NAME

OSSL_PROVIDER_set_default_search_path, OSSL_PROVIDER, OSSL_PROVIDER_load, OSSL_PROVIDER_try_load, OSSL_PROVIDER_unload, OSSL_PROVIDER_available, OSSL_PROVIDER_do_all, OSSL_PROVIDER_gettable_params, OSSL_PROVIDER_get_params, OSSL_PROVIDER_query_operation, OSSL_PROVIDER_unquery_operation, OSSL_PROVIDER_get0_provider_ctx, OSSL_PROVIDER_get0_dispatch, OSSL_PROVIDER_add_builtin, OSSL_PROVIDER_get0_name, OSSL_PROVIDER_get_capabilities, OSSL_PROVIDER_self_test - provider routines

SYNOPSIS

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

typedef struct ossl_provider_st OSSL_PROVIDER;

int OSSL_PROVIDER_set_default_search_path(OSSL_LIB_CTX *libctx,
                                         const char *path);

OSSL_PROVIDER *OSSL_PROVIDER_load(OSSL_LIB_CTX *libctx, const char *name);
OSSL_PROVIDER *OSSL_PROVIDER_try_load(OSSL_LIB_CTX *libctx, const char *name,
                                      int retain_fallbacks);
int OSSL_PROVIDER_unload(OSSL_PROVIDER *prov);
int OSSL_PROVIDER_available(OSSL_LIB_CTX *libctx, const char *name);
int OSSL_PROVIDER_do_all(OSSL_LIB_CTX *ctx,
                         int (*cb)(OSSL_PROVIDER *provider, void *cbdata),
                         void *cbdata);

const OSSL_PARAM *OSSL_PROVIDER_gettable_params(OSSL_PROVIDER *prov);
int OSSL_PROVIDER_get_params(OSSL_PROVIDER *prov, OSSL_PARAM params[]);

const OSSL_ALGORITHM *OSSL_PROVIDER_query_operation(const OSSL_PROVIDER *prov,
                                                    int operation_id,
                                                    int *no_cache);
void OSSL_PROVIDER_unquery_operation(const OSSL_PROVIDER *prov,
                                     int operation_id,
                                     const OSSL_ALGORITHM *algs);
void *OSSL_PROVIDER_get0_provider_ctx(const OSSL_PROVIDER *prov);
const OSSL_DISPATCH *OSSL_PROVIDER_get0_dispatch(const OSSL_PROVIDER *prov);

int OSSL_PROVIDER_add_builtin(OSSL_LIB_CTX *libctx, const char *name,
                              ossl_provider_init_fn *init_fn);
```

```
const char *OSSL_PROVIDER_get0_name(const OSSL_PROVIDER *prov);

int OSSL_PROVIDER_get_capabilities(const OSSL_PROVIDER *prov,
                                   const char *capability,
                                   OSSL_CALLBACK *cb,
                                   void *arg);
int OSSL_PROVIDER_self_test(const OSSL_PROVIDER *prov);
```

DESCRIPTION

OSSL_PROVIDER is a type that holds internal information about implementation providers (see **provider**(7) for information on what a provider is). A provider can be built in to the application or the OpenSSL libraries, or can be a loadable module. The functions described here handle both forms.

Some of these functions operate within a library context, please see **OSSL_LIB_CTX**(3) for further details.

Functions

OSSL_PROVIDER_set_default_search_path() specifies the default search *path* that is to be used for looking for providers in the specified *libctx*. If left unspecified, an environment variable and a fall back default value will be used instead.

OSSL_PROVIDER_add_builtin() is used to add a built in provider to **OSSL_PROVIDER** store in the given library context, by associating a provider name with a provider initialization function. This name can then be used with **OSSL_PROVIDER_load()**.

OSSL_PROVIDER_load() loads and initializes a provider. This may simply initialize a provider that was previously added with **OSSL_PROVIDER_add_builtin()** and run its given initialization function, or load a provider module with the given name and run its provider entry point, "OSSL_provider_init". The *name* can be a path to a provider module, in that case the provider name as returned by **OSSL_PROVIDER_get0_name()** will be the path. Interpretation of relative paths is platform dependent and they are relative to the configured "MODULESDIR" directory or the path set in the environment variable OPENSSL_MODULES if set.

OSSL_PROVIDER_try_load() functions like **OSSL_PROVIDER_load()**, except that it does not disable the fallback providers if the provider cannot be loaded and initialized or if *retain_fallbacks* is nonzero. If the provider loads successfully and *retain_fallbacks* is zero, the fallback providers are disabled.

OSSL_PROVIDER_unload() unloads the given provider. For a provider added with **OSSL_PROVIDER_add_builtin()**, this simply runs its teardown function.

OSSL_PROVIDER_available() checks if a named provider is available for use.

OSSL_PROVIDER_do_all() iterates over all loaded providers, calling *cb* for each one, with the current provider in *provider* and the *cbdata* that comes from the caller. If no other provider has been loaded before calling this function, the default provider is still available as fallback. See **OSSL_PROVIDER-default**(7) for more information on this fallback behaviour.

OSSL_PROVIDER_gettable_params() is used to get a provider parameter descriptor set as a constant **OSSL_PARAM**(3) array.

OSSL_PROVIDER_get_params() is used to get provider parameter values. The caller must prepare the **OSSL_PARAM**(3) array before calling this function, and the variables acting as buffers for this parameter array should be filled with data when it returns successfully.

OSSL_PROVIDER_self_test() is used to run a provider's self tests on demand. If the self tests fail then the provider will fail to provide any further services and algorithms. **OSSL_SELF_TEST_set_callback**(3) may be called beforehand in order to display diagnostics for the running self tests.

OSSL_PROVIDER_query_operation() calls the provider's *query_operation* function (see **provider**(7)), if the provider has one. It returns an array of *OSSL_ALGORITHM* for the given *operation_id* terminated by an all NULL OSSL_ALGORITHM entry. This is considered a low-level function that most applications should not need to call.

OSSL_PROVIDER_unquery_operation() calls the provider's *unquery_operation* function (see **provider**(7)), if the provider has one. This is considered a low-level function that most applications should not need to call.

OSSL_PROVIDER_get0_provider_ctx() returns the provider context for the given provider. The provider context is an opaque handle set by the provider itself and is passed back to the provider by libcrypto in various function calls.

OSSL_PROVIDER_get0_dispatch() returns the provider's dispatch table as it was returned in the *out* parameter from the provider's init function. See **provider-base**(7).

If it is permissible to cache references to this array then **no_store* is set to 0 or 1 otherwise. If the array is not cacheable then it is assumed to have a short lifetime.

OSSL_PROVIDER_get0_name() returns the name of the given provider.

OSSL_PROVIDER_get_capabilities() provides information about the capabilities supported by the provider specified in *prov* with the capability name *capability*. For each capability of that name supported by the provider it will call the callback *cb* and supply a set of **OSSL_PARAM**(3)s describing the capability. It will also pass back the argument *arg*. For more details about capabilities and what they can be used for please see "CAPABILTIIES" in **provider-base**(7).

RETURN VALUES

OSSL_PROVIDER_set_default_search_path(), **OSSL_PROVIDER_add()**, **OSSL_PROVIDER_unload()**, **OSSL_PROVIDER_get_params()** and **OSSL_PROVIDER_get_capabilities()** return 1 on success, or 0 on error.

OSSL_PROVIDER_load() and **OSSL_PROVIDER_try_load()** return a pointer to a provider object on success, or NULL on error.

OSSL_PROVIDER_do_all() returns 1 if the callback *cb* returns 1 for every provider it is called with, or 0 if any provider callback invocation returns 0; callback processing stops at the first callback invocation on a provider that returns 0.

OSSL_PROVIDER_available() returns 1 if the named provider is available, otherwise 0.

OSSL_PROVIDER_gettable_params() returns a pointer to an array of constant **OSSL_PARAM**(3), or NULL if none is provided.

OSSL_PROVIDER_get_params() and returns 1 on success, or 0 on error.

OSSL_PROVIDER_query_operation() returns an array of OSSL_ALGORITHM or NULL on error.

OSSL_PROVIDER_self_test() returns 1 if the self tests pass, or 0 on error.

EXAMPLES

This demonstrates how to load the provider module "foo" and ask for its build information.

```
#include <openssl/params.h>
#include <openssl/provider.h>
#include <openssl/err.h>

OSSL_PROVIDER *prov = NULL;
const char *build = NULL;
OSSL_PARAM request[] = {
   { "buildinfo", OSSL_PARAM_UTF8_PTR, &build, 0, 0 },
```

```
    { NULL, 0, NULL, 0, 0 }
};

if ((prov = OSSL_PROVIDER_load(NULL, "foo")) != NULL
    && OSSL_PROVIDER_get_params(prov, request))
    printf("Provider ’foo’ buildinfo: %s\n", build);
else
    ERR_print_errors_fp(stderr);
```

SEE ALSO

openssl-core.h(7), **OSSL_LIB_CTX**(3), **provider**(7)

HISTORY

The type and functions described here were added in OpenSSL 3.0.

COPYRIGHT