NAME

ares_destroy − Destroy a resolver channel

SYNOPSIS

#include <ares.h>

int ares_destroy(ares_channel *channel***)**

DESCRIPTION

The **ares_destroy** function destroys the name service channel identified by *channel*, freeing all memory and closing all sockets used by the channel. **ares_destroy** invokes the callbacks for each pending query on the channel, passing a status of **ARES_EDESTRUCTION**. These calls give the callbacks a chance to clean up any state which might have been stored in their arguments.

SEE ALSO

ares_init(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_expand_name − Expand a DNS-encoded domain name

SYNOPSIS

#include <ares.h>

int ares_expand_name(const unsigned char ****encoded*,**
const unsigned char ****abuf*, int** *alen***, char *****s***,**
int ****enclen*)**

DESCRIPTION

The **ares_expand_name** function converts a DNS-encoded domain name to a dot-separated C string. The argument *encoded* gives the beginning of the encoded domain name, and the arguments *abuf* and *alen* give the containing message buffer (necessary for the processing of indirection pointers within the encoded domain name). The result is placed in a NUL-terminated allocated buffer, a pointer to which is stored in the variable pointed to by *s*. The length of the encoded name is stored in the variable pointed to by *enclen* so that the caller can advance past the encoded domain name to read further data in the message.

RETURN VALUES

ares_expand_name can return any of the following values:

ARES_SUCCESS
Expansion of the encoded name succeeded.

ARES_EBADNAME
The encoded domain name was malformed and could not be expanded.

ARES_ENOMEM
Memory was exhausted.

SEE ALSO

ares_mkquery(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_fds − Get file descriptors to select on for name service

SYNOPSIS

#include <ares.h>

int ares_fds(ares_channel *channel***, fd_set ****read_fds***,**
fd_set ****write_fds*)**

DESCRIPTION

The **ares_fds** function retrieves the set of file descriptors which the calling application should select on for reading and writing for the processing of name service queries pending on the name service channel identified by *channel*. File descriptors will be set in the file descriptor sets pointed to by *read_fds* and *write_fds* as appropriate. File descriptors already set in *read_fds* and *write_fds* will remain set; initialization of the file descriptor sets (using **FD_ZERO**) is the responsibility of the caller.

RETURN VALUES

ares_fds returns one greater than the number of the highest socket set in either *read_fds* or *write_fds*. If no queries are active, **ares_fds** will return 0.

SEE ALSO

ares_timeout(3), **ares_process**(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_free_errmem − Free memory allocated by ares_strerror

SYNOPSIS

#include <ares.h>

void ares_free_errmem(char **errmem*)

DESCRIPTION

The **ares_free_errmem** function frees any memory which might have been allocated by the **ares_strerror**(3) function. The parameter *errmem* should be set to the variable pointed to by the *memptr* argument previously passed to *ares_strerror*.

SEE ALSO

ares_strerror(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_free_hostent − Free host structure allocated by ares functions

SYNOPSIS

#include <ares.h>

void ares_free_hostent(struct hostent ****host*)**

DESCRIPTION

The *ares_free_hostent* function frees a **struct hostent** allocated by one of the functions *ares_parse_a_reply* or *ares_parse_ptr_reply*.

SEE ALSO

ares_parse_a_reply(3), **ares_parse_ptr_reply**(3)

NOTES

It is not necessary (and is not correct) to free the host structure passed to the callback functions for *ares_gethostbyname* or *ares_gethostbyaddr*. The ares library will automatically free such host structures when the callback returns.

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_free_string − Free strings allocated by ares functions

SYNOPSIS

#include <ares.h>

void ares_free_string(char **str*)

DESCRIPTION

The *ares_free_string* function frees a string allocated by the *ares_mkquery* function.

SEE ALSO

ares_mkquery(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_gethostbyaddr − Initiate a host query by address

SYNOPSIS

#include <ares.h>

typedef void (*ares_host_callback)(void *_arg_, int _status_,
struct hostent *_hostent_)

void ares_gethostbyaddr(ares_channel *channel*, **const void *_addr_,**
int *addrlen*, **int** *family*, **ares_host_callback** *callback*,
void *_arg_)

DESCRIPTION

The **ares_gethostbyaddr** function initiates a host query by address on the name service channel identified by *channel*. The parameters *addr* and *addrlen* give the address as a series of bytes, and *family* gives the type of address. When the query is complete or has failed, the ares library will invoke *callback*. Completion or failure of the query may happen immediately, or may happen during a later call to **ares_process**(3) or **ares_destroy**(3).

The callback argument *arg* is copied from the **ares_gethostbyaddr** argument *arg*. The callback argument *status* indicates whether the query succeeded and, if not, how it failed. It may have any of the following values:

ARES_SUCCESS The host lookup completed successfully.

ARES_ENOTIMP The ares library does not know how to look up addresses of type *family*.

ARES_ENOTFOUND
The address *addr* was not found.

ARES_ENOMEM Memory was exhausted.

ARES_EDESTRUCTION
The name service channel *channel* is being destroyed; the query will not be completed.

On successful completion of the query, the callback argument *hostent* points to a **struct hostent** containing the name of the host returned by the query. The callback need not and should not attempt to free the memory pointed to by *hostent*; the ares library will free it when the callback returns. If the query did not complete successfully, *hostent* will be **NULL**.

SEE ALSO

ares_process(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_gethostbyname − Initiate a host query by name

SYNOPSIS

#include <ares.h>

typedef void (*ares_host_callback)(void *_arg_, int _status_,
struct hostent *_hostent_)

void ares_gethostbyname(ares_channel *channel*, **const char *_name_,**
int *family*, **ares_host_callback** *callback*, **void *_arg_)**

DESCRIPTION

The **ares_gethostbyname** function initiates a host query by name on the name service channel identified by *channel*. The parameter *name* gives the hostname as a NUL-terminated C string, and *family* gives the desired type of address for the resulting host entry. When the query is complete or has failed, the ares library will invoke *callback*. Completion or failure of the query may happen immediately, or may happen during a later call to **ares_process**(3) or **ares_destroy**(3).

The callback argument *arg* is copied from the **ares_gethostbyname** argument *arg*. The callback argument *status* indicates whether the query succeeded and, if not, how it failed. It may have any of the following values:

ARES_SUCCESS The host lookup completed successfully.

ARES_ENOTIMP The ares library does not know how to find addresses of type *family*.

ARES_EBADNAME The hostname **name** is composed entirely of numbers and periods, but is not a valid representation of an Internet address.

ARES_ENOTFOUND
The address *addr* was not found.

ARES_ENOMEM Memory was exhausted.

ARES_EDESTRUCTION
The name service channel *channel* is being destroyed; the query will not be completed.

On successful completion of the query, the callback argument *hostent* points to a **struct hostent** containing the name of the host returned by the query. The callback need not and should not attempt to free the memory pointed to by *hostent*; the ares library will free it when the callback returns. If the query did not complete successfully, *hostent* will be **NULL**.

SEE ALSO

ares_process(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_init, ares_init_options − Initialize a resolver channel

SYNOPSIS

#include <ares.h>

int ares_init(ares_channel ****channel*)**
int ares_init_options(ares_channel ****channel*,**
struct ares_options ****options*, int** *optmask***)**

cc file.c -lares

DESCRIPTION

The **ares_init** function initializes a communications channel for name service lookups. If it returns successfully, **ares_init** will set the variable pointed to by *channel* to a handle used to identify the name service channel. The caller should invoke **ares_destroy**(3) on the handle when the channel is no longer needed.

The **ares_init_options** function also initializes a name service channel, with additional options useful for applications requiring more control over name service configuration. The *optmask* parameter specifies which fields in the structure pointed to by *options* are set, as follows:

ARES_OPT_FLAGS
int *flags***;**
Flags controlling the behavior of the resolver. See below for a description of possible flag values.

ARES_OPT_TIMEOUT
int *timeout***;**
The number of seconds each name server is given to respond to a query on the first try. (After the first try, the timeout algorithm becomes more complicated, but scales linearly with the value of *timeout*.) The default is five seconds.

ARES_OPT_TRIES
int *tries***;**
The number of tries the resolver will try contacting each name server before giving up. The default is four tries.

ARES_OPT_NDOTS
int *ndots***;**
The number of dots which must be present in a domain name for it to be queried for "as is" prior to querying for it with the default domain extensions appended. The default value is 1 unless set otherwise by resolv.conf or the RES_OPTIONS environment variable.

ARES_OPT_PORT **unsigned short** *port***;**
The port to use for queries (both TCP and UDP), in network byte order. The default value is 53 (in network byte order), the standard name service port.

ARES_OPT_SERVERS
struct in_addr ****servers*;**
int *nservers***;**
The list of servers to contact, instead of the servers specified in resolv.conf or the local named.

ARES_OPT_DOMAINS
char *****domains*;**
int *ndomains***;**
The domains to search, instead of the domains specified in resolv.conf or the domain derived from the kernel hostname variable.

ARES_OPT_LOOKUPS

char ****lookups*;**

The lookups to perform for host queries. *lookups* should be set to a string of the characters "b" or "f", where "b" indicates a DNS lookup and "f" indicates a lookup in the hosts file.

The *flags* field should be the bitwise or of some subset of the following values:

ARES_FLAG_USEVC Always use TCP queries (the "virtual circuit") instead of UDP queries. Normally, TCP is only used if a UDP query yields a truncated result.

ARES_FLAG_PRIMARY

Only query the first server in the list of servers to query.

ARES_FLAG_IGNTC If a truncated response to a UDP query is received, do not fall back to TCP; simply continue on with the truncated response.

ARES_FLAG_NORECURSE

Do not set the "recursion desired" bit on outgoing queries, so that the name server being contacted will not try to fetch the answer from other servers if it doesn't know the answer locally.

ARES_FLAG_STAYOPEN

Do not close communciations sockets when the number of active queries drops to zero.

ARES_FLAG_NOSEARCH

Do not use the default search domains; only query hostnames as-is or as aliases.

ARES_FLAG_NOALIASES

Do not honor the HOSTALIASES environment variable, which normally specifies a file of hostname translations.

ARES_FLAG_NOCHECKRESP

Do not discard responses with the SERVFAIL, NOTIMP, or REFUSED response code or responses whose questions don't match the questions in the request. Primarily useful for writing clients which might be used to test or debug name servers.

RETURN VALUES

ares_init or *ares_init_options* can return any of the following values:

ARES_SUCCESS

Initialization succeeded.

ARES_EFILE A configuration file could not be read.

ARES_ENOMEM

The process's available memory was exhausted.

SEE ALSO

ares_destroy(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_mkquery − Compose a single-question DNS query buffer

SYNOPSIS

#include <ares.h>

int ares_mkquery(const char ****name*, int** *dnsclass***, int** *type***,**
unsigned short *id***, int** *rd***, char *****buf***,**
int ****buflen***)

DESCRIPTION

The **ares_mkquery** function composes a DNS query with a single question. The parameter *name* gives the query name as a NUL-terminated C string of period-separated labels optionally ending with a period; periods and backslashes within a label must be escaped with a backlash. The parameters *dnsclass* and *type* give the class and type of the query using the values defined in **<arpa/nameser.h>**. The parameter *id* gives a 16-bit identifier for the query. The parameter *rd* should be nonzero if recursion is desired, zero if not. The query will be placed in an allocated buffer, a pointer to which will be stored in the variable pointed to by *buf*, and the length of which will be stored in the variable pointed to by *buflen*. It is the caller's responsibility to free this buffer using **ares_free_string** when it is no longer needed.

RETURN VALUES

ares_mkquery can return any of the following values:

ARES_SUCCESS
Construction of the DNS query succeeded.

ARES_EBADNAME
The query name *name* could not be encoded as a domain name, either because it contained a zero-length label or because it contained a label of more than 63 characters.

ARES_ENOMEM
Memory was exhausted.

SEE ALSO

ares_expand_name(3), **ares_free_string**(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_parse_a_reply − Parse a reply to a DNS query of type A into a hostent

SYNOPSIS

#include <ares.h>

int ares_parse_a_reply(const unsigned char ****abuf*, int** *alen***,**
struct hostent *****host*);**

DESCRIPTION

The **ares_parse_a_reply** function parses the response to a query of type A into a **struct hostent**. The parameters *abuf* and *alen* give the contents of the response. The result is stored in allocated memory and a pointer to it stored into the variable pointed to by *host*. It is the caller's responsibility to free the resulting host structure using **ares_free_hostent**(3) when it is no longer needed.

RETURN VALUES

ares_parse_a_reply can return any of the following values:

ARES_SUCCESS
The response was successfully parsed.

ARES_EBADRESP
The response was malformatted.

ARES_ENODATA
The response did not contain an answer to the query.

ARES_ENOMEM
Memory was exhausted.

SEE ALSO

ares_gethostbyname(3), **ares_free_hostent**(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_parse_ptr_reply − Parse a reply to a DNS query of type PTR into a hostent

SYNOPSIS

#include <ares.h>

int ares_parse_ptr_reply(const unsigned char ****abuf*, int** *alen***,**
const void ****addr*, int** *addrlen***, int** *family***,**
struct hostent *****host*);**

DESCRIPTION

The **ares_parse_ptr_reply** function parses the response to a query of type PTR into a **struct hostent**. The parameters *abuf* and *alen* give the contents of the response. The parameters *addr*, *addrlen*, and *family* specify which address was queried for; they are not used to verify the response, merely used to fill in the address of the **struct hostent**. The resulting **struct hostent** is stored in allocated memory and a pointer to it stored into the variable pointed to by *host*. It is the caller's responsibility to free the resulting host structure using **ares_free_hostent**(3) when it is no longer needed.

RETURN VALUES

ares_parse_ptr_reply can return any of the following values:

ARES_SUCCESS
The response was successfully parsed.

ARES_EBADRESP
The response was malformatted.

ARES_ENODATA
The response did not contain an answer to the query.

ARES_ENOMEM
Memory was exhausted.

SEE ALSO

ares_gethostbyaddr(3), **ares_free_hostent**(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_process − Process events for name resolution

SYNOPSIS

#include <ares.h>

void ares_process(ares_channel *channel***, fd_set ***read_fds***,**
fd_set ***write_fds*)**

DESCRIPTION

The **ares_process** function handles input/output events and timeouts associated with queries pending on the name service channel identified by *channel*. The file descriptor sets pointed to by *read_fds* and *write_fds* should have file descriptors set in them according to whether the file descriptors specified by **ares_fds**(3) are ready for reading and writing. (The easiest way to determine this information is to invoke **select** with a timeout no greater than the timeout given by **ares_timeout**(3)).

The **ares_process** function will invoke callbacks for pending queries if they complete successfully or fail.

EXAMPLE

The following code fragment waits for all pending queries on a channel to complete:

```
int nfds, count;
fd_set readers, writers;
struct timeval tv, *tvp;

while (1)
  {
    FD_ZERO(&readers);
    FD_ZERO(&writers);
    nfds = ares_fds(channel, &readers, &writers);
    if (nfds == 0)
      break;
    tvp = ares_timeout(channel, NULL, &tv);
    count = select(nfds, &readers, &writers, NULL, tvp);
    ares_process(channel, &readers, &writers);
  }
```

SEE ALSO

ares_fds(3), **ares_timeout**(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_query − Initiate a single-question DNS query

SYNOPSIS

#include <ares.h>

typedef void (*ares_callback)(void * *arg***, int** *status***,**
unsigned char * *abuf***, int** *alen***)**

void ares_query(ares_channel *channel***, const char *** *name***,**
int *dnsclass***, int** *type***, ares_callback** *callback***,**
void * *arg***)**

DESCRIPTION

The **ares_query** function initiates a single-question DNS query on the name service channel identified by *channel*. The parameter *name* gives the query name as a NUL-terminated C string of period-separated labels optionally ending with a period; periods and backslashes within a label must be escaped with a backslash. The parameters *dnsclass* and *type* give the class and type of the query using the values defined in **<arpa/nameser.h>**. When the query is complete or has failed, the ares library will invoke *callback*. Completion or failure of the query may happen immediately, or may happen during a later call to **ares_process**(3) or **ares_destroy**(3).

The callback argument *arg* is copied from the **ares_query** argument *arg*. The callback argument *status* indicates whether the query succeeded and, if not, how it failed. It may have any of the following values:

ARES_SUCCESS The query completed successfully.

ARES_ENODATA The query completed but contains no answers.

ARES_EFORMERR The query completed but the server claims that the query was malformatted.

ARES_ESERVFAIL The query completed but the server claims to have experienced a failure. (This code can only occur if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_ENOTFOUND
The query completed but the queried-for domain name was not found.

ARES_ENOTIMP The query completed but the server does not implement the operation requested by the query. (This code can only occur if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_EREFUSED The query completed but the server refused the query. (This code can only occur if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_EBADNAME The query name *name* could not be encoded as a domain name, either because it contained a zero-length label or because it contained a label of more than 63 characters.

ARES_ETIMEOUT No name servers responded within the timeout period.

ARES_ECONNREFUSED
No name servers could be contacted.

ARES_ENOMEM Memory was exhausted.

ARES_EDESTRUCTION
The name service channel *channel* is being destroyed; the query will not be completed.

If the query completed (even if there was something wrong with it, as indicated by some of the above error codes), the callback argument *abuf* points to a result buffer of length *alen*. If the query did not complete,

abuf will be NULL and *alen* will be 0.

SEE ALSO

ares_process(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_search − Initiate a DNS query with domain search

SYNOPSIS

#include <ares.h>

typedef void (*ares_callback)(void * *arg***, int** *status***,**
unsigned char * *abuf***, int** *alen***)**

void ares_search(ares_channel *channel***, const char *** *name***,**
int *dnsclass***, int** *type***, ares_callback** *callback***,**
void * *arg***)**

DESCRIPTION

The **ares_search** function initiates a series of single-question DNS queries on the name service channel identified by *channel*, using the channel's search domains as well as a host alias file given by the HOSTALIAS environment variable. The parameter *name* gives the alias name or the base of the query name as a NUL-terminated C string of period-separated labels; if it ends with a period, the channel's search domains will not be used. Periods and backslashes within a label must be escaped with a backslash. The parameters *dnsclass* and *type* give the class and type of the query using the values defined in **<arpa/nameser.h>**. When the query sequence is complete or has failed, the ares library will invoke *callback*. Completion or failure of the query sequence may happen immediately, or may happen during a later call to **ares_process**(3) or **ares_destroy**(3).

The callback argument *arg* is copied from the **ares_search** argument *arg*. The callback argument *status* indicates whether the query sequence ended with a successful query and, if not, how the query sequence failed. It may have any of the following values:

ARES_SUCCESS A query completed successfully.

ARES_ENODATA No query completed successfully; when the query was tried without a search domain appended, a response was returned with no answers.

ARES_EFORMERR A query completed but the server claimed that the query was malformatted.

ARES_ESERVFAIL No query completed successfully; when the query was tried without a search domain appended, the server claimed to have experienced a failure. (This code can only occur if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_ENOTFOUND
No query completed successfully; when the query was tried without a search domain appended, the server reported that the queried-for domain name was not found.

ARES_ENOTIMP A query completed but the server does not implement the operation requested by the query. (This code can only occur if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_EREFUSED A query completed but the server refused the query. (This code can only occur returned if the **ARES_FLAG_NOCHECKRESP** flag was specified at channel initialization time; otherwise, such responses are ignored at the **ares_send**(3) level.)

ARES_TIMEOUT No name servers responded to a query within the timeout period.

ARES_ECONNREFUSED
No name servers could be contacted.

ARES_ENOMEM Memory was exhausted.

ARES_EDESTRUCTION
The name service channel *channel* is being destroyed; the query will not be completed.

If a query completed successfully, the callback argument *abuf* points to a result buffer of length *alen*. If the query did not complete successfully, *abuf* will usually be NULL and *alen* will usually be 0, but in some cases an unsuccessful query result may be placed in *abuf*.

SEE ALSO

ares_process(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_send − Initiate a DNS query

SYNOPSIS

#include <ares.h>

typedef void (*ares_callback)(void *_arg_, int _status_,
unsigned char *_abuf_, int _alen_)

void ares_send(ares_channel *channel*, **const unsigned char *_qbuf_,**
int *qlen*, **ares_callback** *callback*, **void *_arg_)**

DESCRIPTION

The **ares_send** function initiates a DNS query on the name service channel identified by *channel*. The parameters *qbuf* and *qlen* give the DNS query, which should already have been formatted according to the DNS protocol. When the query is complete or has failed, the ares library will invoke *callback*. Completion or failure of the query may happen immediately, or may happen during a later call to **ares_process**(3) or **ares_destroy**(3).

The callback argument *arg* is copied from the **ares_send** argument *arg*. The callback argument *status* indicates whether the query succeeded and, if not, how it failed. It may have any of the following values:

ARES_SUCCESS The query completed.

ARES_EBADQUERY
The query buffer was poorly formed (was not long enough for a DNS header or was too long for TCP transmission).

ARES_ETIMEOUT No name servers responded within the timeout period.

ARES_ECONNREFUSED
No name servers could be contacted.

ARES_ENOMEM Memory was exhausted.

ARES_EDESTRUCTION
The name service channel *channel* is being destroyed; the query will not be completed.

If the query completed, the callback argument *abuf* points to a result buffer of length *alen*. If the query did not complete, *abuf* will be NULL and *alen* will be 0.

Unless the flag **ARES_FLAG_NOCHECKRESP** was set at channel initialization time, **ares_send** will normally ignore responses whose questions do not match the questions in *qbuf*, as well as responses with reply codes of **SERVFAIL**, **NOTIMP**, and **REFUSED**. Unlike other query functions in the ares library, however, **ares_send** does not inspect the header of the reply packet to determine the error status, so a callback status of **ARES_SUCCESS** does not reflect as much about the response as for other query functions.

SEE ALSO

ares_process(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_strerror − Get the description of an ares library error code

SYNOPSIS

#include <ares.h>

const char *ares_strerror(int *code***, char ****memptr***)*

DESCRIPTION

The **ares_strerror** function gets the description of the ares library error code *code*, returning the result as a NUL-terminated C string. A pointer to allocated data necessary to compose the error description may be stored in the variable pointed to by *memptr*. It is the caller's responsibility to invoke **ares_free_errmem**(3) with the value of that variable when the error description is no longer needed.

SEE ALSO

ares_free_errmem(3)

AUTHOR

Greg Hudson, MIT Information Systems

NAME

ares_fds − Get file descriptors to select on for name service

SYNOPSIS

#include <ares.h>

struct timeval *ares_timeout(ares_channel *channel***,**
struct timeval ***maxtv*, struct timeval ***tvbuf***)**

DESCRIPTION

The **ares_timeout** function determines the maximum time for which the caller should wait before invoking **ares_process**(3) to process timeouts. The parameter *maxtv* specifies a existing maximum timeout, or **NULL** if the caller does not wish to apply a maximum timeout. The parameter *tvbuf* must point to a writable buffer of type **struct timeval**. It is valid for *maxtv* and *tvbuf* to have the same value.

If no queries have timeouts pending sooner than the given maximum timeout, **ares_timeout** returns the value of *maxtv;* otherwise **ares_timeout** stores the appropriate timeout value into the buffer pointed to by *tvbuf* and returns the value of *tvbuf*.

SEE ALSO

ares_fds(3), **ares_process**(3)

AUTHOR

Greg Hudson, MIT Information Systems