



Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'EVP_MD_CTX_get0_name.3oss1' command

`$ man EVP_MD_CTX_get0_name.3oss1`

EVP_DIGESTINIT(3oss1) OpenSSL EVP_DIGESTINIT(3oss1)

NAME

EVP_MD_fetch, EVP_MD_up_ref, EVP_MD_free, EVP_MD_get_params,
EVP_MD_gettable_params, EVP_MD_CTX_new, EVP_MD_CTX_reset,
EVP_MD_CTX_free, EVP_MD_CTX_copy, EVP_MD_CTX_copy_ex, EVP_MD_CTX_ctrl,
EVP_MD_CTX_set_params, EVP_MD_CTX_get_params,
EVP_MD_settable_ctx_params, EVP_MD_gettable_ctx_params,
EVP_MD_CTX_settable_params, EVP_MD_CTX_gettable_params,
EVP_MD_CTX_set_flags, EVP_MD_CTX_clear_flags, EVP_MD_CTX_test_flags,
EVP_Q_digest, EVP_Digest, EVP_DigestInit_ex2, EVP_DigestInit_ex,
EVP_DigestInit, EVP_DigestUpdate, EVP_DigestFinal_ex,
EVP_DigestFinalXOF, EVP_DigestFinal, EVP_MD_is_a, EVP_MD_get0_name,
EVP_MD_get0_description, EVP_MD_names_do_all, EVP_MD_get0_provider,
EVP_MD_get_type, EVP_MD_get_pkey_type, EVP_MD_get_size,
EVP_MD_get_block_size, EVP_MD_get_flags, EVP_MD_CTX_get0_name,
EVP_MD_CTX_md, EVP_MD_CTX_get0_md, EVP_MD_CTX_get1_md,
EVP_MD_CTX_get_type, EVP_MD_CTX_get_size, EVP_MD_CTX_get_block_size,
EVP_MD_CTX_get0_md_data, EVP_MD_CTX_update_fn,
EVP_MD_CTX_set_update_fn, EVP_md_null, EVP_get_digestbyname,
EVP_get_digestbynid, EVP_get_digestbyobj, EVP_MD_CTX_get_pkey_ctx,
EVP_MD_CTX_set_pkey_ctx, EVP_MD_do_all_provided, EVP_MD_type,
EVP_MD_nid, EVP_MD_name, EVP_MD_pkey_type, EVP_MD_size,
EVP_MD_block_size, EVP_MD_flags, EVP_MD_CTX_size,

EVP_MD_CTX_block_size, EVP_MD_CTX_type, EVP_MD_CTX_pkey_ctx,
EVP_MD_CTX_md_data - EVP digest routines

SYNOPSIS

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

EVP_MD *EVP_MD_fetch(OSSL_LIB_CTX *ctx, const char *algorithm,
                    const char *properties);

int EVP_MD_up_ref(EVP_MD *md);
void EVP_MD_free(EVP_MD *md);

int EVP_MD_get_params(const EVP_MD *digest, OSSL_PARAM params[]);
const OSSL_PARAM *EVP_MD_gettable_params(const EVP_MD *digest);
EVP_MD_CTX *EVP_MD_CTX_new(void);
int EVP_MD_CTX_reset(EVP_MD_CTX *ctx);
void EVP_MD_CTX_free(EVP_MD_CTX *ctx);
void EVP_MD_CTX_ctrl(EVP_MD_CTX *ctx, int cmd, int p1, void* p2);
int EVP_MD_CTX_get_params(EVP_MD_CTX *ctx, OSSL_PARAM params[]);
int EVP_MD_CTX_set_params(EVP_MD_CTX *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM *EVP_MD_settable_ctx_params(const EVP_MD *md);
const OSSL_PARAM *EVP_MD_gettable_ctx_params(const EVP_MD *md);
const OSSL_PARAM *EVP_MD_CTX_settable_params(EVP_MD_CTX *ctx);
const OSSL_PARAM *EVP_MD_CTX_gettable_params(EVP_MD_CTX *ctx);
void EVP_MD_CTX_set_flags(EVP_MD_CTX *ctx, int flags);
void EVP_MD_CTX_clear_flags(EVP_MD_CTX *ctx, int flags);
int EVP_MD_CTX_test_flags(const EVP_MD_CTX *ctx, int flags);
int EVP_Q_digest(OSSL_LIB_CTX *libctx, const char *name, const char *propq,
                const void *data, size_t datalen,
                unsigned char *md, size_t *mdlen);

int EVP_Digest(const void *data, size_t count, unsigned char *md,
              unsigned int *size, const EVP_MD *type, ENGINE *impl);
int EVP_DigestInit_ex2(EVP_MD_CTX *ctx, const EVP_MD *type,
                      const OSSL_PARAM params[]);
int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);
int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s);
```

```

int EVP_DigestFinalXOF(EVP_MD_CTX *ctx, unsigned char *md, size_t len);
int EVP_MD_CTX_copy_ex(EVP_MD_CTX *out, const EVP_MD_CTX *in);
int EVP_DigestInit(EVP_MD_CTX *ctx, const EVP_MD *type);
int EVP_DigestFinal(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s);
int EVP_MD_CTX_copy(EVP_MD_CTX *out, EVP_MD_CTX *in);
const char *EVP_MD_get0_name(const EVP_MD *md);
const char *EVP_MD_get0_description(const EVP_MD *md);
int EVP_MD_is_a(const EVP_MD *md, const char *name);
int EVP_MD_names_do_all(const EVP_MD *md,
                        void (*fn)(const char *name, void *data),
                        void *data);
const OSSL_PROVIDER *EVP_MD_get0_provider(const EVP_MD *md);
int EVP_MD_get_type(const EVP_MD *md);
int EVP_MD_get_pkey_type(const EVP_MD *md);
int EVP_MD_get_size(const EVP_MD *md);
int EVP_MD_get_block_size(const EVP_MD *md);
unsigned long EVP_MD_get_flags(const EVP_MD *md);
const EVP_MD *EVP_MD_CTX_get0_md(const EVP_MD_CTX *ctx);
EVP_MD *EVP_MD_CTX_get1_md(EVP_MD_CTX *ctx);
const char *EVP_MD_CTX_get0_name(const EVP_MD_CTX *ctx);
int EVP_MD_CTX_get_size(const EVP_MD_CTX *ctx);
int EVP_MD_CTX_get_block_size(const EVP_MD_CTX *ctx);
int EVP_MD_CTX_get_type(const EVP_MD_CTX *ctx);
void *EVP_MD_CTX_get0_md_data(const EVP_MD_CTX *ctx);
const EVP_MD *EVP_md_null(void);
const EVP_MD *EVP_get_digestbyname(const char *name);
const EVP_MD *EVP_get_digestbynid(int type);
const EVP_MD *EVP_get_digestbyobj(const ASN1_OBJECT *o);
EVP_PKEY_CTX *EVP_MD_CTX_get_pkey_ctx(const EVP_MD_CTX *ctx);
void EVP_MD_CTX_set_pkey_ctx(EVP_MD_CTX *ctx, EVP_PKEY_CTX *pctx);
void EVP_MD_do_all_provided(OSSL_LIB_CTX *libctx,
                            void (*fn)(EVP_MD *mac, void *arg),
                            void *arg);

```

```

#define EVP_MD_type EVP_MD_get_type
#define EVP_MD_nid EVP_MD_get_type
#define EVP_MD_name EVP_MD_get0_name
#define EVP_MD_pkey_type EVP_MD_get_pkey_type
#define EVP_MD_size EVP_MD_get_size
#define EVP_MD_block_size EVP_MD_get_block_size
#define EVP_MD_flags EVP_MD_get_flags
#define EVP_MD_CTX_size EVP_MD_CTX_get_size
#define EVP_MD_CTX_block_size EVP_MD_CTX_get_block_size
#define EVP_MD_CTX_type EVP_MD_CTX_get_type
#define EVP_MD_CTX_pkey_ctx EVP_MD_CTX_get_pkey_ctx
#define EVP_MD_CTX_md_data EVP_MD_CTX_get0_md_data

```

The following functions have been deprecated since OpenSSL 3.0, and can be hidden entirely by defining `OPENSSL_API_COMPAT` with a suitable version value, see `openssl_user_macros(7)`:

```

const EVP_MD *EVP_MD_CTX_md(const EVP_MD_CTX *ctx);
int (*EVP_MD_CTX_update_fn(EVP_MD_CTX *ctx))(EVP_MD_CTX *ctx,
                                              const void *data, size_t count);
void EVP_MD_CTX_set_update_fn(EVP_MD_CTX *ctx,
                              int (*update)(EVP_MD_CTX *ctx,
                                              const void *data, size_t count));

```

DESCRIPTION

The EVP digest routines are a high-level interface to message digests, and should be used instead of the digest-specific functions.

The `EVP_MD` type is a structure for digest method implementation.

`EVP_MD_fetch()`

Fetches the digest implementation for the given algorithm from any provider offering it, within the criteria given by the properties.

See "ALGORITHM FETCHING" in `crypto(7)` for further information.

The returned value must eventually be freed with `EVP_MD_free()`.

Fetches `EVP_MD` structures are reference counted.

`EVP_MD_up_ref()`

Increments the reference count for an `EVP_MD` structure.

EVP_MD_free()

Decrements the reference count for the fetched EVP_MD structure.

If the reference count drops to 0 then the structure is freed.

EVP_MD_CTX_new()

Allocates and returns a digest context.

EVP_MD_CTX_reset()

Resets the digest context ctx. This can be used to reuse an already existing context.

EVP_MD_CTX_free()

Cleans up digest context ctx and frees up the space allocated to it.

EVP_MD_CTX_ctrl()

This is a legacy method. EVP_MD_CTX_set_params() and EVP_MD_CTX_get_params() is the mechanism that should be used to set and get parameters that are used by providers.

Performs digest-specific control actions on context ctx. The control command is indicated in cmd and any additional arguments in p1 and p2. EVP_MD_CTX_ctrl() must be called after EVP_DigestInit_ex2(). Other restrictions may apply depending on the control type and digest implementation.

If this function happens to be used with a fetched EVP_MD, it will translate the controls that are known to OpenSSL into OSSL_PARAM(3) parameters with keys defined by OpenSSL and call EVP_MD_CTX_get_params() or EVP_MD_CTX_set_params() as is appropriate for each control command.

See "CONTROLS" below for more information, including what translations are being done.

EVP_MD_get_params()

Retrieves the requested list of params from a MD md. See "PARAMETERS" below for more information.

EVP_MD_CTX_get_params()

Retrieves the requested list of params from a MD context ctx. See "PARAMETERS" below for more information.

`EVP_MD_CTX_set_params()`

Sets the list of params into a MD context `ctx`. See "PARAMETERS" below for more information.

`EVP_MD_gettable_params()`

Get a constant `OSSL_PARAM` array that describes the retrievable parameters that can be used with `EVP_MD_get_params()`. See `OSSL_PARAM(3)` for the use of `OSSL_PARAM` as a parameter descriptor.

`EVP_MD_gettable_ctx_params()`, `EVP_MD_CTX_gettable_params()`

Get a constant `OSSL_PARAM` array that describes the retrievable parameters that can be used with `EVP_MD_CTX_get_params()`.

`EVP_MD_gettable_ctx_params()` returns the parameters that can be retrieved from the algorithm, whereas `EVP_MD_CTX_gettable_params()` returns the parameters that can be retrieved in the context's current state. See `OSSL_PARAM(3)` for the use of `OSSL_PARAM` as a parameter descriptor.

`EVP_MD_settable_ctx_params()`, `EVP_MD_CTX_settable_params()`

Get a constant `OSSL_PARAM` array that describes the settable parameters that can be used with `EVP_MD_CTX_set_params()`.

`EVP_MD_settable_ctx_params()` returns the parameters that can be set from the algorithm, whereas `EVP_MD_CTX_settable_params()` returns the parameters that can be set in the context's current state. See `OSSL_PARAM(3)` for the use of `OSSL_PARAM` as a parameter descriptor.

`EVP_MD_CTX_set_flags()`, `EVP_MD_CTX_clear_flags()`,

`EVP_MD_CTX_test_flags()`

Sets, clears and tests `ctx` flags. See "FLAGS" below for more information.

`EVP_Q_digest()` is a quick one-shot digest function.

It hashes `datalen` bytes of data at `data` using the digest algorithm name, which is fetched using the optional `libctx` and `propq` parameters. The digest value is placed in `md` and its length is written at `mdlen` if the pointer is not `NULL`. At most `EVP_MAX_MD_SIZE` bytes will be written.

`EVP_Digest()`

A wrapper around the Digest Init_ex, Update and Final_ex functions. Hashes count bytes of data at data using a digest type from ENGINE impl. The digest value is placed in md and its length is written at size if the pointer is not NULL. At most EVP_MAX_MD_SIZE bytes will be written. If impl is NULL the default implementation of digest type is used.

EVP_DigestInit_ex2()

Sets up digest context ctx to use a digest type. type is typically supplied by a function such as EVP_sha1(), or a value explicitly fetched with EVP_MD_fetch().

The parameters params are set on the context after initialisation. The type parameter can be NULL if ctx has been already initialized with another EVP_DigestInit_ex() call and has not been reset with EVP_MD_CTX_reset().

EVP_DigestInit_ex()

Sets up digest context ctx to use a digest type. type is typically supplied by a function such as EVP_sha1(), or a value explicitly fetched with EVP_MD_fetch().

If impl is non-NULL, its implementation of the digest type is used if there is one, and if not, the default implementation is used.

The type parameter can be NULL if ctx has been already initialized with another EVP_DigestInit_ex() call and has not been reset with EVP_MD_CTX_reset().

EVP_DigestUpdate()

Hashes cnt bytes of data at d into the digest context ctx. This function can be called several times on the same ctx to hash additional data.

EVP_DigestFinal_ex()

Retrieves the digest value from ctx and places it in md. If the s parameter is not NULL then the number of bytes of data written (i.e. the length of the digest) will be written to the integer at s, at most EVP_MAX_MD_SIZE bytes will be written. After calling EVP_DigestFinal_ex() no additional calls to EVP_DigestUpdate() can

be made, but `EVP_DigestInit_ex2()` can be called to initialize a new digest operation.

`EVP_DigestFinalXOF()`

Interfaces to extendable-output functions, XOFs, such as SHAKE128 and SHAKE256. It retrieves the digest value from `ctx` and places it in `len`-sized `md`. After calling this function no additional calls to `EVP_DigestUpdate()` can be made, but `EVP_DigestInit_ex2()` can be called to initialize a new operation.

`EVP_MD_CTX_copy_ex()`

Can be used to copy the message digest state from `in` to `out`. This is useful if large amounts of data are to be hashed which only differ in the last few bytes.

`EVP_DigestInit()`

Behaves in the same way as `EVP_DigestInit_ex2()` except it doesn't set any parameters and calls `EVP_MD_CTX_reset()` so it cannot be used with an type of `NULL`.

`EVP_DigestFinal()`

Similar to `EVP_DigestFinal_ex()` except after computing the digest the digest context `ctx` is automatically cleaned up with `EVP_MD_CTX_reset()`.

`EVP_MD_CTX_copy()`

Similar to `EVP_MD_CTX_copy_ex()` except the destination `out` does not have to be initialized.

`EVP_MD_is_a()`

Returns 1 if `md` is an implementation of an algorithm that's identifiable with `name`, otherwise 0.

If `md` is a legacy digest (it's the return value from the likes of `EVP_sha256()` rather than the result of an `EVP_MD_fetch()`), only cipher names registered with the default library context (see `OSSL_LIB_CTX(3)`) will be considered.

`EVP_MD_get0_name()`, `EVP_MD_CTX_get0_name()`

Return the name of the given message digest. For fetched message digests with multiple names, only one of them is returned; it's

recommended to use `EVP_MD_names_do_all()` instead.

`EVP_MD_names_do_all()`

Traverses all names for the md, and calls fn with each name and data. This is only useful with fetched EVP_MDs.

`EVP_MD_get0_description()`

Returns a description of the digest, meant for display and human consumption. The description is at the discretion of the digest implementation.

`EVP_MD_get0_provider()`

Returns an `OSSL_PROVIDER` pointer to the provider that implements the given EVP_MD.

`EVP_MD_get_size()`, `EVP_MD_CTX_get_size()`

Return the size of the message digest when passed an EVP_MD or an EVP_MD_CTX structure, i.e. the size of the hash.

`EVP_MD_get_block_size()`, `EVP_MD_CTX_get_block_size()`

Return the block size of the message digest when passed an EVP_MD or an EVP_MD_CTX structure.

`EVP_MD_get_type()`, `EVP_MD_CTX_get_type()`

Return the NID of the OBJECT IDENTIFIER representing the given message digest when passed an EVP_MD structure. For example, "`EVP_MD_get_type(EVP_sha1())`" returns `NID_sha1`. This function is normally used when setting ASN1 OIDs.

`EVP_MD_CTX_get0_md_data()`

Return the digest method private data for the passed EVP_MD_CTX. The space is allocated by OpenSSL and has the size originally set with `EVP_MD_meth_set_app_datasize()`.

`EVP_MD_CTX_get0_md()`, `EVP_MD_CTX_get1_md()`

`EVP_MD_CTX_get0_md()` returns the EVP_MD structure corresponding to the passed EVP_MD_CTX. This will be the same EVP_MD object originally passed to `EVP_DigestInit_ex2()` (or other similar function) when the EVP_MD_CTX was first initialised. Note that where explicit fetch is in use (see `EVP_MD_fetch(3)`) the value returned from this function will not have its reference count

incremented and therefore it should not be used after the
EVP_MD_CTX is freed. EVP_MD_CTX_get1_md() is the same except the
ownership is passed to the caller and is from the passed
EVP_MD_CTX.

EVP_MD_CTX_set_update_fn()

Sets the update function for ctx to update. This is the function
that is called by EVP_DigestUpdate(). If not set, the update
function from the EVP_MD type specified at initialization is used.

EVP_MD_CTX_update_fn()

Returns the update function for ctx.

EVP_MD_get_flags()

Returns the md flags. Note that these are different from the
EVP_MD_CTX ones. See EVP_MD_meth_set_flags(3) for more information.

EVP_MD_get_pkey_type()

Returns the NID of the public key signing algorithm associated with
this digest. For example EVP_sha1() is associated with RSA so this
will return NID_sha1WithRSAEncryption. Since digests and signature
algorithms are no longer linked this function is only retained for
compatibility reasons.

EVP_md_null()

A "null" message digest that does nothing: i.e. the hash it returns
is of zero length.

EVP_get_digestbyname(), EVP_get_digestbynid(), EVP_get_digestbyobj()

Returns an EVP_MD structure when passed a digest name, a digest NID
or an ASN1_OBJECT structure respectively.

The EVP_get_digestbyname() function is present for backwards
compatibility with OpenSSL prior to version 3 and is different to
the EVP_MD_fetch() function since it does not attempt to "fetch" an
implementation of the cipher. Additionally, it only knows about
digests that are built-in to OpenSSL and have an associated NID.
Similarly EVP_get_digestbynid() and EVP_get_digestbyobj() also
return objects without an associated implementation.

When the digest objects returned by these functions are used (such

as in a call to `EVP_DigestInit_ex()` an implementation of the digest will be implicitly fetched from the loaded providers. This fetch could fail if no suitable implementation is available. Use `EVP_MD_fetch()` instead to explicitly fetch the algorithm and an associated implementation from a provider.

See "ALGORITHM FETCHING" in `crypto(7)` for more information about fetching.

The digest objects returned from these functions do not need to be freed with `EVP_MD_free()`.

`EVP_MD_CTX_get_pkey_ctx()`

Returns the `EVP_PKEY_CTX` assigned to `ctx`. The returned pointer should not be freed by the caller.

`EVP_MD_CTX_set_pkey_ctx()`

Assigns an `EVP_PKEY_CTX` to `EVP_MD_CTX`. This is usually used to provide a customized `EVP_PKEY_CTX` to `EVP_DigestSignInit(3)` or `EVP_DigestVerifyInit(3)`. The `pctx` passed to this function should be freed by the caller. A `NULL` `pctx` pointer is also allowed to clear the `EVP_PKEY_CTX` assigned to `ctx`. In such case, freeing the cleared `EVP_PKEY_CTX` or not depends on how the `EVP_PKEY_CTX` is created.

`EVP_MD_do_all_provided()`

Traverses all messages digests implemented by all activated providers in the given library context `libctx`, and for each of the implementations, calls the given function `fn` with the implementation method and the given `arg` as argument.

PARAMETERS

See `OSSL_PARAM(3)` for information about passing parameters.

`EVP_MD_CTX_set_params()` can be used with the following `OSSL_PARAM` keys:

"`xoflen`" (`OSSL_DIGEST_PARAM_XOFLEN`) <unsigned integer>

Sets the digest length for extendable output functions. It is used by the SHAKE algorithm and should not exceed what can be given using a `size_t`.

"`pad-type`" (`OSSL_DIGEST_PARAM_PAD_TYPE`) <unsigned integer>

Sets the padding type. It is used by the MDC2 algorithm.

EVP_MD_CTX_get_params() can be used with the following OSSL_PARAM keys:

"micalg" (OSSL_PARAM_DIGEST_KEY_MICALG) <UTF8 string>.

Gets the digest Message Integrity Check algorithm string. This is used when creating S/MIME multipart/signed messages, as specified in RFC 3851. It may be used by external engines or providers.

CONTROLS

EVP_MD_CTX_ctrl() can be used to send the following standard controls:

EVP_MD_CTRL_MICALG

Gets the digest Message Integrity Check algorithm string. This is used when creating S/MIME multipart/signed messages, as specified in RFC 3851. The string value is written to p2.

When used with a fetched EVP_MD, EVP_MD_CTX_get_params() gets called with an OSSL_PARAM(3) item with the key "micalg"

(OSSL_DIGEST_PARAM_MICALG).

EVP_MD_CTRL_XOF_LEN

This control sets the digest length for extendable output functions to p1. Sending this control directly should not be necessary, the use of EVP_DigestFinalXOF() is preferred. Currently used by SHAKE.

When used with a fetched EVP_MD, EVP_MD_CTX_get_params() gets called with an OSSL_PARAM(3) item with the key "xoflen"

(OSSL_DIGEST_PARAM_XOFLLEN).

FLAGS

EVP_MD_CTX_set_flags(), EVP_MD_CTX_clear_flags() and EVP_MD_CTX_test_flags() can be used to manipulate and test these EVP_MD_CTX flags:

EVP_MD_CTX_FLAG_ONESHOT

This flag instructs the digest to optimize for one update only, if possible.

EVP_MD_CTX_FLAG_NO_INIT

This flag instructs EVP_DigestInit() and similar not to initialise the implementation specific data.

EVP_MD_CTX_FLAG_FINALISE

Some functions such as EVP_DigestSign only finalise copies of

internal contexts so additional data can be included after the finalisation call. This is inefficient if this functionality is not required, and can be disabled with this flag.

RETURN VALUES

`EVP_MD_fetch()`

Returns a pointer to a `EVP_MD` for success or `NULL` for failure.

`EVP_MD_up_ref()`

Returns 1 for success or 0 for failure.

`EVP_Q_digest()`, `EVP_Digest()`, `EVP_DigestInit_ex2()`,

`EVP_DigestInit_ex()`, `EVP_DigestUpdate()`, `EVP_DigestFinal_ex()`,

`EVP_DigestFinalXOF()`, and `EVP_DigestFinal()`

return 1 for success and 0 for failure.

`EVP_MD_CTX_ctrl()`

Returns 1 if successful or 0 for failure.

`EVP_MD_CTX_set_params()`, `EVP_MD_CTX_get_params()`

Returns 1 if successful or 0 for failure.

`EVP_MD_CTX_settable_params()`, `EVP_MD_CTX_gettable_params()`

Return an array of constant `OSSL_PARAMS`, or `NULL` if there is none to get.

`EVP_MD_CTX_copy_ex()`

Returns 1 if successful or 0 for failure.

`EVP_MD_get_type()`, `EVP_MD_get_pkey_type()`

Returns the NID of the corresponding OBJECT IDENTIFIER or `NID_undef` if none exists.

`EVP_MD_get_size()`, `EVP_MD_get_block_size()`, `EVP_MD_CTX_get_size()`,

`EVP_MD_CTX_get_block_size()`

Returns the digest or block size in bytes.

`EVP_md_null()`

Returns a pointer to the `EVP_MD` structure of the "null" message digest.

`EVP_get_digestbyname()`, `EVP_get_digestbynid()`, `EVP_get_digestbyobj()`

Returns either an `EVP_MD` structure or `NULL` if an error occurs.

`EVP_MD_CTX_set_pkey_ctx()`

This function has no return value.

`EVP_MD_names_do_all()`

Returns 1 if the callback was called for all names. A return value of 0 means that the callback was not called for any names.

NOTES

The EVP interface to message digests should almost always be used in preference to the low-level interfaces. This is because the code then becomes transparent to the digest used and much more flexible.

New applications should use the SHA-2 (such as `EVP_sha256(3)`) or the SHA-3 digest algorithms (such as `EVP_sha3_512(3)`). The other digest algorithms are still in common use.

For most applications the `impl` parameter to `EVP_DigestInit_ex()` will be set to `NULL` to use the default digest implementation.

The functions `EVP_DigestInit()`, `EVP_DigestFinal()` and `EVP_MD_CTX_copy()` are obsolete but are retained to maintain compatibility with existing code. New applications should use `EVP_DigestInit_ex()`, `EVP_DigestFinal_ex()` and `EVP_MD_CTX_copy_ex()` because they can efficiently reuse a digest context instead of initializing and cleaning it up on each call and allow non default implementations of digests to be specified.

If digest contexts are not cleaned up after use, memory leaks will occur.

`EVP_MD_CTX_get0_name()`, `EVP_MD_CTX_get_size()`, `EVP_MD_CTX_get_block_size()`, `EVP_MD_CTX_get_type()`, `EVP_get_digestbynid()` and `EVP_get_digestbyobj()` are defined as macros. `EVP_MD_CTX_ctrl()` sends commands to message digests for additional configuration or control.

EXAMPLES

This example digests the data "Test Message\n" and "Hello World\n", using the digest name passed on the command line.

```
#include <stdio.h>
```

```
#include <string.h>
```

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

```

int main(int argc, char *argv[])
{
    EVP_MD_CTX *mdctx;
    const EVP_MD *md;
    char mess1[] = "Test Message\n";
    char mess2[] = "Hello World\n";
    unsigned char md_value[EVP_MAX_MD_SIZE];
    unsigned int md_len, i;

    if (argv[1] == NULL) {
        printf("Usage: mdtest digestname\n");
        exit(1);
    }

    md = EVP_get_digestbyname(argv[1]);
    if (md == NULL) {
        printf("Unknown message digest %s\n", argv[1]);
        exit(1);
    }

    mdctx = EVP_MD_CTX_new();
    EVP_DigestInit_ex2(mdctx, md, NULL);
    EVP_DigestUpdate(mdctx, mess1, strlen(mess1));
    EVP_DigestUpdate(mdctx, mess2, strlen(mess2));
    EVP_DigestFinal_ex(mdctx, md_value, &md_len);
    EVP_MD_CTX_free(mdctx);

    printf("Digest is: ");
    for (i = 0; i < md_len; i++)
        printf("%02x", md_value[i]);

    printf("\n");
    exit(0);
}

```

SEE ALSO

[EVP_MD_meth_new\(3\)](#), [openssl-dgst\(1\)](#), [evp\(7\)](#), [OSSL_PROVIDER\(3\)](#),
[OSSL_PARAM\(3\)](#), [property\(7\)](#), "ALGORITHM FETCHING" in [crypto\(7\)](#),
[provider-digest\(7\)](#), [life_cycle-digest\(7\)](#)

The full list of digest algorithms are provided below.

EVP_blake2b512(3), EVP_md2(3), EVP_md4(3), EVP_md5(3), EVP_mdc2(3),
EVP_ripemd160(3), EVP_sha1(3), EVP_sha224(3), EVP_sha3_224(3),
EVP_sm3(3), EVP_whirlpool(3)

HISTORY

The EVP_MD_CTX_create() and EVP_MD_CTX_destroy() functions were renamed to EVP_MD_CTX_new() and EVP_MD_CTX_free() in OpenSSL 1.1.0, respectively.

The link between digests and signing algorithms was fixed in OpenSSL 1.0 and later, so now EVP_sha1() can be used with RSA and DSA.

The EVP_dss1() function was removed in OpenSSL 1.1.0.

The EVP_MD_CTX_set_pkey_ctx() function was added in OpenSSL 1.1.1.

The EVP_Q_digest(), EVP_DigestInit_ex2(), EVP_MD_fetch(), EVP_MD_free(), EVP_MD_up_ref(), EVP_MD_get_params(), EVP_MD_CTX_set_params(), EVP_MD_CTX_get_params(), EVP_MD_gettable_params(), EVP_MD_gettable_ctx_params(), EVP_MD_settable_ctx_params(), EVP_MD_CTX_settable_params() and EVP_MD_CTX_gettable_params() functions were added in OpenSSL 3.0. The EVP_MD_type(), EVP_MD_nid(), EVP_MD_name(), EVP_MD_pkey_type(), EVP_MD_size(), EVP_MD_block_size(), EVP_MD_flags(), EVP_MD_CTX_size(), EVP_MD_CTX_block_size(), EVP_MD_CTX_type(), and EVP_MD_CTX_md_data() functions were renamed to include "get" or "get0" in their names in OpenSSL 3.0, respectively. The old names are kept as non-deprecated alias macros.

The EVP_MD_CTX_md() function was deprecated in OpenSSL 3.0; use EVP_MD_CTX_get0_md() instead. EVP_MD_CTX_update_fn() and EVP_MD_CTX_set_update_fn() were deprecated in OpenSSL 3.0.

COPYRIGHT

Copyright 2000-2021 The OpenSSL Project Authors. All Rights Reserved.
Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at
<<https://www.openssl.org/source/license.html>>.

