NAME

OSSL_DECODER_CTX, OSSL_DECODER_CTX_new, OSSL_DECODER_settable_ctx_params, OSSL_DECODER_CTX_set_params, OSSL_DECODER_CTX_free, OSSL_DECODER_CTX_set_selection, OSSL_DECODER_CTX_set_input_type, OSSL_DECODER_CTX_set_input_structure, OSSL_DECODER_CTX_add_decoder, OSSL_DECODER_CTX_add_extra, OSSL_DECODER_CTX_get_num_decoders, OSSL_DECODER_INSTANCE, OSSL_DECODER_CONSTRUCT, OSSL_DECODER_CLEANUP, OSSL_DECODER_CTX_set_construct, OSSL_DECODER_CTX_set_construct_data, OSSL_DECODER_CTX_set_cleanup, OSSL_DECODER_CTX_get_construct, OSSL_DECODER_CTX_get_construct_data, OSSL_DECODER_CTX_get_cleanup, OSSL_DECODER_export, OSSL_DECODER_INSTANCE_get_decoder, OSSL_DECODER_INSTANCE_get_decoder_ctx, OSSL_DECODER_INSTANCE_get_input_type, OSSL_DECODER_INSTANCE_get_input_structure - Decoder context routines

SYNOPSIS

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

typedef struct ossl_decoder_ctx_st OSSL_DECODER_CTX;

OSSL_DECODER_CTX *OSSL_DECODER_CTX_new(void);
const OSSL_PARAM *OSSL_DECODER_settable_ctx_params(OSSL_DECODER *decoder);
int OSSL_DECODER_CTX_set_params(OSSL_DECODER_CTX *ctx,
                                const OSSL_PARAM params[]);
void OSSL_DECODER_CTX_free(OSSL_DECODER_CTX *ctx);

int OSSL_DECODER_CTX_set_selection(OSSL_DECODER_CTX *ctx, int selection);
int OSSL_DECODER_CTX_set_input_type(OSSL_DECODER_CTX *ctx,
                                    const char *input_type);
int OSSL_DECODER_CTX_set_input_structure(OSSL_DECODER_CTX *ctx,
                                         const char *input_structure);
int OSSL_DECODER_CTX_add_decoder(OSSL_DECODER_CTX *ctx, OSSL_DECODER *decoder);
int OSSL_DECODER_CTX_add_extra(OSSL_DECODER_CTX *ctx,
                               OSSL_LIB_CTX *libctx,
                               const char *propq);
int OSSL_DECODER_CTX_get_num_decoders(OSSL_DECODER_CTX *ctx);

typedef struct ossl_decoder_instance_st OSSL_DECODER_INSTANCE;
OSSL_DECODER *
OSSL_DECODER_INSTANCE_get_decoder(OSSL_DECODER_INSTANCE *decoder_inst);
void *
```

```
OSSL_DECODER_INSTANCE_get_decoder_ctx(OSSL_DECODER_INSTANCE *decoder_inst);
const char *
OSSL_DECODER_INSTANCE_get_input_type(OSSL_DECODER_INSTANCE *decoder_inst);
OSSL_DECODER_INSTANCE_get_input_structure(OSSL_DECODER_INSTANCE *decoder_inst,
                                          int *was_set);

typedef int OSSL_DECODER_CONSTRUCT(OSSL_DECODER_INSTANCE *decoder_inst,
                                   const OSSL_PARAM *object,
                                   void *construct_data);
typedef void OSSL_DECODER_CLEANUP(void *construct_data);

int OSSL_DECODER_CTX_set_construct(OSSL_DECODER_CTX *ctx,
                                   OSSL_DECODER_CONSTRUCT *construct);
int OSSL_DECODER_CTX_set_construct_data(OSSL_DECODER_CTX *ctx,
                                        void *construct_data);
int OSSL_DECODER_CTX_set_cleanup(OSSL_DECODER_CTX *ctx,
                                 OSSL_DECODER_CLEANUP *cleanup);
OSSL_DECODER_CONSTRUCT *OSSL_DECODER_CTX_get_construct(OSSL_DECODER_CTX *ctx);
void *OSSL_DECODER_CTX_get_construct_data(OSSL_DECODER_CTX *ctx);
OSSL_DECODER_CLEANUP *OSSL_DECODER_CTX_get_cleanup(OSSL_DECODER_CTX *ctx);

int OSSL_DECODER_export(OSSL_DECODER_INSTANCE *decoder_inst,
                        void *reference, size_t reference_sz,
                        OSSL_CALLBACK *export_cb, void *export_cbarg);
```

DESCRIPTION

The **OSSL_DECODER_CTX** holds data about multiple decoders, as needed to figure out what the input data is and to attempt to unpack it into one of several possible related results. This also includes chaining decoders, so the output from one can become the input for another. This allows having generic format decoders such as PEM to DER, as well as more specialized decoders like DER to RSA.

The chains may be limited by specifying an input type, which is considered a starting point. This is both considered by **OSSL_DECODER_CTX_add_extra()**, which will stop adding one more decoder implementations when it has already added those that take the specified input type, and functions like **OSSL_DECODER_from_bio**(3), which will only start the decoding process with the decoder implementations that take that input type. For example, if the input type is set to "DER", a PEM to DER decoder will be ignored.

The input type can also be NULL, which means that the caller doesn’t know what type of input they have. In this case, **OSSL_DECODER_from_bio()** will simply try with one decoder implementation

after the other, and thereby discover what kind of input the caller gave it.

For every decoding done, even an intermediary one, a constructor provided by the caller is called to attempt to construct an appropriate type / structure that the caller knows how to handle from the current decoding result. The constructor is set with **OSSL_DECODER_CTX_set_construct()**.

OSSL_DECODER_INSTANCE is an opaque structure that contains data about the decoder that was just used, and that may be useful for the constructor. There are some functions to extract data from this type, described further down.

Functions

OSSL_DECODER_CTX_new() creates a new empty **OSSL_DECODER_CTX**.

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

OSSL_DECODER_CTX_set_params() attempts to set parameters specified with an **OSSL_PARAM**(3) array *params*. These parameters are passed to all decoders that have been added to the *ctx* so far. Parameters that an implementation doesn't recognise should be ignored by it.

OSSL_DECODER_CTX_free() frees the given context *ctx*.

OSSL_DECODER_CTX_add_decoder() populates the **OSSL_DECODER_CTX** *ctx* with a decoder, to be used to attempt to decode some encoded input.

OSSL_DECODER_CTX_add_extra() finds decoders that generate input for already added decoders, and adds them as well. This is used to build decoder chains.

OSSL_DECODER_CTX_set_input_type() sets the starting input type. This limits the decoder chains to be considered, as explained in the general description above.

OSSL_DECODER_CTX_set_input_structure() sets the name of the structure that the input is expected to have. This may be used to determines what decoder implementations may be used. NULL is a valid input structure, when it's not relevant, or when the decoder implementations are expected to figure it out.

OSSL_DECODER_CTX_get_num_decoders() gets the number of decoders currently added to the context *ctx*.

OSSL_DECODER_CTX_set_construct() sets the constructor *construct*.

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

OSSL_DECODER_CTX_set_cleanup() sets the constructor data *cleanup* function. This is called by **OSSL_DECODER_CTX_free**(3).

OSSL_DECODER_CTX_get_construct(), **OSSL_DECODER_CTX_get_construct_data()** and **OSSL_DECODER_CTX_get_cleanup()** return the values that have been set by **OSSL_DECODER_CTX_set_construct()**, **OSSL_DECODER_CTX_set_construct_data()** and **OSSL_DECODER_CTX_set_cleanup()** respectively.

OSSL_DECODER_export() is a fallback function for constructors that cannot use the data they get directly for diverse reasons. It takes the same decode instance *decoder_inst* that the constructor got and an object *reference*, unpacks the object which it refers to, and exports it by creating an **OSSL_PARAM**(3) array that it then passes to *export_cb*, along with *export_arg*.

Constructor

A **OSSL_DECODER_CONSTRUCT** gets the following arguments:

decoder_inst
: The **OSSL_DECODER_INSTANCE** for the decoder from which the constructor gets its data.

object
: A provider-native object abstraction produced by the decoder. Further information on the provider-native object abstraction can be found in **provider-object**(7).

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

The constructor is expected to return 1 when the data it receives can be constructed, otherwise 0.

These utility functions may be used by a constructor:

OSSL_DECODER_INSTANCE_get_decoder() can be used to get the decoder implementation from a decoder instance *decoder_inst*.

OSSL_DECODER_INSTANCE_get_decoder_ctx() can be used to get the decoder implementation's provider context from a decoder instance *decoder_inst*.

OSSL_DECODER_INSTANCE_get_input_type() can be used to get the decoder implementation's

input type from a decoder instance *decoder_inst*.

OSSL_DECODER_INSTANCE_get_input_structure() can be used to get the input structure for the decoder implementation from a decoder instance *decoder_inst*. This may be NULL.

RETURN VALUES

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

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

OSSL_DECODER_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_DECODER_CTX_add_decoder(), **OSSL_DECODER_CTX_add_extra()**, **OSSL_DECODER_CTX_set_construct()**, **OSSL_DECODER_CTX_set_construct_data()** and **OSSL_DECODER_CTX_set_cleanup()** return 1 on success, or 0 on failure.

OSSL_DECODER_CTX_get_construct(), **OSSL_DECODER_CTX_get_construct_data()** and **OSSL_DECODER_CTX_get_cleanup()** return the current pointers to the constructor, the constructor data and the cleanup functions, respectively.

OSSL_DECODER_CTX_num_decoders() returns the current number of decoders. It returns 0 if *ctx* is NULL.

OSSL_DECODER_export() returns 1 on success, or 0 on failure.

OSSL_DECODER_INSTANCE_decoder() returns an **OSSL_DECODER** pointer on success, or NULL on failure.

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

SEE ALSO

provider(7), **OSSL_DECODER**(3), **OSSL_DECODER_from_bio**(3)

HISTORY

The functions described here were added in OpenSSL 3.0.

COPYRIGHT