NAME

ares_parse_caa_reply − Parse a reply to a DNS query of type CAA

SYNOPSIS

```
#include <ares.h>

int ares_parse_caa_reply(const unsigned char* abuf, int alen,
                         struct ares_caa_reply **caa_out);
```

DESCRIPTION

The **ares_parse_caa_reply** function parses the response to a query of type CAA into a linked list (one element per sub-string) of *struct ares_caa_reply* 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 *caa_out*. It is the caller's responsibility to free the resulting *caa_out* structure when it is no longer needed using the function **ares_free_data(3)**

The structure *ares_caa_reply(3)* contains the following fields:

```
struct ares_caa_reply {
  struct ares_caa_reply *next;
  int                 critical;
  unsigned char       *property;
  size_t              plength; /* plength excludes null */
  unsigned char       *value;
  size_t              length;  /* length excludes null */
};
```

RETURN VALUES

ares_parse_caa_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.

EXAMPLE

```
#include <arpa/inet.h>
#include <time.h>
#include <sys/time.h>
#include <netdb.h>

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#include "ares.h"

static void dns_callback(void *arg,
                         int status,
                         int timeouts,
                         unsigned char *abuf,
                         int alen)
 {
```

```
    struct ares_caa_reply *caa_out;
    int err;

    err = ares_parse_caa_reply (abuf, alen, &caa_out);
    if (err == ARES_SUCCESS)
      {
       struct ares_caa_reply *caa_curr;
       for (caa_curr=caa_out; caa_curr; caa_curr=caa_curr->next)
         printf ("%s. CAA %i %s \"%s\"\n", arg,
                                     caa_curr->critical,
                                     caa_curr->property,
                                     caa_curr->value);
      }
    else
      {
       printf ("err=%i\n", err);
      }
    ares_free_data (caa_out);
  }

static void main_loop(ares_channel_t **channel)
  {
    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);
      }
  }

int main(int argc, char **argv)
  {
    const char *sversion;
    int iversion;
    int err;

    sversion = ares_version (&iversion);
    printf ("c-ares version %s\n", sversion);

    char *domain = "wikipedia.org";
    if (argc > 1)
      domain = argv[1];

    ares_channel_t *channel;
    if ((err = ares_init (&channel)) != ARES_SUCCESS)
      {
```

```
    printf ("ares_init() failed (%i)\n", err);
    exit (EXIT_FAILURE);
  }

 ares_query (channel, domain,
         1,   /* ns_c_in */
         257, /* T_CAA */
         dns_callback, domain);

 main_loop (&channel);

 ares_destroy (channel);

 exit (EXIT_SUCCESS);
}
```

AVAILABILITY

This function was first introduced in c-ares version 1.17.0.

SEE ALSO

ares_query(3) **ares_free_data**(3)