



*Full credit is given to the above companies including the OS that this PDF file was generated!*

### ***Rocky Enterprise Linux 9.2 Manual Pages on command 'OSSL\_PARAM.3ossl'***

***\$ man OSSL\_PARAM.3ossl***

OSSL\_PARAM(3ossl)                      OpenSSL                      OSSL\_PARAM(3ossl)

NAME

OSSL\_PARAM - a structure to pass or request object parameters

SYNOPSIS

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

typedef struct ossl_param_st OSSL_PARAM;

struct ossl_param_st {
    const char *key;                      /* the name of the parameter */
    unsigned char data_type;              /* declare what kind of content is in data */
    void *data;                            /* value being passed in or out */
    size_t data_size;                      /* data size */
    size_t return_size;                    /* returned size */
};
```

DESCRIPTION

OSSL\_PARAM is a type that allows passing arbitrary data for some object between two parties that have no or very little shared knowledge about their respective internal structures for that object.

A typical usage example could be an application that wants to set some

parameters for an object, or wants to find out some parameters of an object.

Arrays of this type can be used for the following purposes:

? Setting parameters for some object

The caller sets up the OSSL\_PARAM array and calls some function (the setter) that has intimate knowledge about the object that can take the data from the OSSL\_PARAM array and assign them in a suitable form for the internal structure of the object.

? Request parameters of some object

The caller (the requestor) sets up the OSSL\_PARAM array and calls some function (the responder) that has intimate knowledge about the object, which can take the internal data of the object and copy (possibly convert) that to the memory prepared by the requestor and pointed at with the OSSL\_PARAM data.

? Request parameter descriptors

The caller gets an array of constant OSSL\_PARAM, which describe available parameters and some of their properties; name, data type and expected data size. For a detailed description of each field for this use, see the field descriptions below.

The caller may then use the information from this descriptor array to build up its own OSSL\_PARAM array to pass down to a setter or responder.

Normally, the order of the an OSSL\_PARAM array is not relevant.

However, if the responder can handle multiple elements with the same key, those elements must be handled in the order they are in.

An OSSL\_PARAM array must have a terminating element, where key is NULL.

The usual full terminating template is:

```
{ NULL, 0, NULL, 0, 0 }
```

This can also be specified using OSSL\_PARAM\_END(3).

## Functional support

Libcrypto offers a limited set of helper functions to handle OSSL\_PARAM items and arrays, please see OSSL\_PARAM\_get\_int(3). Developers are free to extend or replace those as they see fit.

## OSSL\_PARAM fields

**key** The identity of the parameter in the form of a string.

In an OSSL\_PARAM array, an item with this field set to NULL is considered a terminating item.

### data\_type

The data\_type is a value that describes the type and organization of the data. See "Supported types" below for a description of the types.

### data

### data\_size

data is a pointer to the memory where the parameter data is (when setting parameters) or shall (when requesting parameters) be stored, and data\_size is its size in bytes. The organization of the data depends on the parameter type and flag.

The data\_size needs special attention with the parameter type OSSL\_PARAM\_UTF8\_STRING in relation to C strings. When setting parameters, the size should be set to the length of the string, not counting the terminating NUL byte. When requesting parameters, the size should be set to the size of the buffer to be populated, which should accommodate enough space for a terminating NUL byte.

When requesting parameters, it's acceptable for data to be NULL.

This can be used by the requestor to figure out dynamically exactly how much buffer space is needed to store the parameter data. In this case, data\_size is ignored.

When the OSSL\_PARAM is used as a parameter descriptor, data should be ignored. If data\_size is zero, it means that an arbitrary data size is accepted, otherwise it specifies the maximum size allowed.

### return\_size

When an array of OSSL\_PARAM is used to request data, the responder must set this field to indicate size of the parameter data, including padding as the case may be. In case the data\_size is an unsuitable size for the data, the responder must still set this field to indicate the minimum data size required. (further notes

on this in "NOTES" below).

When the OSSL\_PARAM is used as a parameter descriptor, return\_size should be ignored.

#### NOTE:

The key names and associated types are defined by the entity that offers these parameters, i.e. names for parameters provided by the OpenSSL libraries are defined by the libraries, and names for parameters provided by providers are defined by those providers, except for the pointer form of strings (see data type descriptions below).

Entities that want to set or request parameters need to know what those keys are and of what type, any functionality between those two entities should remain oblivious and just pass the OSSL\_PARAM array along.

#### Supported types

The data\_type field can be one of the following types:

OSSL\_PARAM\_INTEGER

OSSL\_PARAM\_UNSIGNED\_INTEGER

The parameter data is an integer (signed or unsigned) of arbitrary length, organized in native form, i.e. most significant byte first on Big-Endian systems, and least significant byte first on Little-Endian systems.

OSSL\_PARAM\_REAL

The parameter data is a floating point value in native form.

OSSL\_PARAM\_UTF8\_STRING

The parameter data is a printable string.

OSSL\_PARAM\_OCTET\_STRING

The parameter data is an arbitrary string of bytes.

OSSL\_PARAM\_UTF8\_PTR

The parameter data is a pointer to a printable string.

The difference between this and OSSL\_PARAM\_UTF8\_STRING is that data doesn't point directly at the data, but to a pointer that points to the data.

If there is any uncertainty about which to use,

OSSL\_PARAM\_UTF8\_STRING is almost certainly the correct choice.

This is used to indicate that constant data is or will be passed, and there is therefore no need to copy the data that is passed, just the pointer to it.

`data_size` must be set to the size of the data, not the size of the pointer to the data. If this is used in a parameter request, `data_size` is not relevant. However, the responder will set `return_size` to the size of the data.

Note that the use of this type is fragile and can only be safely used for data that remains constant and in a constant location for a long enough duration (such as the life-time of the entity that offers these parameters).

#### OSSL\_PARAM\_OCTET\_PTR

The parameter data is a pointer to an arbitrary string of bytes.

The difference between this and `OSSL_PARAM_OCTET_STRING` is that data doesn't point directly at the data, but to a pointer that points to the data.

If there is any uncertainty about which to use,

`OSSL_PARAM_OCTET_STRING` is almost certainly the correct choice.

This is used to indicate that constant data is or will be passed, and there is therefore no need to copy the data that is passed, just the pointer to it.

`data_size` must be set to the size of the data, not the size of the pointer to the data. If this is used in a parameter request, `data_size` is not relevant. However, the responder will set `return_size` to the size of the data.

Note that the use of this type is fragile and can only be safely used for data that remains constant and in a constant location for a long enough duration (such as the life-time of the entity that offers these parameters).

#### NOTES

Both when setting and requesting parameters, the functions that are called will have to decide what is and what is not an error. The recommended behaviour is:

- ? Keys that a setter or responder doesn't recognise should simply be ignored. That in itself isn't an error.
- ? If the keys that a called setter recognises form a consistent enough set of data, that call should succeed.
- ? Apart from the return\_size, a responder must never change the fields of an OSSL\_PARAM. To return a value, it should change the contents of the memory that data points at.
- ? If the data type for a key that it's associated with is incorrect, the called function may return an error.  
  
The called function may also try to convert the data to a suitable form (for example, it's plausible to pass a large number as an octet string, so even though a given key is defined as an OSSL\_PARAM\_UNSIGNED\_INTEGER, is plausible to pass the value as an OSSL\_PARAM\_OCTET\_STRING), but this is in no way mandatory.
- ? If a responder finds that some data sizes are too small for the requested data, it must set return\_size for each such OSSL\_PARAM item to the minimum required size, and eventually return an error.
- ? For the integer type parameters (OSSL\_PARAM\_UNSIGNED\_INTEGER and OSSL\_PARAM\_INTEGER), a responder may choose to return an error if the data\_size isn't a suitable size (even if data\_size is bigger than needed). If the responder finds the size suitable, it must fill all data\_size bytes and ensure correct padding for the native endianness, and set return\_size to the same value as data\_size.

## EXAMPLES

A couple of examples to just show how OSSL\_PARAM arrays could be set up.

### Example 1

This example is for setting parameters on some object:

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

const char *foo = "some string";
size_t foo_l = strlen(foo);

const char bar[] = "some other string";

OSSL_PARAM set[] = {
```

```

{ "foo", OSSL_PARAM_UTF8_PTR, &foo, foo_l, 0 },
{ "bar", OSSL_PARAM_UTF8_STRING, (void *)&bar, sizeof(bar) - 1, 0 },
{ NULL, 0, NULL, 0, 0 }
};

```

## Example 2

This example is for requesting parameters on some object:

```

const char *foo = NULL;

size_t foo_l;

char bar[1024];

size_t bar_l;

OSSL_PARAM request[] = {
    { "foo", OSSL_PARAM_UTF8_PTR, &foo, 0 /*irrelevant*/, 0 },
    { "bar", OSSL_PARAM_UTF8_STRING, &bar, sizeof(bar), 0 },
    { NULL, 0, NULL, 0, 0 }
};

```

A responder that receives this array (as params in this example) could fill in the parameters like this:

```

/* OSSL_PARAM *params */

int i;

for (i = 0; params[i].key != NULL; i++) {
    if (strcmp(params[i].key, "foo") == 0) {
        *(char **)params[i].data = "foo value";
        params[i].return_size = 9; /* length of "foo value" string */
    } else if (strcmp(params[i].key, "bar") == 0) {
        memcpy(params[i].data, "bar value", 10);
        params[i].return_size = 9; /* length of "bar value" string */
    }

    /* Ignore stuff we don't know */
}

```

## SEE ALSO

[openssl-core.h\(7\)](#), [OSSL\\_PARAM\\_get\\_int\(3\)](#), [OSSL\\_PARAM\\_dup\(3\)](#)

## HISTORY

OSSL\_PARAM was added in OpenSSL 3.0.

## COPYRIGHT

Copyright 2019-2022 The OpenSSL Project Authors. All Rights Reserved.  
Licensed under the Apache License 2.0 (the "License"). You may not use  
this file except in compliance with the License. You can obtain a copy  
in the file LICENSE in the source distribution or at  
<<https://www.openssl.org/source/license.html>>.

3.0.7                    2023-07-13                    OSSL\_PARAM(3ossl)