NAME

munge_ctx_create, munge_ctx_copy, munge_ctx_destroy, munge_ctx_strerror, munge_ctx_get, munge_ctx_set − MUNGE context functions

SYNOPSIS

#include <munge.h>

munge_ctx_t munge_ctx_create (void);

munge_ctx_t munge_ctx_copy (munge_ctx_t *ctx***);**

void munge_ctx_destroy (munge_ctx_t *ctx***);**

const char * munge_ctx_strerror (munge_ctx_t *ctx***);**

munge_err_t munge_ctx_get (munge_ctx_t *ctx***, munge_opt_t** *opt***, ...);**

munge_err_t munge_ctx_set (munge_ctx_t *ctx***, munge_opt_t** *opt***, ...);**

cc ... -lmunge

DESCRIPTION

The **munge_ctx_create**() function creates and returns a new MUNGE context, or NULL on error.

The **munge_ctx_copy**() function copies the context *ctx* and returns a new MUNGE context, or NULL on error.

The **munge_ctx_destroy**() function destroys the context *ctx*.

The **munge_ctx_strerror**() function returns a descriptive text string describing the MUNGE error number according to the context *ctx*, or NULL if no error condition exists. This may provide a more detailed error message than that returned by **munge_strerror**().

The **munge_ctx_get**() function gets the value for the option *opt* associated with the MUNGE context *ctx*, storing the result in the subsequent pointer argument. If the result is a string, that string should not be freed or modified by the caller.

The **munge_ctx_set**() function sets the value for the option *opt* associated with the MUNGE context *ctx*, using the value of the subsequent argument.

RETURN VALUE

The **munge_ctx_create**() and **munge_ctx_copy**() functions return a newly allocated MUNGE context, or NULL on error.

The **munge_ctx_strerror**() function returns a pointer to a NUL-terminated constant text string, or NULL if no error condition exists. This string should not be freed or modified by the caller.

The **munge_ctx_get**() and **munge_ctx_set**() functions return **EMUNGE_SUCCESS** on success, or a MUNGE error number otherwise.

CONTEXT OPTIONS

The following context options can be queried via **munge_ctx_get**() or specified via **munge_ctx_set**(). The type following each enumeration is the variable type used for the subsequent argument in **munge_ctx_set**(), or the variable type of a pointer used for the subsequent argument in **munge_ctx_get**().

MUNGE_OPT_CIPHER_TYPE , *int*
Get or set the cipher type (cf., **CIPHER TYPES**).

MUNGE_OPT_MAC_TYPE , *int*
Get or set the MAC type (cf., **MAC TYPES**).

MUNGE_OPT_ZIP_TYPE , *int*
Get or set the compression type (cf., **COMPRESSION TYPES**).

MUNGE_OPT_REALM , *char **
Get or set the security realm, where the *char ** type is a NUL-terminated character string. The string returned by **munge_ctx_get**() should not be freed or modified by the caller. **NOT CURRENTLY SUPPORTED**.

MUNGE_OPT_TTL , *int*
Get or set the time-to-live (in seconds) (cf., **TTL TYPES**). This value controls how long the credential is valid once it has been encoded.

MUNGE_OPT_ADDR4 , *struct in_addr*
Get the IPv4 address of the host where the credential was encoded. This option cannot be explicitly set.

MUNGE_OPT_ENCODE_TIME , *time_t*
Get the time (in seconds since the epoch) at which the credential was encoded. This option cannot be explicitly set.

MUNGE_OPT_DECODE_TIME , *time_t*
Get the time (in seconds since the epoch) at which the credential was decoded. This option cannot be explicitly set.

MUNGE_OPT_SOCKET , *char **
Get or set the local domain socket for connecting with **munged**, where the *char ** type is a NUL-terminated character string. The string returned by **munge_ctx_get**() should not be freed or modified by the caller.

MUNGE_OPT_UID_RESTRICTION , *uid_t*
Get or set the UID allowed to decode the credential (cf., **UID & GID TYPES**). This value will be matched against the effective user ID of the process requesting the credential decode.

MUNGE_OPT_GID_RESTRICTION , *gid_t*
Get or set the GID allowed to decode the credential (cf., **UID & GID TYPES**). This value will be matched against the effective group ID of the process requesting the credential decode, as well as each supplementary group of which the effective user ID of that process is a member.

CIPHER TYPES

Credentials can be encrypted using the secret key shared by all **munged** daemons within a security realm. Anyone having access to this key can use it to decrypt a credential, thereby bypassing any restrictions being imposed by **munged**.

MUNGE_CIPHER_NONE
Specify that encryption is to be disabled.

MUNGE_CIPHER_DEFAULT
Specify the default according to the **munged** configuration.

MUNGE_CIPHER_BLOWFISH
Specify the Blowfish cipher designed by Bruce Schneier. This cipher has a 64-bit block-size and a variable key length. MUNGE uses it with a 128-bit key in CBC mode. It is a fast block cipher but suffers from a slow key setup time. Consequently, it underperforms when generating small credentials.

MUNGE_CIPHER_CAST5
Specify the CAST5 cipher designed by Carlisle Adams and Stafford Tavares. This cipher has a 64-bit block-size and a variable key length. MUNGE uses it with a 128-bit key in CBC mode.

MUNGE_CIPHER_AES128
Specify the AES (Advanced Encryption Standard) cipher, also known as Rijndael. It was designed by Joan Daemen and Vincent Rijmen. This cipher has a 128-bit block-size and a key length of 128, 192, or 256 bits. MUNGE uses it here with a 128-bit key in CBC mode.

MUNGE_CIPHER_AES256
Specify the AES (Advanced Encryption Standard) cipher, also known as Rijndael. It was designed by Joan Daemen and Vincent Rijmen. This cipher has a 128-bit block-size and a key length of 128, 192, or 256 bits. MUNGE uses it here with a 256-bit key in CBC mode. Currently, **MUNGE_CIPHER_AES256** requires the use of **MUNGE_MAC_SHA256**.

MAC TYPES

The message authentication code (MAC) is a required component of the credential; consequently, it cannot be disabled.

MUNGE_MAC_DEFAULT
Specify the default according to the **munged** configuration.

MUNGE_MAC_MD5
Specify the MD5 algorithm designed by Ron Rivest and published in 1991. This algorithm has a 128-bit message digest. In 2004, a successful collision attack against MD5 was demonstrated. In 2009, a theoretical pre-image attack against MD5 was published. Consequently, use of MD5 is not recommended due to its lower security margin.

MUNGE_MAC_SHA1
Specify the SHA-1 algorithm designed by the National Security Agency and published in 1995; this is the successor to the original Secure Hash Algorithm (now called SHA-0) published in 1993. This algorithm has a 160-bit message digest. In 2005, successful collision attacks were demonstrated against SHA-1. But since a pre-image attack has not yet been demonstrated, SHA-1 should still be safe to use within MUNGE.

MUNGE_MAC_RIPEMD160
Specify the RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) algorithm designed in Europe by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, and published in 1996. This algorithm has a 160-bit message digest. It is somewhat less popular than SHA-1 and correspondingly less well studied. While slower than SHA-1, it is believed to have a slightly better security margin.

MUNGE_MAC_SHA256
Specify the SHA-256 algorithm designed by the National Security Agency and published in 2002; this is one of the SHA-2 variants in the Secure Hash Algorithm family. This algorithm has a 256-bit message digest. In 2006, NIST began encouraging the use of the SHA-2 family of hash functions for all new applications and protocols.

MUNGE_MAC_SHA512
Specify the SHA-512 algorithm designed by the National Security Agency and published in 2002; this is one of the SHA-2 variants in the Secure Hash Algorithm family. This algorithm has a 512-bit message digest. In 2006, NIST began encouraging the use of the SHA-2 family of hash functions for all new applications and protocols.

COMPRESSION TYPES

If a compression type is specified, a payload-bearing credential will be compressed accordingly. However, if the resulting compressed data is larger than the original uncompressed data, the uncompressed data will be restored and compression will be disabled for that credential.

MUNGE_ZIP_NONE
Specify that compression is to be disabled. This is the recommended setting unless there is a payload of sufficient size to compress.

MUNGE_ZIP_DEFAULT
Specify the default according to the **munged** configuration.

MUNGE_ZIP_BZLIB
Specify the bzip2 library developed by Julian Seward. This is slower and uses more memory, but generally gets better compression on larger payloads.

MUNGE_ZIP_ZLIB
Specify the zlib library developed by Jean-loup Gailly and Mark Adler. This is faster and uses less memory, but gets pretty good compression nonetheless.

TTL TYPES

The time-to-live value specifies the number of seconds after the encode-time that the credential is considered valid. In addition to specifying an integer value, the following types are available.

MUNGE_TTL_MAXIMUM
Specify the maximum allowed by the **munged** configuration.

MUNGE_TTL_DEFAULT
Specify the default according to the **munged** configuration.

UID & GID TYPES

The UID and GID restrictions can be used to restrict the decoding of the credential based on the effective user and group ID of the requesting process. In addition to specifying an integer value, the following types are available.

MUNGE_UID_ANY
Specify that no UID restriction is to take effect; this is the default behavior.

MUNGE_GID_ANY
Specify that no GID restriction is to take effect; this is the default behavior.

ERRORS

Refer to **munge**(3) for a complete list of errors.

EXAMPLE

The following example program illustrates the use of the MUNGE context to query the location of the **munged** domain socket.

```
#include <stdio.h>              /* for printf() */
#include <stdlib.h>             /* for exit() */
#include <munge.h>

int
main (int argc, char *argv[])
{
   munge_ctx_t  ctx;
   munge_err_t  err;
   char        *str;

   if (!(ctx = munge_ctx_create ())) {
      fprintf (stderr, "ERROR: Unable to create MUNGE context\n");
      exit (1);
   }
   err = munge_ctx_get (ctx, MUNGE_OPT_SOCKET, &str);

   if (err != EMUNGE_SUCCESS) {
      fprintf (stderr, "ERROR: %s\n", munge_ctx_strerror (ctx));
      exit (1);
   }
   printf ("socket=%s\n", str);
   /*
    *  Note that 'str' is not to be free()d since
    *    it points to a string within the 'ctx'.
    */
   munge_ctx_destroy (ctx);
   exit (0);
}
```

NOTES

Abandoning a new or copied MUNGE context without destroying it will result in a memory leak.

The context passed to **munge_encode**() is treated read-only except for the error message that is set when an error is returned. The context passed to **munge_decode**() is set according to the context used to encode the credential; however, on error, its settings may be in a state which is invalid for encoding. Consequently, separate contexts should be used for encoding and decoding.

A context should not be shared between threads unless it is protected by a mutex; however, a better alternative is to use a separate context (or two) for each thread, either by creating a new one via **munge_ctx_create**() or copying an existing one via **munge_ctx_copy**().

AUTHOR

Chris Dunlap <cdunlap AT llnl DOT gov>

COPYRIGHT

SEE ALSO

munge(1), **remunge**(1), **unmunge**(1), **munge**(3), **munge_enum**(3), **munge**(7), **munged**(8).

http://munge.googlecode.com/