NAME

X509v3_get_ext_count, X509v3_get_ext, X509v3_get_ext_by_NID, X509v3_get_ext_by_OBJ, X509v3_get_ext_by_critical, X509v3_delete_ext, X509v3_add_ext, X509_get_ext_count, X509_get_ext, X509_get_ext_by_NID, X509_get_ext_by_OBJ, X509_get_ext_by_critical, X509_delete_ext, X509_add_ext, X509_CRL_get_ext_count, X509_CRL_get_ext, X509_CRL_get_ext_by_NID, X509_CRL_get_ext_by_OBJ, X509_CRL_get_ext_by_critical, X509_CRL_delete_ext, X509_CRL_add_ext, X509_REVOKED_get_ext_count, X509_REVOKED_get_ext, X509_REVOKED_get_ext_by_NID, X509_REVOKED_get_ext_by_OBJ, X509_REVOKED_get_ext_by_critical, X509_REVOKED_delete_ext, X509_REVOKED_add_ext - extension stack utility functions

SYNOPSIS

```
#include <openssl/x509.h>

int X509v3_get_ext_count(const STACK_OF(X509_EXTENSION) *x);
X509_EXTENSION *X509v3_get_ext(const STACK_OF(X509_EXTENSION) *x, int loc);

int X509v3_get_ext_by_NID(const STACK_OF(X509_EXTENSION) *x,
                          int nid, int lastpos);
int X509v3_get_ext_by_OBJ(const STACK_OF(X509_EXTENSION) *x,
                          const ASN1_OBJECT *obj, int lastpos);
int X509v3_get_ext_by_critical(const STACK_OF(X509_EXTENSION) *x,
                               int crit, int lastpos);
X509_EXTENSION *X509v3_delete_ext(STACK_OF(X509_EXTENSION) *x, int loc);
STACK_OF(X509_EXTENSION) *X509v3_add_ext(STACK_OF(X509_EXTENSION) **x,
                                         X509_EXTENSION *ex, int loc);

int X509_get_ext_count(const X509 *x);
X509_EXTENSION *X509_get_ext(const X509 *x, int loc);
int X509_get_ext_by_NID(const X509 *x, int nid, int lastpos);
int X509_get_ext_by_OBJ(const X509 *x, const ASN1_OBJECT *obj, int lastpos);
int X509_get_ext_by_critical(const X509 *x, int crit, int lastpos);
X509_EXTENSION *X509_delete_ext(X509 *x, int loc);
int X509_add_ext(X509 *x, X509_EXTENSION *ex, int loc);

int X509_CRL_get_ext_count(const X509_CRL *x);
X509_EXTENSION *X509_CRL_get_ext(const X509_CRL *x, int loc);
int X509_CRL_get_ext_by_NID(const X509_CRL *x, int nid, int lastpos);
int X509_CRL_get_ext_by_OBJ(const X509_CRL *x, const ASN1_OBJECT *obj,
                            int lastpos);
```

```
int X509_CRL_get_ext_by_critical(const X509_CRL *x, int crit, int lastpos);
X509_EXTENSION *X509_CRL_delete_ext(X509_CRL *x, int loc);
int X509_CRL_add_ext(X509_CRL *x, X509_EXTENSION *ex, int loc);

int X509_REVOKED_get_ext_count(const X509_REVOKED *x);
X509_EXTENSION *X509_REVOKED_get_ext(const X509_REVOKED *x, int loc);
int X509_REVOKED_get_ext_by_NID(const X509_REVOKED *x, int nid, int lastpos);
int X509_REVOKED_get_ext_by_OBJ(const X509_REVOKED *x, const ASN1_OBJECT *obj,
                                int lastpos);
int X509_REVOKED_get_ext_by_critical(const X509_REVOKED *x, int crit, int lastpos);
X509_EXTENSION *X509_REVOKED_delete_ext(X509_REVOKED *x, int loc);
int X509_REVOKED_add_ext(X509_REVOKED *x, X509_EXTENSION *ex, int loc);
```

DESCRIPTION

X509v3_get_ext_count() retrieves the number of extensions in *x*.

X509v3_get_ext() retrieves extension *loc* from *x*. The index *loc* can take any value from 0 to X509_get_ext_count(*x*) - 1. The returned extension is an internal pointer which **MUST NOT** be freed by the application.

X509v3_get_ext_by_NID() and **X509v3_get_ext_by_OBJ()** look for an extension with *nid* or *obj* from extension STACK *x*. The search starts from the extension after *lastpos* or from the beginning if *lastpos* is -1. If the extension is found, its index is returned, otherwise -1 is returned.

X509v3_get_ext_by_critical() is similar to **X509v3_get_ext_by_NID()** except it looks for an extension of criticality *crit*. A zero value for *crit* looks for a non-critical extension. A nonzero value looks for a critical extension.

X509v3_delete_ext() deletes the extension with index *loc* from *x*. The deleted extension is returned and must be freed by the caller. If *loc* is an invalid index value, NULL is returned.

X509v3_add_ext() adds extension *ex* to STACK **x* at position *loc*. If *loc* is -1, the new extension is added to the end. If **x* is NULL, a new STACK will be allocated. The passed extension *ex* is duplicated internally so it must be freed after use.

X509_get_ext_count(), **X509_get_ext()**, **X509_get_ext_by_NID()**, **X509_get_ext_by_OBJ()**, **X509_get_ext_by_critical()**, **X509_delete_ext()** and **X509_add_ext()** operate on the extensions of certificate *x*. They are otherwise identical to the X509v3 functions.

X509_CRL_get_ext_count(), **X509_CRL_get_ext()**, **X509_CRL_get_ext_by_NID()**,

X509_CRL_get_ext_by_OBJ(), **X509_CRL_get_ext_by_critical()**, **X509_CRL_delete_ext()** and **X509_CRL_add_ext()** operate on the extensions of CRL *x*. They are otherwise identical to the X509v3 functions.

X509_REVOKED_get_ext_count(), **X509_REVOKED_get_ext()**, **X509_REVOKED_get_ext_by_NID()**, **X509_REVOKED_get_ext_by_OBJ()**, **X509_REVOKED_get_ext_by_critical()**, **X509_REVOKED_delete_ext()** and **X509_REVOKED_add_ext()** operate on the extensions of CRL entry *x*. They are otherwise identical to the X509v3 functions.

NOTES

These functions are used to examine stacks of extensions directly. Applications that want to parse or encode and add an extension should use the extension encode and decode functions instead, such as **X509_add1_ext_i2d()** and **X509_get_ext_d2i()**.

For **X509v3_get_ext_by_NID()**, **X509v3_get_ext_by_OBJ()**, **X509v3_get_ext_by_critical()** and its variants, a zero index return value is not an error since extension STACK *x* indices start from zero. These search functions start from the extension **after** the *lastpos* parameter so it should initially be set to -1. If it is set to zero, the initial extension will not be checked.

X509v3_delete_ext() and its variants are a bit counter-intuitive because these functions do not free the extension they delete. They return an **X509_EXTENSION** object which must be explicitly freed using **X509_EXTENSION_free()**.

RETURN VALUES

X509v3_get_ext_count() returns the extension count or 0 for failure.

X509v3_get_ext(), **X509v3_delete_ext()** and **X509_delete_ext()** return an **X509_EXTENSION** structure or NULL if an error occurs.

X509v3_get_ext_by_OBJ() and **X509v3_get_ext_by_critical()** return the extension index or -1 if an error occurs.

X509v3_get_ext_by_NID() returns the extension index or negative values if an error occurs.

X509v3_add_ext() returns a STACK of extensions or NULL on error.

X509_add_ext() returns 1 on success and 0 on error.

SEE ALSO

X509V3_get_d2i(3)

COPYRIGHT