NAME

OSSL_ENCODER_CTX, OSSL_ENCODER_CTX_new, OSSL_ENCODER_settable_ctx_params, OSSL_ENCODER_CTX_set_params, OSSL_ENCODER_CTX_free, OSSL_ENCODER_CTX_set_selection, OSSL_ENCODER_CTX_set_output_type, OSSL_ENCODER_CTX_set_output_structure, OSSL_ENCODER_CTX_add_encoder, OSSL_ENCODER_CTX_add_extra, OSSL_ENCODER_CTX_get_num_encoders, OSSL_ENCODER_INSTANCE, OSSL_ENCODER_INSTANCE_get_encoder, OSSL_ENCODER_INSTANCE_get_encoder_ctx, OSSL_ENCODER_INSTANCE_get_output_type, OSSL_ENCODER_INSTANCE_get_output_structure, OSSL_ENCODER_CONSTRUCT, OSSL_ENCODER_CLEANUP, OSSL_ENCODER_CTX_set_construct, OSSL_ENCODER_CTX_set_construct_data, OSSL_ENCODER_CTX_set_cleanup - Encoder context routines

SYNOPSIS

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

typedef struct ossl_encoder_ctx_st OSSL_ENCODER_CTX;

OSSL_ENCODER_CTX *OSSL_ENCODER_CTX_new();
const OSSL_PARAM *OSSL_ENCODER_settable_ctx_params(OSSL_ENCODER *encoder);
int OSSL_ENCODER_CTX_set_params(OSSL_ENCODER_CTX *ctx,
                                const OSSL_PARAM params[]);
void OSSL_ENCODER_CTX_free(OSSL_ENCODER_CTX *ctx);

int OSSL_ENCODER_CTX_set_selection(OSSL_ENCODER_CTX *ctx, int selection);
int OSSL_ENCODER_CTX_set_output_type(OSSL_ENCODER_CTX *ctx,
                                     const char *output_type);
int OSSL_ENCODER_CTX_set_output_structure(OSSL_ENCODER_CTX *ctx,
                                          const char *output_structure);

int OSSL_ENCODER_CTX_add_encoder(OSSL_ENCODER_CTX *ctx, OSSL_ENCODER *encoder);
int OSSL_ENCODER_CTX_add_extra(OSSL_ENCODER_CTX *ctx,
                               OSSL_LIB_CTX *libctx, const char *propq);
int OSSL_ENCODER_CTX_get_num_encoders(OSSL_ENCODER_CTX *ctx);

typedef struct ossl_encoder_instance_st OSSL_ENCODER_INSTANCE;
OSSL_ENCODER *
OSSL_ENCODER_INSTANCE_get_encoder(OSSL_ENCODER_INSTANCE *encoder_inst);
void *
OSSL_ENCODER_INSTANCE_get_encoder_ctx(OSSL_ENCODER_INSTANCE *encoder_inst);
```

```
const char *
OSSL_ENCODER_INSTANCE_get_output_type(OSSL_ENCODER_INSTANCE *encoder_inst);
const char *
OSSL_ENCODER_INSTANCE_get_output_structure(OSSL_ENCODER_INSTANCE *encoder_inst);

typedef const void *OSSL_ENCODER_CONSTRUCT(OSSL_ENCODER_INSTANCE *encoder_inst,
                                           void *construct_data);
typedef void OSSL_ENCODER_CLEANUP(void *construct_data);

int OSSL_ENCODER_CTX_set_construct(OSSL_ENCODER_CTX *ctx,
                                   OSSL_ENCODER_CONSTRUCT *construct);
int OSSL_ENCODER_CTX_set_construct_data(OSSL_ENCODER_CTX *ctx,
                                        void *construct_data);
int OSSL_ENCODER_CTX_set_cleanup(OSSL_ENCODER_CTX *ctx,
                                 OSSL_ENCODER_CLEANUP *cleanup);
```

DESCRIPTION

Encoding an input object to the desired encoding may be done with a chain of encoder implementations, which means that the output from one encoder may be the input for the next in the chain. The **OSSL_ENCODER_CTX** holds all the data about these encoders. This allows having generic format encoders such as DER to PEM, as well as more specialized encoders like RSA to DER.

The final output type must be given, and a chain of encoders must end with an implementation that produces that output type.

At the beginning of the encoding process, a constructor provided by the caller is called to ensure that there is an appropriate provider-side object to start with. The constructor is set with **OSSL_ENCODER_CTX_set_construct()**.

OSSL_ENCODER_INSTANCE is an opaque structure that contains data about the encoder that is going to be used, and that may be useful for the constructor. There are some functions to extract data from this type, described in "Constructor" below.

Functions

OSSL_ENCODER_CTX_new() creates a **OSSL_ENCODER_CTX**.

OSSL_ENCODER_settable_ctx_params() returns an **OSSL_PARAM**(3) array of parameter descriptors.

OSSL_ENCODER_CTX_set_params() attempts to set parameters specified with an **OSSL_PARAM**(3)

array *params*. Parameters that the implementation doesn't recognise should be ignored.

OSSL_ENCODER_CTX_free() frees the given context *ctx*.

OSSL_ENCODER_CTX_add_encoder() populates the **OSSL_ENCODER_CTX** *ctx* with a encoder, to be used to encode an input object.

OSSL_ENCODER_CTX_add_extra() finds encoders that further encodes output from already added encoders, and adds them as well. This is used to build encoder chains.

OSSL_ENCODER_CTX_set_output_type() sets the ending output type. This must be specified, and determines if a complete encoder chain is available.

OSSL_ENCODER_CTX_set_output_structure() sets the desired output structure. This may be used to determines what encoder implementations may be used. Depending on the type of object being encoded, the output structure may not be relevant.

OSSL_ENCODER_CTX_get_num_encoders() gets the number of encoders currently added to the context *ctx*.

OSSL_ENCODER_CTX_set_construct() sets the constructor *construct*.

OSSL_ENCODER_CTX_set_construct_data() sets the constructor data that is passed to the constructor every time it's called.

OSSL_ENCODER_CTX_set_cleanup() sets the constructor data *cleanup* function. This is called by **OSSL_ENCODER_CTX_free**(3).

Constructor

A **OSSL_ENCODER_CONSTRUCT** gets the following arguments:

encoder_inst
: The **OSSL_ENCODER_INSTANCE** for the encoder from which the constructor gets its data.

construct_data
: The pointer that was set with **OSSL_ENCODE_CTX_set_construct_data()**.

The constructor is expected to return a valid (non-NULL) pointer to a provider-native object that can be used as first input of an encoding chain, or NULL to indicate that an error has occurred.

These utility functions may be used by a constructor:

OSSL_ENCODER_INSTANCE_get_encoder() can be used to get the encoder implementation of the encoder instance *encoder_inst*.

OSSL_ENCODER_INSTANCE_get_encoder_ctx() can be used to get the encoder implementation's provider context of the encoder instance *encoder_inst*.

OSSL_ENCODER_INSTANCE_get_output_type() can be used to get the output type for the encoder implementation of the encoder instance *encoder_inst*. This will never be NULL.

OSSL_ENCODER_INSTANCE_get_output_structure() can be used to get the output structure for the encoder implementation of the encoder instance *encoder_inst*. This may be NULL.

RETURN VALUES

OSSL_ENCODER_CTX_new() returns a pointer to a **OSSL_ENCODER_CTX**, or NULL if the context structure couldn't be allocated.

OSSL_ENCODER_settable_ctx_params() returns an **OSSL_PARAM**(3) array, or NULL if none is available.

OSSL_ENCODER_CTX_set_params() returns 1 if all recognised parameters were valid, or 0 if one of them was invalid or caused some other failure in the implementation.

OSSL_ENCODER_CTX_add_encoder(), **OSSL_ENCODER_CTX_add_extra()**, **OSSL_ENCODER_CTX_set_construct()**, **OSSL_ENCODER_CTX_set_construct_data()** and **OSSL_ENCODER_CTX_set_cleanup()** return 1 on success, or 0 on failure.

OSSL_ENCODER_CTX_get_num_encoders() returns the current number of encoders. It returns 0 if *ctx* is NULL.

OSSL_ENCODER_INSTANCE_get_encoder() returns an **OSSL_ENCODER** pointer on success, or NULL on failure.

OSSL_ENCODER_INSTANCE_get_encoder_ctx() returns a provider context pointer on success, or NULL on failure.

OSSL_ENCODER_INSTANCE_get_output_type() returns a string with the name of the input type, if relevant. NULL is a valid returned value.

OSSL_ENCODER_INSTANCE_get_output_type() returns a string with the name of the output type.

OSSL_ENCODER_INSTANCE_get_output_structure() returns a string with the name of the output structure.

SEE ALSO

provider(7), **OSSL_ENCODER**(3)

HISTORY

The functions described here were added in OpenSSL 3.0.

COPYRIGHT