libacarsd 1.46

ACARS Decoder Library

©. 2003,2004 by KjM <acarsd@acarsd.org>

It is not allowed to monitor all radio frequencies in every country!
Software that makes use of libacarsd as decoder should make the end users aware of this!
The author cannot be held liable for any legal consequences

1.)Preface

libacarsd is a free library for decoding of ACARS signals into readable messages. ACARS is an abbreviation for:

Aircraft **C**ommunication **A**ddressing and **R**eporting **S**ystem

In order to use libacarsd you need experience in C programming under Windows or under Linux.

On request I may also compile libacarsd as library for other systems.

I can only guarantee that libacarsd occupies disk space and is virus-free in the original package. It is impossible to guarantee that any software functions correctly, since errors can never be excluded. Libacarsd has been successfully tested for months on different Windows and Linux systems, so problems should be quite rare.

In case of problems or to suggest enhancements, please mail to acarsd@acarsd.org.

2.)Integration in programs

You may include libacarsd in any **non-commercial** product. Generally you should refer expressly to your use of libacarsd in the documentation or the tips to your program. If you want to use libacarsd in a commercial application, please get in touch by mail to acarsd@acarsd.org

3.)Contents of the package

The following files are included in the libacarsd package:

libacarsd.dll	dynamic library for Windows
libacarsd.a	static library for Windows
libacarsd.so	dynamic library for Linux
libacarsd.la	static library for Linux
libacarsd.h	header file for libacarsd
sndfile.snd	sample sound file with ACARS messages
test.c	sample C Source

Since version 1.45 the package contains both the libraries for Windows and for Linux. You can test libacarsd on your system with the provided sample source code (test.c). Additional details follow later in this documentation.

4.)libacarsd functions

The following functions are offered by libacarsd and can be used by your application:

libacarsd *acarsd_init(const int buf, const int sample, const int pass);

This is the libaracsd function you must call first. You pass 3 integer values to this function:

buf - Size of the sound buffer which is to be processed
The buffer must not be smaller than the value indicated here, otherwise memory errors will result!

sample - Sampling rate for the sound buffer
The current version of libacarsd supports the following sampling rates:
19500 Hz, 8bit
22050 Hz, 8bit
44100 Hz, 8bit
These are defined through macros in libacarsd.h.
19500Hz = STREAM19500
22050Hz = STREAM22050
44100Hz = STREAM44100
16 bit sound is currently not supported!

pass - Number of passes per sound file.
libacarsd is a real-time multi-pass decoder. Each sound buffer is processed several times with different values until a message has been successfully decoded. On systems with a weak CPU you should not run through more than 6 passes. The default is 7 and a maximum of 10 is possible.

acarsd_init returns a pointer to a libacarsd structure, or NULL in case of an error.

If libacarsd returns NULL you should not make calls to any other functions in the library!

acarsd_destroy(libacarsd *Lib);

This function finishes the use of libacarsd and the libacarsd structure as well as all internal variables are erased.

Following a call to this function you have to re-initialize libacarsd if you would like to make use of libacarsd again!

You must call this function:
- *before terminating your program*
  or
- *when you do not need libacarsd any more*

libacarsd makes use of memory that is only freed up again by calling this function!

int ACARS_Decoder(libacarsd *Lib, unsigned char *buff);

This function tries to convert the buffer (*buff) to readable ACARS messages.

In order to avoid unnecessary calls, this function does not verify that libacarsd has been correctly initialized. Crashes may result if you call ACARS_Decoder() without initializing libacarsd correctly first!

ACARS_Decoder() returns an integer value which can be either ACS_NOTHING or ACD_SUCCESS. If the result is ACS_NOTHING, then nothing was decoded. If ACARS_Decoder() returns ACD_SUCCESS, you need to process the available message in a defined structure, and then **call ACARS_Decoder() again**, since there may be more than one message in one buffer!

int acarsd_goodoffset(libacarsd *Lib, int *num);

Following a return with ACD_SUCCESS you may determine the optimal message yourself from the codeholder structure, or you may leave this to libacarsd. You use the acarsd_goodoffset()function if you want libacarsd to take care of this for you.

As for nearly all functions, what you pass to acarsd_goodoffset() is the initialized libacarsd structure and a pointer to an integer variable.

acarsd_goodoffset() returns the following values:

ACARSD_VALIDATED	A message with correct checksum is present.
ACARSD_WITHCRC	An error-free message is present, but the checksum does not match.
ACARSD_BEST	The best available message is present. "Best available" means that the message has errors in it, but in the set of possible messages this is the one with fewest errors.
ACARSD_NONE	No usable message was decoded.

For all results except ACARSD_NONE, the offset of the readable message can be found in the integer variable indicated as a pointer.

If acarsd_goodoffset() returns ACARSD_NONE, you should not make use of the returned integer value, as it will contain -1 which is an invalid offset!

void acarsd_sorttables(libacarsd *Lib);

libacarsd makes internal use of a range of different code tables that are applied to the sound buffer. The applied code tables depend both on the air-band scanner used and/or the sound card. This function should be called after around 100 successfully decoded messages in order to achieve an optimal sorting of code tables and to reduce processor usage.
This function returns nothing.

char *acarsd_human_readable(libacarsd *Lib, const int force);

This function takes care of creating a readable ACARS message from the codeholder structure. If you use this function, you do not need to use acarsd_goodoffset(), since that function is called from inside acarsd_human_readable().

acarsd_human_readable() transforms the best possible message into the standard ACARS message format and returns the result as char*. In order to avoid memory problems, remember to free up memory by using free() when you have fished processing the result.

If you set force to TRUE (1, that is), messages with errors are also returned. If you set force to FALSE (0, that is), messages are only returned if the checksum is correct or if they are error-free after the libacarsd tests.

acarsd_human_readable() returns a NULL pointer if no message can be created.

***char *ACARS_VInfo**(libacarsd *Lib);*

This function produces a string which represents the volume of the sound buffer that was just processed. The sound buffer is split into blocks of 127 bytes and the volume is calculated for each block. The string always consists of 3 digit values and a divider (|). Example: 017|001|097|101|099|.

char *acarsd_errors_of_msg(libacarsd *Lib, const int offset);

This function returns the error flags that relate to the message referenced by offset. *You have to free the returned pointer of char!*

acarsd_errors_of_msg() returns NULL if offset has an invalid value or if no known errors are found.

The following errors have been defined in libacarsd.h:

MSG_ERR_ALL	although libacarsd correctly identified a header *<SYN><SYN><SOH>, the rest of the message could not be decoded
MSG_ERR_REG	The aircraft registration contains invalid characters. *Messages with correctly verified checksums may still contain 'strange' characters in the registration.*
MSG_ERR_LAB	The message label contains invalid characters
MSG_ERR_BLK	The block id contains invalid characters
MSG_ERR_MSG	The message number contains invalid characters
MSG_ERR_FLI	The flight number contains invalid characters
MSG_ERR_EXT	The message text part of the transmission contains invalid characters

acarsd_errors_of_msg() may return a strings like the following:

„MSG_ERR_BLK MSG_ERR_FLI MSG_ERR_EXT“

long acarsd_option(libacarsd *Lib, int option, int value);

With this function you may select or change different libacarsd options. If you combine option with ACARSD_GET (using OR), the appropriate option will be queried. If you combine option with ACARSD_SET, then the appropriate option will be set to the new value as defined by value.
It is not possible to set all options. acarsd_option() returns TRUE if a value was successfully set. FALSE is returned if a value could not be set. If you try to use an invalid option, -1 will be returned.

The following options are available:

Option	Purpose	Read	Write
ACARSD_CRCMODE	Turn on or off CRC verification	YES	NO
ACARSD_CODETABLES	Read/change the number of code tables used	YES	YES (1-8)
ACARSD_CODEPOS	Number of active entries in the codeholder structure	YES	NO
ACARSD_VOLUME	Calculated volume of the last buffers	YES (0-100)	NO
ACARSD_BUFSIZE	Size of the sound buffer	YES	YES
ACARSD_PASSES	Number of passes per sound buffer	YES	YES (1-10)
ACARSD_SAMPLE	Sampling rate of sound buffer	YES	YES
ACARSD_BUFFERS	Number of decoded buffers	YES	NO

For instance, in order to retrieve the number of decoded sound buffers, you can make the following call:

```
/* Output number of decoded buffers */
printf(„Number of decoded buffers: %ld\n“,
                    acarsd_option(Lib, ACARSD_GET|ACARSD_BUFFERS));
```

Lib is an initialized instance of libacarsd

5.)Available macros in libacarsd

libacarsd.h contains definitions for some macros that shorten or simplify calls. These macros are explained below:

agetopt(L,F);

This macro refers to the acarsd_option() function. The parameter L is an initialized instance of libacarsd and F is the appropriate option which is to be read.

asetopt(L,F);

This macro also refers to the acarsd_option()function. You can use this function to set options. The parameter L is an initialized instance of libacarsd and F is the appropriate option which is to be changed.

isuplink(L,O);

This macro examines if the message in Offset (referred to by O) is an uplink. *Uplinks are messages that are transmitted to aircraft by ground stations. They can only be received in the immediate vicinity of an airport.*

issquitter(L,O);

This macro examines if the message in Offset (referred to by O) is a Squitter. *Squitter messages can be regarded as a form of 'HERE I AM' message from the ground stations*.

6.)Available variables in libacarsd

libacarsd provides access to a set of variables. Firstly, it is the character translation table for characters < 32. This variable has been defined as an array of `char*` and may be referred to with `spec[0..31]`.

The second global variable is also an `array of char *` and contains the error designators that it is possible for a message to have. 8 elements which are adapted to the constant `MSG_ERR_*` have been defined.

In order to determine the exact version of libacarsd, 3 global integer variables are available:

```
libacarsdmajorversion          Major Version of libacarsd (current 1)
libacarsdminorversion          Minor Version of libacarsd (current 45)
libacarsdrevision              Revision number of libacarsd
```

Thus you may communicate to the users of your program which version of libacarsd is being used.

```
/* Output of version information */
printf(„Using libacarsd %d.%dRev%d\n“,
          libacarsdmajorversion,libacarsdminorversion,
          libacarsdrevision);
```

You can call this without initializing libacarsd

7.)The codeholder structure

The ACARS_Decoder() function stores all results in a structure which has been defined as element codeholder within the libacarsd structure.
Codeholder itself has been defined as Array of 100 so as to store the highest number of provided messages. *Codeholder is erased whenever ACARS_Decoder() is called, so you should read and process the values first*.

7.1.)Organization of the structure

Whenever ACD_SUCCESS has been returned from the ACARS_Decoder() function, the number of occupied codeholder structures is available in the variable Lib->acarsd_codepos. The corresponding messages are stored per codeholder. You may write the program logic yourself in order to find the best message, alternatively you may use the acarsd_goodoffset() function. If you write the logic yourself you should first look for messages where CRC = 0, meaning that the checksum was found to be correct. Absent any of these, you should then look for messages with no errors. If you do not find any of these either, the only thing you can do is to determine which message has the lowest number of errors.

Example:

```
    /* Get CRC checked messages */
     for (i=0;i<Lib->acarsd_codepos;i++) {
           if ((!Lib->codeholder[i].errors) && (Lib-
>codeholder[i].closed) &&
               (!Lib->codeholder[i].crc)) {
               m = i;
               crcok = TRUE;
               goto noerror;
           }
     }

     /* Get best message (CRC check failed) */
     for (i=0;i<Lib->acarsd_codepos;i++) {
           if ((!Lib->codeholder[i].errors) && (Lib-
>codeholder[i].closed)) {
               m = i;
               goto noerror;
           }
     }

     /* Search messages without errors */
     for (i=0;i<Lib->acarsd_codepos;i++) {
           if (!Lib->codeholder[i].errors) {
               m = i;
               goto noerror;
           }
     }

    /* Here errorhandling */
    return;
```

```
/* Label for good messages */
noerror:
...
```

In the example above there is first a search for a message with the correct checksum, then a search for one without errors and ending with either <ETX> with or <ETB>. If no such message is found, the message with the lowest number of errors is being selected. However, `acarsd_goodoffset()` can do this for you. You would then get the value of Offset (`m` in the example above) in the pointer variable returned from the function `acarsd_goodoffset()`.

7.2.) Definition of the codeholder structure

You can derive the logic of the `codeholder` structure from the header file libacarsd.h.
The structure is shown here:

```
/* Typedefinition for a single char within ACARS stream */
typedef struct {
    unsigned char error; /* Error indicated by 1. No error = 0 */
    unsigned char data;  /* Decoded char - without error? see above */
} ACD_CONT;

/* Typedefinition for the complete ACARS stream */
typedef struct {
    int len;             /* Count of decoded chars */
    int errors;          /* Errorcounter */
    int closed;          /* Transmission correct closed */
    int flags;           /* Flags for this transmission */
    int squitter;        /* This is a Squitter message */
    int uplink;          /* This is a Uplink message */
    int lastpos;         /* Last position within soundstream */
    unsigned short crc;  /* Calcutated CRC. 0 if message is complete and
validated! */
    ACD_CONT c[1024];    /* Included typedef from above */
} ACD;
```

`ACD` corresponds to an element of the `codeholder` Array. The flags in `flags` correspond to the error flags `MSG_ERR_*`.

8.)Determining the sound volume

The sound volume is calculated automatically during the first processing of the sound buffer. The possible values are between **0-100** . This is the percent value of the actual soundstream.
You can obtain the value of acarsd_volume straight from the libacarsd structure or alternatively by using the following macro: `agetopt(Lib,ACARSD_VOLUME)`.

9.)Sample code

The functioning of libacarsd should be illustrated by the example below.

You can compile the example under Linux by using the following command:

```
gcc test.c -s -Wall -O3 -o libacarsdtest -L./ -lacarsd
```

Under Windows I have tested the library under CYGWIN (www.cygwin.com) and under MINGW (www.mingw.org). For both of these you can use the same command as shown above.

Following a successful compilation you can run the program `libacarsdtest`. Assuming you are using the included sound file (`sndfile.snd`), you should get the following output:

```
Libacarsd version: 1.46Rev2
CRC MODE IS ENABLED: YES
CHECKSUM VALIDATED ON MESSAGE
*<SYN><SYN><SOH>2.VH-OGE<NAK>
80<STX>4026QF0039/MEL.<CR><LF>MVA<CR><LF>QF39/13.VHOGE.MEL<CR><LF>AD082
9/0840 EA1129 AKL<ETX>
---
CHECKSUM VALIDATED ON MESSAGE
*<SYN><SYN><SOH>2.VH-OGE<NAK> H1<STX>D029QF0039#1BBTKO<CR><LF>0840
13JAN04 MEL AKL P207 P165 00480 276 145727 04<CR><LF>L 1022 1035 872
8975 1197 61 095 132 05H 2000<CR><LF>R 1025 1043 858 8964 1199 68 098
127 09B 200<CR><LF>1078 1023 096 032P430<CR><LF>1079 1025 093<ETX>
---
CHECKSUM VALIDATED ON MESSAGE
*<SYN><SYN><SOH>2.VH-OGE<NAK>
86<STX>5208QF0039/AKL.<CR><LF>ARI<CR><LF>AKL ETAB 11:40<CR><LF>WH/CH 01
UN/MNR 02<CR><LF>GIDAY<ETX>
---
CHECKSUM VALIDATED ON MESSAGE
*<SYN><SYN><SOH>2.VH-OGE<NAK> H1<STX>D063QF0034#1FBFCP<CR><LF>2107
14JAN04 AKL MEL P010 M272 24972 759 129461 09<CR><LF>R 0235<CR><LF>NO
087 098 011 000 000 000 000 000 <CR><LF>PC 085 013 000 000 000 000 000
000 <ETX>
---
CHECKSUM VALIDATED ON MESSAGE
*<SYN><SYN><SOH>2.VH-OGE<ACK> _ <STX>1243QF0033<ETX>
```

[...]

9.1.)The source code of test.c

And here is the source code in the file test.c which has been provided for you:

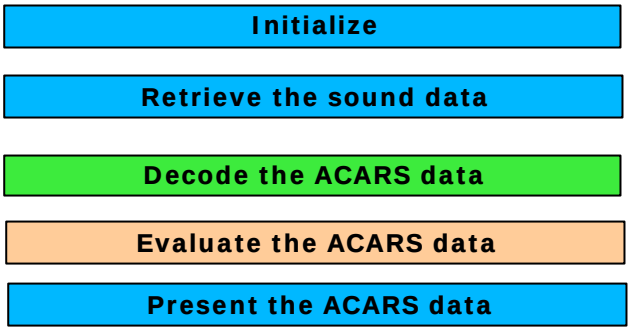
```
/*
  Sample code to test the function of libacarsd
  Written by KjM <acarsd@acarsd.org>
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

/* Include from libacarsd */
#include "libacarsd.h"

/* Filename for testfiles */
#define FILENAME "./sndfile.snd"

/* Simply test */
int main(int argc, char **args) {
    libacarsd *L;
    int i , j, num, succ = 0, err = 0;
    FILE *F;
    unsigned char *c;
    char *a;
    struct stat st;

    /* Get Filesize */
    if (stat(FILENAME,&st)) {
            fprintf(stderr,"Cannot stat() %s\n",FILENAME);
            return 255;
    }

    /* Init lib
       1. Param: Size of buffer
       2. Param: Sampling rate
       3. Param: Number of passes */
    if (!(L = acarsd_init(st.st_size,STREAM19500,8))) {
            printf("acarsd_init() error\n");
            return 255;
    }

    /* Something about libacarsd */
    fprintf(stderr,"Libacarsd version: %d.%dRev%d\n",
                libacarsdmajorversion,libacarsdminorversion,
                libacarsdrevision);

    /* CRC/FCB Check is activ? */
    fprintf(stderr,"CRC MODE IS ENABLED: %s\n",
                (acarsd_option(L,ACARSD_GET|ACARSD_CRCMODE,0))?"YES":"NO");

    /* Disable CRC check */
    asetopt(L,ACARSD_CRCMODE,0);

    /* Set 10 passes on each buffer */
    asetopt(L,ACARSD_PASSES,10);

    /* Use 9 codetables */
    asetopt(L,ACARSD_CODETABLES,9);

    /* Get memory for sample soundfile */
    c = malloc(st.st_size);
    if ((F = fopen(FILENAME,"rb"))) {
            fread(c,1,st.st_size,F); fclose(F);
    } else {
```

```
            /* Free the stream buffer memory */
            free(c);
            printf("error reading soundfile 'sndfile.snd'\n");

            /* Destroy lib */
            acarsd_destroy(L);
            return 254;
    }
```

```
    /* Try to find all messages from soundfile */
    while (ACARS_Decoder(L,c)) {

            /* Get the best message */
            num = acarsd_goodoffset(L,&i);

            if (num != ACARSD_NONE) {
                if (!L->codeholder[i].crc) {
                        printf("CHECKSUM VALIDATED ON MESSAGE\n");
                        succ++;
                } else {
                        if ((a = acarsd_errors_of_msg(L, i))) {
                            printf("Errors: %s\n",a); free(a);
                        }
                        err++;
                }
                for (j=0;j<=L->codeholder[i].len;j++) {
                        if (L->codeholder[i].c[j].data < 32)
                            fprintf(stdout,"<%s>",spec[L->codeholder[i].c[j].data]);
                        else
                            fprintf(stdout,"%c",L->codeholder[i].c[j].data);
                }
                fprintf(stdout,"\n---\n");
            }
    }

    /* Print the volume of the soundstream */
    fprintf(stderr,"Volume: %ld / Buffers: %ld\n",
                agetopt(L,ACARSD_VOLUME),agetopt(L,ACARSD_BUFFERS));

    for (i=0;i<agetopt(L,ACARSD_CODETABLES);i++) {
            printf("Codetable #%d - %ld good messages\n",
                   i+1,L->acarsd_utable[i]);
    }

    printf("+----+----+\n| %02d | %02d |\n+----+----+\n",succ,err);

    /* Destroy libacarsd */
    acarsd_destroy(L);

    /* Free the stream buffer memory */
    free(c);
    return 0;
}
```

10.)Which tasks will libacarsd do for you?

If you would like to program an ACARS decoder yourself, then libacarsd will simplify your work considerably (`I spent weeks working on the algorithms`). What you have to do is the following:



The steps shown above against a blue background you will have to program yourself. The step shown in green will be handled completely by libacarsd. The step marked in orange can either be handled by libacarsd or you may program it yourself.

If you organize the work around these 5 main steps, libacarsd will take care of **up to 40%** of the work involved in developing a fully functioning and effective ACARS decoder.

11.)Licenses

libacarsd can be used free of charge in *non-commercial programs*, but please make a reference to the use of libacarsd. If possible, please also include a link to http://www.acarsd.org/libacarsd.html.
If you wish to use libacarsd in a commercial application you will need to purchase a license. Terms and prices for such a license are available on request.

12.)Source code from other programs

libacarsd does not make use of any source code from other programmers! I have personally made all routines in libacarsd.

13.)Latest versions

You can always download the latest versions from the homepage http://www.acarsd.org.

14.)Acknowledgements

Many thanks to Kjell Fuglestad and François Guillet. Kjell provided information about Squitters and uplink messages (unfortunately I am unable to receive such messages) and François gave me the information required in order to verify the ACARS checksums.

15.) References

libacarsd works successful with the following applications:

- acarsd – Free ACARS Decoder for Linux and Windows (www.acarsd.org)

If you wish that your application is also listed here, feel free to mail me.