

libannodex Reference Manual

0.5.8

Generated by Doxygen 1.3.2

Wed Oct 15 07:18:59 2003

Contents

1	libannodex Main Page	1
1.1	Introduction	1
1.2	Terminology	1
1.3	Licensing	2
2	libannodex Module Index	3
2.1	libannodex Modules	3
3	libannodex Data Structure Index	5
3.1	libannodex Data Structures	5
4	libannodex File Index	7
4.1	libannodex File List	7
5	libannodex Page Index	9
5.1	libannodex Related Pages	9
6	libannodex Module Documentation	11
6.1	Writing Annodex media	11
6.2	Writing to files and file descriptors	12
6.3	Reading from Annodex media	14
6.4	Reading from files and file descriptors	15
6.5	Reading from memory buffers	16
6.6	Advanced management of AnxRead callbacks	17
7	libannodex Data Structure Documentation	21
7.1	_AnxSubstream Struct Reference	21
7.2	my_data Struct Reference	22
8	libannodex File Documentation	23
8.1	annodex.h File Reference	23

8.2	anx_constants.h File Reference	24
8.3	anx_core.h File Reference	25
8.4	anx_general.h File Reference	30
8.5	anx_int64.h File Reference	40
8.6	anx_media.h File Reference	41
8.7	anx_read.h File Reference	45
8.8	anx_types.h File Reference	51
8.9	anx_write.h File Reference	52
9	libannodex Page Documentation	55
9.1	Bug List	55

Chapter 1

libannodex Main Page

1.1 Introduction

This is the documentation for the libannodex C API. libannodex provides a complete programming interface for reading and writing Annodex media. Annodex is a free, open and unpatented format for annotating and indexing media and other forms of continuous data, and can be used on the World Wide Web for "video surfing". For more information about Annodex, see **Annodex.net**.

General functions can be found in **anx_general.h**

1.2 Terminology

libannodex introduces terminology directly related to Annodex format, and also borrows some terminology from the underlying Ogg framework. Ogg is a non-heirarchical container format developed by Monty at Xiph.Org, originally for the Ogg Vorbis audio codec.

1.2.1 Managing multitrack data

Annodex format allows time-synchronous interleaving of multiple data tracks such as audio, video and annotations. In Ogg terms, each track is a "logical bitstream", and is uniquely identified by a serial number or "serialno".

- logical bitstream: a track of media or annotations data
- serialno: an integer identifying a logical bitstream

1.2.2 Time: video frames, sampling rates etc.

All data contained within Annodex tracks must be timed. Annodex format introduces the generic concept of 'granulate' to describe such things as framerates and sampling rates.

- granule: a generic unit of time, specified per logical bitstream

1.3 Licensing

libannodex is provided under the following BSD-style open source license:

Copyright (C) 2003 Commonwealth Scientific and Industrial Research Organisation (CSIRO) Australia

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of CSIRO Australia nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE ORGANISATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 2

libannodex Module Index

2.1 libannodex Modules

Here is a list of all modules:

Writing Annodex media	11
Writing to files and file descriptors	12
Reading from Annodex media	14
Reading from files and file descriptors	15
Reading from memory buffers	16
Advanced management of AnxRead callbacks	17

Chapter 3

libannodex Data Structure Index

3.1 libannodex Data Structures

Here are the data structures with brief descriptions:

_AnxSubstream (A track of data)	21
my_data (Copyright (C) 2003 Commonwealth Scientific and Industrial Research Organisation (CSIRO) Australia)	22

Chapter 4

libannodex File Index

4.1 libannodex File List

Here is a list of all documented files with brief descriptions:

annodex.h (This header is provided for convenience and includes all files required for reading or writing Annodex media)	23
anx_constants.h (Named constants used by the libannodex C API)	24
anx_core.h (Core datatypes etc)	25
anx_general.h (General functions related to Annodex media)	30
anx_int64.h (Platform specific types for int64)	40
anx_media.h (Specification of AnxMediaImporter)	41
anx_read.h (Libannodex provides a convenient callback based framework for reading Annodex media)	45
anx_read_dox.h	??
anx_types.h (Public structures and datatypes)	51
anx_write.h (Writer specific functions)	52

Chapter 5

libannodex Page Index

5.1 libannodex Related Pages

Here is a list of all related documentation pages:

Bug List	55
--------------------	----

Chapter 6

libannodex Module Documentation

6.1 Writing Annodex media

Modules

- Writing to files and file descriptors

6.2 Writing to files and file descriptors

If you wish to write Annodex media to a file or file descriptor (such as a network socket), it can be directly written as follows:

- open an annodex using **anx_open()** or **anx_openfd()**
- import any media using **anx_writer_import()**
- call **anx_write()** repeatedly until it returns 0 or -1
- close the annodex with **anx_close()**

This procedure is illustrated in `src/examples/write-anchor-file.c`:

```
#include <stdio.h>
#include <annodex/annodex.h>

static AnxAnchor my_anchor = {
    NULL, /* id */
    NULL, /* lang */
    NULL, /* dir */
    NULL, /* track */
    "http://www.annodex.net/", /* href */
    "Find out about Annodex media", /* hrefdesc */
    NULL, /* image */
    NULL, /* meta elements */
    NULL /* desc elements */
};

int
main (int argc, char *argv[])
{
    ANNODEX * anx = NULL;
    char * infilename, * outfilename;
    long n;

    if (argc != 3) {
        fprintf (stderr, "Usage: %s infile outfile. anx\n", argv[0]);
        exit (1);
    }

    /* Load all importers */
    anx_init_importers ("/*");

    infilename = argv[1];
    outfilename = argv[2];

    /* Create an ANNODEX* writer, writing to outfilename */
    anx = anx_open (outfilename, ANX_WRITE);

    /* Import infilename into the writer */
    anx_writer_import (anx, infilename, NULL /* id */,
                      NULL /* unknown mime-type */,
                      0 /* seek_offset */, -1 /* seek_end */, 0 /* flags */);

    /* Insert an anchor starting at time 0 */
    anx_insert_anchor (anx, 0, &my_anchor);

    /* End the anchor at time 2.0 seconds */
    anx_insert_anchor (anx, 2.0, NULL);

    while ((n = anx_write (anx, 1024)) > 0);
}
```



```
    anx_close (anx);  
    exit (0);  
}
```

6.3 Reading from Annodex media

6.3.1 Detailed Description

libannodex provides a convenient callback based framework for reading Annodex media.

After opening an annodex for reading, you can attach various callbacks relevant to the parts of the file you are interested in, including the stream header, track headers, head element, anchors and media data. Then, as bytes are read, libannodex will call your callbacks as appropriate.

- General functions related to opening, closing and seeking are located in **anx_general.h** .
- Functions directly related to reading are located in **anx_read.h**

Modules

- Reading from files and file descriptors
- Reading from memory buffers
- Advanced management of AnxRead callbacks

6.4 Reading from files and file descriptors

If the Annodex media you wish to access is directly available as a local file or via a file descriptor (such as a network socket), it can be directly opened as follows:

- open an annodex using **anx_open()** or **anx_openfd()**
- attach read callbacks using **anx_set_read_*_callback()**
- call **anx_read()** repeatedly until it returns 0 or -1
- close the annodex with **anx_close()**

This procedure is illustrated in `src/examples/print-title-file.c`:

```
#include <stdio.h>
#include <annodex/annodex.h>

static int
read_head (ANNODEX * anx, const AnxHead * head, void * user_data)
{
    puts (head->title);
    return ANX_CONTINUE;
}

int
main (int argc, char *argv[])
{
    ANNODEX * anx = NULL;
    char * filename;
    long n;

    if (argc != 2) {
        fprintf (stderr, "Usage: %s file.anx\n", argv[0]);
        exit (1);
    }

    filename = argv[1];

    anx = anx_open (filename, ANX_READ);

    anx_set_read_head_callback (anx, read_head);

    while ((n = anx_read (anx, 1024)) > 0);

    anx_close (anx);

    exit (0);
}
```

6.5 Reading from memory buffers

Sometimes it is not possible to provide a file descriptor for a data source; for example, if the data is extracted from a high level library. In this case, you must directly input the data to libannodex using memory buffers as follows:

- open an annodex using **anx_new()**
- attach read callbacks using **anx_set_read_*_callback()**
- call **anx_reader_input()** repeatedly until it returns 0 or -1
- close the annodex with **anx_close()**

This procedure is illustrated in `src/examples/print-title-memory.c`:

```
#include <stdio.h>
#include <annodex/annodex.h>

static int
read_head (ANNODEX * anx, const AnxHead * head, void * user_data)
{
    puts (head->title);
    return ANX_CONTINUE;
}

int
main (int argc, char *argv[])
{
    ANNODEX * anx = NULL;
    unsigned char buf[1024];
    long n = 1024;

    anx = anx_new (ANX_READ);

    anx_set_read_head_callback (anx, read_head);

    while (n > 0) {
        /* Fill a memory buffer */
        n = fread (buf, 1, 1024, stdin);

        /* Input that memory buffer into the annodex */
        n = anx_reader_input (anx, buf, n);
    }

    anx_close (anx);

    exit (0);
}
```

6.6 Advanced management of AnxRead callbacks

You retain control of the number of bytes read or input, and the callbacks you provide can instruct libannodex to immediately return control back to your application.

6.6.1 Callbacks

It is not required to implement all callbacks.

6.6.2 Return values

- ANX_CONTINUE on success, and to inform `anx_read*()` functions to continue on to the next packet
- ANX_STOP_OK on success, to inform `anx_read*()` functions to return without further processing
- ANX_STOP_ERR on error, to inform `anx_read*()` functions to return without further processing

These mechanisms are illustrated in `src/examples/print-lots.c`:

```
#include <stdio.h>
#include <string.h>
#include <annodex/annodex.h>

struct my_data {
    char * filename;
    long interesting_serialno;
    int interesting_media_packets;
    int done;
};

static int
read_stream (ANNODEX * anx, double timebase, char * utc, void * user_data)
{
    struct my_data * happy = (struct my_data *) user_data;

    printf ("Welcome to %s! The timebase is %f\n", happy->filename, timebase);
    return ANX_CONTINUE;
}

static int
read_track (ANNODEX * anx, long serialno, char * id, char * mime_type,
            anx_int64_t granule_rate_n, anx_int64_t granule_rate_d,
            int nr_header_packets, void * user_data)
{
    struct my_data * happy = (struct my_data *) user_data;

    /* Ignore the annotations track, we don't find it interesting! */
    if (!strcmp (mime_type, "text/x-annodex")) return ANX_CONTINUE;

    printf ("Our first track has mime-type %s and granule rate %lld/%lld.\n",
            mime_type, granule_rate_n, granule_rate_d);

    printf ("We will remember it by its serial number %ld and mark it with crosses.\n", serialno);
    happy->interesting_serialno = serialno;

    /* We don't care about any other tracks! */
    anx_set_read_substream_callback (anx, NULL);
}
```

```

    return ANX_CONTINUE;
}

static int
read_media (ANNODEX * annodex, unsigned char * buf, long n,
            long serialno, anx_int64_t granulepos, void * user_data)
{
    struct my_data * happy = (struct my_data *) user_data;

    if (happy->done) {
        putchar ('!');
    } else if (serialno == happy->interesting_serialno) {
        happy->interesting_media_packets++;
        putchar ('+');
    } else {
        putchar ('.');
    }

    return ANX_CONTINUE;
}

static int
read_anchor3 (ANNODEX * anx, const AnxAnchor * anchor, void * user_data)
{
    struct my_data * happy = (struct my_data *) user_data;

    printf ("\nAnd the third anchor links to %s\n", anchor->href);

    happy->done = 1;

    printf ("This completes our whirlwind tour of the first three anchors!\n");
    return ANX_STOP_OK;
}

static int
read_anchor2 (ANNODEX * anx, const AnxAnchor * anchor, void * user_data)
{
    printf ("\nThe second anchor links to %s\n", anchor->href);
    anx_set_read_anchor_callback (anx, read_anchor3);
    return ANX_CONTINUE;
}

static int
read_anchor1 (ANNODEX * anx, const AnxAnchor * anchor, void * user_data)
{
    printf ("\nThe first anchor links to %s\n", anchor->href);
    anx_set_read_anchor_callback (anx, read_anchor2);
    return ANX_CONTINUE;
}

int
main (int argc, char *argv[])
{
    ANNODEX * anx = NULL;
    struct my_data me;
    long n;

    if (argc != 2) {
        fprintf (stderr, "Usage: %s file.ann\n", argv[0]);
        exit (1);
    }

    me.filename = argv[1];
    me.interesting_serialno = -1;
    me.interesting_media_packets = 0;
    me.done = 0;

```

```

anx = anx_open (me.filename, ANX_READ);

anx_set_read_stream_callback (anx, read_stream);
anx_set_read_substream_callback (anx, read_track);
anx_set_read_media_callback (anx, read_media);
anx_set_read_anchor_callback (anx, read_anchor1);


anx_set_user_data (anx, &me);


while (!me.done && (n = anx_read (anx, 1024)) > 0);


printf ("%d packets from the first track (serialno %ld) were received\n",
        me.interesting_media_packets, me.interesting_serialno);
printf ("before the third anchor.\n");


anx_close (anx);


exit (0);
}

```

which produces output like:

[illegible]

And the third anchor links to <http://www.venus.int/>
This completes our whirlwind tour of the first three anchors!

639 packets from the first track (serialno 1810353996) were received before the third anchor.

Chapter 7

libannodex Data Structure Documentation

7.1 _AnxSubstream Struct Reference

```
#include <anx_types.h>
```

7.1.1 Detailed Description

A track of data.

Data Fields

- long **serialno**
The serialno of the Ogg logical bitstream.
- char * **id**
The text id of the track.
- char * **mime_type**
Mime type of track contents.
- long **nr_header_packets**
number of extra header packets
- anx_int64_t **granule_rate_n**
the granule rate numerator
- anx_int64_t **granule_rate_d**
the granule rate denominator

The documentation for this struct was generated from the following file:

- **anx_types.h**

7.2 my_data Struct Reference

7.2.1 Detailed Description

Copyright (C) 2003 Commonwealth Scientific and Industrial Research Organisation (CSIRO) Australia.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of CSIRO Australia nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE ORGANISATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Data Fields

- char * **filename**
- long **interesting_serialno**
- int **interesting_media_packets**
- int **done**

The documentation for this struct was generated from the following file:

- print-lots.c

Chapter 8

libannodex File Documentation

8.1 annodex.h File Reference

8.1.1 Detailed Description

This header is provided for convenience and includes all files required for reading or writing Annodex media.

```
#include <stdlib.h>
#include <annodex/anx_int64.h>
#include <annodex/anx_types.h>
#include <annodex/anx_constants.h>
#include <annodex/anx_general.h>
#include <annodex/anx_read.h>
#include <annodex/anx_write.h>
```

8.2 anx_constants.h File Reference

8.2.1 Detailed Description

Named constants used by the libannodex C API.

Defines

- `#define ANX_READ 00`
- `#define ANX_WRITE 01`
- `#define ANX_CONTINUE 0`
- `#define ANX_STOP_OK 1`
- `#define ANX_STOP_ERR -1`
- `#define ANX_IGNORE_ANNO 0x1`
- `#define ANX_IGNORE_MEDIA 0x2`
- `#define ANX_IGNORE_IMPORT_IMPORT 0x4`
- `#define ANX_ALLOW_RECURSIVE_IMPORT 0x8`
- `#define ANX_SEEK_SET 0`
- `#define ANX_SEEK_CUR 1`
- `#define ANX_SEEK_END 2`
- `#define ANX_SEEK_HEAD 3`
- `#define ANX_SEEK_PREV 4`
- `#define ANX_SEEK_NEXT 5`
- `#define ANX_EOK 0`
- `#define ANX_ESYSTEM 1`
- `#define ANX_EINVALID 2`
- `#define ANX_ENOTSUPPORTED 3`
- `#define ANX_ENOTREADY 10`
- `#define ANX_ENOSEEK 11`
- `#define ANX_EEOF 12`
- `#define ANX_ENOIMPORTER 13`
- `#define ANX_ENOTIMPLEMENTED 99`

8.3 anx_core.h File Reference

8.3.1 Detailed Description

Core datatypes etc.

8.3.2 AnxList

AnxList implements a doubly linked list. Use like GList ...

```
#include <annodex/anx_types.h>
```

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

Defines

- `#define anx_malloc malloc`
malloc() wrapper to allow easy overriding
- `#define anx_free free`
free() wrapper to allow easy overriding
- `#define anx_strdup(s) ((s)==NULL?NULL:strdup(s))`
simple strdup() wrapper to allow 'duping' NULL strings

Functions

- `double anx_parse_time (const char *str)`
Time parsing.
- `AnxList * anx_list_new (void)`
Create a new list.
- `AnxList * anx_list_clone (AnxList *list)`
Clone a list using the default clone function.
- `AnxList * anx_list_clone_with (AnxList *list, AnxCloneFunc clone)`
Clone a list using a custom clone function.
- `AnxList * anx_list_tail (AnxList *list)`
Return the tail element of a list.
- `AnxList * anx_list_prepend (AnxList *list, void *data)`
Prepend a new node to a list containing given data.
- `AnxList * anx_list_append (AnxList *list, void *data)`
Append a new node to a list containing given data.
- `AnxList * anx_list_add_before (AnxList *list, void *data, AnxList *node)`

Add a new node containing given data before a given node.

- **AnxList * anx_list_add_after** (**AnxList** *list, void *data, **AnxList** *node)

Add a new node containing given data after a given node.

- **AnxList * anx_list_find** (**AnxList** *list, void *data)

Find the first node containing given data in a list.

- **AnxList * anx_list_remove** (**AnxList** *list, **AnxList** *node)

Remove a node from a list.

- **int anx_list_length** (**AnxList** *list)

Query the number of items in a list.

- **int anx_list_is_empty** (**AnxList** *list)

Query if a list is empty, ie.

- **int anx_list_is_singleton** (**AnxList** *list)

Query if the list is singleton, ie.

- **AnxList * anx_list_free_with** (**AnxList** *list, **AnxFreeFunc** free_func)

Free a list, using a given function to free each data element.

- **AnxList * anx_list_free** (**AnxList** *list)

*Free a list, using **anx_free()** to free each data element.*

8.3.3 Function Documentation

8.3.3.1 **AnxList* anx_list_add_after** (**AnxList** * *list*, void * *data*, **AnxList** * *node*)

Add a new node containing given data after a given node.

Parameters:

list the list

data the data element of the newly created node

node the node after which to add the newly created node

Returns:

the head of the list

8.3.3.2 **AnxList* anx_list_add_before** (**AnxList** * *list*, void * *data*, **AnxList** * *node*)

Add a new node containing given data before a given node.

Parameters:

list the list

data the data element of the newly created node

node the node before which to add the newly created node

Returns:

the head of the list (which may have changed)

8.3.3.3 AnxList* anx_list_append (AnxList * *list*, void * *data*)

Append a new node to a list containing given data.

Parameters:

list the list

data the data element of the newly created node

Returns:

the head of the list

8.3.3.4 AnxList* anx_list_clone (AnxList * *list*)

Clone a list using the default clone function.

Parameters:

list the list to clone

Returns:

a newly cloned list

8.3.3.5 AnxList* anx_list_clone_with (AnxList * *list*, AnxCloneFunc *clone*)

Clone a list using a custom clone function.

Parameters:

list the list to clone

clone the function to use to clone a list item

Returns:

a newly cloned list

8.3.3.6 AnxList* anx_list_find (AnxList * *list*, void * *data*)

Find the first node containing given data in a list.

Parameters:

list the list

data the data element to find

Returns:

the first node containing given data, or NULL if it is not found

8.3.3.7 AnxList* anx_list_free (AnxList * *list*)

Free a list, using **anx_free()** to free each data element.

Parameters:

list the list

Returns:

NULL on success

8.3.3.8 AnxList* anx_list_free_with (AnxList * *list*, AnxFreeFunc *free_func*)

Free a list, using a given function to free each data element.

Parameters:

list the list

free_func a function to free each data element

Returns:

NULL on success

8.3.3.9 int anx_list_is_empty (AnxList * *list*)

Query if a list is empty, ie.

contains no items

Parameters:

list the list

Returns:

1 if the list is empty, 0 otherwise

8.3.3.10 int anx_list_is_singleton (AnxList * *list*)

Query if the list is singleton, ie.

contains exactly one item

Parameters:

list the list

Returns:

1 if the list is singleton, 0 otherwise

8.3.3.11 int anx_list_length (AnxList * *list*)

Query the number of items in a list.

Parameters:

list the list

Returns:

the number of nodes in the list

8.3.3.12 AnxList* anx_list_new (void)

Create a new list.

Returns:

a new list

8.3.3.13 AnxList* anx_list_prepend (AnxList * *list*, void * *data*)

Prepend a new node to a list containing given data.

Parameters:

list the list

data the data element of the newly created node

Returns:

the new list head

8.3.3.14 AnxList* anx_list_remove (AnxList * *list*, AnxList * *node*)

Remove a node from a list.

Parameters:

list the list

node the node to remove

Returns:

the head of the list (which may have changed)

8.3.3.15 AnxList* anx_list_tail (AnxList * *list*)

Return the tail element of a list.

Parameters:

list the list

Returns:

the tail element

8.4 anx_general.h File Reference

8.4.1 Detailed Description

General functions related to Annodex media.

```
#include <annodex/anx_types.h>
```

Functions

- **int** **anx_last_error** (**ANNODEX** *annodex)
Retrieve the error code of the most recent error on an annodex.
- **const char *** **anx_strerror** (**ANNODEX** *annodex)
Retrieve a printable error string corresponding to the most recent error on annodex.
- **ANNODEX *** **anx_open** (char *filename, int flags)
Open a file containing Annodex media.
- **ANNODEX *** **anx_openfd** (int fd, int flags)
Attach to an existing file descriptor.
- **ANNODEX *** **anx_new** (int flags)
Create a managed Annodex handle This is an alternative interface for non-file annodexes.
- **int** **anx_flush** (**ANNODEX** *annodex)
Flush any unwritten data associated with an annodex.
- **ANNODEX *** **anx_close** (**ANNODEX** *annodex)
Close an annodex.
- **int** **anx_destroy** (**ANNODEX** *annodex)
Forcefully close an annodex without flushing any file descriptors.
- **int** **anx_ready** (**ANNODEX** *annodex)
Query if an annodex is ready.
- **int** **anx_eos** (**ANNODEX** *annodex)
Query if an annodex has reached its 'End Of Stream' markers.
- **AnxHead *** **anx_set_head** (**ANNODEX** *annodex, **AnxHead ***head)
Set the head element of the Annodex.
- **AnxHead *** **anx_get_head** (**ANNODEX** *annodex)
Get a copy of the head element of an annodex.
- **anx_int64_t** **anx_tell** (**ANNODEX** *annodex)
Query the current byte offset of an annodex.
- **int** **anx_seek_id** (**ANNODEX** *annodex, const char *id)

Seek to an anchor identified by id.

- double **anx_get_timebase** (ANNODEX *annodex)
Query the timebase of an annodex.
- double **anx_set_timebase** (ANNODEX *annodex, double seconds)
- double **anx_tell_time** (ANNODEX *annodex)
Query the current offset of an annodex expressed as time in seconds.
- double **anx_seek_time** (ANNODEX *annodex, double seconds, int whence)
Seek to a time point in an annodex.
- AnxList * **anx_get_substream_list** (ANNODEX *annodex)
Query the contents of tracks in an annodex.
- char * **anx_get_mime_type** (ANNODEX *annodex, long serialno)
Query the mime type of a track in an annodex.
- long **anx_get_nr_headers** (ANNODEX *annodex, long serialno)
Query the number of header packets of a track in an annodex.
- int **anx_get_granule_rate** (ANNODEX *annodex, long serialno, anx_int64_t *granule_rate_n, anx_int64_t *granule_rate_d)
Query the granule rate of a track in an annodex.
- anx_int64_t **anx_time_to_granules** (ANNODEX *annodex, long serialno, double seconds)
Convert a time in seconds to a granule position for a track in an annodex.
- double **anx_granules_to_time** (ANNODEX *annodex, long serialno, anx_int64_t granules)
Convert a granule position to time in seconds for a track in an annodex.
- int **anx_snprint_head** (char *buf, int n, AnxHead *h)
Print an AnxHead structure to a memory buffer.
- int **anx_snprint_anchor** (char *buf, int n, AnxAnchor *a, double start, double end)
Print an AnxAnchor structure to a memory buffer.
- AnxHead * **anx_head_free** (AnxHead *head)
Free an AnxHead structure.
- AnxAnchor * **anx_anchor_free** (AnxAnchor *anchor)
Free an AnxAnchor structure.
- AnxHead * **anx_head_clone** (AnxHead *head)
Clone an AnxHead structure.
- AnxAnchor * **anx_anchor_clone** (AnxAnchor *anchor)
Clone an AnxAnchor structure.

- AnxMetaElement * **anx_meta_element_clone** (AnxMetaElement *meta)
Clone an AnxMetaElement structure.
- AnxDescElement * **anx_desc_element_clone** (AnxDescElement *desc)
Clone an AnxDescElement structure.

8.4.2 Function Documentation

8.4.2.1 AnxAnchor* anx_anchor_clone (AnxAnchor * *anchor*)

Clone an AnxAnchor structure.

Parameters:

anchor the anchor to clone

Returns:

a new anchor structure

Note:

All components of the original anchor are cloned, including all strings and its lists of meta and desc tags.

8.4.2.2 AnxAnchor* anx_anchor_free (AnxAnchor * *anchor*)

Free an AnxAnchor structure.

Parameters:

anchor the structure to free

Returns:

NULL on success

8.4.2.3 ANNODEX* anx_close (ANNODEX * *annodex*)

Close an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

NULL on success; returns the unchanged annodex on failure (eg. due to system close() failing)

8.4.2.4 AnxDescElement* anx_desc_element_clone (AnxDescElement * *desc*)

Clone an AnxDescElement structure.

Parameters:

desc the desc tag structure to clone

Returns:

a new desc tag structure

8.4.2.5 int anx_destroy (ANNODEX * *annodex*)

Forcefully close an annodex without flushing any file descriptors.

Parameters:

annodex an ANNODEX* handle

Returns:

0 on success, -1 on failure

8.4.2.6 int anx_eos (ANNODEX * *annodex*)

Query if an annodex has reached its 'End Of Stream' markers.

Parameters:

annodex an ANNODEX* handle

Returns:

0 if annodex is not at eos, 1 if it is

8.4.2.7 int anx_flush (ANNODEX * *annodex*)

Flush any unwritten data associated with an annodex.

Parameters:

annodex An ANNODEX* handle

Returns:

0 on success, -1 on failure

8.4.2.8 int anx_get_granule_rate (ANNODEX * *annodex*, long *serialno*, anx_int64_t * *granule_rate_n*, anx_int64_t * *granule_rate_d*)

Query the granule rate of a track in an annodex.

Parameters:

annodex an ANNODEX* handle

serialno the serialno of the track to query

granule_rate_n return granule_rate numerator

granule_rate_d return granule_rate denominator

Returns:

0 on success, -1 on failure

8.4.2.9 AnxHead* anx_get_head (ANNODEX * *annodex*)

Get a copy of the head element of an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

a copy of the head element

8.4.2.10 char* anx_get_mime_type (ANNODEX * *annodex*, long *serialno*)

Query the mime type of a track in an annodex.

Parameters:

annodex an ANNODEX* handle

serialno the serialno of the track to query

8.4.2.11 long anx_get_nr_headers (ANNODEX * *annodex*, long *serialno*)

Query the number of header packets of a track in an annodex.

Parameters:

annodex an ANNODEX* handle

serialno the serialno of the track to query

8.4.2.12 AnxList* anx_get_substream_list (ANNODEX * *annodex*)

Query the contents of tracks in an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

a list of AnxSubstream *

8.4.2.13 double anx_get_timebase (ANNODEX * *annodex*)

Query the timebase of an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

the timebase of the annodex

8.4.2.14 `double anx_granules_to_time (ANNODEX * annodex, long serialno, anx_int64_t granules)`

Convert a granule position to time in seconds for a track in an annodex.

Parameters:

annodex an ANNODEX* handle
serialno the serialno of the track to query
granules the granule position to convert

8.4.2.15 `AnxHead* anx_head_clone (AnxHead * head)`

Clone an AnxHead structure.

Parameters:

head the head to clone

Returns:

a new head structure

Note:

All components of the original head are cloned, including all strings and its list of meta tags.

8.4.2.16 `AnxHead* anx_head_free (AnxHead * head)`

Free an AnxHead structure.

Parameters:

head the structure to free

Returns:

NULL on success

8.4.2.17 `int anx_last_error (ANNODEX * annodex)`

Retrieve the error code of the most recent error on an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

the error code of the most recent error

8.4.2.18 `AnxMetaElement* anx_meta_element_clone (AnxMetaElement * meta)`

Clone an AnxMetaElement structure.

Parameters:

meta the meta tag structure to clone

Returns:

a new meta tag structure

8.4.2.19 ANNODEX* anx_new (int *flags*)

Create a managed Annodex handle This is an alternative interface for non-file annodexes.

Parameters:

flags ANX_READ or ANX_WRITE

Returns:

an ANNODEX* handle

8.4.2.20 ANNODEX* anx_open (char * *filename*, int *flags*)

Open a file containing Annodex media.

Parameters:

filename path to the file

flags ANX_READ or ANX_WRITE

Returns:

an ANNODEX* handle

8.4.2.21 ANNODEX* anx_openfd (int *fd*, int *flags*)

Attach to an existing file descriptor.

Parameters:

fd an open file descriptor

flags ANX_READ or ANX_WRITE

Returns:

an ANNODEX* handle

8.4.2.22 int anx_ready (ANNODEX * *annodex*)

Query if an annodex is ready.

Parameters:

annodex an ANNODEX* handle

Returns:

0 if annodex is not ready, 1 if it is

8.4.2.23 int anx_seek_id (ANNODEX * *annodex*, const char * *id*)

Seek to an anchor identified by id.

Parameters:

annodex an ANNODEX* handle

id The id of the anchor to seek to

8.4.2.24 double anx_seek_time (ANNODEX * *annodex*, double *seconds*, int *whence*)

Seek to a time point in an annodex.

Parameters:

- annodex* an ANNODEX* handle
- seconds* the time to seek to
- whence* whence parameter (defined in **anx_constants.h**)

8.4.2.25 AnxHead* anx_set_head (ANNODEX * *annodex*, AnxHead * *head*)

Set the head element of the Annodex.

Parameters:

- annodex* an ANNODEX* handle
- head* a complete AnxHead structure to set

Returns:

the head if successful, NULL on failure

Note:

libannodex makes a copy of the head; it can be safely freed after returning from this call.

8.4.2.26 double anx_set_timebase (ANNODEX * *annodex*, double *seconds*)**Parameters:**

- annodex* an ANNODEX* handle
- seconds* the new timebase

8.4.2.27 int anx_snprint_anchor (char * *buf*, int *n*, AnxAnchor * *a*, double *start*, double *end*)

Print an AnxAnchor structure to a memory buffer.

The head is serialized to XML conformant with the anxa DTD.

Parameters:

- buf* a buffer to print into
- n* the maximum number of characters to print
- a* the anchor to print
- start* the value of the 'start' attribute, as a time in seconds. If start is negative, the 'start' attribute is not printed.

Returns:

the number of characters printed, -1 on potential overrun

Bug

This should follow C99 semantics

8.4.2.28 `int anx_snprint_head (char * buf, int n, AnxHead * h)`

Print an AnxHead structure to a memory buffer.

The head is serialized to XML conformant with the anxhead DTD.

Parameters:

buf a buffer to print into

n the maximum number of characters to print

h the head to print

Returns:

the number of characters printed, -1 on potential overrun

Bug

This should follow C99 semantics

8.4.2.29 `const char* anx_strerror (ANNODEX * annodex)`

Retrieve a printable error string corresponding to the most recent error on annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

an error string

8.4.2.30 `anx_int64_t anx_tell (ANNODEX * annodex)`

Query the current byte offset of an annodex.

Parameters:

annodex an ANNODEX* handle

Returns:

the current byte offset XXX: Deprecated?

8.4.2.31 `double anx_tell_time (ANNODEX * annodex)`

Query the current offset of an annodex expressed as time in seconds.

Parameters:

annodex an ANNODEX* handle

Returns:

the current time offset

8.4.2.32 `anx_int64_t anx_time_to_granules` (ANNODEX * *annodex*, long *serialno*, double *seconds*)

Convert a time in seconds to a granule position for a track in an annodex.

Parameters:

annodex an ANNODEX* handle

serialno the serialno of the track to query

seconds the time to convert

Returns:

the granulepos corresponding to seconds

8.5 anx_int64.h File Reference

8.5.1 Detailed Description

Platform specific types for int64.

Adapted from libogg's method.

```
#include <sys/types.h>
```

Typedefs

- typedef int64_t **anx_int64_t**

This typedef was determined on the system on which the documentation was generated.

8.5.2 Typedef Documentation

8.5.2.1 typedef int64_t anx_int64_t

This typedef was determined on the system on which the documentation was generated.

To query this on your system, do eg.

```
echo "#include <annodex/anx_int64.h>" | gcc -E - | grep anx_int64_t
```

8.6 anx_media.h File Reference

8.6.1 Detailed Description

Specification of AnxMediaImporter.

- When read requires starting a new frame or GOP, check the need_sync flag. If this is set, output a sync frame.

Note:

Always check not to seek before start_packet and not to seek or read past end_packet.

- seek should return the actual position after seeking (as the backend may not be able to seek exactly to the requested position).

Backends must implement the following function:

- AnxMediaImporter * anx_media_importer_init (int i);

Importers are accessed by index i using values from 0 upwards. For out of range indexes, this function must return NULL.

No other symbols in a backend should be visible (ie. declare all other globals and functions as static).

```
#include <annodex/anx_int64.h>
```

```
#include <annodex/anx_types.h>
```

```
#include <annodex/anx_core.h>
```

Data Structures

- struct **_AnxMedia**
- struct **_AnxMediaImporter**
- struct **_AnxMediaSubstream**

Typedefs

- typedef **_AnxMediaImporter** **AnxMediaImporter**
An AnxMediaImporter implements generic functions for retrieving data from sources of a particular MIME type.
- typedef **_AnxMediaSubstream** **AnxMediaSubstream**
An AnxMediaSubstream contains one track of data.
- typedef **_AnxMedia** **AnxMedia**
An AnxMedia contains an instance of an active media object, which may in turn contain several substreams.
- typedef **AnxMedia** *(* **AnxMediaOpenFunc**)(const char *path, const char *id, int ignore_media, double start_time, double end_time, AnxImportCallbacks *import_callbacks)

Signature of a function for opening a media object by filename.

- typedef **AnxMedia** *(* **AnxMediaOpenFDFunc**)(int fd, const char *id, int ignore_media, double start_time, double end_time, AnxImportCallbacks *import_callbacks)

Signature of a function for opening a media object attached to an open file descriptor.

- typedef long(* **AnxMediaReadFunc**)(AnxMedia *media, unsigned char *buf, long n, long bound)

Signature of a function for reading bytes from a media object.

- typedef long(* **AnxMediaSizeofNextReadFunc**)(AnxMedia *media, long bound)

Signature of a function to return the preferred next read size.

- typedef int(* **AnxMediaCloseFunc**)(AnxMedia *media)

Signature of a function to close a media object.

Functions

- int **anx_register_media_importer** (AnxMediaImporter *importer)

Register an AnxMediaImporter object with libannodex.

- int **anx_unregister_media_importer** (AnxMediaImporter *importer)

Unregister an importer previously registered with libannodex.

8.6.2 Typedef Documentation

8.6.2.1 typedef int(* AnxMediaCloseFunc)(AnxMedia * media)

Signature of a function to close a media object.

Parameters:

media an AnxMedia* handle

8.6.2.2 typedef AnxMedia*(* AnxMediaOpenFDFunc)(int fd, const char * id, int ignore_media, double start_time, double end_time, AnxImportCallbacks * import_callbacks)

Signature of a function for opening a media object attached to an open file descriptor.

Parameters:

fd an open file descriptor

id the id of this

ignore_media a flag to indicate that the importer should ignore any media read requests, ie. just deliver anchors

start_time the start time to initially seek to

end_time a time bound to end on

import_callbacks callbacks to call when further importing or anchor inserting is required.

Returns:

a new AnxMedia* handle

8.6.2.3 `typedef AnxMedia*(* AnxMediaOpenFunc)(const char * path, const char * id, int ignore_media, double start_time, double end_time, AnxImportCallbacks * import_callbacks)`

Signature of a function for opening a media object by filename.

Parameters:

path the path to the media file

id the id of this

ignore_media a flag to indicate that the importer should ignore any media read requests, ie. just deliver anchors

start_time the start time to initially seek to

end_time a time bound to end on

import_callbacks callbacks to call when further importing or anchor inserting is required.

Returns:

a new AnxMedia* handle

8.6.2.4 `typedef long(* AnxMediaReadFunc)(AnxMedia * media, unsigned char * buf, long n, long bound)`

Signature of a function for reading bytes from a media object.

Parameters:

media an AnxMedia* handle

buf a buffer to read data into

n a maximum number of bytes to read

bound a maximum granulecount to read

8.6.2.5 `typedef long(* AnxMediaSizeofNextReadFunc)(AnxMedia * media, long bound)`

Signature of a function to return the preferred next read size.

Parameters:

media an AnxMedia* handle

bound a maximum granulecount to read

8.6.3 Function Documentation

8.6.3.1 `int anx_register_media_importer (AnxMediaImporter * importer)`

Register an AnxMediaImporter object with libannodex.

Parameters:

importer the importer

Returns:

0 on success, -1 on failure

8.6.3.2 `int anx_unregister_media_importer (AnxMediaImporter * importer)`

Unregister an importer previously registered with libannodex.

Parameters:

importer the importer

Returns:

0 on success, -1 on failure

8.7 anx_read.h File Reference

8.7.1 Detailed Description

libannodex provides a convenient callback based framework for reading Annodex media.

After opening an annodex for reading, you can attach various callbacks relevant to the parts of the file you are interested in, including the stream header, track headers, head element, anchors and media data. Then, as bytes are read, libannodex will call your callbacks as appropriate.

For tutorial information and examples, refer to the following:

- Reading from files and file descriptors
- Reading from memory buffers
- Advanced management of AnxRead* callbacks

```
#include <annodex/anx_types.h>
```

Typedefs

- typedef int(* **AnxReadStream**)(ANNODEX *annodex, double timebase, char *utc, void *user_data)
Signature for a callback called when the 'Annodex' stream header is parsed.
- typedef int(* **AnxReadSubstream**)(ANNODEX *annodex, long serialno, char *id, char *mime_type, **anx_int64_t** granule_rate_n, **anx_int64_t** granule_rate_d, int nr_header_packets, void *user_data)
Signature for a callback called each time an 'AnxData' track header is parsed.
- typedef int(* **AnxReadHead**)(ANNODEX *annodex, const AnxHead *head, void *user_data)
Signature for a callback called when the head element is parsed.
- typedef int(* **AnxReadAnchor**)(ANNODEX *annodex, const AnxAnchor *anchor, void *user_data)
Signature for a callback called each time an anchor element is parsed.
- typedef int(* **AnxReadMedia**)(ANNODEX *annodex, unsigned char *data, long n, long serialno, **anx_int64_t** granulepos, void *user_data)
Signature for a callback called each time a media packet is found.

Functions

- int **anx_set_read_stream_callback** (ANNODEX *annodex, **AnxReadStream** read_stream)
Set the function to call each time an 'Annodex' stream header is parsed.
- int **anx_set_read_substream_callback** (ANNODEX *annodex, **AnxReadSubstream** read_substream)

Set the function to call each time an 'AnxData' track header is parsed.

- **int `anx_set_read_head_callback` (`ANNODEX` *annodex, `AnxReadHead` read_head)**
Set the function to call when the head element is read.
- **int `anx_set_read_anchor_callback` (`ANNODEX` *annodex, `AnxReadAnchor` read_anchor)**
Set the function to call each time an anchor is parsed.
- **int `anx_set_read_media_callback` (`ANNODEX` *annodex, `AnxReadMedia` read_media)**
Set the function to call each time a media packet is read.
- **int `anx_set_user_data` (`ANNODEX` *annodex, void *user_data)**
Associate arbitrary data with read callbacks.
- **long `anx_read` (`ANNODEX` *annodex, long n)**
Read from an annodex opened with `anx_open()` or `anx_openfd()`.
- **long `anx_reader_input` (`ANNODEX` *annodex, unsigned char *buf, long n)**
Input data from a memory buffer into an annodex.

8.7.2 Typedef Documentation

8.7.2.1 `typedef int(* AnxReadAnchor)(ANNODEX * annodex, const AnxAnchor * anchor, void * user_data)`

Signature for a callback called each time an anchor element is parsed.

Parameters:

annodex the `ANNODEX*` handle in use

anchor the anchor element

user_data user defined data previously attached with `anx_set_user_data()`

Returns:

- `ANX_CONTINUE` on success, and to inform `anx_read*()` functions to continue on to the next packet
- `ANX_STOP_OK` on success, to inform `anx_read*()` functions to return without further processing
- `ANX_STOP_ERR` on error, to inform `anx_read*()` functions to return without further processing

8.7.2.2 `typedef int(* AnxReadHead)(ANNODEX * annodex, const AnxHead * head, void * user_data)`

Signature for a callback called when the head element is parsed.

Parameters:

- annodex* the ANNODEx* handle in use
head the head element
user_data user defined data previously attached with **anx_set_user_data()**

Returns:

- ANX_CONTINUE on success, and to inform anx_read*() functions to continue on to the next packet
- ANX_STOP_OK on success, to inform anx_read*() functions to return without further processing
- ANX_STOP_ERR on error, to inform anx_read*() functions to return without further processing

8.7.2.3 typedef int(* AnxReadMedia)(ANNODEx * annodex, unsigned char * data, long n, long serialno, anx_int64_t granulepos, void * user_data)

Signature for a callback called each time a media packet is found.

Parameters:

- annodex* the ANNODEx* handle in use
data a pointer to the data read from the packet
n the length in bytes of the data
serialno the track serial number of the packet
granulepos the granule position of the packet
user_data user defined data previously attached with **anx_set_user_data()**

Returns:

- ANX_CONTINUE on success, and to inform anx_read*() functions to continue on to the next packet
- ANX_STOP_OK on success, to inform anx_read*() functions to return without further processing
- ANX_STOP_ERR on error, to inform anx_read*() functions to return without further processing

8.7.2.4 typedef int(* AnxReadStream)(ANNODEx * annodex, double timebase, char * utc, void * user_data)

Signature for a callback called when the 'Annodex' stream header is parsed.

Parameters:

- annodex* the ANNODEx* handle in use
timebase the timebase of the annodex
utc the UTC timebase of the annodex
user_data user defined data previously attached with **anx_set_user_data()**

Returns:

- ANX_CONTINUE on success, and to inform anx_read*() functions to continue on to the next packet

- ANX_STOP_OK on success, to inform `anx_read*()` functions to return without further processing
- ANX_STOP_ERR on error, to inform `anx_read*()` functions to return without further processing

8.7.2.5 `typedef int(* AnxReadSubstream)(ANNODEX * annodex, long serialno, char * id, char * mime_type, anx_int64_t granule_rate_n, anx_int64_t granule_rate_d, int nr_header_packets, void * user_data)`

Signature for a callback called each time an 'AnxData' track header is parsed.

Parameters:

annodex the ANNODEX* handle in use
serialno the track serial number
id the id attribute of the track
mime_type the MIME type of the track
granule_rate_n the numerator of the granule rate
granule_rate_d the denominator of the granule rate
nr_header_packets the number of header packets of the track
user_data user defined data previously attached with `anx_set_user_data()`

Returns:

- ANX_CONTINUE on success, and to inform `anx_read*()` functions to continue on to the next packet
- ANX_STOP_OK on success, to inform `anx_read*()` functions to return without further processing
- ANX_STOP_ERR on error, to inform `anx_read*()` functions to return without further processing

8.7.3 Function Documentation

8.7.3.1 `long anx_read (ANNODEX * annodex, long n)`

Read from an annodex opened with `anx_open()` or `anx_openfd()`.

The annodex is automatically parsed and relevant read callbacks are called, stopping when the required number of bytes have been read, or earlier if any of the read callbacks returned other than ANX_CONTINUE.

Parameters:

annodex an ANNODEX* handle
n count of bytes to read

Returns:

the count of bytes actually read

8.7.3.2 long anx_reader_input (ANNODEX * *annodex*, unsigned char * *buf*, long *n*)

Input data from a memory buffer into an annodex.

Usually the annodex would have been created with ***anx_new()***. The annodex is automatically parsed and relevant read callbacks are called, stopping when the required number of bytes have been read, or earlier if any of the read callbacks returned other than ANX_CONTINUE.

Parameters:

annodex an ANNODEX* handle
buf a memory buffer containing data to input
n count of bytes to input

Returns:

the count of bytes actually input

8.7.3.3 int anx_set_read_anchor_callback (ANNODEX * *annodex*, AnxReadAnchor *read_anchor*)

Set the function to call each time an anchor is parsed.

Parameters:

annodex an ANNODEX* handle
read_anchor the callback to call

Returns:

0 on success

8.7.3.4 int anx_set_read_head_callback (ANNODEX * *annodex*, AnxReadHead *read_head*)

Set the function to call when the head element is read.

Parameters:

annodex an ANNODEX* handle
read_head the callback to call

Returns:

0 on success, -1 on failure

8.7.3.5 int anx_set_read_media_callback (ANNODEX * *annodex*, AnxReadMedia *read_media*)

Set the function to call each time a media packet is read.

Parameters:

annodex an ANNODEX* handle
read_media the callback to call

Returns:

0 on success, -1 on failure

8.7.3.6 `int anx_set_read_stream_callback (ANNODEX * annodex, AnxReadStream read_stream)`

Set the function to call each time an 'Annodex' stream header is parsed.

Parameters:

annodex an ANNODEX* handle

read_stream the callback to call

Returns:

0 on success, -1 on failure

8.7.3.7 `int anx_set_read_substream_callback (ANNODEX * annodex, AnxReadSubstream read_substream)`

Set the function to call each time an 'AnxData' track header is parsed.

Parameters:

annodex an ANNODEX* handle

read_substream the callback to call

Returns:

0 on success, -1 on failure

8.7.3.8 `int anx_set_user_data (ANNODEX * annodex, void * user_data)`

Associate arbitrary data with read callbacks.

Parameters:

annodex an ANNODEX* handle

user_data a pointer to a user defined object to be passed to each of the read callbacks.

Returns:

0 on success, -1 on failure

8.8 anx_types.h File Reference

8.8.1 Detailed Description

Public structures and datatypes.

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

```
#include <annodex/anx_int64.h>
```

Data Structures

- struct **_AnxAnchor**
- struct **_AnxDescElement**
- struct **_AnxHead**
- struct **_AnxImportCallbacks**
- struct **_AnxList**
- struct **_AnxMetaElement**
- struct **_AnxSubstream**

A track of data.

Typedefs

- typedef void **ANNODEX**
An ANNODEX handle.
- typedef _AnxList **AnxList**
A doubly linked list.
- typedef void (*)(**AnxCloneFunc**)(void *data)
Signature of a cloning function.
- typedef void (*)(**AnxFreeFunc**)(void *data)
Signature of a freeing function.
- typedef _AnxSubstream **AnxSubstream**
Track.
- typedef _AnxHead **AnxHead**
- typedef _AnxAnchor **AnxAnchor**
- typedef _AnxMetaElement **AnxMetaElement**
- typedef _AnxDescElement **AnxDescElement**
- typedef int (*)(**AnxImportStream**)(double timebase, char *utc, void *user_data)
- typedef int (*)(**AnxImportHead**)(AnxHead *head, void *user_data)
- typedef int (*)(**AnxImportAnchor**)(AnxAnchor *anchor, double time_offset, void *user_data)
- typedef int (*)(**AnxImportImport**)(double start_time, char *filename, char *id, char *mime_type, double seek_offset, double seek_end, void *user_data)
- typedef _AnxImportCallbacks **AnxImportCallbacks**

8.9 anx_write.h File Reference

8.9.1 Detailed Description

Writer specific functions.

```
#include <annodex/anx_types.h>
```

Functions

- void **anx_init_importers** (char *mime_type_pattern)
Initialise system importers matching a given mime type pattern.
- long **anx_get_next_page_size** (ANNODEX *annodex)
Query the size of the next page to be written.
- int **anx_writer_import** (ANNODEX *annodex, char *filename, char *id, char *mime_type, double seek_offset, double seek_end, int flags)
Import a file into the current ANNODEX writer.*
- int **anx_writer_set_anno_callbacks** (ANNODEX *annodex, AnxImportHead import_head_callback, AnxImportAnchor import_anchor_callback, void *user_data)
Override the callbacks that importers should call when they wish to handle annotation elements.
- int **anx_writer_set_ii_callback** (ANNODEX *annodex, AnxImportImport import_import_callback, void *user_data)
Override the Import Import callback.
- int **anx_insert_anchor** (ANNODEX *annodex, double at_time, AnxAnchor *anchor)
Insert an anchor into an ANNODEX writer.*
- double **anx_writer_get_end** (ANNODEX *annodex)
- int **anx_writer_set_end** (ANNODEX *annodex, double end_time)
- long **anx_write** (ANNODEX *annodex, long n)
*Write to an annodex opened with **anx_open()** or **anx_openfd()**.*
- long **anx_writer_output** (ANNODEX *annodex, unsigned char *buf, long n)
Output data from an annodex into a memory buffer.
- int **anx_request_header** (ANNODEX *annodex)
- int **anx_request_media_sync** (ANNODEX *annodex)

8.9.2 Function Documentation

8.9.2.1 long anx_get_next_page_size (ANNODEX * annodex)

Query the size of the next page to be written.

Returns:

the size in bytes of the next page to be written

8.9.2.2 void anx_init_importers (char * *mime_type_pattern*)

Initialise system importers matching a given mime type pattern.

Parameters:

mime_type_pattern A mime type or range of mime types

8.9.2.3 int anx_insert_anchor (ANNODEX * *annodex*, double *at_time*, AnxAnchor * *anchor*)

Insert an anchor into an ANNODEX* writer.

Parameters:

annodex An ANNODEX* writer

at_time Time in seconds to schedule the anchor

anchor An anchor

8.9.2.4 long anx_write (ANNODEX * *annodex*, long *n*)

Write to an annodex opened with **anx_open()** or **anx_openfd()**.

Parameters:

annodex An ANNODEX* writer

n count of bytes to write

Returns:

the count of bytes actually written

8.9.2.5 int anx_writer_import (ANNODEX * *annodex*, char * *filename*, char * *id*, char * *mime_type*, double *seek_offset*, double *seek_end*, int *flags*)

Import a file into the current ANNODEX* writer.

Parameters:

annodex An ANNODEX* writer

filename the file to import

id the id of the file

mime_type the mime-type of the whole import file.

seek_offset a time in seconds to begin importing data from

seek_end a time in seconds to finish importing data

flags writer import flags (RECURSIVE etc.)

8.9.2.6 `long anx_writer_output (ANNODEX * annodex, unsigned char * buf, long n)`

Output data from an annodex into a memory buffer.

Usually the annodex would have been created with `anx_new()`.

Parameters:

annodex an ANNODEX* writer

buf a memory buffer of size at least n bytes in which to output

n a count of bytes to output

Returns:

the count of bytes actually output

8.9.2.7 `int anx_writer_set_anno_callbacks (ANNODEX * annodex, AnxImportHead import_head_callback, AnxImportAnchor import_anchor_callback, void * user_data)`

Override the callbacks that importers should call when they wish to handle annotation elements.

The default head callback overwrites the head and the default anchor callback inserts an anchor into the scheduling of the ANNODEX* writer.

8.9.2.8 `int anx_writer_set_ii_callback (ANNODEX * annodex, AnxImportImport import_import_callback, void * user_data)`

Override the Import Import callback.

Parameters:

annodex An ANNODEX* writer

import_import_callback The new Import Import callback

user_data User-defined data to pass to the callback

Chapter 9

libannodex Page Documentation

9.1 Bug List

Global `anx_snprint_anchor(char *buf, int n, AnxAnchor *a, double start, double end)`

This should follow C99 semantics

Global `anx_snprint_head(char *buf, int n, AnxHead *h)` This should follow C99 semantics

Index

- `_AnxSubstream`, 21
- Advanced management of `AnxRead` calls, 17
- `annodex.h`, 23
- `anx_anchor_clone`
 - `anx_general.h`, 32
- `anx_anchor_free`
 - `anx_general.h`, 32
- `anx_close`
 - `anx_general.h`, 32
- `anx_constants.h`, 24
- `anx_core.h`, 25
 - `anx_list_add_after`, 26
 - `anx_list_add_before`, 26
 - `anx_list_append`, 27
 - `anx_list_clone`, 27
 - `anx_list_clone_with`, 27
 - `anx_list_find`, 27
 - `anx_list_free`, 27
 - `anx_list_free_with`, 28
 - `anx_list_is_empty`, 28
 - `anx_list_is_singleton`, 28
 - `anx_list_length`, 28
 - `anx_list_new`, 28
 - `anx_list_prepend`, 29
 - `anx_list_remove`, 29
 - `anx_list_tail`, 29
- `anx_desc_element_clone`
 - `anx_general.h`, 32
- `anx_destroy`
 - `anx_general.h`, 32
- `anx_eos`
 - `anx_general.h`, 33
- `anx_flush`
 - `anx_general.h`, 33
- `anx_general.h`, 30
 - `anx_anchor_clone`, 32
 - `anx_anchor_free`, 32
 - `anx_close`, 32
 - `anx_desc_element_clone`, 32
 - `anx_destroy`, 32
 - `anx_eos`, 33
 - `anx_flush`, 33
 - `anx_get_granule_rate`, 33
 - `anx_get_head`, 33
 - `anx_get_mime_type`, 34
 - `anx_get_nr_headers`, 34
 - `anx_get_substream_list`, 34
 - `anx_get_timebase`, 34
 - `anx_granules_to_time`, 34
 - `anx_head_clone`, 35
 - `anx_head_free`, 35
 - `anx_last_error`, 35
 - `anx_meta_element_clone`, 35
 - `anx_new`, 35
 - `anx_open`, 36
 - `anx_openfd`, 36
 - `anx_ready`, 36
 - `anx_seek_id`, 36
 - `anx_seek_time`, 36
 - `anx_set_head`, 37
 - `anx_set_timebase`, 37
 - `anx_snprint_anchor`, 37
 - `anx_snprint_head`, 37
 - `anx_strerror`, 38
 - `anx_tell`, 38
 - `anx_tell_time`, 38
 - `anx_time_to_granules`, 38
- `anx_get_granule_rate`
 - `anx_general.h`, 33
- `anx_get_head`
 - `anx_general.h`, 33
- `anx_get_mime_type`
 - `anx_general.h`, 34
- `anx_get_next_page_size`
 - `anx_write.h`, 52
- `anx_get_nr_headers`
 - `anx_general.h`, 34
- `anx_get_substream_list`
 - `anx_general.h`, 34
- `anx_get_timebase`
 - `anx_general.h`, 34
- `anx_granules_to_time`
 - `anx_general.h`, 34
- `anx_head_clone`
 - `anx_general.h`, 35
- `anx_head_free`
 - `anx_general.h`, 35
- `anx_init_importers`

- anx_write.h, 52
- anx_insert_anchor
 - anx_write.h, 53
- anx_int64.h, 40
 - anx_int64_t, 40
- anx_int64_t
 - anx_int64.h, 40
- anx_last_error
 - anx_general.h, 35
- anx_list_add_after
 - anx_core.h, 26
- anx_list_add_before
 - anx_core.h, 26
- anx_list_append
 - anx_core.h, 27
- anx_list_clone
 - anx_core.h, 27
- anx_list_clone_with
 - anx_core.h, 27
- anx_list_find
 - anx_core.h, 27
- anx_list_free
 - anx_core.h, 27
- anx_list_free_with
 - anx_core.h, 28
- anx_list_is_empty
 - anx_core.h, 28
- anx_list_is_singleton
 - anx_core.h, 28
- anx_list_length
 - anx_core.h, 28
- anx_list_new
 - anx_core.h, 28
- anx_list_prepend
 - anx_core.h, 29
- anx_list_remove
 - anx_core.h, 29
- anx_list_tail
 - anx_core.h, 29
- anx_media.h, 41
 - anx_register_media_importer, 44
 - anx_unregister_media_importer, 44
 - AnxMediaCloseFunc, 42
 - AnxMediaOpenFDFunc, 42
 - AnxMediaOpenFunc, 43
 - AnxMediaReadFunc, 43
 - AnxMediaSizeofNextReadFunc, 43
- anx_meta_element_clone
 - anx_general.h, 35
- anx_new
 - anx_general.h, 35
- anx_open
 - anx_general.h, 36
- anx_openfd
 - anx_general.h, 36
- anx_read
 - anx_read.h, 48
- anx_read.h, 45
 - anx_read, 48
 - anx_reader_input, 48
 - anx_set_read_anchor_callback, 49
 - anx_set_read_head_callback, 49
 - anx_set_read_media_callback, 49
 - anx_set_read_stream_callback, 49
 - anx_set_read_substream_callback, 50
 - anx_set_user_data, 50
 - AnxReadAnchor, 46
 - AnxReadHead, 46
 - AnxReadMedia, 47
 - AnxReadStream, 47
 - AnxReadSubstream, 48
- anx_reader_input
 - anx_read.h, 48
- anx_ready
 - anx_general.h, 36
- anx_register_media_importer
 - anx_media.h, 44
- anx_seek_id
 - anx_general.h, 36
- anx_seek_time
 - anx_general.h, 36
- anx_set_head
 - anx_general.h, 37
- anx_set_read_anchor_callback
 - anx_read.h, 49
- anx_set_read_head_callback
 - anx_read.h, 49
- anx_set_read_media_callback
 - anx_read.h, 49
- anx_set_read_stream_callback
 - anx_read.h, 49
- anx_set_read_substream_callback
 - anx_read.h, 50
- anx_set_timebase
 - anx_general.h, 37
- anx_set_user_data
 - anx_read.h, 50
- anx_snprint_anchor
 - anx_general.h, 37
- anx_snprint_head
 - anx_general.h, 37
- anx_strerror
 - anx_general.h, 38
- anx_tell
 - anx_general.h, 38
- anx_tell_time
 - anx_general.h, 38
- anx_time_to_granules

- `anx_general.h`, 38
- `anx_types.h`, 51
- `anx_unregister_media_importer`
 - `anx_media.h`, 44
- `anx_write`
 - `anx_write.h`, 53
- `anx_write.h`, 52
 - `anx_get_next_page_size`, 52
 - `anx_init_importers`, 52
 - `anx_insert_anchor`, 53
 - `anx_write`, 53
 - `anx_writer_import`, 53
 - `anx_writer_output`, 53
 - `anx_writer_set_anno_callbacks`, 54
 - `anx_writer_set_ii_callback`, 54
- `anx_writer_import`
 - `anx_write.h`, 53
- `anx_writer_output`
 - `anx_write.h`, 53
- `anx_writer_set_anno_callbacks`
 - `anx_write.h`, 54
- `anx_writer_set_ii_callback`
 - `anx_write.h`, 54
- `AnxMediaCloseFunc`
 - `anx_media.h`, 42
- `AnxMediaOpenFDFunc`
 - `anx_media.h`, 42
- `AnxMediaOpenFunc`
 - `anx_media.h`, 43
- `AnxMediaReadFunc`
 - `anx_media.h`, 43
- `AnxMediaSizeofNextReadFunc`
 - `anx_media.h`, 43
- `AnxReadAnchor`
 - `anx_read.h`, 46
- `AnxReadHead`
 - `anx_read.h`, 46
- `AnxReadMedia`
 - `anx_read.h`, 47
- `AnxReadStream`
 - `anx_read.h`, 47
- `AnxReadSubstream`
 - `anx_read.h`, 48
- `my_data`, 22
- Reading from Annodex media, 14
- Reading from files and file descriptors, 15
- Reading from memory buffers, 16
- Writing Annodex media, 11
- Writing to files and file descriptors, 12