NAME

EVP_MD_CTX_init, EVP_MD_CTX_create, EVP_DigestInit_ex, EVP_DigestUpdate, EVP_DigestFinal_ex, EVP_MD_CTX_cleanup, EVP_MD_CTX_destroy, EVP_MAX_MD_SIZE, EVP_MD_CTX_copy_ex, EVP_DigestInit, EVP_DigestFinal, EVP_MD_CTX_copy, EVP_MD_type, EVP_MD_pkey_type, EVP_MD_size, EVP_MD_block_size, EVP_MD_CTX_md, EVP_MD_CTX_size, EVP_MD_CTX_block_size, EVP_MD_CTX_type, EVP_md_null, EVP_md2, EVP_md5, EVP_sha, EVP_sha1, EVP_sha224, EVP_sha256, EVP_sha384, EVP_sha512, EVP_dss, EVP_dss1, EVP_mdc2, EVP_ripemd160, EVP_get_digestbyname, EVP_get_digestbynid, EVP_get_digestbyobj - EVP digest routines

SYNOPSIS

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

void EVP_MD_CTX_init(EVP_MD_CTX *ctx);
EVP_MD_CTX *EVP_MD_CTX_create(void);

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_MD_CTX_cleanup(EVP_MD_CTX *ctx);
void EVP_MD_CTX_destroy(EVP_MD_CTX *ctx);

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);

#define EVP_MAX_MD_SIZE 64 /* SHA512 */

int EVP_MD_type(const EVP_MD *md);
int EVP_MD_pkey_type(const EVP_MD *md);
int EVP_MD_size(const EVP_MD *md);
int EVP_MD_block_size(const EVP_MD *md);

const EVP_MD *EVP_MD_CTX_md(const EVP_MD_CTX *ctx);
#define EVP_MD_CTX_size(e) EVP_MD_size(EVP_MD_CTX_md(e))
#define EVP_MD_CTX_block_size(e) EVP_MD_block_size((e)->digest)
#define EVP_MD_CTX_type(e) EVP_MD_type((e)->digest)

const EVP_MD *EVP_md_null(void);
const EVP_MD *EVP_md2(void);
const EVP_MD *EVP_md5(void);
const EVP_MD *EVP_sha(void);
const EVP_MD *EVP_sha1(void);
const EVP_MD *EVP_dss(void);
const EVP_MD *EVP_dss1(void);
const EVP_MD *EVP_mdc2(void);
const EVP_MD *EVP_ripemd160(void);

const EVP_MD *EVP_sha224(void);
```

```
const EVP_MD *EVP_sha256(void);
const EVP_MD *EVP_sha384(void);
const EVP_MD *EVP_sha512(void);

const EVP_MD *EVP_get_digestbyname(const char *name);
#define EVP_get_digestbynid(a) EVP_get_digestbyname(OBJ_nid2sn(a))
#define EVP_get_digestbyobj(a) EVP_get_digestbynid(OBJ_obj2nid(a))
```

DESCRIPTION

The EVP digest routines are a high level interface to message digests.

EVP_MD_CTX_init() initializes digest context **ctx**.

EVP_MD_CTX_create() allocates, initializes and returns a digest context.

EVP_DigestInit_ex() sets up digest context **ctx** to use a digest **type** from ENGINE **impl**. **ctx** must be initialized before calling this function. **type** will typically be supplied by a functionsuch as *EVP_sha1()*. If**impl** is NULL then the default implemen tation of digest **type** is used.

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_ex()* can be called to initialize a new digest operation.

EVP_MD_CTX_cleanup() cleans up digest context **ctx**, it should be called after a digest context is no longer needed.

EVP_MD_CTX_destroy() cleans up digest context **ctx** and frees up the space allocated to it, it should be called only on a context created using *EVP_MD_CTX_create()*.

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. **out** must be initialized before calling this function.

EVP_DigestInit() behaves in the same way as *EVP_DigestInit_ex()* except the passed context **ctx** does not have to be initialized, and it always uses the default digest implementation.

EVP_DigestFinal() is similar to *EVP_DigestFinal_ex()* except the digest context **ctx** is automatically cleaned up.

EVP_MD_CTX_copy() is similar to *EVP_MD_CTX_copy_ex()* except the destination **out** does not have to be initialized.

EVP_MD_size() and *EVP_MD_CTX_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_block_size() and *EVP_MD_CTX_block_size()* return the block size of the message digest when passed an **EVP_MD** or an **EVP_MD_CTX** structure.

EVP_MD_type() and *EVP_MD_CTX_type()* return the NID of the OBJECT IDENTIFIER representing the given message digest when passed an **EVP_MD** structure. For example EVP_MD_type(*EVP_sha1()*) returns **NID_sha1**. This function is normally used when setting ASN1 OIDs.

EVP_MD_CTX_md() returns the **EVP_MD** structure corresponding to the passed **EVP_MD_CTX**.

EVP_MD_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_md2(), *EVP_md5()*, *EVP_sha()*, *EVP_sha1()*, *EVP_sha224()*, *EVP_sha256()*, *EVP_sha384()*, *EVP_sha512()*, *EVP_mdc2()* and *EVP_ripemd160()* return **EVP_MD** structures for the MD2, MD5, SHA, SHA1, SHA224, SHA256, SHA384, SHA512, MDC2 and RIPEMD160 digest algorithms respectively.

EVP_dss() and *EVP_dss1()* return **EVP_MD** structures for SHA and SHA1 digest algorithms but using DSS (DSA) for the signature algorithm. Note: there is no need to use these pseudo-digests in OpenSSL 1.0.0 and later, they are however retained for compatibility.

EVP_md_null() is a "null" message digest that does nothing: i.e. the hash it returns is of zero length.

EVP_get_digestbyname(), *EVP_get_digestbynid()* and *EVP_get_digestbyobj()* return an **EVP_MD** structure when passed a digest name, a digest NID or an ASN1_OBJECT structure respectively. The digest table must be initialized using, for example, *OpenSSL_add_all_digests()* for these functions to work.

RETURN VALUES

EVP_DigestInit_ex(), *EVP_DigestUpdate()* and *EVP_DigestFinal_ex()* return 1 for success and 0 for failure.

EVP_MD_CTX_copy_ex() returns 1 if successful or 0 for failure.

EVP_MD_type(), *EVP_MD_pkey_type()* and *EVP_MD_type()* return the NID of the corresponding OBJECT IDENTIFIER or NID_undef if none exists.

EVP_MD_size(), *EVP_MD_block_size()*, *EVP_MD_CTX_size()* and *EVP_MD_CTX_block_size()* return the digest or block size in bytes.

EVP_md_null(), *EVP_md2()*, *EVP_md5()*, *EVP_sha()*, *EVP_sha1()*, *EVP_dss()*, *EVP_dss1()*, *EVP_mdc2()* and *EVP_ripemd160()* return pointers to the corresponding EVP_MD structures.

EVP_get_digestbyname(), *EVP_get_digestbynid()* and *EVP_get_digestbyobj()* return either an **EVP_MD** structure or NULL if an error occurs.

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 SHA2 digest algorithms such as SHA256. 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.

In OpenSSL 0.9.7 and later if digest contexts are not cleaned up after use memory leaks will occur.

Stack allocation of EVP_MD_CTX structures is common, for example:

```
EVP_MD_CTX mctx;
EVP_MD_CTX_init(&mctx);
```

This will cause binary compatibility issues if the size of EVP_MD_CTX structure changes (this will only happen with a major release of OpenSSL). Applications wishing to avoid this should use *EVP_MD_CTX_create()* instead:

```
EVP_MD_CTX *mctx;
mctx = EVP_MD_CTX_create();
```

EXAMPLE

This example digests the data "Test Messagen" and "Hello Worldn", using the digest name passed on the command line.

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

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];
int md_len, i;

OpenSSL_add_all_digests();

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

md = EVP_get_digestbyname(argv[1]);

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

mdctx = EVP_MD_CTX_create();
EVP_DigestInit_ex(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_destroy(mdctx);

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

/* Call this once before exit. */
EVP_cleanup();
exit(0)
}
```

SEE ALSO

dgst(1), *evp(3)*

HISTORY

EVP_DigestInit(), *EVP_DigestUpdate()* and *EVP_DigestFinal()* are available in all versions of SSLeay and OpenSSL.

EVP_MD_CTX_init(), *EVP_MD_CTX_create()*, *EVP_MD_CTX_copy_ex()*, *EVP_MD_CTX_cleanup()*, *EVP_MD_CTX_destroy()*, *EVP_DigestInit_ex()* and *EVP_DigestFinal_ex()* were added in OpenSSL 0.9.7.

EVP_md_null(), *EVP_md2()*, *EVP_md5()*, *EVP_sha()*, *EVP_sha1()*, *EVP_dss()*, *EVP_dss1()*, *EVP_mdc2()* and *EVP_ripemd160()* were changed to return truly const EVP_MD * in OpenSSL 0.9.7.

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; there is no need to use *EVP_dss1()* any more.

OpenSSL 1.0 and later does not include the MD2 digest algorithm in the default configuration due to its security weaknesses.