NAME

libsoxformat_ng – libsox_ng format handlers' internals

SYNOPSIS

```
#include "sox_i.h"

typedef struct {
    ...
} priv_t;

static int startread(sox_format_t *ft);
static size_t read(sox_format_t *ft, sox_sample_t *buf, size_t len);
static int stopread(sox_format_t *ft);
static int startwrite(sox_format_t *ft);
static size_t write(sox_format_t *ft, sox_sample_t *buf, size_t len);
static int stopwrite(sox_format_t *ft);
static int seek(sox_format_t *ft, sox_uint64_t offset);

sox_format_handler_t const *lsx_whatever_format_fn(void)
{
  static char const description[] = "Whatever format";
  static char const * const names = { "wvr", NULL };
  static unsigned const write_encodings[] = {
    SOX_ENCODING_*, {bitwidth,} 0,
    ..., 0,
    0};
  static sox_format_handler_t const handler = { SOX_LIB_VERSION_CODE,
    description, names, {SOX_FILE_* |} 0,
    startread, read, stopread,
    startwrite, write, stopwrite,
    seek, write_encodings,
    sizeof(priv_t)
  };
  return &handler;
}
```

DESCRIPTION

How SoX format handlers work and how to write a new one.

SoX's formats and effects operate with an internal sample format of signed 32-bit integers. The data processing routines are called with buffers of these samples and buffer sizes which refer to the number of samples processed, not the number of bytes. File readers translate input samples to signed 32-bit integers and return the number of samples read. For example, data in linear signed byte format is left-shifted 24 bits.

Stereo data is stored with the left and right channels' data in successive samples and quadraphonic data is stored left front, right front, left rear, right rear.

A format handler is responsible for translating between sound sample files and an internal buffer in which sound data is stored as signed 32-bit integers with a fixed sampling rate.

```
struct sox_format_handler {
  unsigned     sox_lib_version_code;        /* Checked on load           */
  char       * description;                 /* Description of the format */
  char     * * names;                       /* Filename extensions       */
  unsigned int flags;                       /* File flags (SOX_FILE_*)   */
  sox_format_handler_startread  startread;  /* Initialize decoder        */
  sox_format_handler_read       read;       /* Decode a block of samples */
```

```
  sox_format_handler_stopread   stopread;   /* Close decoder             */
  sox_format_handler_startwrite startwrite; /* Initialize encoder        */
  sox_format_handler_write      write;      /* Encode a block of samples */
  sox_format_handler_stopwrite  stopwrite;  /* Close encoder             */
  sox_format_handler_seek       seek;       /* Reposition reader         */
  unsigned   * write_formats;               /* Encodings and precision   */
  unsigned   * write_rates;                 /* Sample rates              */
  size_t       priv_size;                   /* sizeof(priv_t)            */
};
```

sox_uint32_t sox_lib_version_code
: Checked when format plugins are loaded to ensure they were compiled for the same version of libsox that is calling them.

char *description
: A short description of the format, printed by **sox_ng --help-format**.

char **names
: A null-terminated array of filename extensions (without the dot) that are handled by this format.

unsigned int flags
: The logical OR of:

SOX_FILE_NOSTDIO	The handler does not use stdio routines
SOX_FILE_DEVICE	The file is an audio device
SOX_FILE_PHONY	Phony file/device (for example /dev/null)
SOX_FILE_REWIND	File should be rewound to write header
SOX_FILE_BIT_REV	Is the file bit-reversed?
SOX_FILE_NIB_REV	Is the file nibble-reversed?
SOX_FILE_ENDIAN	Is the file format endian?
SOX_FILE_ENDBIG	For an endian file format, is it big endian?
SOX_FILE_MONO	Do channel restrictions allow mono?
SOX_FILE_STEREO	Do channel restrictions allow stereo?
SOX_FILE_QUAD	Do channel restrictions allow quad?
SOX_FILE_LIT_END	A mask to OR in if the file is little-endian
SOX_FILE_BIG_END	A mask to OR in if the file is big-endian
SOX_FILE_CHANS	A mask to interrogate channels restrictions. If **flags & SOX_FILE_CHANS** is 0, there are no restrictions

int (*startread)(sox_format_t *_fp_)
: A function to set up the format parameters, read in a data header and do anything else that needs to be done before decoding data. It returns **SOX_SUCCESS** if all went well, or a non-zero value if there was some problem with the file, in which case **read** and **stopread** will not be called.

  At exit from **startread**, **sox_format_t.signal** should be completely filled in, using either data from the file's headers (if available) or whatever the format is guessing/assuming if header data is not available.

size_t (*read)(sox_format_t *_ft_, sox_sample_t *_buf_, size_t _len_)
: Given a buffer and a length, read up to that many samples, transform them into signed long integers and copy them into the buffer. It returns the number of samples actually read.

int (*stopread)(sox_format_t *_ft_)
: Does what needs to be done when it has finished reading.

int (*startwrite)(sox_format_t *_ft_)
: Set up the format parameters, maybe write out a data header, and any other preliminaries for encoding the format. It returns **SOX_SUCCESS** if all went well, or a non-zero value if there was some problem, in which case **write** and **stopwrite** will not be called.

At exit from **startwrite**, **sox_format_t.signal** should be completely filled in, using either the data that was specified, or values chosen by the format based on the format's defaults or capabilities.

size_t (*write)(sox_format_t *_ft_, sox_sample_t *_buf_, size_t _len_)
Given a buffer and a length, copy that many samples out of the buffer, convert them from signed longs to the appropriate data types and write them to the file. If it can't write all the samples out, it returns a lesser number of samples.

int (*stopwrite)(sox_format_t *_ft_)
Fix up the file header or whatever else needs to be done before closing the file.

unsigned *write_formats
An array of values indicating the encodings and precisions supported for writing (encoding). Precisions are specified with default precision first, and with 0 and repeat, and list ends with with one more 0.

An example for a format that supports signed integers at depths of 16 and 24 bits with a default of 16, and 8-bit unsigned:

```
unsigned const * formats = {
  SOX_ENCODING_SIGN2, 16, 24, 0,
  SOX_ENCODING_UNSIGNED, 8, 0,
  0 };
```

sox_rate_t *write_rates
An zero-terminated array of sample rates that are supported for writing, NULL if all rates are supported.

size_t priv_size
The size of the format's **priv_t**. SoX automatically allocates a buffer in which the handler can store its private data, of which the size is specified here. Usually this will be **sizeof(priv_t)**. The buffer is allocated and zeroed before the call to **startread/startwrite**, is freed after the call to **stopread/stopwrite** and is provided via **ft->priv** in each call to the handler's functions.

EXAMPLES

The SoX source code includes a skeleton C file to assist you in writing new formats (**src/skelform.c**). New formats can often just deal with the header and then use **raw.c**'s routines for reading and writing.

SEE ALSO

sox_ng(1), **libsox_ng**(3), **soxformat_ng**(7)**and** `src/skelform.c` in the SoX source code.