Rocky Enterprise Linux 9.2 Manual Pages on command 'OSSL_PARAM_allocate_from_text.3ossl'

$ man OSSL_PARAM_allocate_from_text.3ossl

OSSL_PARAM_ALLOCATE_FROM_TEXT(3ossl)OpenSSOSSL_PARAM_ALLOCATE_FROM_TEXT(3ossl)

NAME

OSSL_PARAM_allocate_from_text - OSSL_PARAM construction utilities

SYNOPSIS

```
#include <openssl/params.h>

int OSSL_PARAM_allocate_from_text(OSSL_PARAM *to,
                                  const OSSL_PARAM *paramdefs,
                                  const char *key, const char *value,
                                  size_t value_n,
                                  int *found);
```

DESCRIPTION

With OpenSSL before version 3.0, parameters were passed down to or retrieved from algorithm implementations via control functions. Some of these control functions existed in variants that took string

parameters, for example EVP_PKEY_CTX_ctrl_str(3).

OpenSSL 3.0 introduces a new mechanism to do the same thing with an array of parameters that contain name, value, value type and value size (see OSSL_PARAM(3) for more information).

OSSL_PARAM_allocate_from_text() uses key to look up an item in paramdefs. If an item was found, it converts value to something suitable for that item's data_type, and stores the result in to->data as well as its size in to->data_size. to->key and to->data_type are assigned the corresponding values from the item that was found, and to->return_size is set to zero.

to->data is always allocated using OPENSSL_zalloc(3) and needs to be freed by the caller when it's not useful any more, using OPENSSL_free(3).

If found is not NULL, *found is set to 1 if key could be located in paramdefs, and to 0 otherwise.

The use of key and value in detail

OSSL_PARAM_allocate_from_text() takes note if key starts with "hex", and will only use the rest of key to look up an item in paramdefs in that case. As an example, if key is "hexid", "id" will be looked up in paramdefs.

When an item in paramdefs has been found, value is converted depending on that item's data_type, as follows:

OSSL_PARAM_INTEGER and OSSL_PARAM_UNSIGNED_INTEGER

If key didn't start with "hex", value is assumed to contain value_n decimal characters, which are decoded, and the resulting bytes become the number stored in the to->data storage.

If value starts with "0x", it is assumed to contain value_n
hexadecimal characters.

If key started with "hex", value is assumed to contain value_n
hexadecimal characters without the "0x" prefix.

If value contains characters that couldn't be decoded as
hexadecimal or decimal characters, OSSL_PARAM_allocate_from_text()
considers that an error.

OSSL_PARAM_UTF8_STRING

If key started with "hex", OSSL_PARAM_allocate_from_text()
considers that an error.

Otherwise, value is considered a C string and is copied to the
to->data storage. On systems where the native character encoding
is EBCDIC, the bytes in to->data are converted to ASCII.

OSSL_PARAM_OCTET_STRING

If key started with "hex", value is assumed to contain value_n
hexadecimal characters, which are decoded, and the resulting bytes
are stored in the to->data storage. If value contains characters
that couldn't be decoded as hexadecimal or decimal characters,
OSSL_PARAM_allocate_from_text() considers that an error.

If key didn't start with "hex", value_n bytes from value are copied
to the to->data storage.

RETURN VALUES

OSSL_PARAM_allocate_from_text() returns 1 if key was found in paramdefs
and there was no other failure, otherwise 0.

NOTES

The parameter descriptor array comes from functions dedicated to return them. The following OSSL_PARAM attributes are used:

key

data_type

data_size

All other attributes are ignored.

The data_size attribute can be zero, meaning that the parameter it describes expects arbitrary length data.

EXAMPLES

Code that looked like this:

```
 int mac_ctrl_string(EVP_PKEY_CTX *ctx, const char *value)
 {
     int rv;
     char *stmp, *vtmp = NULL;

     stmp = OPENSSL_strdup(value);
     if (stmp == NULL)
         return -1;
     vtmp = strchr(stmp, ':');
     if (vtmp != NULL)
         *vtmp++ = '\0';
     rv = EVP_MAC_ctrl_str(ctx, stmp, vtmp);
     OPENSSL_free(stmp);
     return rv;
 }

 ...
```

```
 for (i = 0; i < sk_OPENSSL_STRING_num(macopts); i++) {
     char *macopt = sk_OPENSSL_STRING_value(macopts, i);

     if (pkey_ctrl_string(mac_ctx, macopt) <= 0) {
         BIO_printf(bio_err,
                    "MAC parameter error \"%s\"\n", macopt);
         ERR_print_errors(bio_err);
         goto mac_end;
     }
 }
```

Can be written like this instead:

```
 OSSL_PARAM *params =
     OPENSSL_zalloc(sizeof(*params)
                    * (sk_OPENSSL_STRING_num(opts) + 1));
 const OSSL_PARAM *paramdefs = EVP_MAC_settable_ctx_params(mac);
 size_t params_n;
 char *opt = "<unknown>";

 for (params_n = 0; params_n < (size_t)sk_OPENSSL_STRING_num(opts);
      params_n++) {
     char *stmp, *vtmp = NULL;

     opt = sk_OPENSSL_STRING_value(opts, (int)params_n);
     if ((stmp = OPENSSL_strdup(opt)) == NULL
             || (vtmp = strchr(stmp, ':')) == NULL)
         goto err;

     *vtmp++ = '\0';
     if (!OSSL_PARAM_allocate_from_text(&params[params_n],
                                        paramdefs, stmp,
```

```
                                vtmp, strlen(vtmp), NULL))
            goto err;
        }
        params[params_n] = OSSL_PARAM_construct_end();
        if (!EVP_MAC_CTX_set_params(ctx, params))
            goto err;
        while (params_n-- > 0)
            OPENSSL_free(params[params_n].data);
        OPENSSL_free(params);
        /* ... */
        return;

       err:
        BIO_printf(bio_err, "MAC parameter error '%s'\n", opt);
        ERR_print_errors(bio_err);
```

SEE ALSO

OSSL_PARAM(3), OSSL_PARAM_int(3)

COPYRIGHT