Rocky Enterprise Linux 9.2 Manual Pages on command 'OSSL_TRACE1.3ossl'

$ man OSSL_TRACE1.3ossl

OSSL_TRACE_ENABLED(3ossl) OpenSSL OSSL_TRACE_ENABLED(3ossl)

NAME

OSSL_trace_enabled, OSSL_trace_begin, OSSL_trace_end, OSSL_TRACE_BEGIN, OSSL_TRACE_END, OSSL_TRACE_CANCEL, OSSL_TRACE, OSSL_TRACE1, OSSL_TRACE2, OSSL_TRACE3, OSSL_TRACE4, OSSL_TRACE5, OSSL_TRACE6, OSSL_TRACE7, OSSL_TRACE8, OSSL_TRACE9, OSSL_TRACEV, OSSL_TRACE_ENABLED - OpenSSL Tracing API

SYNOPSIS

```
#include <openssl/trace.h>
int OSSL_trace_enabled(int category);
BIO *OSSL_trace_begin(int category);
void OSSL_trace_end(int category, BIO *channel);
/* trace group macros */
OSSL_TRACE_BEGIN(category) {
    ...
    if (some_error) {
        /* Leave trace group prematurely in case of an error */
        OSSL_TRACE_CANCEL(category);
```

```
        goto err;
    }
    ...
} OSSL_TRACE_END(category);
/* one-shot trace macros */
OSSL_TRACE1(category, format, arg1)
OSSL_TRACE2(category, format, arg1, arg2)
...
OSSL_TRACE9(category, format, arg1, ..., arg9)
/* check whether a trace category is enabled */
if (OSSL_TRACE_ENABLED(category)) {
    ...
}
```

DESCRIPTION

The functions described here are mainly interesting for those who provide OpenSSL functionality, either in OpenSSL itself or in engine modules or similar.

If tracing is enabled (see "NOTES" below), these functions are used to generate free text tracing output.

The tracing output is divided into types which are enabled individually by the application. The tracing types are described in detail in "Trace types" in OSSL_trace_set_callback(3). The fallback type OSSL_TRACE_CATEGORY_ALL should not be used with the functions described here.

Tracing for a specific category is enabled if a so called trace channel is attached to it. A trace channel is simply a BIO object to which the application can write its trace output.

The application has two different ways of registering a trace channel, either by directly providing a BIO object using OSSL_trace_set_channel(), or by providing a callback routine using OSSL_trace_set_callback(). The latter is wrapped internally by a dedicated BIO object, so for the tracing code both channel types are effectively indistinguishable. We call them a simple trace channel and

a callback trace channel, respectively.

To produce trace output, it is necessary to obtain a pointer to the trace channel (i.e., the BIO object) using OSSL_trace_begin(), write to it using arbitrary BIO output routines, and finally releases the channel using OSSL_trace_end(). The OSSL_trace_begin()/OSSL_trace_end() calls surrounding the trace output create a group, which acts as a critical section (guarded by a mutex) to ensure that the trace output of different threads does not get mixed up.

The tracing code normally does not call OSSL_trace_{begin,end}() directly, but rather uses a set of convenience macros, see the "Macros" section below.

Functions

OSSL_trace_enabled() can be used to check if tracing for the given category is enabled.

OSSL_trace_begin() is used to starts a tracing section, and get the channel for the given category in form of a BIO. This BIO can only be used for output.

OSSL_trace_end() is used to end a tracing section.

Using OSSL_trace_begin() and OSSL_trace_end() to wrap tracing sections is mandatory. The result of trying to produce tracing output outside of such sections is undefined.

Macros

There are a number of convenience macros defined, to make tracing easy and consistent.

OSSL_TRACE_BEGIN() and OSSL_TRACE_END() reserve the BIO "trc_out" and are used as follows to wrap a trace section:

```
 OSSL_TRACE_BEGIN(TLS) {
     BIO_fprintf(trc_out, ... );
 } OSSL_TRACE_END(TLS);
```

This will normally expand to:

```
 do {
     BIO *trc_out = OSSL_trace_begin(OSSL_TRACE_CATEGORY_TLS);
     if (trc_out != NULL) {
```

```
        ...
        BIO_fprintf(trc_out, ...);
    }
    OSSL_trace_end(OSSL_TRACE_CATEGORY_TLS, trc_out);
 } while (0);
```

OSSL_TRACE_CANCEL() must be used before returning from or jumping out
of a trace section:

```
 OSSL_TRACE_BEGIN(TLS) {
    if (some_error) {
        OSSL_TRACE_CANCEL(TLS);
        goto err;
    }
    BIO_fprintf(trc_out, ... );
 } OSSL_TRACE_END(TLS);
```

This will normally expand to:

```
 do {
    BIO *trc_out = OSSL_trace_begin(OSSL_TRACE_CATEGORY_TLS);
    if (trc_out != NULL) {
        if (some_error) {
            OSSL_trace_end(OSSL_TRACE_CATEGORY_TLS, trc_out);
            goto err;
        }
        BIO_fprintf(trc_out, ... );
    }
    OSSL_trace_end(OSSL_TRACE_CATEGORY_TLS, trc_out);
 } while (0);
```

OSSL_TRACE() and OSSL_TRACE1(), OSSL_TRACE2(), ... OSSL_TRACE9() are
so-called one-shot macros:

The macro call "OSSL_TRACE(category, text)", produces literal text
trace output.

The macro call "OSSL_TRACEn(category, format, arg1, ..., argn)"
produces printf-style trace output with n format field arguments
(n=1,...,9). It expands to:

```
OSSL_TRACE_BEGIN(category) {
    BIO_printf(trc_out, format, arg1, ..., argN)
} OSSL_TRACE_END(category)
```

Internally, all one-shot macros are implemented using a generic OSSL_TRACEV() macro, since C90 does not support variadic macros. This helper macro has a rather weird synopsis and should not be used directly.

The OSSL_TRACE_ENABLED() macro can be used to conditionally execute some code only if a specific trace category is enabled. In some situations this is simpler than entering a trace section using OSSL_TRACE_BEGIN() and OSSL_TRACE_END(). For example, the code

```
if (OSSL_TRACE_ENABLED(TLS)) {
    ...
}
```

expands to

```
if (OSSL_trace_enabled(OSSL_TRACE_CATEGORY_TLS) {
    ...
}
```

NOTES

If producing the trace output requires carrying out auxiliary calculations, this auxiliary code should be placed inside a conditional block which is executed only if the trace category is enabled.

The most natural way to do this is to place the code inside the trace section itself because it already introduces such a conditional block.

```
OSSL_TRACE_BEGIN(TLS) {
    int var = do_some_auxiliary_calculation();
    BIO_printf(trc_out, "var = %d\n", var);
} OSSL_TRACE_END(TLS);
```

In some cases it is more advantageous to use a simple conditional group instead of a trace section. This is the case if calculations and tracing happen in different locations of the code, or if the calculations are so time consuming that placing them inside a (critical) trace section would create too much contention.

```
if (OSSL_TRACE_ENABLED(TLS)) {
    int var = do_some_auxiliary_calculation();
    OSSL_TRACE1("var = %d\n", var);
}
```

Note however that premature optimization of tracing code is in general futile and it's better to keep the tracing code as simple as possible. Because most often the limiting factor for the application's speed is the time it takes to print the trace output, not to calculate it.

Configure Tracing

By default, the OpenSSL library is built with tracing disabled. To use the tracing functionality documented here, it is therefore necessary to configure and build OpenSSL with the 'enable-trace' option.

When the library is built with tracing disabled:

- The macro OPENSSL_NO_TRACE is defined in <openssl/opensslconf.h>.
- all functions are still present, but OSSL_trace_enabled() will always report the categories as disabled, and all other functions will do nothing.
- the convenience macros are defined to produce dead code. For example, take this example from "Macros" section above:

```
OSSL_TRACE_BEGIN(TLS) {
    if (condition) {
        OSSL_TRACE_CANCEL(TLS);
        goto err;
    }
    BIO_fprintf(trc_out, ... );
} OSSL_TRACE_END(TLS);
```

When the tracing API isn't operational, that will expand to:

```
do {
    BIO *trc_out = NULL;
    if (0) {
        if (condition) {
            ((void)0);
            goto err;
```

```
            }
            BIO_fprintf(trc_out, ... );
        }
    } while (0);
```

RETURN VALUES

OSSL_trace_enabled() returns 1 if tracing for the given type is operational and enabled, otherwise 0.

OSSL_trace_begin() returns a BIO pointer if the given type is enabled, otherwise NULL.

HISTORY

The OpenSSL Tracing API was added in OpenSSL 3.0.

COPYRIGHT