NAME

ASN1_TIME_set, ASN1_UTCTIME_set, ASN1_GENERALIZEDTIME_set, ASN1_TIME_adj, ASN1_UTCTIME_adj, ASN1_GENERALIZEDTIME_adj, ASN1_TIME_check, ASN1_UTCTIME_check, ASN1_GENERALIZEDTIME_check, ASN1_TIME_set_string, ASN1_UTCTIME_set_string, ASN1_GENERALIZEDTIME_set_string, ASN1_TIME_set_string_X509, ASN1_TIME_normalize, ASN1_TIME_to_tm, ASN1_TIME_print, ASN1_TIME_print_ex, ASN1_UTCTIME_print, ASN1_GENERALIZEDTIME_print, ASN1_TIME_diff, ASN1_TIME_cmp_time_t, ASN1_UTCTIME_cmp_time_t, ASN1_TIME_compare, ASN1_TIME_to_generalizedtime, ASN1_TIME_dup, ASN1_UTCTIME_dup, ASN1_GENERALIZEDTIME_dup - ASN.1 Time functions

SYNOPSIS

```
ASN1_TIME *ASN1_TIME_set(ASN1_TIME *s, time_t t);
ASN1_UTCTIME *ASN1_UTCTIME_set(ASN1_UTCTIME *s, time_t t);
ASN1_GENERALIZEDTIME *ASN1_GENERALIZEDTIME_set(ASN1_GENERALIZEDTIME *s,
                                               time_t t);

ASN1_TIME *ASN1_TIME_adj(ASN1_TIME *s, time_t t, int offset_day,
                         long offset_sec);
ASN1_UTCTIME *ASN1_UTCTIME_adj(ASN1_UTCTIME *s, time_t t,
                               int offset_day, long offset_sec);
ASN1_GENERALIZEDTIME *ASN1_GENERALIZEDTIME_adj(ASN1_GENERALIZEDTIME *s,
                                               time_t t, int offset_day,
                                               long offset_sec);

int ASN1_TIME_set_string(ASN1_TIME *s, const char *str);
int ASN1_TIME_set_string_X509(ASN1_TIME *s, const char *str);
int ASN1_UTCTIME_set_string(ASN1_UTCTIME *s, const char *str);
int ASN1_GENERALIZEDTIME_set_string(ASN1_GENERALIZEDTIME *s,
                                    const char *str);

int ASN1_TIME_normalize(ASN1_TIME *s);

int ASN1_TIME_check(const ASN1_TIME *t);
int ASN1_UTCTIME_check(const ASN1_UTCTIME *t);
int ASN1_GENERALIZEDTIME_check(const ASN1_GENERALIZEDTIME *t);

int ASN1_TIME_print(BIO *b, const ASN1_TIME *s);
int ASN1_TIME_print_ex(BIO *bp, const ASN1_TIME *tm, unsigned long flags);
int ASN1_UTCTIME_print(BIO *b, const ASN1_UTCTIME *s);
```

```
int ASN1_GENERALIZEDTIME_print(BIO *b, const ASN1_GENERALIZEDTIME *s);

int ASN1_TIME_to_tm(const ASN1_TIME *s, struct tm *tm);
int ASN1_TIME_diff(int *pday, int *psec, const ASN1_TIME *from,
                   const ASN1_TIME *to);

int ASN1_TIME_cmp_time_t(const ASN1_TIME *s, time_t t);
int ASN1_UTCTIME_cmp_time_t(const ASN1_UTCTIME *s, time_t t);

int ASN1_TIME_compare(const ASN1_TIME *a, const ASN1_TIME *b);

ASN1_GENERALIZEDTIME *ASN1_TIME_to_generalizedtime(ASN1_TIME *t,
                                                   ASN1_GENERALIZEDTIME **out);

ASN1_TIME *ASN1_TIME_dup(const ASN1_TIME *t);
ASN1_UTCTIME *ASN1_UTCTIME_dup(const ASN1_UTCTIME *t);
ASN1_GENERALIZEDTIME *ASN1_GENERALIZEDTIME_dup(const ASN1_GENERALIZEDTIME *t);
```

DESCRIPTION

The **ASN1_TIME_set()**, **ASN1_UTCTIME_set()** and **ASN1_GENERALIZEDTIME_set()** functions set the structure *s* to the time represented by the time_t value *t*. If *s* is NULL a new time structure is allocated and returned.

The **ASN1_TIME_adj()**, **ASN1_UTCTIME_adj()** and **ASN1_GENERALIZEDTIME_adj()** functions set the time structure *s* to the time represented by the time *offset_day* and *offset_sec* after the time_t value *t*. The values of *offset_day* or *offset_sec* can be negative to set a time before *t*. The *offset_sec* value can also exceed the number of seconds in a day. If *s* is NULL a new structure is allocated and returned.

The **ASN1_TIME_set_string()**, **ASN1_UTCTIME_set_string()** and **ASN1_GENERALIZEDTIME_set_string()** functions set the time structure *s* to the time represented by string *str* which must be in appropriate ASN.1 time format (for example YYMMDDHHMMSSZ or YYYYMMDDHHMMSSZ). If *s* is NULL this function performs a format check on *str* only. The string *str* is copied into *s*.

ASN1_TIME_set_string_X509() sets **ASN1_TIME** structure *s* to the time represented by string *str* which must be in appropriate time format that RFC 5280 requires, which means it only allows YYMMDDHHMMSSZ and YYYYMMDDHHMMSSZ (leap second is rejected), all other ASN.1 time format are not allowed. If *s* is NULL this function performs a format check on *str* only.

The **ASN1_TIME_normalize()** function converts an **ASN1_GENERALIZEDTIME** or **ASN1_UTCTIME** into a time value that can be used in a certificate. It should be used after the **ASN1_TIME_set_string()** functions and before **ASN1_TIME_print()** functions to get consistent (i.e. GMT) results.

The **ASN1_TIME_check()**, **ASN1_UTCTIME_check()** and **ASN1_GENERALIZEDTIME_check()** functions check the syntax of the time structure *s*.

The **ASN1_TIME_print()**, **ASN1_UTCTIME_print()** and **ASN1_GENERALIZEDTIME_print()** functions print the time structure *s* to BIO *b* in human readable format. It will be of the format MMM DD HH:MM:SS YYYY [GMT], for example "Feb 3 00:55:52 2015 GMT", which does not include a newline. If the time structure has invalid format it prints out "Bad time value" and returns an error. The output for generalized time may include a fractional part following the second.

ASN1_TIME_print_ex() provides *flags* to specify the output format of the datetime. This can be either **ASN1_DTFLGS_RFC822** or **ASN1_DTFLGS_ISO8601**.

ASN1_TIME_to_tm() converts the time *s* to the standard *tm* structure. If *s* is NULL, then the current time is converted. The output time is GMT. The *tm_sec*, *tm_min*, *tm_hour*, *tm_mday*, *tm_wday*, *tm_yday*, *tm_mon* and *tm_year* fields of *tm* structure are set to proper values, whereas all other fields are set to 0. If *tm* is NULL this function performs a format check on *s* only. If *s* is in Generalized format with fractional seconds, e.g. YYYYMMDDHHMMSS.SSSZ, the fractional seconds will be lost while converting *s* to *tm* structure.

ASN1_TIME_diff() sets **pday* and **psec* to the time difference between *from* and *to*. If *to* represents a time later than *from* then one or both (depending on the time difference) of **pday* and **psec* will be positive. If *to* represents a time earlier than *from* then one or both of **pday* and **psec* will be negative. If *to* and *from* represent the same time then **pday* and **psec* will both be zero. If both **pday* and **psec* are nonzero they will always have the same sign. The value of **psec* will always be less than the number of seconds in a day. If *from* or *to* is NULL the current time is used.

The **ASN1_TIME_cmp_time_t()** and **ASN1_UTCTIME_cmp_time_t()** functions compare the two times represented by the time structure *s* and the time_t *t*.

The **ASN1_TIME_compare()** function compares the two times represented by the time structures *a* and *b*.

The **ASN1_TIME_to_generalizedtime()** function converts an **ASN1_TIME** to an **ASN1_GENERALIZEDTIME**, regardless of year. If either *out* or **out* are NULL, then a new object is allocated and must be freed after use.

The **ASN1_TIME_dup()**, **ASN1_UTCTIME_dup()** and **ASN1_GENERALIZEDTIME_dup()** functions duplicate the time structure *t* and return the duplicated result correspondingly.

NOTES

The **ASN1_TIME** structure corresponds to the ASN.1 structure **Time** defined in RFC5280 et al. The time setting functions obey the rules outlined in RFC5280: if the date can be represented by UTCTime it is used, else GeneralizedTime is used.

The **ASN1_TIME**, **ASN1_UTCTIME** and **ASN1_GENERALIZEDTIME** structures are represented as an **ASN1_STRING** internally and can be freed up using **ASN1_STRING_free()**.

The **ASN1_TIME** structure can represent years from 0000 to 9999 but no attempt is made to correct ancient calendar changes (for example from Julian to Gregorian calendars).

ASN1_UTCTIME is limited to a year range of 1950 through 2049.

Some applications add offset times directly to a time_t value and pass the results to **ASN1_TIME_set()** (or equivalent). This can cause problems as the time_t value can overflow on some systems resulting in unexpected results. New applications should use **ASN1_TIME_adj()** instead and pass the offset value in the *offset_sec* and *offset_day* parameters instead of directly manipulating a time_t value.

ASN1_TIME_adj() may change the type from **ASN1_GENERALIZEDTIME** to **ASN1_UTCTIME**, or vice versa, based on the resulting year. **ASN1_GENERALIZEDTIME_adj()** and **ASN1_UTCTIME_adj()** will not modify the type of the return structure.

It is recommended that functions starting with **ASN1_TIME** be used instead of those starting with **ASN1_UTCTIME** or **ASN1_GENERALIZEDTIME**. The functions starting with **ASN1_UTCTIME** and **ASN1_GENERALIZEDTIME** act only on that specific time format. The functions starting with **ASN1_TIME** will operate on either format.

BUGS

ASN1_TIME_print(), **ASN1_UTCTIME_print()** and **ASN1_GENERALIZEDTIME_print()** do not print out the timezone: it either prints out "GMT" or nothing. But all certificates complying with RFC5280 et al use GMT anyway.

ASN1_TIME_print(), **ASN1_TIME_print_ex()**, **ASN1_UTCTIME_print()** and **ASN1_GENERALIZEDTIME_print()** do not distinguish if they fail because of an I/O error or invalid time format.

Use the **ASN1_TIME_normalize()** function to normalize the time value before printing to get GMT

results.

RETURN VALUES

ASN1_TIME_set(), **ASN1_UTCTIME_set()**, **ASN1_GENERALIZEDTIME_set()**, **ASN1_TIME_adj()**, **ASN1_UTCTIME_adj()** and **ASN1_GENERALIZEDTIME_set()** return a pointer to a time structure or NULL if an error occurred.

ASN1_TIME_set_string(), **ASN1_UTCTIME_set_string()**, **ASN1_GENERALIZEDTIME_set_string()** and **ASN1_TIME_set_string_X509()** return 1 if the time value is successfully set and 0 otherwise.

ASN1_TIME_normalize() returns 1 on success, and 0 on error.

ASN1_TIME_check(), ASN1_UTCTIME_check and **ASN1_GENERALIZEDTIME_check()** return 1 if the structure is syntactically correct and 0 otherwise.

ASN1_TIME_print(), **ASN1_UTCTIME_print()** and **ASN1_GENERALIZEDTIME_print()** return 1 if the time is successfully printed out and 0 if an I/O error occurred an error occurred (I/O error or invalid time format).

ASN1_TIME_to_tm() returns 1 if the time is successfully parsed and 0 if an error occurred (invalid time format).

ASN1_TIME_diff() returns 1 for success and 0 for failure. It can fail if the passed-in time structure has invalid syntax, for example.

ASN1_TIME_cmp_time_t() and **ASN1_UTCTIME_cmp_time_t()** return -1 if *s* is before *t*, 0 if *s* equals *t*, or 1 if *s* is after *t*. -2 is returned on error.

ASN1_TIME_compare() returns -1 if *a* is before *b*, 0 if *a* equals *b*, or 1 if *a* is after *b*. -2 is returned on error.

ASN1_TIME_to_generalizedtime() returns a pointer to the appropriate time structure on success or NULL if an error occurred.

ASN1_TIME_dup(), **ASN1_UTCTIME_dup()** and **ASN1_GENERALIZEDTIME_dup()** return a pointer to a time structure or NULL if an error occurred.

EXAMPLES

Set a time structure to one hour after the current time and print it out:

```
#include <time.h>
#include <openssl/asn1.h>

ASN1_TIME *tm;
time_t t;
BIO *b;

t = time(NULL);
tm = ASN1_TIME_adj(NULL, t, 0, 60 * 60);
b = BIO_new_fp(stdout, BIO_NOCLOSE);
ASN1_TIME_print(b, tm);
ASN1_STRING_free(tm);
BIO_free(b);
```

Determine if one time is later or sooner than the current time:

```
int day, sec;

if (!ASN1_TIME_diff(&day, &sec, NULL, to))
   /* Invalid time format */

if (day > 0 || sec > 0)
   printf("Later\n");
else if (day < 0 || sec < 0)
   printf("Sooner\n");
else
   printf("Same\n");
```

HISTORY

The **ASN1_TIME_to_tm()** function was added in OpenSSL 1.1.1. The **ASN1_TIME_set_string_X509()** function was added in OpenSSL 1.1.1. The **ASN1_TIME_normalize()** function was added in OpenSSL 1.1.1. The **ASN1_TIME_cmp_time_t()** function was added in OpenSSL 1.1.1. The **ASN1_TIME_compare()** function was added in OpenSSL 1.1.1.

COPYRIGHT