NAME

OSSL_trace_set_channel, OSSL_trace_set_prefix, OSSL_trace_set_suffix, OSSL_trace_set_callback, OSSL_trace_cb - Enabling trace output

SYNOPSIS

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

 typedef size_t (*OSSL_trace_cb)(const char *buf, size_t cnt,
                                 int category, int cmd, void *data);

 void OSSL_trace_set_channel(int category, BIO *bio);
 void OSSL_trace_set_prefix(int category, const char *prefix);
 void OSSL_trace_set_suffix(int category, const char *suffix);
 void OSSL_trace_set_callback(int category, OSSL_trace_cb cb, void  *data);
```

DESCRIPTION

If available (see "NOTES" below), the application can request internal trace output. This output comes in form of free text for humans to read.

The trace output is divided into categories which can be enabled individually. Every category can be enabled individually by attaching a so called *trace channel* to it, which in the simplest case is just a BIO object to which the application can write the tracing output for this category. Alternatively, the application can provide a tracer callback in order to get more finegrained trace information. This callback will be wrapped internally by a dedicated BIO object.

For the tracing code, both trace channel types are indistinguishable. These are called a *simple trace channel* and a *callback trace channel*, respectively.

Functions

OSSL_trace_set_channel() is used to enable the given trace "category" by attaching the **BIO** *bio* object as (simple) trace channel. On success the ownership of the BIO is transferred to the channel, so the caller must not free it directly.

OSSL_trace_set_prefix() and **OSSL_trace_set_suffix()** can be used to add an extra line for each channel, to be output before and after group of tracing output. What constitutes an output group is decided by the code that produces the output. The lines given here are considered immutable; for more dynamic tracing prefixes, consider setting a callback with **OSSL_trace_set_callback()** instead.

OSSL_trace_set_callback() is used to enable the given trace *category* by giving it the tracer callback *cb* with the associated data *data*, which will simply be passed through to *cb* whenever it’s called. The callback function is internally wrapped by a dedicated BIO object, the so called *callback trace channel*. This should be used when it's desirable to do form the trace output to something suitable for application needs where a prefix and suffix line aren't enough.

OSSL_trace_set_channel() and **OSSL_trace_set_callback()** are mutually exclusive, calling one of them will clear whatever was set by the previous call.

Calling **OSSL_trace_set_channel()** with NULL for *channel* or **OSSL_trace_set_callback()** with NULL for *cb* disables tracing for the given *category*.

Trace callback

The tracer callback must return a **size_t**, which must be zero on error and otherwise return the number of bytes that were output. It receives a text buffer *buf* with *cnt* bytes of text, as well as the *category*, a control number *cmd*, and the *data* that was passed to **OSSL_trace_set_callback()**.

The possible control numbers are:

OSSL_TRACE_CTRL_BEGIN

The callback is called from **OSSL_trace_begin()**, which gives the callback the possibility to output a dynamic starting line, or set a prefix that should be output at the beginning of each line, or something other.

OSSL_TRACE_CTRL_WRITE

This callback is called whenever data is written to the BIO by some regular BIO output routine. An arbitrary number of **OSSL_TRACE_CTRL_WRITE** callbacks can occur inside a group marked by a pair of **OSSL_TRACE_CTRL_BEGIN** and **OSSL_TRACE_CTRL_END** calls, but never outside such a group.

OSSL_TRACE_CTRL_END

The callback is called from **OSSL_trace_end()**, which gives the callback the possibility to output a dynamic ending line, or reset the line prefix that was set with **OSSL_TRACE_CTRL_BEGIN**, or something other.

Trace categories

The trace categories are simple numbers available through macros.

OSSL_TRACE_CATEGORY_TRACE

Traces the OpenSSL trace API itself.

More precisely, this will generate trace output any time a new trace hook is set.

OSSL_TRACE_CATEGORY_INIT

Traces OpenSSL library initialization and cleanup.

This needs special care, as OpenSSL will do automatic cleanup after exit from "main()", and any tracing output done during this cleanup will be lost if the tracing channel or callback were cleaned away prematurely. A suggestion is to make such cleanup part of a function that's registered very early with **atexit**(3).

OSSL_TRACE_CATEGORY_TLS

Traces the TLS/SSL protocol.

OSSL_TRACE_CATEGORY_TLS_CIPHER

Traces the ciphers used by the TLS/SSL protocol.

OSSL_TRACE_CATEGORY_CONF

Traces details about the provider and engine configuration.

OSSL_TRACE_CATEGORY_ENGINE_TABLE

Traces the ENGINE algorithm table selection.

More precisely, functions like **ENGINE_get_pkey_asn1_meth_engine()**, **ENGINE_get_pkey_meth_engine()**, **ENGINE_get_cipher_engine()**, **ENGINE_get_digest_engine()**, will generate trace summaries of the handling of internal tables.

OSSL_TRACE_CATEGORY_ENGINE_REF_COUNT

Traces the ENGINE reference counting.

More precisely, both reference counts in the ENGINE structure will be monitored with a line of trace output generated for each change.

OSSL_TRACE_CATEGORY_PKCS5V2

Traces PKCS#5 v2 key generation.

OSSL_TRACE_CATEGORY_PKCS12_KEYGEN

Traces PKCS#12 key generation.

OSSL_TRACE_CATEGORY_PKCS12_DECRYPT

Traces PKCS#12 decryption.

OSSL_TRACE_CATEGORY_X509V3_POLICY

Traces X509v3 policy processing.

More precisely, this generates the complete policy tree at various point during evaluation.

OSSL_TRACE_CATEGORY_BN_CTX
Traces BIGNUM context operations.

OSSL_TRACE_CATEGORY_CMP
Traces CMP client and server activity.

OSSL_TRACE_CATEGORY_STORE
Traces STORE operations.

OSSL_TRACE_CATEGORY_DECODER
Traces decoder operations.

OSSL_TRACE_CATEGORY_ENCODER
Traces encoder operations.

OSSL_TRACE_CATEGORY_REF_COUNT
Traces decrementing certain ASN.1 structure references.

There is also **OSSL_TRACE_CATEGORY_ALL**, which works as a fallback and can be used to get *all* trace output.

Note, however, that in this case all trace output will effectively be associated with the ’ALL’ category, which is undesirable if the application intends to include the category name in the trace output. In this case it is better to register separate channels for each trace category instead.

RETURN VALUES

OSSL_trace_set_channel(), **OSSL_trace_set_prefix()**, **OSSL_trace_set_suffix()**, and **OSSL_trace_set_callback()** return 1 on success, or 0 on failure.

EXAMPLES

In all examples below, the trace producing code is assumed to be the following:

```
int foo = 42;
const char bar[] = { 0,  1,  2,  3,  4,  5,  6,  7,
                     8,  9, 10, 11, 12, 13, 14, 15 };
```

```
 OSSL_TRACE_BEGIN(TLS) {
     BIO_puts(trc_out, "foo: ");
     BIO_printf(trc_out, "%d\n", foo);
     BIO_dump(trc_out, bar, sizeof(bar));
 } OSSL_TRACE_END(TLS);
```

Simple example

An example with just a channel and constant prefix / suffix.

```
 int main(int argc, char *argv[])
 {
     BIO *err = BIO_new_fp(stderr, BIO_NOCLOSE | BIO_FP_TEXT);
     OSSL_trace_set_channel(OSSL_TRACE_CATEGORY_SSL, err);
     OSSL_trace_set_prefix(OSSL_TRACE_CATEGORY_SSL, "BEGIN TRACE[TLS]");
     OSSL_trace_set_suffix(OSSL_TRACE_CATEGORY_SSL, "END TRACE[TLS]");

     /* ... work ... */
 }
```

When the trace producing code above is performed, this will be output on standard error:

```
 BEGIN TRACE[TLS]
 foo: 42
 0000 - 00 01 02 03 04 05 06 07-08 09 0a 0b 0c 0d 0e 0f   ................
 END TRACE[TLS]
```

Advanced example

This example uses the callback, and depends on pthreads functionality.

```
 static size_t cb(const char *buf, size_t cnt,
                 int category, int cmd, void *vdata)
 {
     BIO *bio = vdata;
     const char *label = NULL;

     switch (cmd) {
     case OSSL_TRACE_CTRL_BEGIN:
         label = "BEGIN";
         break;
     case OSSL_TRACE_CTRL_END:
         label = "END";
         break;
     }

     if (label != NULL) {
         union {
             pthread_t tid;
             unsigned long ltid;
         } tid;

         tid.tid = pthread_self();
         BIO_printf(bio, "%s TRACE[%s]:%lx\n",
                    label, OSSL_trace_get_category_name(category), tid.ltid);
     }
     return (size_t)BIO_puts(bio, buf);
 }

 int main(int argc, char *argv[])
 {
     BIO *err = BIO_new_fp(stderr, BIO_NOCLOSE | BIO_FP_TEXT);
     OSSL_trace_set_callback(OSSL_TRACE_CATEGORY_SSL, cb, err);

     /* ... work ... */
 }
```

The output is almost the same as for the simple example above.

```
    BEGIN TRACE[TLS]:7f9eb0193b80
    foo: 42
    0000 - 00 01 02 03 04 05 06 07-08 09 0a 0b 0c 0d 0e 0f   ................
    END TRACE[TLS]:7f9eb0193b80
```

NOTES

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>* and all functions described here are inoperational, i.e. will do nothing.

HISTORY

OSSL_trace_set_channel(), **OSSL_trace_set_prefix()**, **OSSL_trace_set_suffix()**, and **OSSL_trace_set_callback()** were all added in OpenSSL 3.0.

COPYRIGHT