NAME

provider-storemgmt - The OSSL_STORE library <-> provider functions

SYNOPSIS

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

/*
 * None of these are actual functions, but are displayed like this for
 * the function signatures for functions that are offered as function
 * pointers in OSSL_DISPATCH arrays.
 */

void *OSSL_FUNC_store_open(void *provctx, const char *uri);
void *OSSL_FUNC_store_attach(void *provctx, OSSL_CORE_BIO *bio);
const OSSL_PARAM *store_settable_ctx_params(void *provctx);
int OSSL_FUNC_store_set_ctx_params(void *loaderctx, const OSSL_PARAM[]);
int OSSL_FUNC_store_load(void *loaderctx,
                         OSSL_CALLBACK *object_cb, void *object_cbarg,
                         OSSL_PASSPHRASE_CALLBACK *pw_cb, void *pw_cbarg);
int OSSL_FUNC_store_eof(void *loaderctx);
int OSSL_FUNC_store_close(void *loaderctx);

int OSSL_FUNC_store_export_object
    (void *loaderctx, const void *objref, size_t objref_sz,
     OSSL_CALLBACK *export_cb, void *export_cbarg);
```

DESCRIPTION

The STORE operation is the provider side of the **ossl_store**(7) API.

The primary responsibility of the STORE operation is to load all sorts of objects from a container indicated by URI. These objects are given to the OpenSSL library in provider-native object abstraction form (see **provider-object**(7)). The OpenSSL library is then responsible for passing on that abstraction to suitable provided functions.

Examples of functions that the OpenSSL library can pass the abstraction to include **OSSL_FUNC_keymgmt_load()** (**provider-keymgmt**(7)), **OSSL_FUNC_store_export_object()** (which exports the object in parameterized form).

All "functions" mentioned here are passed as function pointers between *libcrypto* and the provider in **OSSL_DISPATCH**(3) arrays via **OSSL_ALGORITHM**(3) arrays that are returned by the provider's

provider_query_operation() function (see "Provider Functions" in **provider-base**(7)).

All these "functions" have a corresponding function type definition named **OSSL_FUNC_{name}_fn**, and a helper function to retrieve the function pointer from a **OSSL_DISPATCH**(3) element named **OSSL_get_{name}**. For example, the "function" **OSSL_FUNC_store_attach()** has these:

```
typedef void *(OSSL_FUNC_store_attach_fn)(void *provctx,
                                          OSSL_CORE_BIO * bio);
static ossl_inline OSSL_FUNC_store_attach_fn
    OSSL_FUNC_store_attach(const OSSL_DISPATCH *opf);
```

OSSL_DISPATCH(3) arrays are indexed by numbers that are provided as macros in **openssl-core_dispatch.h**(7), as follows:

```
OSSL_FUNC_store_open              OSSL_FUNC_STORE_OPEN
OSSL_FUNC_store_attach            OSSL_FUNC_STORE_ATTACH
OSSL_FUNC_store_settable_ctx_params  OSSL_FUNC_STORE_SETTABLE_CTX_PARAMS
OSSL_FUNC_store_set_ctx_params      OSSL_FUNC_STORE_SET_CTX_PARAMS
OSSL_FUNC_store_load              OSSL_FUNC_STORE_LOAD
OSSL_FUNC_store_eof               OSSL_FUNC_STORE_EOF
OSSL_FUNC_store_close             OSSL_FUNC_STORE_CLOSE
OSSL_FUNC_store_export_object       OSSL_FUNC_STORE_EXPORT_OBJECT
```

Functions

OSSL_FUNC_store_open() should create a provider side context with data based on the input *uri*. The implementation is entirely responsible for the interpretation of the URI.

OSSL_FUNC_store_attach() should create a provider side context with the core **BIO** *bio* attached. This is an alternative to using a URI to find storage, supporting **OSSL_STORE_attach**(3).

OSSL_FUNC_store_settable_ctx_params() should return a constant array of descriptor **OSSL_PARAM**(3), for parameters that **OSSL_FUNC_store_set_ctx_params()** can handle.

OSSL_FUNC_store_set_ctx_params() should set additional parameters, such as what kind of data to expect, search criteria, and so on. More on those below, in "Load Parameters". Whether unrecognised parameters are an error or simply ignored is at the implementation's discretion. Passing NULL for *params* should return true.

OSSL_FUNC_store_load() loads the next object from the URI opened by **OSSL_FUNC_store_open()**, creates an object abstraction for it (see **provider-object**(7)), and calls *object_cb* with it as well as

object_cbarg. *object_cb* will then interpret the object abstraction and do what it can to wrap it or decode it into an OpenSSL structure. In case a passphrase needs to be prompted to unlock an object, *pw_cb* should be called.

OSSL_FUNC_store_eof() indicates if the end of the set of objects from the URI has been reached. When that happens, there's no point trying to do any further loading.

OSSL_FUNC_store_close() frees the provider side context *ctx*.

When a provider-native object is created by a store manager it would be unsuitable for direct use with a foreign provider. The export function allows for exporting the object to that foreign provider if the foreign provider supports the type of the object and provides an import function.

OSSL_FUNC_store_export_object() should export the object of size *objref_sz* referenced by *objref* as an **OSSL_PARAM**(3) array and pass that to the *export_cb* as well as the given *export_cbarg*.

Load Parameters

"expect" (**OSSL_STORE_PARAM_EXPECT**) <integer>

Is a hint of what type of data the OpenSSL library expects to get. This is only useful for optimization, as the library will check that the object types match the expectation too.

The number that can be given through this parameter is found in <*openssl/store.h*>, with the macros having names starting with "OSSL_STORE_INFO_". These are further described in "SUPPORTED OBJECTS" in **OSSL_STORE_INFO**(3).

"subject" (**OSSL_STORE_PARAM_SUBJECT**) <octet string>

Indicates that the caller wants to search for an object with the given subject associated. This can be used to select specific certificates by subject.

The contents of the octet string is expected to be in DER form.

"issuer" (**OSSL_STORE_PARAM_ISSUER**) <octet string>

Indicates that the caller wants to search for an object with the given issuer associated. This can be used to select specific certificates by issuer.

The contents of the octet string is expected to be in DER form.

"serial" (**OSSL_STORE_PARAM_SERIAL**) <integer>

Indicates that the caller wants to search for an object with the given serial number associated.

"digest" (**OSSL_STORE_PARAM_DIGEST**) <UTF8 string>
"fingerprint" (**OSSL_STORE_PARAM_FINGERPRINT**) <octet string>
Indicates that the caller wants to search for an object with the given fingerprint, computed with the given digest.

"alias" (**OSSL_STORE_PARAM_ALIAS**) <UTF8 string>
Indicates that the caller wants to search for an object with the given alias (some call it a "friendly name").

"properties" (**OSSL_STORE_PARAM_PROPERTIES) <utf8 string**
Property string to use when querying for algorithms such as the **OSSL_DECODER** decoder implementations.

"input-type" (**OSSL_STORE_PARAM_INPUT_TYPE) <utf8 string**
Type of the input format as a hint to use when decoding the objects in the store.

Several of these search criteria may be combined. For example, to search for a certificate by issuer+serial, both the "issuer" and the "serial" parameters will be given.

SEE ALSO

provider(7)

HISTORY

The STORE interface was introduced in OpenSSL 3.0.

COPYRIGHT