if the file is compressed and uncompression failed, or `-XMP_ERROR_SYSTEM` in case of a system error (the system error code is set in `errno`).

int xmp_load_module(xmp_context c, char *path)

Load a module into the specified player context.

Parameters:

c: the player context handle.

path: pathname of the module to load.

Returns:

0 if sucessful, or a negative error code in case of error. Error codes can be `-XMP_ERROR_FORMAT` in case of an unrecognized file format, `-XMP_ERROR_DEPACK` if the file is compressed and uncompression failed, `-XMP_ERROR_LOAD` if the file format was recognized but the file loading failed, or `-XMP_ERROR_SYSTEM` in case of a system error (the system error code is set in `errno`).

void xmp_release_module(xmp_context c)

Release memory allocated by a module from the specified player context.

Parameters:

c: the player context handle.

void xmp_scan_module(xmp_context c)

Scan the loaded module for sequences and timing. Scanning is automatically performed by xmp_load_module() and this function should be called only if xmp_set_player() is used to change player timing (with parameter `XMP_PLAYER_VBLANK`) in libxmp 4.0.2 or older.

Parameters:

c: the player context handle.

void xmp_get_module_info(xmp_context c, struct xmp_module_info *info)

Retrieve current module data.

Parameters:

c: the player context handle.

info: pointer to structure containing the module data. `struct xmp_module_info` is defined as follows:

```
struct xmp_module_info {
    unsigned char md5[16];          /* MD5 message digest */
    int vol_base;                   /* Volume scale */
    struct xmp_module *mod;         /* Pointer to module data */
    char *comment;                  /* Comment text, if any */
    int num_sequences;              /* Number of valid sequences */
    struct xmp_sequence *seq_data;  /* Pointer to sequence data */
};
```

Detailed module data is exposed in the `mod` field:

```
struct xmp_module {
    char name[XMP_NAME_SIZE];       /* Module title */
    char type[XMP_NAME_SIZE];       /* Module format */
    int pat;                        /* Number of patterns */
    int trk;                        /* Number of tracks */
    int chn;                        /* Tracks per pattern */
    int ins;                        /* Number of instruments */
    int smp;                        /* Number of samples */
    int spd;                        /* Initial speed */
    int bpm;                        /* Initial BPM */
    int len;                        /* Module length in patterns */
    int rst;                        /* Restart position */
    int gvl;                        /* Global volume */

    struct xmp_pattern **xxp;       /* Patterns */
    struct xmp_track **xxt;         /* Tracks */
    struct xmp_instrument *xxi;     /* Instruments */
    struct xmp_sample *xxs;         /* Samples */
    struct xmp_channel xxc[64];     /* Channel info */
    unsigned char xxo[XMP_MAX_MOD_LENGTH];  /* Orders */
};
```

See the header file for more information about pattern and instrument data.

Module playing

int xmp_start_player(xmp_context c, int rate, int format)

Start playing the currently loaded module.

Parameters:

c:	the player context handle.
rate:	the sampling rate to use, in Hz (tipically 44100). Valid values range from 8kHz to 48kHz.
flags:	bitmapped configurable player flags, one or more of the following:

```
XMP_FORMAT_8BIT          /* Mix to 8-bit instead of 16 */
XMP_FORMAT_UNSIGNED      /* Mix to unsigned samples */
XMP_FORMAT_MONO          /* Mix to mono instead of stereo */
```

Returns:

0 if sucessful, or a negative error code in case of error. Error codes can be `-XMP_ERROR_INTERNAL` in case of a internal player error, or `-XMP_ERROR_SYSTEM` in case of a system error (the system error code is set in `errno`).

int xmp_play_frame(xmp_context c)

Play one frame of the module. Modules usually play at 50 frames per second. Use xmp_get_frame_info() to retrieve the buffer containing audio data.

Parameters:

c:	the player context handle.

Returns:

0 if sucessful or -1 if the module was stopped.

void xmp_get_frame_info(xmp_context c, struct xmp_frame_info *info)

Retrieve current frame data.

Parameters:

c:	the player context handle.

info: pointer to structure containing current frame data. `struct xmp_frame_info` is defined as follows:

```
struct xmp_frame_info {           /* Current frame information */
    int pos;            /* Current position */
    int pattern;        /* Current pattern */
    int row;            /* Current row in pattern */
    int num_rows;       /* Number of rows in current pattern */
    int frame;          /* Current frame */
    int speed;          /* Current replay speed */
    int bpm;            /* Current bpm */
    int time;           /* Current module time in ms */
    int total_time;     /* Estimated replay time in ms*/
    int frame_time;     /* Frame replay time in us */
    void *buffer;       /* Pointer to sound buffer */
    int buffer_size;    /* Used buffer size */
    int total_size;     /* Total buffer size */
    int volume;         /* Current master volume */
    int loop_count;     /* Loop counter */
    int virt_channels;  /* Number of virtual channels */
    int virt_used;      /* Used virtual channels */
    int sequence;       /* Current sequence */

    struct xmp_channel_info {     /* Current channel information */
        unsigned int period;      /* Sample period */
        unsigned int position;    /* Sample position */
        short pitchbend;          /* Linear bend from base note*/
        unsigned char note;       /* Current base note number */
        unsigned char instrument; /* Current instrument number */
        unsigned char sample;     /* Current sample number */
        unsigned char volume;     /* Current volume */
        unsigned char pan;        /* Current stereo pan */
        unsigned char reserved;   /* Reserved */
        struct xmp_event event;   /* Current track event */
    } channel_info[XMP_MAX_CHANNELS];
};
```

This function should be used to retrieve sound buffer data after xmp_play_frame() is called. Fields `buffer` and `buffer_size` contain the pointer to the sound buffer PCM data and its size. The buffer size will be no larger than `XMP_MAX_FRAMESIZE`.

Returns:

0 if sucessful or -1 if the module was stopped.

void xmp_end_player(xmp_context c)

End module replay and releases player memory.

Parameters:

c: the player context handle.

Player control

int xmp_next_position(xmp_context c)

Skip replay to the start of the next position.

Parameters:

c: the player context handle.

Returns:

The new position index.

int xmp_prev_position(xmp_context c)

Skip replay to the start of the previous position.

Parameters:

c:	the player context handle.

Returns:

The new position index.

int xmp_set_position(xmp_context c, int pos)

Skip replay to the start of the given position.

Parameters:

c:	the player context handle.
pos:	the position index to set.

Returns:

The new position index.

void xmp_stop_module(xmp_context c)

Stop the currently playing module.

Parameters:

c:	the player context handle.

void xmp_restart_module(xmp_context c)

Restart the currently playing module.

Parameters:

c:	the player context handle.

int xmp_seek_time(xmp_context c, int time)

Skip replay to the specified time.

Parameters:

c:	the player context handle.
time:	time to seek in milliseconds.

Returns:

The new position index.

int xmp_channel_mute(xmp_context c, int channel, int status)

Mute or unmute the specified channel.

Parameters:

c: the player context handle.
channel: the channel to mute or unmute.
status: 0 to mute channel, 1 to unmute or -1 to query the current channel status.

Returns:

The previous channel status.

int xmp_channel_vol(xmp_context c, int channel, int vol)

Set or retrieve the volume of the specified channel.

Parameters:

c: the player context handle.
channel: the channel to set or get volume.
vol: a value from 0-100 to set the channel volume, or -1 to retrieve the current volume.

Returns:

The previous channel volume.

void xmp_inject_event(xmp_context c, int channel, struct xmp_event *event)

Dynamically insert a new event into a playing module.

Parameters:

c: the player context handle.
channel: the channel to insert the new event.
event: the event to insert. `struct xmp_event` is defined as:

```
struct xmp_event {
    unsigned char note;   /* Note number (0 means no note) */
    unsigned char ins;    /* Patch number */
    unsigned char vol;    /* Volume (0 to basevol) */
    unsigned char fxt;    /* Effect type */
    unsigned char fxp;    /* Effect parameter */
    unsigned char f2t;    /* Secondary effect type */
    unsigned char f2p;    /* Secondary effect parameter */
    unsigned char _flag;  /* Internal (reserved) flags */
};
```

Player parameter setting

int xmp_set_player(xmp_context c, int param, int val)

Set mixer parameter with the specified value.

Parameters:

param: player parameter to set. Valid parameters are:

```
XMP_PLAYER_AMP      /* Amplification factor */
XMP_PLAYER_MIX      /* Stereo mixing */
XMP_PLAYER_INTERP   /* Interpolation type */
XMP_PLAYER_DSP      /* DSP effect flags */
XMP_PLAYER_FLAGS    /* Player flags */
```

val: the value to set. Valid values are:

- Amplification factor: ranges from 0 to 3. Default value is 1.
- Stereo mixing: percentual left/right channel separation. Default is 70.
- Interpolation type: can be one of the following values:

```
XMP_INTERP_NEAREST  /* Nearest neighbor */
XMP_INTERP_LINEAR   /* Linear (default) */
XMP_INTERP_SPLINE   /* Cubic spline */
```

- DSP effects flags: enable or disable DSP effects. Valid effects are:

```
XMP_DSP_LOWPASS     /* Lowpass filter effect */
XMP_DSP_ALL         /* All effects */
```

- Player flags: tweakable player parameters. Valid flags are:

```
XMP_FLAGS_VBLANK    /* Use vblank timing */
XMP_FLAGS_FX9BUG    /* Emulate Protracker 2.x FX9 bug */
XMP_FLAGS_FIXLOOP   /* Make sample loop value / 2 */
```

Returns:

0 if parameter was correctly set, or `-XMP_ERROR_INVALID` if parameter or values are out of the valid ranges.

int xmp_get_player(xmp_context c, int param)

Retrieve current value of the specified mixer parameter.

Parameters:

c: the player context handle.
param: player parameter to get. See xmp_set_player() for a list of valid parameters.

Returns:

The parameter value.

int xmp_set_instrument_path(xmp_context c, char *path)

Set the path to retrieve external instruments or samples. Used by some formats (such as MED2) to read sample files from a different directory in the filesystem.

Parameters:

c: the player context handle.
path: the path to retrieve instrument files.

Returns:

0 if the instrument path was correctly set, or `-XMP_ERROR_SYSTEM` in case of error (the system error code is set in `errno`).