Universal Data Serializer (Low Level)

Generated by Doxygen 1.5.8

Sat May 30 22:50:06 2009

Contents

1 Main Page

This documentation is for the low-level UDS library. You probably do not want to use this.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

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

4 Data Structure Documentation

4.1 uds_bnode_t Struct Reference

This structure represents both a build key and a build tree.

`#include <uds_lowlevel.h>`

Data Fields

- char **key** [KEYNAME_SIZE]
- int type

  This can be:
  - *NODE_ROOT for a tree root*
  - *NODE_BASIC for a basic key/string*
  - *NODE_FILE for a key/filename*
  - *NODE_MEMR for a memory region*
  - *NODE_TREE for a subtree.*

- char * x

  This represents either the string or the filename, depending on type.

- size_t **x_len**
- uint8_t * **memregion**
- size_t **memr_len**
- char **comment** [COMMENT_SIZE]
- struct uds_bnode_t * **subtree**
- struct uds_bnode_t * **next**
- struct uds_bnode_t * root

  This should be NULL if instance is a root.

- size_t list_count

  This should be zero if instance is not a root.

4.1.1 Detailed Description

This structure represents both a build key and a build tree.

Build keys and trees are not complete, but will be compiled on write.

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

- uds_lowlevel.h

4.2 uds_open_file_t Struct Reference

Used internally to represent an open file.

`#include <uds_lowlevel.h>`

Data Fields

- int **fd**
- char * **mapped**
- size_t **len**

4.2.1 Detailed Description

Used internally to represent an open file.

Should never be modified.

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

- uds_lowlevel.h

5 File Documentation

5.1 uds_lowlevel.h File Reference

Low-level tree functions and structures.

`#include <stddef.h>`

`#include <stdio.h>`

`#include <inttypes.h>`

Data Structures

- struct uds_bnode_t

  This structure represents both a build key and a build tree.

- struct uds_open_file_t

  Used internally to represent an open file.

Defines

- #define **KEYNAME_SIZE** 50
- #define **COMMENT_SIZE** 50
- #define **USER_MAGIC_SIZE** 4
- #define **NODE_ROOT** 0
- #define **NODE_BASIC** 1
- #define **NODE_FILE** 2
- #define **NODE_MEMR** 3
- #define **NODE_TREE** 4

Typedefs

- typedef struct uds_bnode_t uds_bnode

  This structure represents both a build key and a build tree.

- typedef struct uds_open_file_t uds_open_file

  Used internally to represent an open file.

Functions

Build functions

These functions are used for building and writing UDS files.

- int uds_bnode_addkey (uds_bnode *tree, char key[KEYNAME_SIZE], char *value, size_t value_len, char comment[COMMENT_SIZE])

  Adds a key to a build tree.

- int uds_bnode_addkey_keep (uds_bnode *tree, char key[KEYNAME_SIZE], char *filename, size_t filename_len, char comment[COMMENT_SIZE])

  Adds a key to a build tree, keeping a filename for later addition.

- int uds_bnode_addkey_memregion (uds_bnode *tree, char key[KEYNAME_SIZE], uint8_t *memregion, size_t len, char comment[COMMENT_SIZE])
  Adds a key to a build tree from a memory region.

- uds_bnode * uds_bnode_addkey_subtree (uds_bnode *tree, char key[KEYNAME_SIZE], char comment[COMMENT_SIZE])
  Adds a key to a build tree as a subtree.

- uds_bnode * uds_bnode_getkey (uds_bnode *tree, char key[KEYNAME_SIZE])
  Checks whether a key exists in a tree.

- int uds_write_tree (uds_bnode *tree, char *filename, size_t filename_len, char magic[USER_MAGIC_SIZE])
  Builds a tree and writes to disk.

- void uds_bnode_free_tree (uds_bnode *tree)
  Frees a build tree.

- void uds_bnode_init (uds_bnode *node)
  Initializes a bnode.

Read functions

These are used to read from a UDS file.

- uds_open_file * uds_open_from_mem (char *mem, size_t len)
  Treats a region of memory as a UDS file.

- uds_open_file * uds_open (char *filename)
  Opens a UDS file.

- void uds_close (uds_open_file *file)
  Closes a UDS file.

- int uds_check_internal_magic (uds_open_file *file)
  Checks the internal magic number, which should be UDS1.

- int uds_get_user_magic (uds_open_file *file, char magic[USER_MAGIC_SIZE])
  Gets the user magic number from a UDS file.

- size_t uds_traverse_file (uds_open_file *file, char key[KEYNAME_SIZE], size_t start)
  Looks through a file, starting at the current cursor position, for a key.

- void uds_name_comment (uds_open_file *file, size_t location, char name[KEYNAME_SIZE], char comment[COMMENT_SIZE])
  Gets the name and comment from a key into arrays of size KEYNAME_SIZE (50) and COMMENT_SIZE (50).
- int uds_key_type (uds_open_file *file, size_t location)
  Gets the type of the key pointed to by 'buffer'.
- char * uds_get_value (uds_open_file *file, size_t location)
  Gets the value of a key as a string.
- char * uds_get_memregion (uds_open_file *file, size_t location)
  Gets the value of a key as a memory region.
- size_t uds_get_memr_len (uds_open_file *file, size_t location)
  Gets the length of a memory region.
- size_t uds_get_subtree (uds_open_file *file, size_t location)
  Gets the location of a subtree.

5.1.1 Detailed Description

Low-level tree functions and structures.

5.1.2 Typedef Documentation

5.1.2.1 typedef struct uds_bnode_t uds_bnode

This structure represents both a build key and a build tree.

Build keys and trees are not complete, but will be compiled on write.

5.1.2.2 typedef struct uds_open_file_t uds_open_file

Used internally to represent an open file.

Should never be modified.

5.1.3 Function Documentation

5.1.3.1 int uds_bnode_addkey (uds_bnode * *tree*, char *key*[KEYNAME_SIZE], char * *value*, size_t *value_len*, char *comment*[COMMENT_SIZE])

Adds a key to a build tree.

Note that this does not seek into subtrees.

Parameters:

tree The tree to add the key to.

key The key name.

value The key value.

value_len Length of key value, not counting NULL terminator.

comment Internal comment for the key.

Returns:

- 0 if addition was successful.
- 1 if key already exists.

5.1.3.2 int uds_bnode_addkey_keep (uds_bnode * *tree*, char *key*[KEYNAME_SIZE], char * *filename*, size_t *filename_len*, char *comment*[COMMENT_SIZE])

Adds a key to a build tree, keeping a filename for later addition.

Note that this does not seek into subtrees.

Parameters:

tree The tree to add the key to.

key The key name.

filename The name of the file to be added.

filename_len Length of file name, not counting NULL terminator.

comment Internal comment for the key.

Returns:

- 0 if addition was successful.
- 1 if key already exists.

5.1.3.3 int uds_bnode_addkey_memregion (uds_bnode * *tree*, char *key*[KEYNAME_SIZE], uint8_t * *memregion*, size_t *len*, char *comment*[COMMENT_SIZE])

Adds a key to a build tree from a memory region.

The memory region will not be read until uds_write_tree() is called. Note that this does not seek into subtrees.

Parameters:

tree The tree to add the key to.

key The key name.

memregion Pointer to the memory region to be added, as an an array of uint8_t.

len Length of the memory region.

comment Internal comment for the key.

Returns:

- 0 if addition was successful.
- 1 if key already exists or memregion was NULL.

5.1.3.4 uds_bnode* uds_bnode_addkey_subtree (uds_bnode * *tree*, char *key*[KEYNAME_SIZE], char *comment*[COMMENT_SIZE])

Adds a key to a build tree as a subtree.

Parameters:

tree The tree to add the key to.

key The key name.

comment Internal comment for the key.

Returns:

- Pointer to subtree if successful.
- NULL if not successful.

5.1.3.5 void uds_bnode_free_tree (uds_bnode * *tree*)

Frees a build tree.

This one is recursive; note however that memory regions 'submitted' with uds_btree_addkey_memregion are not freed.

5.1.3.6 uds_bnode* uds_bnode_getkey (uds_bnode * *tree*, char *key*[KEYNAME_SIZE])

Checks whether a key exists in a tree.

Note that this does not seek into subtrees.

Parameters:

tree The tree to search.

key The key name to look for.

Returns:

- Pointer to key if the key exists.
- NULL if it does not.

5.1.3.7 int uds_check_internal_magic (uds_open_file * *file*)

Checks the internal magic number, which should be UDS1.

Returns:

- 0 on success.
- 1 if the file was too short.
- 2 if the magic number is invalid.

5.1.3.8 size_t uds_get_memr_len (uds_open_file * *file*, size_t *location*)

Gets the length of a memory region.

See also:

uds_get_memregion

5.1.3.9 char* uds_get_memregion (uds_open_file * *file*, size_t *location*)

Gets the value of a key as a memory region.

This is a pointer into the file, as a mapped memory region. Do not free it. Note that no null terminator is guaranteed.

See also:

uds_get_memr_len

5.1.3.10 size_t uds_get_subtree (uds_open_file * *file*, size_t *location*)

Gets the location of a subtree.

This can then be passed to uds_traverse_file().

5.1.3.11 int uds_get_user_magic (uds_open_file * *file*, char *magic*[USER_MAGIC_SIZE])

Gets the user magic number from a UDS file.

Does not verify file type.

Returns:

- 0 on success.
- 1 if file was too short.

5.1.3.12 char* uds_get_value (uds_open_file * *file*, size_t *location*)

Gets the value of a key as a string.

Note that this copies the value, so don't use it for long values.

You must free this after using it.

5.1.3.13 int uds_key_type (uds_open_file * *file*, size_t *location*)

Gets the type of the key pointed to by 'buffer'.

Returns:

- 1 for data
- 2 for a subtree

5.1.3.14 uds_open_file* uds_open (char * *filename*)

Opens a UDS file.

Note that this uses mmap().

Returns:

- NULL if the file could not be opened

5.1.3.15 uds_open_file* uds_open_from_mem (char * *mem*, size_t *len*)

Treats a region of memory as a UDS file.

Returns a uds_open_file, simply free() this when done.

5.1.3.16 size_t uds_traverse_file (uds_open_file * *file*, char *key*[KEYNAME_SIZE], size_t *start*)

Looks through a file, starting at the current cursor position, for a key.

Does not look into subtrees.

Parameters:

start The location in the file to start at. Should be 1 to search the root, or the location of a subtree to search it.

Returns:

- Greater than zero if the key was found. The returned number can be used as the 'location' parameter for the information functions.

- Zero if the key was not found.

5.1.3.17 int uds_write_tree (uds_bnode * *tree*, char * *filename*, size_t *filename_len*, char *magic*[USER_MAGIC_SIZE])

Builds a tree and writes to disk.

Parameters:

tree The tree to write.

filename The filename to write to.

filename_len Length of key, not counting NULL terminator.

magic A four-character magic number. This will be concatenated to the end of 'UDS1' (UDS one) to make the actual magic number. This should not be null-terminated.

Returns:

- 0 on success.
- 1 on general failure.

Index