

# Libxmp 4.2 API documentation

## Contents

|  |           |
|--|-----------|
| <b>Introduction</b>  | <b>2</b>  |
| Concepts   | 2         |
| A simple example   | 2         |
| SDL example  | 3         |
| <b>API reference</b>   | <b>5</b>  |
| Version and player information   | 5         |
| const char *xmp_version  | 5         |
| const unsigned int xmp_vercode   | 5         |
| char **xmp_get_format_list()   | 5         |
| Context creation   | 5         |
| xmp_context xmp_create_context()   | 5         |
| void xmp_free_context(xmp_context c)   | 5         |
| Module loading   | 5         |
| int xmp_test_module(char *path, struct xmp_test_info *test_info)                 | 5         |
| int xmp_load_module(xmp_context c, char *path)                                   | 6         |
| int xmp_load_module_from_memory(xmp_context c, void *mem, long size)             | 6         |
| void xmp_release_module(xmp_context c)   | 6         |
| void xmp_scan_module(xmp_context c)  | 6         |
| void xmp_get_module_info(xmp_context c, struct xmp_module_info *info)            | 6         |
| Module playing   | 7         |
| int xmp_start_player(xmp_context c, int rate, int format)                        | 7         |
| int xmp_play_frame(xmp_context c)  | 8         |
| void xmp_get_frame_info(xmp_context c, struct xmp_frame_info *info)              | 8         |
| int xmp_play_buffer(xmp_context c, void *buffer, int size, int loop)             | 9         |
| void xmp_end_player(xmp_context c)   | 9         |
| Player control   | 9         |
| int xmp_next_position(xmp_context c)   | 9         |
| int xmp_prev_position(xmp_context c)   | 9         |
| int xmp_set_position(xmp_context c, int pos)                                     | 9         |
| void xmp_stop_module(xmp_context c)  | 10        |
| void xmp_restart_module(xmp_context c)   | 10        |
| int xmp_seek_time(xmp_context c, int time)                                       | 10        |
| int xmp_channel_mute(xmp_context c, int channel, int status)                     | 10        |
| int xmp_channel_vol(xmp_context c, int channel, int vol)                         | 10        |
| void xmp_inject_event(xmp_context c, int channel, struct xmp_event *event)       | 10        |
| Player parameter setting   | 11        |
| int xmp_set_instrument_path(xmp_context c, char *path)                           | 11        |
| int xmp_get_player(xmp_context c, int param)                                     | 11        |
| int xmp_set_player(xmp_context c, int param, int val)                            | 12        |
| <b>External sample mixer API</b>   | <b>13</b> |
| Example  | 13        |
| SMIX API reference   | 14        |
| int xmp_start_smix(xmp_context c, int nch, int nsmp)                             | 14        |
| int xmp_smix_play_instrument(xmp_context c, int ins, int note, int vol, int chn) | 14        |
| int xmp_smix_play_sample(xmp_context c, int ins, int vol, int chn)               | 15        |
| int xmp_smix_channel_pan(xmp_context c, int chn, int pan)                        | 15        |
| int xmp_smix_load_sample(xmp_context c, int num, char *path)                     | 15        |
| int xmp_smix_release_sample(xmp_context c, int num)                              | 15        |
| void xmp_end_smix(xmp_context c)   | 16        |

# 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.
- **State:** *[Added in libxmp 4.2]* The player can be in one of three possible states: *unloaded*, *loaded*, or *playing*. The player is in unloaded state after context creation, changing to other states when a module is loaded or played.
- **External sample mixer:** *[Added in libxmp 4.2]* Special sound channels can be reserved using `xmp_start_smix()` to play module instruments or external samples. This is useful when libxmp is used to provide background music to games or other applications where sound effects can be played in response to events or user actions

## 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 unpacking file */
-XMP_ERROR_SYSTEM      /* System error */
-XMP_ERROR_STATE       /* Incorrect player state */

```

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.

## SDL example

To use libxmp with SDL, just provide a callback function that renders module data. The module will play when `SDL_PauseAudio(0)` is called:

```

#include <SDL/SDL.h>
#include <xmp.h>

static void fill_audio(void *udata, unsigned char *stream, int len)
{
    xmp_play_buffer(udata, stream, len, 0);
}

```

```

int sound_init(xmp_context ctx, int sampling_rate, int channels)
{
    SDL_AudioSpec a;

    a.freq = sampling_rate;
    a.format = (AUDIO_S16);
    a.channels = channels;
    a.samples = 2048;
    a.callback = fill_audio;
    a.userdata = ctx;

    if (SDL_OpenAudio(&a, NULL) < 0) {
        fprintf(stderr, "%s\n", SDL_GetError());
        return -1;
    }
}

int main(int argc, char **argv)
{
    xmp_context ctx;

    if ((ctx = xmp_create_context()) == NULL)
        return 1;

    sound_init(ctx, 44100, 2);
    xmp_load_module(ctx, argv[1]);
    xmp_start_player(ctx, 44100, 0);

    SDL_PauseAudio(0);

    sleep(10);           // Do something important here

    SDL_PauseAudio(1);

    xmp_end_player(ctx);
    xmp_release_module(ctx);
    xmp_free_context(ctx);

    SDL_CloseAudio();
    return 0;
}

```

SDL callbacks run in a separate thread, so don't forget to protect sections that manipulate module data with `SDL_LockAudio()` and `SDL_UnlockAudio()`.

# 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. Testing a file does not affect the current player context or any currently loaded module.

**Parameters:**

**path:** pathname of the module to test.

**test\_info:** NULL, or 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 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 successful, 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 system error (the system error code is set in `errno`).

## **int xmp\_load\_module\_from\_memory(xmp\_context c, void \*mem, long size)**

*[Added in libxmp 4.2]* Load a module from memory into the specified player context.

### **Parameters:**

- c:** the player context handle.
- mem:** a pointer to the module file image in memory. Multi-file modules or compressed modules can't be loaded from memory.
- size:** the size of the module, or 0 if the size is unknown or not specified. If size is set to 0 certain module formats won't be recognized, the MD5 digest will not be set, and module-specific quirks won't be applied.

### **Returns:**

0 if successful, 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_LOAD` if the file format was recognized but the file loading failed, or `-XMP_ERROR_SYSTEM` in case of 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 **xpx;         /* 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 (typically 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 successful, or a negative error code in case of error. Error codes can be -XMP\_ERROR\_INTERNAL in case of a internal player error, -XMP\_ERROR\_INVALID if the sampling rate is invalid, or -XMP\_ERROR\_SYSTEM in case of 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 successful, `-XMP_END` if the module ended or was stopped, or `-XMP_ERROR_STATE` if the player is not in playing state.

## 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`.



## **int xmp\_play\_buffer(xmp\_context c, void \*buffer, int size, int loop)**

*[Added in libxmp 4.1]* Fill the buffer with PCM data up to the specified size. This is a convenience function that calls `xmp_play_frame()` internally to fill the user-supplied buffer — don't call both functions in the same replay loop, choose one of them. If you don't need equally sized data chunks, `xmp_play_frame()` will result in better performance.

### **Parameters:**

- c:** the player context handle.
- buffer:** the buffer to fill with PCM data, or NULL to reset the internal state.
- size:** buffer size in bytes.
- loop:** stop replay when the loop counter reaches the specified value, or 0 to disable loop checking.

### **Returns:**

0 if successful, `-XMP_END` if module was stopped or the loop counter was reached, or `-XMP_ERROR_STATE` if the player is not in playing state.

## **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, or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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, or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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, `-XMP_ERROR_INVALID` if the new position is invalid or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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, or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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, or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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, or `-XMP_ERROR_STATE` if the player is not in playing state.

### **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_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`).

### `int xmp_get_player(xmp_context c, int param)`

Retrieve current value of the specified player parameter.

**Parameters:**

- c:** the player context handle.
- param:** player parameter to get. 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 */
XMP_PLAYER_CFLAGS   /* Player flags for current module*/
XMP_PLAYER_SMPCTL    /* Control sample loading */
XMP_PLAYER_VOLUME   /* Player master volume */
XMP_PLAYER_STATE     /* Current player state*/
```

See `xmp_set_player` for the list of valid values for each parameter. Valid states are:

```
XMP_STATE_UNLOADED /* Context created */
XMP_STATE_LOADED   /* Module loaded */
XMP_STATE_PLAYING  /* Module playing */
```

**Returns:**

The parameter value, or `-XMP_ERROR_STATE` if the parameter is not `XMP_PLAYER_STATE` and the player is not in playing state.

## **int xmp\_set\_player(xmp\_context c, int param, int val)**

Set player 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 */
XMP_PLAYER_CFLAGS    /* Player flags for current module*/
XMP_PLAYER_SMPCTL    /* Control sample loading */
XMP_PLAYER_VOLUME    /* Player master volume */
```

**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 */
```

- *[Added in libxmp 4.1]* Player flags for current module: same flags as above but after applying module-specific quirks (if any).
- *[Added in libxmp 4.1]* Control sample load. Valid values are:

```
XMP_SMPCTL_SKIP     /* Don't load samples */
```

Disabling sample loading when loading a module allows computation of module duration without decompressing and loading large sample data, and is useful when duration information is needed for a module that won't be played immediately.

- *[Added in libxmp 4.2]* Set the player master volume, in a 0-100 scale.

### **Returns:**

0 if parameter was correctly set, `-XMP_ERROR_INVALID` if parameter or values are out of the valid ranges, or `-XMP_ERROR_STATE` if the player is not in playing state.

# External sample mixer API

Libxmp 4.2 includes a mini-API that can be used to add sound effects to games and similar applications, provided that you have a low latency sound system. It allows module instruments or external sample files in WAV format to be played in response to arbitrary events.

## Example

This example using SDL loads a module and a sound sample, plays the module as background music, and plays the sample when a key is pressed:

```
#include <SDL/SDL.h>
#include <xmp.h>

static void fill_audio(void *udata, unsigned char *stream, int len)
{
    xmp_play_buffer(udata, stream, len, 0);
}

int sound_init(xmp_context ctx, int sampling_rate, int channels)
{
    SDL_AudioSpec a;

    a.freq = sampling_rate;
    a.format = (AUDIO_S16);
    a.channels = channels;
    a.samples = 2048;
    a.callback = fill_audio;
    a.userdata = ctx;

    if (SDL_OpenAudio(&a, NULL) < 0) {
        fprintf(stderr, "%s\n", SDL_GetError());
        return -1;
    }
}

int video_init()
{
    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        fprintf(stderr, "%s\n", SDL_GetError());
        return -1;
    }
    if (SDL_SetVideoMode(640, 480, 8, 0) == NULL) {
        fprintf(stderr, "%s\n", SDL_GetError());
        return -1;
    }
    atexit(SDL_Quit);
}

int main(int argc, char **argv)
{
    SDL_Event event;
    xmp_context ctx;

    if ((ctx = xmp_create_context()) == NULL)
        return 1;

    video_init();
    sound_init(ctx, 44100, 2);
}
```

```

xmp_start_smix(ctx, 1, 1);
xmp_smix_load_sample(ctx, 0, "blip.wav");

xmp_load_module(ctx, "music.mod");
xmp_start_player(ctx, 44100, 0);
xmp_set_player(ctx, XMP_PLAYER_VOLUME, 40);

SDL_PauseAudio(0);

while (1) {
    if (SDL_WaitEvent(&event)) {
        if (event.type == SDL_KEYDOWN) {
            if (event.key.keysym.sym == SDLK_ESCAPE)
                break;
            xmp_smix_play_sample(ctx, 0, 60, 64, 0);
        }
    }
}

SDL_PauseAudio(1);

xmp_end_player(ctx);
xmp_release_module(ctx);
xmp_free_context(ctx);
xmp_end_smix(ctx);

SDL_CloseAudio();
return 0;
}

```

## SMIX API reference

### int xmp\_start\_smix(xmp\_context c, int nch, int nsmp)

Initialize the external sample mixer subsystem with the given number of reserved channels and samples.

#### Parameters:

- c:** the player context handle.
- nch:** number of reserved sound mixer channels (1 to 64).
- nsmp:** number of external samples.

#### Returns:

0 if the external sample mixer system was correctly initialized, `-XMP_ERROR_INVALID` in case of invalid parameters, `-XMP_ERROR_STATE` if the player is already in playing state, or `-XMP_ERROR_SYSTEM` in case of system error (the system error code is set in `errno`).

### int xmp\_smix\_play\_instrument(xmp\_context c, int ins, int note, int vol, int chn)

Play a note using an instrument from the currently loaded module in one of the reserved sound mixer channels.

#### Parameters:

- c:** the player context handle.
- ins:** the instrument to play.
- note:** the note number to play (60 = middle C).
- vol:** the volume to use (0 to the maximum volume value used by the current module).

**chn:** the reserved channel to use to play the instrument.

**Returns:**

0 if the instrument was correctly played, `-XMP_ERROR_INVALID` in case of invalid parameters, or `-XMP_ERROR_STATE` if the player is not in playing state.

## **int xmp\_smix\_play\_sample(xmp\_context c, int ins, int vol, int chn)**

Play an external sample file in one of the reserved sound channels. The sample must have been previously loaded using `xmp_smix_load_sample()`.

**Parameters:**

**c:** the player context handle.  
**ins:** the sample to play.  
**vol:** the volume to use (0 to the maximum volume value used by the current module).  
**chn:** the reserved channel to use to play the sample.

**Returns:**

0 if the sample was correctly played, `-XMP_ERROR_INVALID` in case of invalid parameters, or `-XMP_ERROR_STATE` if the player is not in playing state.

## **int xmp\_smix\_channel\_pan(xmp\_context c, int chn, int pan)**

Set the reserved channel pan value.

**Parameters:**

**c:** the player context handle.  
**chn:** the reserved channel number.  
**pan:** the pan value to set (0 to 255).

**Returns:**

0 if the pan value was set, or `-XMP_ERROR_INVALID` if parameters are invalid.

## **int xmp\_smix\_load\_sample(xmp\_context c, int num, char \*path)**

Load a sound sample from a file. Samples should be in mono WAV (RIFF) format.

**Parameters:**

**c:** the player context handle.  
**num:** the slot number of the external sample to load.  
**path:** pathname of the file to load.

**Returns:**

0 if the sample was correctly loaded, `-XMP_ERROR_INVALID` if the sample slot number is invalid (not reserved using `xmp_start_smix()`), `-XMP_ERROR_FORMAT` if the file format is unsupported, or `-XMP_ERROR_SYSTEM` in case of system error (the system error code is set in `errno`).

## **int xmp\_smix\_release\_sample(xmp\_context c, int num)**

Release memory allocated by an external sample in the specified player context.

**Parameters:**

**c:** the player context handle.  
**num:** the sample slot number to release.

**Returns:**

0 if memory was correctly released, or `-XMP_ERROR_INVALID` if the sample slot number is invalid.

## **void xmp\_end\_smix(xmp\_context c)**

Deinitialize and release memory used by the external sample mixer subsystem.

### **Parameters:**

**c:** the player context handle.