librsync Reference Manual

Generated by Doxygen 1.2.14

Fri Mar 22 11:07:26 2002

Contents

1 librsync Data Structure Index

1.1 librsync Data Structures

Here are the data structures with brief descriptions:

2 librsync File Index

2.1 librsync File List

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

3 librsync Page Index

3.1 librsync Related Pages

Here is a list of all related documentation pages:

4 librsync Data Structure Documentation

4.1 rs_buffers_s Struct Reference

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

Data Fields

- char * next_in

    Next input byte.

- size_t 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.

- size_t avail_out

    Remaining free space at next_out.

4.1.1 Detailed Description

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

On each call to rs_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

Buffers must be allocated and passed in by the caller. This routine never allocates, reallocates or frees buffers.

Pay attention to the meaning of the returned pointer and length values. They do **not** indicate the location and amount of returned data. Rather, if `*out_ptr` was originally set to `out_buf`, then the output data begins at `out_buf`, and has length `*out_ptr - out_buf`.

Note also that if `*avail_in` is nonzero on return, then not all of the input data has been consumed. The caller should either provide more output buffer space and call rs_work() again passing the same `next_in` and `avail_in`, or put the remaining input data into some persistent buffer and call rs_work() with it again when there is more output space.

Parameters:

next_in References a pointer which on entry should point to the start of the data to be encoded. Updated to point to the byte after the last one consumed.

avail_in References the length of available input. Updated to be the number of unused data bytes, which will be zero if all the input was consumed. May be zero if there is no new input, but the caller just wants to drain output.

next_out References a pointer which on entry points to the start of the output buffer. Updated to point to the byte after the last one filled.

avail_out References the size of available output buffer. Updated to the size of unused output buffer.

Returns:
The rs_result that caused iteration to stop.

See also:
rs_buffers_t , api_buffers

4.1.2 Field Documentation

4.1.2.1 char∗ rs_buffers_s::next_in

Next input byte.

4.1.2.2 size_t rs_buffers_s::avail_in

Number of bytes available at next_in.

4.1.2.3 int rs_buffers_s::eof_in

True if there is no more data after this.

4.1.2.4 char∗ rs_buffers_s::next_out

Next output byte should be put there.

4.1.2.5 size_t rs_buffers_s::avail_out

Remaining free space at next_out.

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

- rsync.h

4.2 rs_stats Struct Reference

Performance statistics from a librsync encoding or decoding operation.

Data Fields

- char const * op

  Human-readable name of current operation.

- int lit_cmds

  Number of literal commands.

- rs_long_t lit_bytes

  Number of literal bytes.

- rs_long_t lit_cmdbytes

  Number of bytes used in literal command headers.

- rs_long_t **copy_cmds**
- rs_long_t **copy_bytes**
- rs_long_t **copy_cmdbytes**
- rs_long_t **sig_cmds**
- rs_long_t **sig_bytes**
- int **false_matches**
- rs_long_t sig_blocks

  Number of blocks described by the signature.

- size_t **block_len**
- rs_long_t in_bytes

  Total bytes read from input.

- rs_long_t out_bytes

  Total bytes written to output.

4.2.1 Detailed Description

Performance statistics from a librsync encoding or decoding operation.

See also:
rs_format_stats(), rs_log_stats()

4.2.2 Field Documentation

4.2.2.1 char const* rs_stats::op

Human-readable name of current operation.

For example, "delta".

4.2.2.2 int rs_stats::lit_cmds

Number of literal commands.

4.2.2.3 rs_long_t rs_stats::lit_bytes

Number of literal bytes.

4.2.2.4 rs_long_t rs_stats::lit_cmdbytes

Number of bytes used in literal command headers.

4.2.2.5 rs_long_t rs_stats::sig_blocks

Number of blocks described by the signature.

4.2.2.6 rs_long_t rs_stats::in_bytes

Total bytes read from input.

4.2.2.7 rs_long_t rs_stats::out_bytes

Total bytes written to output.

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

- rsync.h

4.3 rs_target Struct Reference

Description of the match described by a signature.

Data Fields

- unsigned short **t**
- int **i**

4.3.1 Detailed Description

Description of the match described by a signature.

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

- sumset.h

5 librsync File Documentation

5.1 rsync.h File Reference

Main public interface to librsync.

Data Structures

- struct rs_stats

  Performance statistics from a librsync encoding or decoding operation.

- struct **rs_mdfour**
- struct rs_buffers_s

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

Defines

- #define **RS_MD4_LENGTH** 16
- #define RS_DEFAULT_STRONG_LEN 8

  Default length of strong signatures, in bytes.

- #define RS_DEFAULT_BLOCK_LEN 2048

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

Typedefs

- typedef long rs_long_t

  A long integer type that can handle the largest file offsets.

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

  Callback to write out log messages.

- typedef rs_stats rs_stats_t

  Performance statistics from a librsync encoding or decoding operation.

- typedef rs_mdfour rs_mdfour_t

  MD4 message-digest accumulator.

- typedef unsigned int **rs_weak_sum_t**
- typedef unsigned char **rs_strong_sum_t** [RS_MD4_LENGTH]
- typedef rs_signature **rs_signature_t**
- typedef rs_buffers_s rs_buffers_t

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

- typedef rs_job rs_job_t

    Job of work to be done.

- typedef enum rs_work_options rs_work_options

    Bitmask values that may be passed to the options parameter of rs_work().

- typedef rs_result **rs_driven_cb** (rs_job_t ∗job, rs_buffers_t ∗buf, void ∗opaque)
- typedef rs_result rs_copy_cb (void ∗opaque, off_t pos, size_t ∗len, void ∗∗buf)

    Callback used to retrieve parts of the basis file.

Enumerations

- enum rs_loglevel { RS_LOG_EMERG = 0, RS_LOG_ALERT = 1, RS_LOG_CRIT = 2, RS_LOG_ERR = 3, RS_LOG_WARNING = 4, RS_LOG_NOTICE = 5, RS_LOG_INFO = 6, RS_LOG_DEBUG = 7 }

    Log severity levels.

- enum rs_result { RS_DONE = 0, RS_BLOCKED = 1, RS_RUNNING = 2, RS_TEST_SKIPPED = 77, RS_IO_ERROR = 100, RS_SYNTAX_ERROR = 101, RS_MEM_ERROR = 102, RS_INPUT_ENDED = 103, RS_BAD_MAGIC = 104, RS_UNIMPLEMENTED = 105, RS_CORRUPT = 106, RS_INTERNAL_ERROR = 107, RS_PARAM_ERROR = 108 }

    Return codes from nonblocking rsync operations.

- enum rs_work_options { RS_END = 0x01 }

    Bitmask values that may be passed to the options parameter of rs_work().

Functions

- void rs_trace_set_level (rs_loglevel level)

    Set the least important message severity that will be output.

- void rs_trace_to (rs_trace_fn_t ∗)

    Set trace callback.

- void rs_trace_stderr (int level, char const ∗msg)

    Default trace callback that writes to stderr.

- int rs_supports_trace (void)

    Check whether the library was compiled with debugging trace suport.

- void rs_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.

- size_t rs_unbase64 (char ∗s)

    Decode a base64 buffer in place.

- void rs_base64 (unsigned char const ∗buf, int n, char ∗out)

Encode a buffer as base64.

- char const ∗ rs_strerror (rs_result r)

  Return an English description of a rs_result value.

- void **rs_mdfour** (unsigned char ∗out, void const ∗in, size_t)
- void **rs_mdfour_begin** (rs_mdfour_t ∗md)
- void rs_mdfour_update (rs_mdfour_t ∗md, void const ∗, size_t n)

  Feed some data into the MD4 accumulator.

- void **rs_mdfour_result** (rs_mdfour_t ∗md, unsigned char ∗out)
- char ∗ rs_format_stats (rs_stats_t const ∗, char ∗, size_t)

  Return a human-readable representation of statistics.

- int **rs_log_stats** (rs_stats_t const ∗stats)
- void rs_free_sumset (rs_signature_t ∗)

  Deep deallocation of checksums.

- void rs_sumset_dump (rs_signature_t const ∗)

  Dump signatures to the log.

- rs_result rs_job_iter (rs_job_t ∗, rs_buffers_t ∗)

  Run a rs_job_t state machine until it blocks (RS_BLOCKED), returns an error, or completes (RS_-COMPLETE).

- rs_result rs_job_drive (rs_job_t ∗job, rs_buffers_t ∗buf, rs_driven_cb in_cb, void ∗in_opaque, rs_-driven_cb out_cb, void ∗out_opaque)

  Actively process a job, by making callbacks to fill and empty the buffers until the job is done.

- const rs_stats_t ∗ rs_job_statistics (rs_job_t ∗job)

  Return pointer to statistics accumulated about this job.

- rs_result **rs_job_free** (rs_job_t ∗)
- int **rs_accum_value** (rs_job_t ∗, char ∗sum, size_t sum_len)
- rs_job_t ∗ rs_sig_begin (size_t new_block_len, size_t strong_sum_len)

  Set up a new encoding job.

- rs_job_t ∗ rs_delta_begin (rs_signature_t ∗)

  Prepare to compute a streaming delta.

- rs_job_t ∗ rs_loadsig_begin (rs_signature_t ∗∗)

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

- rs_job_t ∗ rs_patch_begin (rs_copy_cb ∗, void ∗copy_arg)

  Apply a gloss_delta to a gloss_basis to recreate the new file.

- rs_result **rs_build_hash_table** (rs_signature_t ∗sums)
- void rs_mdfour_file (FILE ∗in_file, char ∗result)

  Calculate the MD4 sum of a file.

- rs_result rs_sig_file (FILE *old_file, FILE *sig_file, size_t block_len, size_t strong_len, rs_stats_t *)

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

- rs_result rs_loadsig_file (FILE *, rs_signature_t **, rs_stats_t *)

  Load signatures from a signature file into memory.

- rs_result rs_file_copy_cb (void *arg, off_t pos, size_t *len, void **buf)

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

- rs_result **rs_delta_file** (rs_signature_t *, FILE *new_file, FILE *delta_file, rs_stats_t *)
- rs_result **rs_patch_file** (FILE *basis_file, FILE *delta_file, FILE *new_file, rs_stats_t *)

Variables

- char const rs_librsync_version [ ]

  Library version string.

- char const **rs_licence_string** [ ]
- int rs_inbuflen

  Buffer sizes for file IO.

- int rs_outbuflen

  Buffer sizes for file IO.

5.1.1 Detailed Description

Main public interface to librsync.

Author:

Martin Pool <`mbp@samba.org`>

Version:

librsync-0.9.1

Id:

rsync.h,v 1.13 2001/04/09 15:02:40 mbp Exp

See intro for an introduction to use of this library.

5.1.2 Define Documentation

5.1.2.1 #define RS_DEFAULT_STRONG_LEN 8

Default length of strong signatures, in bytes.

The MD4 checksum is truncated to this size.

5.1.2.2 #define RS_DEFAULT_BLOCK_LEN 2048

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

5.1.3 Typedef Documentation

5.1.3.1 typedef long rs_long_t

A long integer type that can handle the largest file offsets.

Perhaps this might have to be configured to be 'long long', 'long', or something else depending on the platform.

5.1.3.2 rs_trace_fn_t

Callback to write out log messages.

Parameters:

level a syslog level.

msg message to be logged.

5.1.3.3 typedef struct rs_stats rs_stats_t

Performance statistics from a librsync encoding or decoding operation.

See also:

rs_format_stats(), rs_log_stats()

5.1.3.4 struct rs_mdfour rs_mdfour_t

MD4 message-digest accumulator.

See also:

rs_mdfour(), rs_mdfour_begin(), rs_mdfour_update(), rs_mdfour_result()

5.1.3.5 typedef struct rs_buffers_s rs_buffers_t

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

See also:

struct rs_buffers_s , api_buffers

5.1.3.6 struct rs_job rs_job_t

Job of work to be done.

Created by functions such as rs_sig_begin(), and then iterated over by rs_job_iter().

5.1.3.7 typedef enum rs_work_options rs_work_options

Bitmask values that may be passed to the options parameter of rs_work().

5.1.3.8 typedef rs_result rs_copy_cb(void *opaque, off_t pos, size_t *len, void **buf)

Callback used to retrieve parts of the basis file.

Parameters:

pos Position where copying should begin.

len On input, the amount of data that should be retrieved. Updated to show how much is actually available.

buf On input, a buffer of at least `*len` bytes. May be updated to point to a buffer allocated by the callback if it prefers.

5.1.4 Enumeration Type Documentation

5.1.4.1 enum rs_loglevel

Log severity levels.

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

See also:

rs_trace_set_level()

Enumeration values:

RS_LOG_EMERG System is unusable.

RS_LOG_ALERT Action must be taken immediately.

RS_LOG_CRIT Critical conditions.

RS_LOG_ERR Error conditions.

RS_LOG_WARNING Warning conditions.

RS_LOG_NOTICE Normal but significant condition.

RS_LOG_INFO Informational.

RS_LOG_DEBUG Debug-level messages.

5.1.4.2 enum rs_result

Return codes from nonblocking rsync operations.

Enumeration values:

- **RS_DONE** Completed successfully.
- **RS_BLOCKED** Blocked waiting for more data.
- **RS_RUNNING** Not yet finished or blocked.
  This value should never be returned to the caller.
- **RS_TEST_SKIPPED** Test neither passed or failed.
- **RS_IO_ERROR** Error in file or network IO.
- **RS_SYNTAX_ERROR** Command line syntax error.
- **RS_MEM_ERROR** Out of memory.
- **RS_INPUT_ENDED** End of input file, possibly unexpected.
- **RS_BAD_MAGIC** Bad magic number at start of stream.
  Probably not a librsync file, or possibly the wrong kind of file or from an incompatible library version.
- **RS_UNIMPLEMENTED** Author is lazy.
- **RS_CORRUPT** Unbelievable value in stream.
- **RS_INTERNAL_ERROR** Probably a library bug.
- **RS_PARAM_ERROR** Bad value passed in to library, probably an application bug.

5.1.4.3 enum rs_work_options

Bitmask values that may be passed to the options parameter of rs_work().

Enumeration values:

- **RS_END** End of input file; please finish up.

5.1.5 Function Documentation

5.1.5.1 void rs_trace_set_level (rs_loglevel *level*)

Set the least important message severity that will be output.

5.1.5.2 void rs_trace_to (rs_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?

5.1.5.3 void rs_trace_stderr (int *level*, char const * *msg*)

Default trace callback that writes to stderr.

Implements rs_trace_fn_t, and may be passed to rs_trace_to().

5.1.5.4 int rs_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.

5.1.5.5 void rs_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.

5.1.5.6 size_t rs_unbase64 (char * *s*)

Decode a base64 buffer in place.

Returns:
the number of binary bytes.

5.1.5.7 void rs_base64 (unsigned char const * *buf*, int *n*, char * *out*)

Encode a buffer as base64.

5.1.5.8 char const* rs_strerror (rs_result *r*)

Return an English description of a rs_result value.

5.1.5.9 void rs_mdfour_update (rs_mdfour_t * *md*, void const * *in_void*, size_t *n*)

Feed some data into the MD4 accumulator.

Parameters:
n Number of bytes fed in.

5.1.5.10 char* rs_format_stats (rs_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

5.1.5.11 void rs_free_sumset (rs_signature_t * *psums*)

Deep deallocation of checksums.

5.1.5.12 void rs_sumset_dump (rs_signature_t const * *sums*)

Dump signatures to the log.

5.1.5.13 rs_result rs_job_iter (rs_job_t * *job*, rs_buffers_t * *buffers*)

Run a rs_job_t state machine until it blocks (RS_BLOCKED), returns an error, or completes (RS_COMPLETE).

Returns:
The rs_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.

5.1.5.14 rs_result rs_job_drive (rs_job_t * *job*, rs_buffers_t * *buf*, rs_driven_cb *in_cb*, void * *in_opaque*, rs_driven_cb *out_cb*, void * *out_opaque*)

Actively process a job, by making callbacks to fill and empty the buffers until the job is done.

5.1.5.15 const rs_stats_t* rs_job_statistics (rs_job_t * *job*)

Return pointer to statistics accumulated about this job.

5.1.5.16 rs_job_t* rs_sig_begin (size_t *new_block_len*, size_t *strong_sum_len*)

Set up a new encoding job.

See also:
rs_sig_file()

5.1.5.17 rs_job_t* rs_delta_begin (rs_signature_t *)

Prepare to compute a streaming delta.

5.1.5.18 rs_job_t* rs_loadsig_begin (rs_signature_t ** *signature*)

Read a signature from a file into an rs_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 rs_build_hash_table() before you can use them.

5.1.5.19 rs_job_t* rs_patch_begin (rs_copy_cb * *copy_cb*, void * *copy_arg*)

Apply a gloss_delta to a gloss_basis to recreate the new file.

This gives you back a rs_job_t object, which can be cranked by calling rs_job_iter() and updating the stream pointers. When finished, call rs_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:
rs_patch_file()

5.1.5.20 void rs_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.

5.1.5.21 rs_result rs_sig_file (FILE * *old_file*, FILE * *sig_file*, size_t *new_block_len*, size_t *strong_len*, rs_stats_t * *stats*)

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:
rs_sig_begin()

5.1.5.22 rs_result rs_loadsig_file (FILE ∗ *sig_file*, rs_signature_t ∗∗ *sumset*, rs_stats_t ∗ *stats*)

Load signatures from a signature file into memory.

Return a pointer to the newly allocated structure in SUMSET.

See also:
rs_readsig_begin()

5.1.5.23 rs_result rs_file_copy_cb (void ∗ *arg*, off_t *pos*, size_t ∗ *len*, void ∗∗ *buf*)

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

5.1.6 Variable Documentation

5.1.6.1 char const rs_librsync_version[ ]

Library version string.

5.1.6.2 int rs_inbuflen

Buffer sizes for file IO.

You probably only need to change these in testing.

5.1.6.3 int rs_outbuflen

Buffer sizes for file IO.

You probably only need to change these in testing.

6 librsync Page Documentation

6.1 Todo List

Global rs_mdfour64(rs_mdfour_t ∗m, const void ∗p) 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?

Global rs_patch_begin(rs_copy_cb ∗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 rs_trace_to(rs_trace_fn_t ∗) Do we really need such fine-grained control, or just yes/no tracing?

Index