libhsync Reference Manual

Generated by Doxygen 1.2.5

Wed Feb 28 16:01:46 2001

Contents

1 The libhsync delta-encoding library 1

2 Introduction 1

3 Programming interface 2

4 libhsync Data Structure Index 4

5 libhsync File Index 4

6 libhsync Page Index 4

7 libhsync Class Documentation 4

8 libhsync File Documentation 7

9 libhsync Page Documentation 17

10 Glossary 18

11 rdiff network delta tool 18

12 Todo List 18

1 The libhsync delta-encoding library

Author(s):
Martin Pool <mbp@samba.org>

Id:
main.dox,v 1.7 2001/02/26 13:27:12 mbp Exp

This document is also available in printable form:

- http://rproxy.samba.org/doxygen/libhsync/refman.ps.gz
- http://rproxy.samba.org/doxygen/libhsync/refman.pdf

2 Introduction

libhsync is a library for calculating and applying network deltas, with an interface designed to ease integration into diverse network applications. *libhsync* is being developed as part of the **rproxy** <http://rproxy.samba.org/> and **rsync** <http://rsync.samba.org/> projects.

libhsync encapsulates the core algorithms of the rsync protocol, which help with efficient calculation of the differences between two files. The rsync algorithm is different from most differencing algorithms because it does not require the presence of the two files to calculate the delta. Instead, it requires a set of checksums of each block of one file, which together form a signature for that file. Blocks at any in the other file which have the same checksum are likely to be identical, and whatever remains is the difference.

The library does not deal with file metadata or structure, such as filenames, permissions, or directories. To this library, a file is just a stream of bytes. Higher-level tools, such as rsync <http://rsync.samba.org> can deal with such issues in a way appropriate to their users.

The library supports three basic operations:

1. Generating the signature S of a file A .
2. Calculating a delta D from S and a new file B.
3. Applying D to A to reconstruct B.

The library also provides the rdiff network delta tool command-line tool, which makes this functionality available to users and scripting languages.

3 Programming interface

The public interface to libhsync (hsync.h) has functions in several main areas:

- Data streaming
- Generating and applying deltas
- Processing whole files
- Debugging trace and error logging
- Encoding statistics
- Utility functions

All external symbols have the prefix "hs_", or "HS_" in the case of preprocessor symbols.

3.1 Data streaming

A key design requirement for libhsync is that it should handle data as and when the hosting application requires it. libhsync can be used inside applications that do non-blocking IO or filtering of network streams, because it never does IO directly, or needs to block waiting for data.

The programming interface to libhsync is similar to that of zlib and bzlib. Arbitrary-length input and output buffers are passed to the library by the application, through an instance of hs_stream_t. The library proceeds as far as it can, and returns an hs_result value indicating whether it needs more data or space.

All the state needed by the library to resume processing when more data is available is kept in a small opaque hs_job_t structure. After creation of a job, repeated calls to hs_job_iter() in between filling and emptying the buffers keeps data flowing through the stream. The hs_result_t values returned may indicate

- HS_DONE: processing is complete

- HS_BLOCKED: processing has blocked pending more data
- one of various possible errors in processing

These can be converted to a human-readable string by hs_strerror().

3.2 Generating and applying deltas

All encoding operations are performed by using a `*_begin` function to create a hs_job_t object, passing in any necessary initialization parameters. The various jobs available are:

- hs_sig_begin(): Calculate the signature of a file.
- hs_loadsig_begin(): Load a signature into memory.
- hs_delta_begin(): Calculate the delta between a signature and a new file.
- hs_patch_begin(): Apply a delta to a basis to recreate the new file.

3.3 Processing whole files

Some applications do not require fine-grained control over IO, but rather just want to process a whole file with a single call. libhsync provides 'whole-file' functionality to do exactly that.

Processing of a whole file begins with creation of a hs_job_t object for the appropriate operation, just as if the application was going to do buffering itself. After creation, the job may be passed to hs_whole_run(), which will feed it to and from two FILEs as necessary until end of file is reached or the operation completes.

3.4 Debugging trace and error logging

libhsync can output trace or log messages as it proceeds. These follow a fairly standard priority-based filtering system (hs_trace_set_level()), using the same severity levels as UNIX syslog. Messages by default are sent to stderr, but may be passed to an application-provided callback (hs_trace_to(), hs_trace_fn_t).

3.5 Encoding statistics

Encoding and decoding routines accumulate compression performance statistics in a hs_stats_t structure as they run. These may be converted to human-readable form or written to the log file using hs_format_stats() or hs_log_stats() respectively.

3.6 Utility functions

Some additional functions are used internally and also exposed in the API:

- encoding/decoding binary data: hs_base64(), hs_unbase64(), hs_hexify().
- MD4 message digests: hs_mdfour(), hs_mdfour_begin(), hs_mdfour_update(), hs_mdfour_result().

4 libhsync Data Structure Index

4.1 libhsync Data Structures

Here are the data structures with brief descriptions:

hs_stats (Performance statistics from a libhsync encoding or decoding operation) 4

hs_stream_s (Stream through which the calling application feeds data to and from the library) 6

5 libhsync File Index

5.1 libhsync File List

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

hsync.h (Main public interface to libhsync) 8

hsyncfile.h (High-level file-based interfaces) 15

6 libhsync Page Index

6.1 libhsync Related Pages

Here is a list of all related documentation pages:

Glossary 18

rdiff network delta tool 18

Todo List 18

7 libhsync Class Documentation

7.1 hs_stats Struct Reference

Performance statistics from a libhsync encoding or decoding operation.

```
#include <hsync.h>
```

Data Fields

- char const* op

  Human-readable name of current operation.

- char const* algorithm

  Algorithm used to perform the operation.

- int lit_cmds

  Number of literal commands.

- int lit_bytes

  Number of literal bytes.

- int **copy_cmds**
- int **copy_bytes**
- int **sig_cmds**
- int **sig_bytes**
- int **false_matches**

7.1.1 Detailed Description

Performance statistics from a libhsync encoding or decoding operation.

See also:
hs_format_stats(), hs_log_stats()

7.1.2 Field Documentation

7.1.2.1 char const * hs_stats::op

Human-readable name of current operation.

For example, "delta".

7.1.2.2 char const * hs_stats::algorithm

Algorithm used to perform the operation.

7.1.2.3 int hs_stats::lit_cmds

Number of literal commands.

7.1.2.4 int hs_stats::lit_bytes

Number of literal bytes.

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

- hsync.h

7.2 hs_stream_s Struct Reference

Stream through which the calling application feeds data to and from the library.

```
#include <hsync.h>
```

Data Fields

- int dogtag
  To identify mutilated corpse.
- char* next_in
  Next input byte.
- unsigned int avail_in
  Number of bytes available at next_in.
- int eof_in
  True if there is no more data after this.
- char* next_out
  Next output byte should be put there.
- unsigned int avail_out
  Remaining free space at next_out.
- struct hs_simpl* impl

7.2.1 Detailed Description

Stream through which the calling application feeds data to and from the library.

On each call to hs_job_iter, the caller can make available

- avail_in bytes of input data at next_in
- avail_out bytes of output space at next_out
- some of both

There is some internal state in impl. Streams are initialized by hs_stream_init, and then used to create a job by hs_sig_begin or similar functions.

See also:
hs_stream_t

7.2.2 Field Documentation

7.2.2.1 int hs_stream_s::dogtag

To identify mutilated corpse.

7.2.2.2 char * hs_stream_s::next_in

Next input byte.

7.2.2.3 unsigned int hs_stream_s::avail_in

Number of bytes available at next_in.

7.2.2.4 int hs_stream_s::eof_in

True if there is no more data after this.

7.2.2.5 char * hs_stream_s::next_out

Next output byte should be put there.

7.2.2.6 unsigned int hs_stream_s::avail_out

Remaining free space at next_out.

7.2.2.7 struct hs_simpl * hs_stream_s::impl

For internal use only.

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

- hsync.h

8 libhsync File Documentation

8.1 hsync.h File Reference

Main public interface to libhsync.

Data Structures

- struct hs_stats

  Performance statistics from a libhsync encoding or decoding operation.

- struct **hs_mdfour**
- struct hs_stream_s

  Stream through which the calling application feeds data to and from the library.

Defines

- #define **HS_MD4_LENGTH** 16
- #define HS_DEFAULT_STRONG_LEN 8

  Default length of strong signatures, in bytes.

- #define HS_DEFAULT_BLOCK_LEN 2048

  Default block length, if not determined by any other factors.

Typedefs

- typedef void hs_trace_fn_t (int level, char const *msg)

  Callback to write out log messages.

- typedef struct hs_stats hs_stats_t

  Performance statistics from a libhsync encoding or decoding operation.

- typedef struct hs_mdfour hs_mdfour_t

  MD4 message-digest accumulator.

- typedef struct hs_signature **hs_signature_t**
- typedef struct hs_stream_s hs_stream_t

  Stream through which the calling application feeds data to and from the library.

- typedef struct hs_job hs_job_t

  Job of work to be done.

- typedef hs_result hs_copy_cb (void *opaque, size_t *len, void **result)

  Callback used to retrieve parts of the basis file.

Enumerations

- enum hs_loglevel { HS_LOG_EMERG = 0, HS_LOG_ALERT = 1, HS_LOG_CRIT = 2, HS_LOG_ERR = 3, HS_LOG_WARNING = 4, HS_LOG_NOTICE = 5, HS_LOG_INFO = 6, HS_LOG_DEBUG = 7 }

  Log severity levels.

- enum hs_result { HS_DONE = 0, HS_BLOCKED = 1, HS_RUNNING = 2, HS_TEST_SKIPPED = 77, HS_IO_ERROR = 100, HS_SYNTAX_ERROR = 101, HS_MEM_ERROR = 102, HS_INPUT_ENDED = 103, HS_BAD_MAGIC = 104, HS_UNIMPLEMENTED = 105, HS_CORRUPT = 106 }

  Return codes from nonblocking hsync operations.

Functions

- void hs_trace_set_level (hs_loglevel level)

  Set the least important message severity that will be output.

- void hs_trace_to (hs_trace_fn_t *)

  Set trace callback.

- void hs_trace_stderr (int level, char const *msg)

  Default trace callback that writes to stderr.

- int hs_supports_trace (void)

  Check whether the library was compiled with debugging trace suport.

- void hs_hexify (char *to_buf, void const *from_buf, int from_len)
- size_t hs_unbase64 (char *s)

  Decode a base64 buffer in place.

- void hs_base64 (unsigned char const *buf, int n, char *out)

  Encode a buffer as base64.

- char const* hs_strerror (hs_result r)

  Return an English description of a hs_result value.

- void **hs_mdfour** (unsigned char *out, void const *in, int n)
- void **hs_mdfour_begin** (hs_mdfour_t *md)
- void **hs_mdfour_update** (hs_mdfour_t *md, void const *, size_t n)
- void **hs_mdfour_result** (hs_mdfour_t *md, unsigned char *out)
- char* hs_format_stats (hs_stats_t const *, char *, size_t)

  Return a human-readable representation of statistics.

- int **hs_log_stats** (hs_stats_t const *stats)
- void hs_free_sumset (hs_signature_t *)

  Deep deallocation of checksums.

- void hs_sumset_dump (hs_signature_t const *)

Dump signatures to the log.

- void **hs_stream_init** (hs_stream_t *)
- hs_job_t* **hs_accum_begin** (hs_stream_t *)
- hs_result hs_job_iter (hs_job_t *)

  Run a hs_job_t state machine until it blocks (HS_BLOCKED), returns an error, or completes (HS_COMPLETE).

- hs_result **hs_job_free** (hs_job_t *)
- int **hs_accum_value** (hs_job_t *, char *sum, size_t sum_len)
- hs_job_t* hs_sig_begin (hs_stream_t *stream, size_t new_block_len, size_t strong_sum_len)

  Set up a new encoding job.

- hs_job_t* hs_delta_begin (hs_stream_t *stream, hs_signature_t *)

  Prepare to compute a delta on a stream.

- hs_job_t* hs_loadsig_begin (hs_stream_t *, hs_signature_t **)

  Read a signature from a file into an hs_signature_t structure in memory.

- hs_job_t* hs_patch_begin (hs_stream_t *, hs_copy_cb *, void *copy_arg)

  Apply a delta to a basis to recreate the new file.

- hs_result **hs_build_hash_table** (hs_signature_t *sums)

Variables

- char const hs_libhsync_version [ ] = (PACKAGE " " VERSION)

  Library version string.

- char const **hs_licence_string** [ ]

8.1.1 Detailed Description

Main public interface to libhsync.

Author(s):
Martin Pool <mbp@samba.org>

Id:
hsync.h,v 1.69 2001/02/27 22:48:13 mbp Exp

See Introduction for an introduction to use of this library.

8.1.2 Define Documentation

8.1.2.1 #define HS_DEFAULT_STRONG_LEN 8

Default length of strong signatures, in bytes.

The MD4 checksum is truncated to this size.

8.1.2.2 #define HS_DEFAULT_BLOCK_LEN 2048

Default block length, if not determined by any other factors.

8.1.3 Typedef Documentation

8.1.3.1 typedef void hs_trace_fn_t

Callback to write out log messages.

Parameters:

level a syslog level.

msg message to be logged.

8.1.3.2 typedef struct hs_stats hs_stats_t

Performance statistics from a libhsync encoding or decoding operation.

See also:

hs_format_stats(), hs_log_stats()

8.1.3.3 struct hs_mdfour hs_mdfour_t

MD4 message-digest accumulator.

See also:

hs_mdfour(), hs_mdfour_begin(), hs_mdfour_update(), hs_mdfour_result()

8.1.3.4 typedef struct hs_stream_s hs_stream_t

Stream through which the calling application feeds data to and from the library.

See also:

struct hs_stream_s

8.1.3.5 typedef struct hs_job hs_job_t

Job of work to be done.

Created by functions such as hs_sig_begin(), and then iterated over by hs_job_iter().

8.1.3.6 typedef hs_result hs_copy_cb

Callback used to retrieve parts of the basis file.

8.1.4 Enumeration Type Documentation

8.1.4.1 enum hs_loglevel

Log severity levels.

These are the same as syslog, at least in glibc.

See also:
hs_trace_set_level()

Enumeration values:

HS_LOG_EMERG System is unusable.

HS_LOG_ALERT Action must be taken immediately.

HS_LOG_CRIT Critical conditions.

HS_LOG_ERR Error conditions.

HS_LOG_WARNING Warning conditions.

HS_LOG_NOTICE Normal but significant condition.

HS_LOG_INFO Informational.

HS_LOG_DEBUG Debug-level messages.

8.1.4.2 enum hs_result

Return codes from nonblocking hsync operations.

Enumeration values:

HS_DONE Completed successfully.

HS_BLOCKED Blocked waiting for more data.

HS_RUNNING Not yet finished or blocked.
This value should never be returned to the caller.

HS_TEST_SKIPPED Test neither passed or failed.

HS_IO_ERROR Error in file or network IO.

HS_SYNTAX_ERROR Command line syntax error.

HS_MEM_ERROR Out of memory.

HS_INPUT_ENDED End of input file, possibly unexpected.

HS_BAD_MAGIC Bad magic number at start of stream.
Probably not a libhsync file, or possibly the wrong kind of file or from an incompatible library version.

HS_UNIMPLEMENTED Author is lazy.

HS_CORRUPT Unbelievable value in stream.

8.1.5 Function Documentation

8.1.5.1 void hs_trace_set_level (hs_loglevel *level*)

Set the least important message severity that will be output.

8.1.5.2 void hs_trace_to (hs_trace_fn_t * *new_impl*)

Set trace callback.

The callback scheme allows for use within applications that may have their own particular ways of reporting errors: log files for a web server, perhaps, and an error dialog for a browser.

Todo:
Do we really need such fine-grained control, or just yes/no tracing?

8.1.5.3 void hs_trace_stderr (int *level*, char const * *msg*)

Default trace callback that writes to stderr.

Implements hs_trace_fn_t, and may be passed to hs_trace_to().

8.1.5.4 int hs_supports_trace (void)

Check whether the library was compiled with debugging trace suport.

If this returns false, then trying to turn trace on will achieve nothing.

8.1.5.5 void hs_hexify (char * *to_buf*, void const * *from_buf*, int *from_len*)

Convert FROM_LEN bytes at FROM_BUF into a hex representation in TO_BUF, which must be twice as long plus one byte for the null terminator.

8.1.5.6 size_t hs_unbase64 (char * *s*)

Decode a base64 buffer in place.

Returns:
the number of binary bytes.

8.1.5.7 void hs_base64 (unsigned char const * *buf*, int *n*, char * *out*)

Encode a buffer as base64.

8.1.5.8 char const * hs_strerror (hs_result *r*)

Return an English description of a hs_result value.

8.1.5.9 char * hs_format_stats (hs_stats_t const * *stats*, char * *buf*, size_t *size*)

Return a human-readable representation of statistics.

The string is truncated if it does not fit. 100 characters should be sufficient space.

Parameters:

stats Statistics from an encoding or decoding operation.

buf Buffer to receive result.

size Size of buffer.

Returns:

buf

8.1.5.10 void hs_free_sumset (hs_signature_t * *psums*)

Deep deallocation of checksums.

8.1.5.11 void hs_sumset_dump (hs_signature_t const * *sums*)

Dump signatures to the log.

8.1.5.12 hs_result hs_job_iter (hs_job_t * *job*)

Run a hs_job_t state machine until it blocks (HS_BLOCKED), returns an error, or completes (HS_-COMPLETE).

Returns:

The hs_result that caused iteration to stop.

Parameters:

ending True if there is no more data after what's in the input buffer. The final block checksum will run across whatever's in there, without trying to accumulate anything else.

8.1.5.13 hs_job_t * hs_sig_begin (hs_stream_t * *stream*, size_t *new_block_len*, size_t *strong_sum_len*)

Set up a new encoding job.

See also:

hs_sig_file()

8.1.5.14 hs_job_t* hs_delta_begin (hs_stream_t * *stream*, hs_signature_t * *sig*)

Prepare to compute a delta on a stream.

8.1.5.15 hs_job_t * hs_loadsig_begin (hs_stream_t * *stream*, hs_signature_t ** *signature*)

Read a signature from a file into an hs_signature_t structure in memory.

Once there, it can be used to generate a delta to a newer version of the file.

Note:
After loading the signatures, you must call hs_build_hash_table() before you can use them.

8.1.5.16 hs_job_t * hs_patch_begin (hs_stream_t * *stream*, hs_copy_cb * *copy_cb*, void * *copy_arg*)

Apply a delta to a basis to recreate the new file.

This gives you back a hs_job_t object, which can be cranked by calling hs_job_iter() and updating the stream pointers. When finished, call hs_job_finish() to dispose of it.

Parameters:

stream Contains pointers to input and output buffers, to be adjusted by caller on each iteration.

copy_cb Callback used to retrieve content from the basis file.

copy_arg Opaque environment pointer passed through to the callback.

Todo:
As output is produced, accumulate the MD4 checksum of the output. Then if we find a CHECKSUM command we can check it's contents against the output.
Implement COPY commands.

See also:
hs_patch_file()

8.1.6 Variable Documentation

8.1.6.1 char const hs_libhsync_version[ ] = (PACKAGE " " VERSION)

Library version string.

8.2 hsyncfile.h File Reference

High-level file-based interfaces.

Functions

- void hs_mdfour_file (FILE *in_file, char *result)

  Calculate the MD4 sum of a file.

- hs_result hs_sig_file (FILE *old_file, FILE *sig_file, size_t, size_t)

  Generate the signature of a basis file, and write it out to another.

- hs_result hs_loadsig_file (FILE *sig_file, hs_signature_t **sumset)

  Load signatures from a signature file into memory.

- hs_result hs_file_copy_cb (void *arg, size_t *len, void **buf)

  Default copy implementation that retrieves a part of a stdio file.

- hs_result **hs_delta_file** (hs_signature_t *, FILE *new_file, FILE *delta_file)
- hs_result **hs_patch_file** (FILE *basis_file, FILE *delta_file, FILE *new_file)

Variables

- int hs_inbuflen
- int hs_outbuflen

8.2.1 Detailed Description

High-level file-based interfaces.

Author(s):
Martin Pool <mbp@samba.org>

Id:
hsyncfile.h,v 1.10 2001/02/27 22:48:27 mbp Exp

8.2.2 Function Documentation

8.2.2.1 void hs_mdfour_file (FILE * *in_file*, char * *result*)

Calculate the MD4 sum of a file.

Parameters:
result Binary (not hex) MD4 of the whole contents of the file.

8.2.2.2 hs_result hs_sig_file (FILE * *old_file*, FILE * *sig_file*, size_t *new_block_len*, size_t *strong_len*)

Generate the signature of a basis file, and write it out to another.

Parameters:

new_block_len block size for signature generation, in bytes

strong_len truncated length of strong checksums, in bytes

See also:

hs_sig_begin()

8.2.2.3 hs_result hs_loadsig_file (FILE * *sig_file*, hs_signature_t ** *sumset*)

Load signatures from a signature file into memory.

Return a pointer to the newly allocated structure in SUMSET.

See also:

hs_readsig_begin()

8.2.2.4 hs_result hs_file_copy_cb (void * *arg*, size_t * *len*, void ** *buf*)

Default copy implementation that retrieves a part of a stdio file.

8.2.3 Variable Documentation

8.2.3.1 int hs_inbuflen

Buffer sizes for file IO.

You probably only need to change these in testing.

8.2.3.2 int hs_outbuflen

Buffer sizes for file IO.

You probably only need to change these in testing.

9 libhsync Page Documentation

10 Glossary

Author(s):
Martin Pool <mbp@samba.org>

Id:
gloss.dox,v 1.1 2001/02/26 10:19:12 mbp Exp

10.1 delta

A description of changes necessary to bring a basis file to the new state.

10.2 basis

The old version of a file, to which a delta is applied. The delta may use blocks from the basis file to reconstruct the new file.

11 rdiff network delta tool

Author(s):
Martin Pool

Version:

Id:
rdiff.dox,v 1.1 2001/02/25 03:32:06 mbp Exp

Foo.

12 Todo List

global hs_mdfour64(hs_mdfour_t *m, uint32_t *M) Recode to be fast, and to use system integer types. Perhaps if we can find an mdfour implementation already on the system (e.g. in OpenSSL) then we should use it instead of our own?

Apparently rsync 2.4 now has a fast MD4 routine. So we should copy that into here.

global hs_patch_begin(hs_stream_t *, hs_copy_cb *, void *copy_arg) As output is produced, accumulate the MD4 checksum of the output. Then if we find a CHECKSUM command we can check it's contents against the output.

Implement COPY commands.

global hs_trace_to(hs_trace_fn_t *) Do we really need such fine-grained control, or just yes/no tracing?

Index

algorithm
    hs_stats, 5
avail_in
    hs_stream_s, 7
avail_out
    hs_stream_s, 7

basis, 18

copy_bytes
    hs_stats, 5
copy_cmds
    hs_stats, 5

delta, 18
dogtag
    hs_stream_s, 7

eof_in
    hs_stream_s, 7

false_matches
    hs_stats, 5

hs_accum_begin
    hsync.h, 10
hs_accum_value
    hsync.h, 10
HS_BAD_MAGIC
    hsync.h, 12
hs_base64
    hsync.h, 13
HS_BLOCKED
    hsync.h, 12
hs_build_hash_table
    hsync.h, 10
hs_copy_cb
    hsync.h, 11
HS_CORRUPT
    hsync.h, 12
HS_DEFAULT_BLOCK_LEN
    hsync.h, 10
HS_DEFAULT_STRONG_LEN
    hsync.h, 10
hs_delta_begin
    hsync.h, 14
hs_delta_file
    hsyncfile.h, 16
HS_DONE
    hsync.h, 12
hs_file_copy_cb
    hsyncfile.h, 17
hs_format_stats
    hsync.h, 14
hs_free_sumset
    hsync.h, 14
hs_hexify
    hsync.h, 13
hs_inbuflen
    hsyncfile.h, 17
HS_INPUT_ENDED
    hsync.h, 12
HS_IO_ERROR
    hsync.h, 12
hs_job_free
    hsync.h, 10
hs_job_iter
    hsync.h, 14
hs_job_t
    hsync.h, 11
hs_libhsync_version
    hsync.h, 15
hs_licence_string
    hsync.h, 10
hs_loadsig_begin
    hsync.h, 15
hs_loadsig_file
    hsyncfile.h, 17
HS_LOG_ALERT
    hsync.h, 12
HS_LOG_CRIT
    hsync.h, 12
HS_LOG_DEBUG
    hsync.h, 12
HS_LOG_EMERG
    hsync.h, 12
HS_LOG_ERR
    hsync.h, 12
HS_LOG_INFO
    hsync.h, 12
HS_LOG_NOTICE
    hsync.h, 12
hs_log_stats
    hsync.h, 9
HS_LOG_WARNING
    hsync.h, 12
hs_loglevel
    hsync.h, 12
HS_MD4_LENGTH
    hsync.h, 8
hs_mdfour
    hsync.h, 9

hs_mdfour_begin
hsync.h, 9
hs_mdfour_file
hsyncfile.h, 16
hs_mdfour_result
hsync.h, 9
hs_mdfour_t
hsync.h, 11
hs_mdfour_update
hsync.h, 9
HS_MEM_ERROR
hsync.h, 12
hs_outbuflen
hsyncfile.h, 17
hs_patch_begin
hsync.h, 15
hs_patch_file
hsyncfile.h, 16
hs_result
hsync.h, 12
HS_RUNNING
hsync.h, 12
hs_sig_begin
hsync.h, 14
hs_sig_file
hsyncfile.h, 16
hs_signature_t
hsync.h, 8
hs_stats, 4
algorithm, 5
copy_bytes, 5
copy_cmds, 5
false_matches, 5
lit_bytes, 5
lit_cmds, 5
op, 5
sig_bytes, 5
sig_cmds, 5
hs_stats_t
hsync.h, 11
hs_stream_init
hsync.h, 10
hs_stream_s, 6
avail_in, 7
avail_out, 7
dogtag, 7
eof_in, 7
impl, 7
next_in, 7
next_out, 7
hs_stream_t
hsync.h, 11
hs_strerror
hsync.h, 13
hs_sumset_dump
hsync.h, 14
hs_supports_trace
hsync.h, 13
HS_SYNTAX_ERROR
hsync.h, 12
HS_TEST_SKIPPED
hsync.h, 12
hs_trace_fn_t
hsync.h, 11
hs_trace_set_level
hsync.h, 13
hs_trace_stderr
hsync.h, 13
hs_trace_to
hsync.h, 13
hs_unbase64
hsync.h, 13
HS_UNIMPLEMENTED
hsync.h, 12
hsync.h, 8
hs_accum_begin, 10
hs_accum_value, 10
HS_BAD_MAGIC, 12
hs_base64, 13
HS_BLOCKED, 12
hs_build_hash_table, 10
hs_copy_cb, 11
HS_CORRUPT, 12
HS_DEFAULT_BLOCK_LEN, 10
HS_DEFAULT_STRONG_LEN, 10
hs_delta_begin, 14
HS_DONE, 12
hs_format_stats, 14
hs_free_sumset, 14
hs_hexify, 13
HS_INPUT_ENDED, 12
HS_IO_ERROR, 12
hs_job_free, 10
hs_job_iter, 14
hs_job_t, 11
hs_libhsync_version, 15
hs_licence_string, 10
hs_loadsig_begin, 15
HS_LOG_ALERT, 12
HS_LOG_CRIT, 12
HS_LOG_DEBUG, 12
HS_LOG_EMERG, 12
HS_LOG_ERR, 12
HS_LOG_INFO, 12
HS_LOG_NOTICE, 12
hs_log_stats, 9
HS_LOG_WARNING, 12
hs_loglevel, 12

HS_MD4_LENGTH, 8
hs_mdfour, 9
hs_mdfour_begin, 9
hs_mdfour_result, 9
hs_mdfour_t, 11
hs_mdfour_update, 9
HS_MEM_ERROR, 12
hs_patch_begin, 15
hs_result, 12
HS_RUNNING, 12
hs_sig_begin, 14
hs_signature_t, 8
hs_stats_t, 11
hs_stream_init, 10
hs_stream_t, 11
hs_strerror, 13
hs_sumset_dump, 14
hs_supports_trace, 13
HS_SYNTAX_ERROR, 12
HS_TEST_SKIPPED, 12
hs_trace_fn_t, 11
hs_trace_set_level, 13
hs_trace_stderr, 13
hs_trace_to, 13
hs_unbase64, 13
HS_UNIMPLEMENTED, 12
hsyncfile.h, 15
hs_delta_file, 16
hs_file_copy_cb, 17
hs_inbuflen, 17
hs_loadsig_file, 17
hs_mdfour_file, 16
hs_outbuflen, 17
hs_patch_file, 16
hs_sig_file, 16

impl
hs_stream_s, 7

lit_bytes
hs_stats, 5
lit_cmds
hs_stats, 5

next_in
hs_stream_s, 7
next_out
hs_stream_s, 7

op
hs_stats, 5

sig_bytes
hs_stats, 5
sig_cmds
hs_stats, 5