Communication components

Frantisek Vacek
CTU

Pavel Pisa
CTU
# Table of Contents

1. CAN/CANopen user guide .............................................................................1
   Introduction .................................................................................................1
   Virtual CAN API (VCA) and libvca ..............................................................1
     Description and implementation issues ......................................................1
   CAN device ..................................................................................................5
     CAN slave ...............................................................................................5
     CAN master .............................................................................................7
   CAN monitor ................................................................................................7
     Description and implementation issues ......................................................7
     CanMonitor testing ..................................................................................8
   Installation ...................................................................................................10
   API / Compatibility ....................................................................................11
     VCA base API ..........................................................................................11
     SDO processing API ..................................................................................22
     PDO processing API ..................................................................................33
     OD access API ..........................................................................................42
     libulut API ..............................................................................................51
     libsuiut API ..............................................................................................118
Chapter 1. CAN/CANopen user guide

Introduction

The CAN/CANopen component consists of four main parts.

- LinCAN driver
- Virtual CAN API (VCA) and libvca
- CAN device
- CAN monitor

Virtual CAN API (VCA) and libvca

Description and implementation issues

The libvca consists of five parts.

- VCA base (vca_base.c)
- Object dictionary acces (vca_od.c)
- SDO processing (vcasdo_fsm.c)
- PDO processing (vcapdo.c)
- Miscellaneous and utility functions

The main idea of VCA is to have only one interface between application/library and CAN driver (LinCAN). The access to the CAN driver is different in RTLinux and Linux user space. While the function calls are used in RTLinux, the user space application uses /dev/can device. This is why we need VCA.

Next figures shows how to incorporate all the parts of libvca to work together in both spaces.

Figure 1-1. Usage of libvca in the Linux user space
VCA base

VCA base is primarily a set of primitive functions used open/close CAN driver and send/receive CAN message. Where are also couple of IOCTLs shielded by VCA base. Most of VCA base is implemented in vca_base.c. One part of VCA base is also logging support used by whole CAN/CANopen component. Log support is implemented in vca_log.c.

Next code fragment shows simple usage of VCA primitives.

```c
vca_handle_t canhandle;
const char *candev = "/dev/can0";
printf("Opening %s\n", candev);
if (vca_open_handle(&canhandle, candev, NULL, 0) != VCA_OK) {
    perror("open");
    exit(1);
}

while (1) {
    struct canmsg_t readmsg;
    int ret = vca_rec_msg_seq(canhandle, &readmsg, 1);
    if(ret < 0) {
        vca_log("cantest", LOG_ERR, "Error reading message from '%s\n", candev);
    }
    else {
        printf("Received message #\n: id:%lx data:[",i,readmsg.id);
        for(n=0 ; n<readmsg.length ; n++) {
            if(n > 0) printf(" ");
            printf("%.2x", (unsigned char)readmsg.data[n]);
        }
        printf("]\n"),
        i++;
    }
}
```

Object dictionary access

The Object Dictionary (OD) is implemented as a GAVL tree of vcaod_object_t objects. It can be also a GSA array for embedded devices with small amount of memory, but this feature is not implemented yet.

There are three main functions for access to objects in OD.

- vcaod_find_object
- vcaod_get_value
- vcaod_set_value
Chapter 1. CAN/CANopen user guide

Using this function one can read or change objects in OD.

```c
vca_handle_t canhandle;
const char *candev = "/dev/can0";
printf("Opening %s\n", candev);
if (vca_open_handle(&canhandle, candev, NULL, 0) != VCA_OK) {
    perror("open");
    exit(1);
}

while (1) {
    struct canmsg_t readmsg;
    int ret = vca_rec_msg_seq(canhandle, &readmsg, 1);
    if(ret < 0) {
        vca_log("cantest", LOG_ERR, "Error reading message from '%s\n', candev);
    } else {
        printf("Received message #%lu: id:%lx data:[", i, readmsg.id);
        for(n=0 ; n<readmsg.length ; n++) {
            if(n > 0) printf(" ");
            printf("%.2x", (unsigned char)readmsg.data[n]);
        }
        printf("]"
    }
    i++;
}
```

**SDO processing**

The core structure for the SDO processing is a vcasdo_fsm_t. This structure holds all the status information about current SDO handshake and also other information like SDO COB IDs, node number etc. SDO processing library do not contain any synchronous call like select(), read(), write() etc. This aproach gives it independancy on used communication model. Next code example showes, how to deploy SDO library. Fragment is taken from canslave.c.

```
// slave SDO communication loop
vcasdo_fsm_t fsm;
// use default SDO COB IDs
vcasdo_init_fsm(&fsm, 0, 0, node);
while(1) {
    // read CAN driver loop
    struct canmsg_t readmsg;
    int ret = vca_rec_msg_seq(canhandle, &readmsg, 1); // 1.
    if(ret <= 0) continue;
    if(fsm->state == sdofsmIdle) { // 2.
        // init communicated data in fsm for new communication
        // load object data and prepare FSM for new communication handshake
        do { // 3.
            int cmd;
            vcasdo_read_multiplexor(readmsg.data + 1, &fsm->index, &fsm->subindex);
            cmd = VCA_SDO_GET_COMMAND(readmsg.data[0]);
            if(cmd == VCA_SDO_INIT_UPLOAD_R) {
                uint32_t abort_code;
                ul_dbuff_t *db = &fsm->data;
                vcaod_object_t *odo;
                int l;
                // load data from OD
                odo = vcaod_find_object(&od_root, fsm->index, fsm->subindex, &abort_code);
                if(!odo) {
                    mylog(LOG_ERR, "[%04x:%02x] not found, ABORTING transfer\n", fsm->index, fsm->subindex);
                    vcasdo_fsm_abort(fsm, abort_code);
                    break;
                }
            }
        }
    }
```

3
Chapter 1. CAN/CANopen user guide

// further function calls returns check are omitted for the example
}
l = vcaod_get_object_data_size(odo, &abort_code);
ul_dbuff_set_len(db, l);
// because object can be an array, we should set parameter array_index
l = vcaod_get_value(odo, fsm->subindex, db->data, db->len, &abort_code);
// FSM is prepared, make it run
vcasdo_fsm_run(fsm);
}
else if(cmd == VCA_SDO_INIT_DOWNLOAD_R) {
// in case of download nothing special should be done with FSM
vcasdo_fsm_run(fsm);
}
}
while(0);

// now run new communication or continue in previous one (segmented or block transfer)
// vcasdo_fsm_taste_msg() generate answer CAN message for incoming CAN message
// according to state of fsm
// if(fsm->state != sdofsmRun) vcasdo_fsm_taste_msg() returns -1 and it does not do anything more.
if(vcasdo_fsm_taste_msg(fsm, &readmsg) == 0) { // 4.
    // bad cobid
    mylog(LOG_DEB, "message REFUSED\n");
} else { // 5.
    if(fsm->state == sdofsmDone) {
        do {
            // SDO transfer complete
            mylog(LOG_INF, "SDO transfer done\n");
            if(!fsm->is_uploader) {
                // store downloaded data to OD
                uint32_t abort_code;if
                ul_dbuff_t *db = &fsm->data;
                int l;
                // store downloaded data to OD
                vcaod_object_t *odo;
                odo = vcaod_find_object(&od_root, fsm->index, fsm->subindex, &abort_code);
                l = vcaod_set_value(odo, fsm->subindex, db->data, db->len, &abort_code);
                // transfer is done, no answer will be sent to the CAN
                ul_dbuff_set_len(&fsm->data, 0);
            }
        } while(0);
    } else if(fsm->state == sdofsmAbort) { // SDO transfer aborted
        mylog(LOG_MSG, "SDO transfer ABORTED: error %x 's'", fsm->err_no, vcasdo_abort_msg(fsm->err_no));
    } else if(fsm->state == sdofsmError) { // SDO transfer error
        mylog(LOG_MSG, "SDO transfer ERROR: error %x 's'", fsm->err_no, vcasdo_abort_msg(fsm->err_no));
    } else if(fsm->state == sdofsmRun) { // SDO transfer RUNNING
        mylog(LOG_DEB, "SDO transfer RUNNING\n");
    } else { // unexpected state
        mylog(LOG_ERR, "SDO FSM unexpected state: %i", fsm->state);
        fsm->out_msg.length = 0;
    }
}

if(fsm->out_msg.length > 0) { // 7.
    // fsm->out_msg.length > 0 signals that message should be sent to CAN
    vca_send_msg_seq(canhandle, &fsm->out_msg, 1);
}
// if fsm is not still running reinit communication after all errors, aborting
if(fsm->state != sdofsmRun) {
The example above is long but lot of code is OD object getting/setting and extraordinary states logging. The main idea is following.

1. get one CAN message
2. check FSM state, if it is currently serving SDO communication or if it is idle
3. if FSM is idle, parse message, get SDO command and get data from OD in case of upload, than make FSM run.
4. give CAN message to the FSM’s vcasdo_fsm_taste_msg() function, it ether process message (and change FSM state in appropriate way) or simply refuses it.
5. if message is not refused, check FSM state again.
6. if fsm->out_msg contains data, send data to CAN
7. go to 1. again

CANopen master SDO communication uses the same library in a very similar manner (see canmaster.c). The difference is only in fact that master initialize communication (starts with send) while slave always starts with CAN message read. If the master wants to start SDO communication it should init SDO FSM for upload or download calling vcasdo_fsm_upload1() or vcasdo_fsm_download1().

PDO processing
PDO processing is made using a structure vcaPDOProcessor_t. PDO prosessor knows which OD objects are PDO mapped (because it is written in EDS) and it can store/retrieve them to/from OD automatically. Core function is vcaPDOProcessor_processMsg(). If one call this function with a message just read from CAN, PDO processor check if it is PDO object and it takes care about appropriate behaviour. For more details see canslave.c.

Miscellaneous and utility functions
This is set of help function to parse text, convert it to number or serialize CAN messages to human readable form.

CAN device

CAN slave

Description and implementation issues
Can slave consist of two main parts
• CANslave core
• HW module
Chapter 1. CAN/CANopen user guide

The core is a canslave part which is always the same. It is responsible for PDO & SDO communication and OD storage. Slave message loop takes CAN messages from CAN. Every message is passed to PDO processor, if it is not processed here than it is passed to SDO FSM (Service Data Object Finite State Machine). SDO FSM process message, if it is part of SDO communication frame or refuse it. If it is SDO, FSM makes all necessary actions (mainly OD exchange) and prepare CAN answer message. Slave message loop sends such a message back to CAN bus. CAN slave core has also timer (timer is missing in figure below), which is responsible for triggering of synchronous PDOs.

When one starts CAN slave, he should provide EDS and HDS file to it. Slave reads EDS file and it build OD tree according to its contents. In future work will be possible to compile parsed EDS directly to CAN slave core.

**EDS** Electronic Data Sheet file is a text file describing all objects in the slave object dictionary and its mapping into the PDOs. It has normalized form according to CiA Draft Standard 301.

**HDS** file contains information which CANopen object in OD is linked to which dinfo in HW module. It is a simple text file with following structure.

```
6000:01 /nascanhw/input01
6200:01 /nascanhw/output01
```

Every line of HDS contains OD object index and subindex and dinfo name to be connected to. Dinfos can be stored in an arbitrary tree like files in directories. First directory on dinfo path is name of DLL which contains this dinfo. For example /nascanhw/input01 is in libnascanhw.so. When HDS file is parsed all needed DLLs are dynamically loaded. That means that HW module can consist of more than one *.so file.

What is dinfo? Dinfo is generic structure for passing arbitrary data type among canslave components. Every process value has to have its dinfo in HW module. Every dinfo has getter and setter functions for the primitive data types. At present only the long int and ul_dbuff_t is supported.

You can see the dinfo table on figure inside the OD and also inside the HW module. During CAN device initialization some dinfo structures are allocated. There are two kind of them. HW dinfos resiststing in the driver module are initialized when module is loaded. Every object mentioned in HDS file has also HW dinfo reference in OD. When some object, that do not have HW dinfo (not connected to the hardware), is PDO mapped the fake dinfo is created for that object in OD because the PDO processor always use only dinfo API for access to any PDO processed object data whether it comes from HW module or not. All dinfo structures are reference counted, so they are destroyed automatically when they are not needed anymore.
Chapter 1. CAN/CANopen user guide

Testing
See the section called CanMonitor testing.

**CAN master**

**Description and implementation issues**
CAN master is very similar to the CAN slave. Big difference is in ability of CAN-master to communicate with hierarchically higher application via named pipes. This gives an application opportunity to communicate with CANmaster placed in the user space (via named pipes) or in the RT-linux space (via /dev/rtfxx).

For more information see canmaster.c.

Testing
See the section called CanMonitor testing.

**CAN monitor**

**Description and implementation issues**
Can monitor component consists of two parts. CAN proxy - **canmond** and Java GUI canmond client **CanMonitor**.

![Diagram](image.png)

**Figure 1-4. CAN monitor component**
Canmond is the heard of component. It works like CAN proxy, it is connected using named pipes to the canmaster and resends CANopen objects to the all connected applications. Actualy it is a TCP server listenning on port 10001. TCP connection allows clients to be placed wherever on Internet. One can for example read/send CAN messages using a Java applet on his HTML browser. Canmond uses named pipes for communication with canmaster because canmaster can be placed in kernel space (using /dev/rtfxx) or in user space and use arbitrary couple of named pipes. This decomposition gives us opurtunity place canmaster in every memory space.
CanMonitor is a GUI Java based application connected to the canmond using UNIX TCP socket. One can send/monitor CAN messages using it. If one has slave EDS (Electronic Data Sheet), he can read/write device OD (Object Dictionary) just by clicking on the mouse.

**CanMonitor testing**

Program from this package does not need special installation. They can run from any directory. Just type `make` in `can/canmon` directory. And copy desired files from `can/_compiled` directory. If you want to compile only one component, type `make` in the component's directory.

Restrictions on versions of GNU C or glibc are not known in this stage of project.

Java SDK ver. 1.4 or above is recommended.

Component was tested with real CANopen device WAGO 750-307.

All VCA sources were compiled by GNU C ver. 3.2 and linked with glibc ver. 2.2.5.

All components were also tested with `canmaster` and `canslave` components. In following example is written how.

**Example 1 - connecting to real CANopen device**

Make sure, that CAN driver and the CAN monitor component is installed and works properly. Check that real CANopen device is connected to your CAN card.

Open two terminal windows. In first window launch `canmaster`.

You should see something like this

```
[fanda@mandrake bin]$ ./canmaster
CANMASTER - CANopen master
canmaster: entering state STATE_INITIALIZING
canmaster: entering state STATE_PREOPERATIONAL
canmaster: entering state STATE_OPERATIONAL
```

Than you should launch `canmond` on the same machine.

```
[fanda@mandrake bin]$ ./canmond
CANMOND - CAN monitor server
```

If you have a graphical environment with Java installed, you can launch `CanMonitor` with CANopen device EDS file issuing:

```
[fanda@mandrake bin]$ canmonitor -e nascan.eds
loading config from '/home/fanda/.canmonitor/CanMonitor.conf.xml'
connecting to localhost/127.0.0.1
connected OK
```

If everything works right, you should see Java application window.
Now you can load device EDS file and upload/download CANopen objects.

**Example 2 - connecting to canslave**

In this example **canslave** is tested, that means that you do not need any real CANopen device. Tested canslave can resist on same computer as canmaster on can be on other computer connected by CAN bus. If both programs resist on same computer make sure that CAN driver **lincan** was configured to send CAN messages to all other who have open CAN driver on same computer.

Do all steps from example 1. Open terminal window and launch **canslave**. You can launch more canslaves with different node numbers. Do not forget introduce *.EDS file name after `-e` switch in command line. You should see something like this

```
[fanda@mandrake bin]$ canslave -e nascan.eds
CANSLAVE - CAN slave
canslave: Opening CAN driver: /dev/can0
canslave: Opening EDS: nascan.eds
canslave: entering state STATE_INITIALIZING
canslave: SYNC COB_ID: 0, SYNC period: 0
canslave: entering state STATE_PREOPERATIONAL
canslave: entering state STATE_OPERATIONAL
```
Than you can load to the running CanMonitor next EDS file and work with canslave OD or scan the CAN bus traffic.

**Installation**

CAN component uses the OMK make system. There is no `./configure` script. The component can be built as a part of OCERA tree or as a standalone. If it is build as a standalone you should run script `can/switch2standalone`.

```
[fanda@lab3-2 can]$ ./switch2standalone
Default config for /utils/suiut
Default config for /utils/ulut
Default config for /utils/flib
Default config for /utils
Default config for /canvca/libvca
Default config for /canvca/cantest
Default config for /canvca
Default config for /candev/cpickle
Default config for /candev/nascanhw
Default config for /candev/canmon/canmon
Default config for /canmon
Default config for /lincan/src
Default config for /lincan/utils
Default config for /lincan
Default config for
```

To modify required configuration options, create "config.omk" file and add modified lines from "config.omk-default" file into it.

```
To build project, call simple "make"
```

GNU make program version 3.81beta1 or newer is required to build project check by "make --version" command.

Default configuration of any subcomponent can be changed by introducing a file `config.omk` in the subcomponent directory. Defines in this file simply beats defines in file `config.omk-default`, so you can put there only defines that are different that the default ones in the `config.omk-default`.

For example by default the building of Java application is disabled. That means that there is a line `CONFIG_OC_CANMONITOR=n` in the `config.omk-default`. If you have the Java SDK and the ant build system installed, add the line `CONFIG_OC_CANMONITOR=y` to the file `config.omk` to enable the Java applications to be build.

When you switch to standalone, you can build any particular component by running make in the component directory.

For more details see file `can/README.makerules`.


Programs in this package does not need special installation. They can run from any directory. Just type `make` in `can/canmon` directory and copy desired files wherever you want. The make process is an out source build. After make you can find your binaries in directory `can/_compiled/bin`. If you want to compile only one component,
type **make** in the component’s directory. That component and all components in subdirectories will be build.

Restrictions on versions of GNU C or glibc are not known in this stage of project but gcc ver >= 3.0 is recommended. Java SDK ver. 1.4 or above is also recommended (assert keyword support).

**API / Compatibility**

**VCA base API**

**struct canmsg_t**

**Name**

*struct canmsg_t* — structure representing CAN message

**Synopsis**

```c
struct canmsg_t {
    int flags;
    int cob;
    canmsg_id_t id;
    canmsg_tstamp_t timestamp;
    unsigned short length;
    unsigned char data[CAN_MSG_LENGTH];
};
```

**Members**

*flags*  
message flags **MSG_RTR** .. message is Remote Transmission Request, **MSG_EXT** .. message with extended ID, **MSG_OVR** .. indication of queue overflow condition, **MSG_LOCAL** .. message originates from this node.

*cob*  
communication object number (not used)

*id*  
ID of CAN message

*timestamp*  
not used

*length*  
length of used data

*data[CAN_MSG_LENGTH]*  
data bytes buffer
struct canfilt_t

Name

struct canfilt_t — structure for acceptance filter setup

Synopsis

struct canfilt_t {
    int flags;
    int queid;
    int cob;
    canmsg_id_t id;
    canmsg_id_t mask;
};

Members

flags

message flags MSG_RTR .. message is Remote Transmission Request, MSG_EXT .. message with extended ID, MSG_OVR .. indication of queue overflow condition, MSG_LOCAL .. message originates from this node. there are corresponding mask bits MSG_RTR_MASK, MSG_EXT_MASK, MSG_LOCAL_MASK. MSG_PROCESSLOCAL enables local messages processing in the combination with global setting

queid

CAN queue identification in the case of the multiple queues per one user (open instance)

cob

communication object number (not used)

id

selected required value of cared ID id bits

mask

select bits significand for the comparation; 1 .. take care about corresponding ID bit, 0 .. don’t care
vca_h2log

Name
vca_h2log — converts VCA handle to printable number

Synopsis

long vca_h2log (vca_handle_t vcah);

Arguments

vcah
VCA handle

Header
can_vca.h

Return Value
unique printable VCA handle number

vca_open_handle

Name
vca_open_handle — opens new VCA handle from CAN driver

Synopsis

int vca_open_handle (vca_handle_t * vcah_p, const char * dev_name,
const char * options, int flags);

Arguments

vcah_p
points to location filled by new VCA handle
dev_name
   name of requested CAN device, if NULL, default VCA_DEV_NAME is used
options
   options argument, can be NULL
flags
   flags modifying style of open (VCA_O_NOBLOCK)

Header
can_vca.h

Return Value
VCA_OK in case of success

vca_close_handle

Name
vca_close_handle — closes previously acquired VCA handle

Synopsis

int vca_close_handle (vca_handle_t vcah);

Arguments

vcah
   VCA handle

Header
can_vca.h

Return Value
Same as libc close returns.
**vca_send_msg_seq**

**Name**

vca_send_msg_seq — sends sequentially block of CAN messages

**Synopsis**

```c
int vca_send_msg_seq (vca_handle_t vcah, canmsg_t * messages, int count);
```

**Arguments**

- **vcah**
  - VCA handle
- **messages**
  - points to continuous array of CAN messages to send
- **count**
  - count of messages in array

**Header**

can_vca.h

**Return Value**

Number of sucessfully sent messages or error < 0

---

**vca_rec_msg_seq**

**Name**

vca_rec_msg_seq — receive sequential block of CAN messages

**Synopsis**

```c
int vca_rec_msg_seq (vca_handle_t vcah, canmsg_t * messages, int count);
```
Arguments

\textit{vcah}
  \begin{itemize}
  \item VCA handle
  \end{itemize}

\textit{messages}
  \begin{itemize}
  \item points to array for received CAN messages
  \end{itemize}

\textit{count}
  \begin{itemize}
  \item number of message slots in array
  \end{itemize}

Header

\texttt{can_vca.h}

Return Value

number of received messages or error < 0

\textbf{vca\_wait}

Name

\textit{vca\_wait} — blocking wait for the new message(s)

Synopsis

\begin{verbatim}
int vca_wait (vca_handle_t vcah, int wait_msec, int what);
\end{verbatim}

Arguments

\textit{vcah}
  \begin{itemize}
  \item VCA handle
  \end{itemize}

\textit{wait\_msec}
  \begin{itemize}
  \item number of miliseconds to wait, 0 => forever
  \end{itemize}

\textit{what}
  \begin{itemize}
  \item 0,1 => wait for Rx message, 2 => wait for Tx - free 3 => wait for both
  \end{itemize}
Header
can_vca.h

Return Value
Positive value if wait condition is satisfied

vca_gethex

Name
vca_gethex — gets one hexadecimal number from string

Synopsis

int vca_gethex (const char * str, int * u);

Arguments

str
scanned string

u
pointer to store got value

Return
the number of eaten chars

Header
can_vca.h

vca_strmatch

Name
vca_strmatch — get token from string
Synopsis

```c
int vca_strmatch (const char * str, const char * template);
```

Arguments

- `str`: scanned string
- `template`: token template template consists of characters and '~' matching one or more of spaces ie. '~hello' matches ' hello', ' hello', ' hello' etc.

Return

the number of used chars from str if match or negative value (number of partially matched chars from str - 1) if template does not match

Header

can_vca.h

vca_msg2str

Name

vca_msg2str — converts canmsg_t to the string

Synopsis

```c
int vca_msg2str (const struct canmsg_t * can_msg, char * buff, int buff_len);
```

Arguments

- `can_msg`: pointer to the serialized CAN message
- `buff`: buffer for the serialized string
buff_len
    max length of serialized string, including terminating zero

Return
the number of written chars not including terminating zero

Header
can_vca.h

vca_byte2str

Name
vca_byte2str — converts byte to the string

Synopsis

const char* vca_byte2str (unsigned char b, int base);

Arguments

b
    byte to convert

base
    base, can be (2, 8, 16)

Return
string representation of b in chosen base

Header
can_vca.h
vca_str2msg

Name
vca_str2msg — converts the string to the canmsg_t object

Synopsis

int vca_str2msg (struct canmsg_t * can_msg, const char * str);

Arguments

can_msg
pointer to the serialized CAN message

str
string representing CAN message

Return
number of read chars if succeed else zero or negative value.

Header
can_vca.h

vca_cmp_terminated

Name
vca_cmp_terminated — compares two strings terminated either by ‘\0’ or by terminator.

Synopsis

int vca_cmp_terminated (const char * pa, const char * pb, char terminator);
Arguments

\( pa \)

first string

\( pb \)

second string

\( \text{terminator} \)

additional char (\0 stil terminates string too), that indicates end of string

Description

Usefull when one works with the path names.

Return

the same value like libc strcmp does.

Header

can_vca.h

\textbf{vca\_log}

Name

\texttt{vca\_log} — generic logging facility for VCA library

Synopsis

\begin{verbatim}
void vca_log (const char * \texttt{domain}, int \texttt{level}, const char * \texttt{format}, ...
\end{verbatim}

Arguments

\( \texttt{domain} \)

pointer to character string representing source of logged event, it is VCA\_LDOMAIN for library itself

\( \texttt{level} \)

severity level
printf style format followed by arguments

variable arguments

Description
This function is used for logging of various events. If not overridden by application, logged messages goes to the stderr. Environment variable VCA_LOG_FILENAME can be used to redirect output to file. Environment variable VCA_DEBUG_FLG can be used to select different set of logged events through vca_debug_flg.

Note
There is a global variable vca_log_cutoff_level. Only the messages with level <= vca_log_cutoff_level will be logged. see can_vca.h

vca_log_redir

Name
vca_log_redir — redirects default log output function

Synopsis

void vca_log_redir (vca_log_fnc_t * log_fnc, int add_flags);

Arguments

log_fnc
new log output function. Value NULL resets to default function

add_flags
some more flags
Chapter 1. CAN/CANopen user guide

SDO processing API

struct vcasdo_fsm_t

Name

struct vcasdo_fsm_t — structure representing SDO FSM

Synopsis

struct vcasdo_fsm_t {
    unsigned srvcli_cob_id;
    unsigned clisrv_cob_id;
    unsigned node;
    unsigned index;
    unsigned subindex;
    struct timeval last_activity;
    int bytes_to_load;
    int toggle_bit:1;
    int is_server:1;
    int is_uploader:1;
    int state;
    vcasdo_fsm_state_fnc_t * statefnc;
    int err_no;
    ul_dbuff_t data;
    canmsg_t out_msg;
};

Members

srvcli_cob_id

    SDO server-client COB_ID (default is 0x580 + node), port on which master listen

clisrv_cob_id

    SDO client-server COB_ID (default is 0x600 + node), port on which slave listen

node

    CANopen node number

index

    index of communicated object

subindex

    subindex of communicated object

last_activity

    time of last FSM activity (internal use)

bytes_to_load

    number of stil not uploaded SDO data bytes (internal use)

toggle_bit

    (internal use)
is_server
   type of FSM client or server (Master or Slave) (internal use)

is_uploader
   processing upload/download in state sdo_fsmRun, sdo_fsmDone

state
   state of SDO (sdo_fsmIdle = 0, sdo_fsmRun, sdo_fsmDone, sdo_fsmError, sdo_fsmAbort)

statefnc
   pointer to the state function (internal use)

err_no
   error number in state sdo_fsmError.

data
   uploaded/downloaded bytes (see ul_dbuff.h)

out_msg
   if vcasdo_taste_msg generates answer, it is stored in the out_msg

Header
   vcasdo_fsm.h

vcasdo_fsm_upload1

Name
   vcasdo_fsm_upload1 — starts SDO upload using parameters set by previous calling vcasdo_init_fsm

Synopsis

   int vcasdo_fsm_upload1 (vcasdo_fsm_t * fsm);

Arguments

fsm
   FSM to work with
Return
the same as vcasdo_fsm_upload1

See also
vcasdo_fsm_upload1.

Header
vcasdo_fsm.h

vcasdo_fsm_download1

Name
vcasdo_fsm_upload1 — starts SDO download using parameters set by previous calling vcasdo_init_fsm

Synopsis

int vcasdo_fsm_download1 (vcasdo_fsm_t * fsm, ul_dbuff_t * data);

Arguments

fsm
FSM to work with

data
pointer to &ul_dbuff_t structure where downloaded data will be stored

Return
the same as vcasdo_fsm_download

See also
vcasdo_fsm_download.
vcasdo_read_multiplexor

Name
vcasdo_read_multiplexor — reads index and subindex from multiplexor part of CANopen message

Synopsis

void vcasdo_read_multiplexor (const byte * mult, unsigned * index, unsigned * subindex);

Arguments

mult
    pointer to the multiplexor part of CANopen message

index
    pointer to place to store read index

subindex
    pointer to place to store read subindex

vcasdo_error_msg

Name
vcasdo_error_msg — translates err_no to the string message

Synopsis

const char* vcasdo_error_msg (int err_no);
Arguments

err_no

number of error, if FSM state == sdofsmError

Return
textual error description.

Header
vcasdo_fsm.h

vcasdo_init_fsm

Name
vcasdo_init_fsm — init SDO FSM

Synopsis

void vcasdo_init_fsm (vcasdo_fsm_t * fsm, unsigned srvcli_cob_id, unsigned clisrv_cob_id, unsigned node);

Arguments

fsm

fsm to init

srvcli_cob_id

port to use for server->client communication (default 0x850 used if srvcli_cob_id==0)

clisrv_cob_id

port to use for client->server communication (default 0x600 used if clisrv_cob_id==0)

node

number of node on CAN bus to communicate with
vcasdo_destroy_fsm

Name
vcasdo_destroy_fsm — frees all SDO FSM resources (destructor)

Synopsis

```c
void vcasdo_destroy_fsm (vcasdo_fsm_t * fsm);
```

Arguments

- `fsm`:
  - fsm to destroy

vcasdo_fsm_idle

Name
vcasdo_fsm_idle — sets SDO FSM to idle state

Synopsis

```c
void vcasdo_fsm_idle (vcasdo_fsm_t * fsm);
```

Arguments

- `fsm`:
  - SDO FSM
vcasdo_fsm_run

**Name**

vcasdo_fsm_run — starts SDO communication protocol for this FSM

**Synopsis**

```c
void vcasdo_fsm_run (vcasdo_fsm_t * fsm);
```

**Arguments**

- `fsm`
  
  SDO FSM

vcasdo_fsm_abort

**Name**

vcasdo_fsm_abort — aborts SDO communication for this FSM, fill abort out_msg

**Synopsis**

```c
void vcasdo_fsm_abort (vcasdo_fsm_t * fsm, uint32_t abort_code);
```

**Arguments**

- `fsm`
  
  SDO FSM
abort_code
    code to fill to out_msg

Header
vcasdo_fsm.h

vcasdo_fsm_upload

Name
vcasdo_fsm_upload — starts upload SDO communication protocol for this FSM

Synopsis

int vcasdo_fsm_upload (vcasdo_fsm_t * fsm, int node, unsigned index,
    byte subindex, unsigned srvcli_cob_id, unsigned clisrv_cob_id);

Arguments

fsm
    SDO FSM

node
    CANopen device node to upload from

index
    uploaded object index

subindex
    uploaded object subindex

srvcli_cob_id
    port to use for server->client communication (default 0x850 used if
    srvcli_cob_id==0)

clisrv_cob_id
    port to use for client->server communication (default 0x600 used if
    clisrv_cob_id==0)

Return
not 0 if fsm->out_msg contains CAN message to sent
vcasdo_fsm_download

Name
vcasdo_fsm_download — starts download SDO communication protocol for this FSM

Synopsis

int vcasdo_fsm_download (vcasdo_fsm_t * fsm, ul_dbuff_t * dbuff, int node, unsigned index, byte subindex, unsigned srvcli_cob_id, unsigned clisrv_cob_id);

Arguments

fsm
SDO FSM
dbuff
pointer to a ul_dbuff structure to store received/transmitted data
node
CANopen device node to upload from
index
uploaded object index
subindex
uploaded object subindex
srvcli_cob_id
port to use for server->client communication (default 0x850 used if srvcli_cob_id==0)
clisrv_cob_id
port to use for client->server communication (default 0x600 used if clisrv_cob_id==0)

Return
not 0 if fsm->out_msg contains CAN message to sent
vcasdo_fsm_taste_msg

Name
vcasdo_fsm_taste_msg — try to process msg in FSM

Synopsis

int vcasdo_fsm_taste_msg(vcasdo_fsm_t * fsm, const canmsg_t * msg);

Arguments

fsm
  fsm to process msg
msg
  tasted msg

Return

0 if msg is not eatable for FSM, -1 if message has correct CobID but can’t be processed in current FSM state, 1 if message is processed,

vcasdo_abort_msg

Name
vcasdo_abort_msg — translates SDO abort_code to the string message
Synopsis

const char* vcasdo_abort_msg (uint32_t abort_code);

Arguments

abort_code

abort code

Header

vcasdo_msg.h

../../canvca/libvca/vcasdo_msg.c

Name

../../canvca/libvca/vcasdo_msg.c — Document generation inconsistency

Oops

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The template for this document tried to insert the structured comment from the file ../../canvca/libvca/vcasdo_msg.c at this point, but none was found. This dummy section is inserted to allow generation to continue.</td>
</tr>
</tbody>
</table>

PDO processing API

struct vcapdo_mapping_t

Name

struct vcapdo_mapping_t — structure representing mapping of sigle object in PDO
Synopsis

```c
struct vcapdo_mapping_t {
    vcaod_object_t * object;
    unsigned char start;
    unsigned char len;
    sui_dinfo_t * dinfo;
};
```

Members

- **object**: pointer to the mapped object
- **start**: bit offset of object value in PDO
- **len**: bit length of object value in PDO
- **dinfo**: pointer to object data source. Every PDO can be read/written through `dinfo` to the OD or to hardware. Actually, there is no other way for PDO object to do that.

Header

`vca_pdo.h`

```c
struct vcapdolst_object_t
```

Name

`struct vcapdolst_object_t — structure representing single PDO object`

Synopsis

```c
struct vcapdolst_object_t {
    gavl_node_t my_node;
    struct vcaPDOProcessor_t * pdo_processor;
    unsigned long cob_id;
    unsigned char transmission_type;
    unsigned flags;
    unsigned char sync_every;
    unsigned char sync_counter;
    uint16_t inhibit_time;
    uint16_t event_timer;
    unsigned char pdo_buff[8];
    int mapped_cnt;
    vcapdo_mapping_t * mapped_objects;
    evc_rx_hub_t rx_hub;
};
```
**Members**

- **my_node**
  - structure necessary for storing node in GAVL tree

- **pdo_processor**
  - pointer to PDO processor servicing this PDO

- **cob_id**
  - COB ID of PDO

- **transmition_type**
  - type of PDO transmission according to DS301 table 55

- **flags**
  - PDO characteristics and parsed transmission_type

- **sync_every**
  - synchronous PDO will be processed every n-th SYNC message

- **sync_counter**
  - auxiliary variable for `sync_every`

- **inhibit_time**
  - minimum gap between two PDO transmissions (multiples of 100 us)

- **event_timer**
  - if nonzero, PDO is transmitted every `event_timer` ms. Valid only in transmission modes 254, 255. (!vcapdoFlagSynchronous && !vcapdoFlagRTROnly)

- **pdo_buff[8]**
  - buffer for received/transmitted PDO

- **mapped_cnt**
  - number of mapped objects in OD

- **mapped_objects**
  - array to structures describing mapping details for all mapped objects

- **rx_hub**
  - If PDO communication is event driven, appropriate events are connected to this hub

**See also**

GAVL usage (ul_gavlchk.c)

**Header**

vca_pdo.h
**struct vcapdolst_root_t**

**Name**

struct vcapdolst_root_t — structure representing root of OD

**Synopsis**

struct vcapdolst_root_t {
    gavl_node_t * my_root;
};

**Members**

my_root

    object dictionary GAVL tree root

**See also**

GAVL usage (u1_gavchk.c)

**Header**

vcapdo.h

---

**struct vcaPDOProcessor_t**

**Name**

struct vcaPDOProcessor_t — structure used for PDO communication

**Synopsis**

struct vcaPDOProcessor_t {
    vcapdolst_root_t pdolst_root;
    // TODO send_to_can_fnc: remove this hack and add queue of outcoming CAN messages// to make this library thread safe.
    vcapdo_send_to_can_fnc_t *;
    vcaod_root_t * od_root;
    //vcaDinfoManager_t * dinfo_mgr;
    int node_id;
};

**Members**

pdolst_root

    GAVL containing all defined &vcapdolst_object_t structures
send_to_can_fnc

PDOProcessor should use this function if it needs to send CAN message during processing

od_root

pointer to used OD (necessary for PD0s creation and initialization in vcaPDOProcessor_createPDOList)

dinfo_mngr

pointer to used DinfoManager (providing HW dinfos during initialization)

node_id

Node number, optional parameter, if it is specified, default PDO COB-IDs can be assigned if they are not specified in EDS. If node_id is 0, then it is ignored.

Description

vcaPDOProcessor is responsible for all PDO related tasks in CANopen device

Header

vca_pdo.h

vcaPDOProcessor_init

Name

vcaPDOProcessor_init — vcaPDOProcessor constructor

Synopsis

void vcaPDOProcessor_init (vcaPDOProcessor_t * proc);

Arguments

proc

pointer to PDO processor to work with

Header

vca_pdo.h
vcaPDOProcessor_destroy

Name
vcaPDOProcessor_destroy — vcaPDOProcessor destructor

Synopsis

void vcaPDOProcessor_destroy (vcaPDOProcessor_t * proc);

Arguments

proc
  pointer to PDO processor to work with

Description
It releases all PDO objects

Header
vca_pdo.h

vcaPDOProcessor_setOD

Name
vcaPDOProcessor_setOD — assign OD to PDOProcessor

Synopsis

void vcaPDOProcessor_setOD (vcaPDOProcessor_t * proc, vcaod_root_t * od_root);

Arguments

proc
  pointer to PDO processor to work with
od_root
assigned root of Object Dictionary

Header
vca pdo.h

vcaPDOProcessor_createPDOList

Name
vcaPDOProcessor_createPDOList — scans OD and creates all valid PDO structures.

Synopsis

int vcaPDOProcessor_createPDOList (vcaPDOProcessor_t * proc);

Arguments

proc
pointer to PDO processor to work with

Description
It also deletes previously created PDO structures (if any).

Return
0 or negative number in case of an error

Header
vca pdo.h
_vcaPDOProcessor_disconnectDinfoLinks

Name

_vcaPDOProcessor_disconnectDinfoLinks — disconnect all PDOs and their dinfo structures

Synopsis

void _vcaPDOProcessor_disconnectDinfoLinks (vcaPDOProcessor_t * proc);

Arguments

proc

pointer to PDO processor to work with

Description

Actualy it only decrements RefCnt, so only dinfos with RefCnt==1 will be deleted

Note

this function is internal and it is not a part of VCA PDO public interface.

Header

vca_pdo.h

vcaPDOProcessor_connectDinfoLinks

Name

vcaPDOProcessor_connectDinfoLinks — scans defined PDOs and makes necessary data links from PDOs to OD and HW

Synopsis

void vcaPDOProcessor_connectDinfoLinks (vcaPDOProcessor_t * proc);
Arguments

proc
    pointer to PDO processor to work with

Description
Disconnect all connected dinfos. For each mapped object tries to find appropriate dinfo asking DinfoManager. If DinfoManager returns NULL, that means, that no HW is connected to this object. In such case function creates dbuff_dinfo for data stored in OD and connect it to mapped PDO.

Header
vca_pdo.h

vcaPDOProcessor_processMsg

Name
vcaPDOProcessor_processMsg — tries to process msg

Synopsis

int vcaPDOProcessor_processMsg (vcaPDOProcessor_t * proc, canmsg_t * msg);

Arguments

proc
    pointer to PDO processor to work with

msg
    CAN msg to proceed

Return
zero if msg is processed
OD access API

struct vcaod_root_t

Name

struct vcaod_root_t — structure representing root of OD

Synopsis

struct vcaod_root_t {
    gsa_array_field_t my_root;
};

Members

my_root

    object dictionary GAVL tree root
struct vcaod_object_t

Name

struct vcaod_object_t — structure representing single object in OD

Synopsis

struct vcaod_object_t {
    #ifndef CONFIG_OD_GSA
        gavl_node_t my_node;
    #endif
    unsigned index;
    int subindex;
    unsigned char data_type;
    unsigned object_type;
    int access;
    unsigned flags;
    char name[VCAOD_OBJECT_NAME_LEN];
    struct vcaod_object_t * subobjects;
    int subcnt;
    vcaod_dbuff_t value;
    sui_dinfo_t * dinfo;
};

Members

my_node
    structure neccessary for storing node in GAVL tree, is NULL for subindices
index
    index of object
subindex
    subindex of subobject or -1 if object is not subobject
data_type
    can be one of (BOOLEAN, INTEGER8, ...)
object_type
    type of object (DOMAIN=2, DEFTYPE=5, DEFSTRUCT=6, VAR=7, ARRAY=8, RECORD=9)
access
    access attributes (RW, WO, RO, CONST)
flags

flags can be: VCAOD_OBJECT_FLAG_MANDATORY object is mandatory/optional,
VCAOD_OBJECT_FLAG_PDO_MAPPING object is supposed to be PDO mapped,
VCAOD_OBJECT_FLAG_WEAK_DINFO dinfo is weak pointer

name[VCAOD_OBJECT_NAME_LEN]

textual name of object

subobjects

pointer to array of subobjects (definition==DEFSTRUCT, RECORD) or NULL

subcnt

number of subobjects

value

object values (definition==ARRAY) or single value (other definitions). If definition==ARRAY all values have the same length and they are stored sequently in

value

dinfo

Reference to dinfo associated with current object. There are couple of reasons for such a association. 1. Object is PDO mapped but its value doesn’t come from HW dinfo (it is not tecnological value) - in such a case dbuff dinfo is created and referenced from that OD object. 2. Object is PDO mapped and its value comes from HW dinfo (it is tecnological value) - in such a case only weak reference is in OD object. When HW module is unloaded or dinfo will be destroyed from any reason, also weak reference to it will be cleared to NULL. 3. Object is not PDO mapped but its value comes from HW dinfo - in such a case even SDO communication should read that dinfo to get the proper object value.

Header

vca_od.h

vcaod_find_object

Name

vcaod_find_object — finds object in OD. This function is not a part of the SDO API

Synopsis

vcaod_object_t* vcaod_find_object (vcaod_root_t * odroot, unsigned ix,
unsigned subix, uint32_t * abort_code);
Arguments

odroot
  object dictionary
ix
  object index
subix
  object subindex, ignored if object does not have subobjects
abort_code
  Pointer to the abort code in case of an ERROR. It can be NULL, than it is ignored. Abort codes are defined in CANopen standard 301 and can be translated to text calling vcasdo_abort_msg.

Return
found object or NULL

Header
vca_od.h

vcaod_get_value

Name
vcaod_get_value — reads object value from Object Dictionary and copies them to caller buffer

Synopsis

int vcaod_get_value (const vcaod_object_t * object, int array_index, void * buff, int len, uint32_t * abort_code);

Arguments

object
  object from dictionary, see. vcaod_find_object
array_index
  if object is an array array_index specifies which index to get, otherwise it is ignored.
buff
buffer to write requested data

len
length of the buffer

abort_code
Pointer to the abort code in case of an ERROR. It can be NULL, than it is ignored. Abort codes are defined in CANopen standard 301 and can be translated to text calling `vcaod_abort_msg`.

Return
number of read bytes negative value in case of an error

Header
vca_od.h

vcaod_set_value

Name
vcaod_set_value — copies object value from caller’s buffer to Object Dictionary

Synopsis

```c
int vcaod_set_value (vcaod_object_t * object, int array_index, const void * buff, int len, uint32_t * abort_code);
```

Arguments

object
object from dictionary, see `vcaod_find_object`

array_index
if object is an array, array_index, tells which item to get, in other case it is simply ignored.

buff
buffer containing written data

len
length of the data
abort_code

area to fill the abort code in case of an ERROR. It can be NULL, than it is ignored. Abort codes are defined in CANopen standard 301 and can be translated to text calling vcasdo_abort_msg.

Description

Function sets whole buffer to zeros before it starts to copy object data to it, even if buffer is larger than data.

Return

number of stored data bytes negative value in case of an error

Header

vca_od.h

vcaod_get_object_data_size

Name

vcaod_get_object_data_size — get size of object in bytes

Synopsis

int vcaod_get_object_data_size (const vcaod_object_t * object, uint32_t * abort_code);

Arguments

object

object from dictionary, see. vcaod_find_object

abort_code

area to fill the abort code in case of an ERROR. It can be NULL, than it is ignored. Abort codes are defined in CANopen standard 301 and can be translated to text calling vcasdo_abort_msg.

Return

number of stored data bytes negative value in case of an error
Chapter 1. CAN/CANopen user guide

Header
vca_od.h

od_item_set_value_as_str

Name
od_item_set_value_as_str — set object value from its string representation.

Synopsis

int od_item_set_value_as_str (vcaod_object_t * item, const char * valstr);

Arguments

item
    object to set
value
    string representation of object value

Return
negative value in case of an error

Header
vca_od.h

vcaod_od_free

Name
vcaod_od_free — release all OD memory
**Synopsis**

```c
void vcaod_od_free (vcaod_root_t * odroot);
```

**Arguments**

- `odroot`  
  pointer to the object dictionary root

**Header**

`vca_od.h`

---

**vcaod_dump_od**

**Name**

`vcaod_dump_od` — debug function, dumps OD to log

**Synopsis**

```c
void vcaod_dump_od (vcaod_root_t * odroot);
```

**Arguments**

- `odroot`  
  root, which contains OD

**Header**

`vca_od.h`

---

**vcaod_get_dinfo_ref**

**Name**

`vcaod_get_dinfo_ref` — returns reference to dinfo corresponding to `obj`
Synopsis

sui_dinfo_t * vcaod_get_dinfo_ref (vcaod_object_t * obj, int create_weak);

Arguments

obj
    object from OD
create_weak
    if there is no HW dinfo for object, creates temporary dbuff dinfo

Description

If \texttt{obj} allready has its &dinfo assigned \texttt{vcaod_get_dinfo_ref} returns this pointer, if it is not function creates new &dinfo object.

Return

pointer to associated dinfo with reference count increased or NULL if creation fails

Header

\texttt{vca_od.h}

\texttt{../../canvca/libvca/vca_od.c}

Name

\texttt{../../canvca/libvca/vca_od.c} — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file \texttt{../../canvca/libvca/vca_od.c} at this point, but none was found. This dummy section is inserted to allow generation to continue.
libulut API

**ul_dbuff_init**

Name

*ul_dbuff_init* — init memory allocated for dynamic buffer
Synopsis

```c
int ul_dbuff_init (ul_dbuff_t * buf, int flags);
```

Arguments

- **buf**: buffer structure
- **flags**: flags describing behaviour of the buffer only UL_DBUFF_IS_STATIC flag is supported. In this case buffer use only static array sbuf

Description

Returns capacity of initialised buffer

---

**ul_dbuff_destroy**

Name

ul_dbuff_destroy — frees all resources allocated by buf

Synopsis

```c
void ul_dbuff_destroy (ul_dbuff_t * buf);
```

Arguments

- **buf**: buffer structure

---

**ul_dbuff_prep**

Name

ul_dbuff_prep — sets a new len and capacity of the buffer
Synopsis

```c
int ul_dbuff_prep (ul_dbuff_t * buf, int new_len);
```

Arguments

- `buf` - buffer structure
- `new_len` - new desired buffer length

Description

Returns new buffer length

```c
struct ul_dbuff
```

Name

`struct ul_dbuff` — Generic Buffer for Dynamic Data

Synopsis

```c
struct ul_dbuff {
    unsigned long len;
    unsigned long capacity;
    int flags;
    unsigned char * data;
    unsigned char sbuff[UL_DBUFF_SLEN];
};
```

Members

- `len` - actual length of stored data
- `capacity` - capacity of allocated buffer
- `flags` - only one flag (`UL_DBUFF_IS_STATIC`) used now
- `data` - pointer to dynamically allocated buffer
ul_dbuff_set_capacity

**Name**
ul_dbuff_set_capacity — change capacity of buffer to at least new_capacity

**Synopsis**

```c
int ul_dbuff_set_capacity (ul_dbuff_t * buf, int new_capacity);
```

**Arguments**

- `buf`
  - buffer structure
- `new_capacity`
  - new capacity

**Description**

Returns real capacity of reallocated buffer

ul_dbuff_set_len

**Name**
ul_dbuff_set_len — sets a new len of the buffer, change the capacity if necessary

**Synopsis**

```c
int ul_dbuff_set_len (ul_dbuff_t * buf, int new_len);
```
Arguments

\( buf \)

buffer structure

\( new\_len \)

new desired buffer length

Description

Returns new buffer length

ul_dbuff_set

Name

ul_dbuff_set — copies bytes to buffer and change its capacity if neccessary like memset

Synopsis

\[
\text{int ul\_dbuff\_set (ul\_dbuff\_t * buf, byte b, int n);}\]

Arguments

\( buf \)

buffer structure

\( b \)

appended bytes

\( n \)

number of appended bytes

Returns

length of buffer
ul_dbuff_cpy

Name
ul_dbuff_cpy — copies bytes to buffer and change its capacity if necessary

Synopsis

int ul_dbuff_cpy (ul_dbuff_t * buf, const void * b, int n);

Arguments

buf
  buffer structure

b
  appended bytes

n
  number of appended bytes

Returns
length of buffer

ul_dbuff_cat

Name
ul_dbuff_cat — appends bytes at end of buffer and change its capacity if necessary

Synopsis

int ul_dbuff_cat (ul_dbuff_t * buf, const void * b, int n);

Arguments

buf
  buffer structure
b
  appended bytes

n
  number of appended bytes

**Returns**

length of buffer

---

**ul_dbuff_strcat**

**Name**

ul_dbuff_strcat — appends str at the end of buffer and change its capacity if necessary

**Synopsis**

```c
int ul_dbuff_strcat (ul_dbuff_t * buf, const char * str);
```

**Arguments**

- **buf**
  
  buffer structure

- **str**
  
  string to append

**Description**

Returns number length of buffer (including terminating \'\0\')

---

**ul_dbuff_strcpy**

**Name**

ul_dbuff_strcpy — copy str to the buffer and change its capacity if necessary
Synopsis

```c
int ul_dbuff_strcpy (ul_dbuff_t * buf, const char * str);
```

Arguments

- `buf` buffer structure
- `str` string to copy

Description

Returns number length of buffer (including terminating '\0')

ul_dbuff_append_byte

Name

ul_dbuff_append_byte — appends byte at the end of buffer and change its capacity if necessary

Synopsis

```c
int ul_dbuff_append_byte (ul_dbuff_t * buf, unsigned char b);
```

Arguments

- `buf` buffer structure
- `b` appended byte

Description

Returns number length of buffer (including terminating '\0')
ul_dbuff_ltrim

Name
ul_dbuff_ltrim — remove all white space characters from the left

Synopsis

int ul_dbuff_ltrim (ul_dbuff_t * buf);

Arguments

buf
    buffer structure

Return
new length of buffer

ul_dbuff_rtrim

Name
ul_dbuff_rtrim — remove all white space characters from the right

Synopsis

int ul_dbuff_rtrim (ul_dbuff_t * buf);

Arguments

buf
    buffer structure

Description
if buffer is terminated by ‘\0’, than is also terminated after rtrim
Return
new length of buffer

ul_dbuff_trim

Name
ul_dbuff_trim — remove all white space characters from the right and from the left

Synopsis

int ul_dbuff_trim (ul_dbuff_t * buf);

Arguments

buf
buffer structure

Description
Returns number length of buffer (including terminating '\0')

ul_dbuff_cpos

Name
ul_dbuff_cpos — searches string for char

Synopsis

int ul_dbuff_cpos (const ul_dbuff_t * buf, unsigned char what, unsigned char quote);
Arguments

*buf*
  searched dbuff

*what*
  char to find

*quote*
  skip str areas quoted in quote chars If you want to ignore quotes assign '\0' to quote in function call

Return

position of what char or negative value

ul_str_cpos

Name

ul_str_cpos — searches string for char

Synopsis

```c
int ul_str_cpos (const unsigned char * str, unsigned char what, unsigned char quote);
```

Arguments

*str*
  zero terminated string

*what*
  char to find

*quote*
  skip str areas quoted in quote chars If you want to ignore quotes assign '\0' to quote in function call

Return

position of what char or negative value
ul_str_pos

Name
ul_str_pos — searches string for substring

Synopsis

int ul_str_pos (const unsigned char * str, const unsigned char * what, unsigned char quote);

Arguments

str
   zero terminated string
what
   string to find
quote
   skip str areas quoted in quote chars If you want to ignore quotes assign \0 to quote in function call

Return

position of what string or negative value

ul_strncpy

Name
ul_strncpy — copies string to the buffer

Synopsis

int ul_strncpy (unsigned char * to, const unsigned char * from, int buff_size);
Arguments

to
    buffer where to copy str
from
    zero terminated string
buff_size
    size of the to buffer (including terminating zero)

Description

Standard strncpy function have some disadvantages (ie. do not append term. zero if copied string doesn’t fit in to buffer, fills whole rest of buffer with zeros) Returns strlen(to) or negative value in case of error

ul_dbuff_cut_pos

Name

ul_dbuff_cut_pos — cut first n bytes from fromdb and copies it to todb.

Synopsis

void ul_dbuff_cut_pos (ul_dbuff_t * fromdb, ul_dbuff_t * todb, int n);

Arguments

fromdb
    buffer to cut from
todb
    buffer to copy to
n
    position where to cut

Description

If n is greater than fromdb.len whole fromdb is copied to todb. If n is negative position to cut is counted from the end of fromdb. If n is zero fromdb stays unchanged and todb is resized to len equal zero.
ul_dbuff_cut_delimited

Name
ul_dbuff_cut_delimited — cuts bytes before delimiter + delimiter char from fromdb and copies them to the todb

Synopsis

void ul_dbuff_cut_delimited (ul_dbuff_t * fromdb, ul_dbuff_t * todb, char delimiter, char quote);

Arguments

fromdb
buffer to cut from
todb
buffer to copy to
delimiter
delimiter char
quote
quoted delimiters are ignored, quote can be '\0', than it is ignored.

Description
If fromdb doesn’t contain delimiter todb is trimmed to zero length.

ul_dbuff_cut_token

Name
ul_dbuff_cut_token — cuts not whitespaces from fromdb to todb.

Synopsis

void ul_dbuff_cut_token (ul_dbuff_t * fromdb, ul_dbuff_t * todb);
Arguments

fromdb
  buffer to cut from
todb
  buffer to copy to

Description
Leading whitespaces are ignored. Cut string is trimmed.

**evc_link_init**

Name

evclink_init — Initialize Event Connector Link

Synopsis

```c
int evc_link_init (evc_link_t * link);
```

Arguments

link
  pointer to the link

Description
Link reference count is set to 1 by this function

Return Value
negative value informs about failure.

**evc_link_new**

Name

evclink_new — Allocates New Event Connector Link
Chapter 1. CAN/CANopen user guide

Synopsis

evc_link_t * evc_link_new ( void);

Arguments

void

no arguments

Description

Link reference count is set to 1 by this function

Return Value

pointer to the new link or NULL.

evc_link_connect

Name

evc_link_connect — Connects Link between Two Hubs

Synopsis

int evc_link_connect (evc_link_t * link, evc_tx_hub_t * src, evc_rx_hub_t * dst, evc_prop_fnc_t * prop);

Arguments

link

pointer to the non-connected initialized link

src

pointer to the source hub of type &evc_tx_hub_t

dst

pointer to the destination hub of type &evc_rx_hub_t
prop

  propagation function corresponding to source and destination expected event arguments

**Description**

If ready flag is not set, link state is set to ready and reference count is increased.

**Return Value**

negative return value indicates fail.

evc_link_init_standalone

**Name**

evc_link_init_standalone — Initialize Standalone Link

**Synopsis**

```c
int evc_link_init_standalone (evc_link_t * link, evc_rx_fnc_t * rx_fnc, void * context);
```

**Arguments**

- **link**
  
  pointer to the link

- **rx_fnc**
  
  pointer to the function invoked by event reception

- **context**
  
  context for the rx_fnc function invocation

**Description**

Link reference count is set 1 by this function

**Return Value**

negative value informs about failure.
**evc_link_new_standalone**

**Name**  
evc_link_new_standalone — Allocates New Standalone Link

**Synopsis**  
evclink_t * evc_link_new_standalone (evc_rx_fnc_t * rx_fnc, void * context);

**Arguments**

- **rx_fnc**  
  callback function invoked if event is delivered
- **context**  
  context provided to the callback function

**Description**  
Link reference count is set to 1 by this function

**Return Value**  
pointer to the new link or NULL.

**evc_link_connect_standalone**

**Name**  
evc_link_connect_standalone — Connects Standalone Link to Source Hubs

**Synopsis**  
int evc_link_connect_standalone (evc_link_t * link, evc_tx_hub_t * src, evc_prop_fnc_t * prop);
Arguments

link
   pointer to the non-connected initialized link

tsrc
   pointer to the source hub of type &evc_tx_hub_t

prop
   propagation function corresponding to hub source and standalone rx_fnc expected event arguments

Description
If ready flag is not set, link state is set to ready and reference count is increased.

Return Value
negative return value indicates failure.

evc_link_delete

Name
evc_link_delete — Deletes Link from Hubs Lists

Synopsis

int evc_link_delete (evc_link_t * link);

Arguments

link
   pointer to the possibly connected initialized link

Description
If ready flag is set, link ready flag is cleared and reference count is decreased. This could lead to link disappear, if nobody is holding reference.
Return Value
positive return value indicates immediate delete, zero return value informs about delayed delete.

evc_link_dispose

Name
evc_link_dispose — Disposes Link

Synopsis

void evc_link_dispose (evc_link_t * link);

Arguments

link
pointer to the possibly connected initialized link

Description
Deletes link from hubs, marks it as dead, calls final death propagate for the link and if link is malloced, releases link occupied memory.

evc_tx_hub_init

Name
evc_tx_hub_init — Initializes Event Transmition Hub

Synopsis

int evc_tx_hub_init (evc_tx_hub_t * hub);
Arguments

\textit{hub}

pointer to the \texttt{evc\_tx\_hub\_t} type hub

Return Value

negative return value indicates failure.

evc\_tx\_hub\_done

Name

\textit{evc\_tx\_hub\_done} — Initializes Event Transmission Hub

Synopsis

\begin{verbatim}
void evc\_tx\_hub\_done (evc\_tx\_hub\_t * hub);
\end{verbatim}

Arguments

\textit{hub}

pointer to the \texttt{evc\_tx\_hub\_t} type hub

evc\_tx\_hub\_propagate

Name

\textit{evc\_tx\_hub\_propagate} — Propagate Event to Links Destinations

Synopsis

\begin{verbatim}
void evc\_tx\_hub\_propagate (evc\_tx\_hub\_t * hub, va\_list args);
\end{verbatim}
Arguments

hub
    pointer to the &evc_tx_hub_t type hub

args
    pointer to the variable arguments list

Description
The function propagates event to the connected links, it skips links marked as dead, blocked or delete_pend. If the link is not marked as recursive, it ensures, that link is not called twice.

evc_tx_hub_emit

Name
evc_tx_hub_emit — Emits Event to Hub

Synopsis

void evc_tx_hub_emit (evc_tx_hub_t * hub, ...);

Arguments

hub
    pointer to the &evc_tx_hub_t type hub

... variable arguments

Description
The function hands over arguments to evc_tx_hub_propagate as &va_list.

evc_rx_hub_init

Name
evc_rx_hub_init — Initializes Event Reception Hub
Synopsis

```c
int evc_rx_hub_init (evc_rx_hub_t * hub, evc_rx_fnc_t * rx_fnc, void * context);
```

Arguments

- **hub**: pointer to the `evc_rx_hub_t` type hub
- **rx_fnc**: pointer to the function invoked by event reception
- **context**: context for the `rx_fnc` function invocation

Return Value

Negative return value indicates failure.

evc_rx_hub_done

Name

evc_rx_hub_done — Finalize Event Reception Hub

Synopsis

```c
void evc_rx_hub_done (evc_rx_hub_t * hub);
```

Arguments

- **hub**: pointer to the `evc_rx_hub_t` type hub
struct evc_link

Name
struct evc_link — Event Connector Link

Synopsis
struct evc_link {
    struct src;
    unsigned standalone:1;
    dst;
evc_prop_fnc_t * propagate;
    int refcnt;
    unsigned recursive:1;
    unsigned blocked:1;
    unsigned ready:1;
    unsigned dead:1;
    unsigned delete_pend:1;
    unsigned malloced:1;
    unsigned standalone:1;
    unsigned tx_full_hub:1;
    unsigned rx_full_hub:1;
    short taken;
};

Members
src
    describes source of the event link, contains pointer to &evc_tx_hub_t and peers
    links list
standalone
    link is used for standalone function invocation
dst
    determines destination of the event, it can be standalone rx_fnc function
    with context or &evc_tx_hub_t in the multi case
propagate
    pointer to the arguments propagation function,
refcnt
    link reference counter
recursive
    link can propagate could be invoked recursively, else recursive events are ig-
    nored by link
blocked
    event propagation is blocked for the link, can be used by application
ready
    link is ready and has purpose to live - it connects two active entities
dead
  link is dead and cannot propagate events
delete_pend
  link is being deleted, but it is taken simultaneously, delete has to wait for finish of the propagate and to moving to the next link
malloced
  link has been malloced and should be automatically freed when referenc counts drop to zero
standalone
  link is used for standalone function invocation
tx_full_hub
  src points to the full hub structure
rx_full_hub
  dst points to the full hub structure	
taken
  link is in middle of the propagation process

Description
The link delivers events from the source to the destination. The link specific function propagate is called for each link leading from the hub activated by evc_tx_hub_emit and evc_tx_hub_propagate. The propagate function is responsible for parameters transformation before invocation of standalone or destination hub rx_fnc function.

struct evc_tx_hub

Name
struct evc_tx_hub — Event Transmit Hub

Synopsis
struct evc_tx_hub {
  ul_list_head_t links;
};

Members
links
  list of links outgoing from the hub
struct evc_rx_hub

Name
struct evc_rx_hub — Event Receiving Hub

Synopsis
struct evc_rx_hub {
    ul_list_head_t links;
    evc_rx_fnc_t * rx_fnc;
    void * context;
};

Members
links
    list of links incoming to the hub
rx_fnc
    function invoked when event arrives
context
    context for rx_fnc

evc_link_inc_refcnt

Name
evc_link_inc_refcnt — Increment Link Reference Count

Synopsis
void evc_link_inc_refcnt (evc_link_t * link);

Arguments
link
    pointer to link
**evc_link_dec_refcnt**

**Name**
evclink_dec_refcnt — Decrement Link Reference Count

**Synopsis**

```c
void evc_link_dec_refcnt (evc_link_t * link);
```

**Arguments**

`link`

pointer to link

**Description**

if the link reference count drops to 0, link is deleted from hubs by evc_link_dispose function and if malloced is sed, link memory is disposed by free. Special handlink can be achieved if propagate returns non-zero value if called with ded link.

**gavl_first_node**

**Name**
gavl_first_node — Returns First Node of GAVL Tree

**Synopsis**

```c
gavl_node_t * gavl_first_node (const gavl_root_t * root);
```

**Arguments**

`root`

GAVL tree root
Return Value
pointer to the first node of tree according to ordering

gavl_last_node

Name
gavl_last_node — Returns Last Node of GAVL Tree

Synopsis

gavl_node_t * gavl_last_node (const gavl_root_t * root);

Arguments

root
    GAVL tree root

Return Value
pointer to the last node of tree according to ordering

gavl_is_empty

Name
gavl_is_empty — Check for Empty GAVL Tree

Synopsis

int gavl_is_empty (const gavl_root_t * root);

Arguments

root
    GAVL tree root
**Return Value**

returns non-zero value if there is no node in the tree

---

**gavl_search_node**

**Name**

*gavl_search_node* — Search for Node or Place for Node by Key

**Synopsis**

```c
int gavl_search_node (const gavl_root_t * root, const void * key, int mode, gavl_node_t ** nodep);
```

**Arguments**

- **root**
  - GAVL tree root
- **key**
  - key value searched for
- **mode**
  - mode of the search operation
- **nodep**
  - pointer to place for storing of pointer to found node or pointer to node which should be parent of inserted node

**Description**

Core search routine for GAVL trees searches in tree starting at *root* for node of item with value of item field at offset *key_off* equal to provided *key* value. Values are compared by function pointed by *cmp_fnc* field in the tree *root*. Integer *mode* modifies search algorithm:
- **GAVL_FANY** — finds node of any item with field value *key*,
- **GAVL_FFIRST** — finds node of first item with *key*,
- **GAVL_FAFTER** — node points after last item with *key* value, reworded - index points at first item with higher value of field or after last item

**Return Value**

Return of nonzero value indicates match found. If the *mode* is ored with **GAVL_FCMP**, result of last compare is returned.
gavl_find

Name

gavl_find — Find Item for Provided Key

Synopsis

void * gavl_find (const gavl_root_t * root, const void * key);

Arguments

root
    GAVL tree root

key
    key value searched for

Return Value

pointer to item associated to key value.

gavl_find_first

Name

gavl_find_first — Find the First Item with Provided Key Value

Synopsis

void * gavl_find_first (const gavl_root_t * root, const void * key);

Arguments

root
    GAVL tree root

key
    key value searched for


**gavl_find_after**

**Name**
gavl_find_after — Find the First Item with Higher Key Value

**Synopsis**

```c
void * gavl_find_after (const gavl_root_t * root, const void * key);
```

**Arguments**

*root*

- GAVL tree root

*key*

- key value searched for

**Description**

same as above, but points to item with first key value above searched key.

**Return Value**

pointer to the first item associated to key value.

**gavl_insert_node_at**

**Name**
gavl_insert_node_at — Insert Existing Node to Already Computed Place into GAVL Tree

---

**Chapter 1. CAN/CANopen user guide**
Synopsis

int gavl_insert_node_at (gavl_root_t * root, gavl_node_t * node, gavl_node_t * where, int leftright);

Arguments

root
  GAVL tree root

node
  pointer to inserted node

where
  pointer to found parent node

leftright
  left (1) or right (0) branch

Return Value

positive value informs about success

gavl_insert_node

Name

gavl_insert_node — Insert Existing Node into GAVL Tree

Synopsis

int gavl_insert_node (gavl_root_t * root, gavl_node_t * node, int mode);

Arguments

root
  GAVL tree root

node
  pointer to inserted node
mode
  if mode is GAVL_FAFTER, multiple items with same key can be used, else strict
  ordering is required

Return Value
positive value informs about success

gavl_insert

Name
gavl_insert — Insert New Item into GAVL Tree

Synopsis

int gavl_insert (gavl_root_t * root, void * item, int mode);

Arguments

root
  GAVL tree root

item
  pointer to inserted item

mode
  if mode is GAVL_FAFTER, multiple items with same key can be used, else strict
  ordering is required

Return Value
positive value informs about success, negative value indicates malloc fail or attempt
  to insert item with already defined key.

gavl_delete_node

Name
gavl_delete_node — Deletes/Unlinks Node from GAVL Tree
Synopsis

int gavl_delete_node (gavl_root_t * root, gavl_node_t * node);

Arguments

root
GAVL tree root

node
pointer to deleted node

Return Value
positive value informs about success.

gavl_delete

Name
gavl_delete — Delete/Unlink Item from GAVL Tree

Synopsis

int gavl_delete (gavl_root_t * root, void * item);

Arguments

root
GAVL tree root

item
pointer to deleted item

Return Value
positive value informs about success, negative value indicates that item is not found in tree defined by root
gavl_delete_and_next_node

Name

gavl_delete_and_next_node — Delete/Unlink Item from GAVL Tree

Synopsis

gavl_node_t * gavl_delete_and_next_node (gavl_root_t * root,
gavl_node_t * node);

Arguments

root
    GAVL tree root

node
    pointer to actual node which is unlinked from tree after function call, it can be
    unallocated or reused by application code after this call.

Description

This function can be used after call gavl_first_node for destructive traversal
through the tree, it cannot be combined with gavl_next_node or gavl_prev_node
and root is emptied after the end of traversal. If the tree is used after
unsuccessful/unfinished traversal, it must be balanced again. The height differences
are inconsistent in other case. If traversal could be interrupted, the function
gavl_cut_first could be better choice.

Return Value

pointer to next node or NULL, when all nodes are deleted

gavl_cut_first

Name

  gavl_cut_first — Cut First Item from Tree

Synopsis

void * gavl_cut_first (gavl_root_t * root);
Arguments

root

GAVL tree root

Description

This enables fast delete of the first item without tree balancing. The resulting tree is degraded but height differences are kept consistent. Use of this function can result in height of tree maximally one greater than the tree managed by optimal AVL functions.

Return Value

returns the first item or NULL if the tree is empty

struct gavl_node

Name

struct gavl_node — Structure Representing Node of Generic AVL Tree

Synopsis

struct gavl_node {
    struct gavl_node * left;
    struct gavl_node * right;
    struct gavl_node * parent;
    int hdiff;
};

Members

left

pointer to left child or NULL

right

pointer to right child or NULL

parent

pointer to parent node, NULL for root

hdiff

difference of height between left and right child
Chapter 1. CAN/CANopen user guide

Description
This structure represents one node in the tree and links left and right to nodes with lower and higher value of order criterion. Each tree is built from one type of items defined by user. User can decide to include node structure inside item representation or GAVL can malloc node structures for each inserted item. The GAVL allocates memory space with capacity sizeof(gavl_node_t)+sizeof(void*) in the second case. The item pointer is stored following node structure (void**)(node+1);

struct gavl_root

Name
struct gavl_root — Structure Representing Root of Generic AVL Tree

Synopsis
struct gavl_root {
  gavl_node_t * root_node;
  int node_offs;
  int key_offs;
  gavl_cmp_fnc_t * cmp_fnc;
};

Members
root_node
  pointer to root node of GAVL tree
node_offs
  offset between start of user defined item representation and included GAVL node structure. If negative value is stored there, user item does not contain node structure and GAVL manages standalone ones with item pointers.
key_offs
  offset to compared (ordered) fields in the item representation
cmp_fnc
  function defining order of items by comparing fields at offset key_offs.

gavl_node2item

Name
gavl_node2item — Conversion from GAVL Tree Node to User Defined Item
Synopsis

void * gavl_node2item (const gavl_root_t * root, const gavl_node_t * node);

Arguments

root
GAVL tree root

node
node belonging to root GAVL tree

Return Value
pointer to item corresponding to node

gavl_node2item_safe

Name

 gavl_node2item_safe — Conversion from GAVL Tree Node to User Defined Item

Synopsis

void * gavl_node2item_safe (const gavl_root_t * root, const gavl_node_t * node);

Arguments

root
GAVL tree root

node
node belonging to root GAVL tree
Return Value
pointer to item corresponding to node

gavl_node2key

Name
gavl_node2key — Conversion from GAVL Tree Node to Ordering Key

Synopsis

void * gavl_node2key (const gavl_root_t * root, const gavl_node_t * node);

Arguments

root
  GAVL tree root

node
  node belonging to root GAVL tree

Return Value
pointer to key corresponding to node

gavl_next_node

Name
gavl_next_node — Returns Next Node of GAVL Tree

Synopsis

gavl_node_t * gavl_next_node (const gavl_node_t * node);
Arguments

node

node for which accessor is looked for

Return Value

pointer to next node of tree according to ordering

gavl_prev_node

Name

gavl_prev_node — Returns Previous Node of GAVL Tree

Synopsis

```c
const gavl_node_t * gavl_prev_node (const gavl_node_t * node);
```

Arguments

node

node for which predecessor is looked for

Return Value

pointer to previous node of tree according to ordering

gavl_balance_one

Name

gavl_balance_one — Balance One Node to Enhance Balance Factor

Synopsis

```c
int gavl_balance_one (gavl_node_t ** subtree);
```
Arguments

subtree

pointer to pointer to node for which balance is enhanced

Return Value

returns nonzero value if height of subtree is lowered by one

gavl_insert_primitive_at

Name
gavl_insert_primitive_at — Low Level Routine to Insert Node into Tree

Synopsis

```c
int gavl_insert_primitive_at (gavl_node_t ** root_nodep, gavl_node_t * node, gavl_node_t * where, int leftright);
```

Arguments

root_nodep

pointer to pointer to GAVL tree root node

node

pointer to inserted node

where

pointer to found parent node

leftright

left (>=1) or right (<=0) branch

Description

This function can be used for implementing AVL trees with custom root definition. The value of the selected left or right pointer of provided node has to be NULL before insert operation, i.e. node has to be end node in the selected direction.
**Return Value**
positive value informs about success

---

**gavl_delete_primitive**

**Name**
gavl_delete_primitive — Low Level Deletes/Unlinks Node from GAVL Tree

**Synopsis**

```c
int gavl_delete_primitive (gavl_node_t ** root_nodep, gavl_node_t * node);
```

**Arguments**

- `root_nodep`
  pointer to pointer to GAVL tree root node
- `node`
  pointer to deleted node

**Return Value**
positive value informs about success.

---

**gavl_cut_first_primitive**

**Name**
gavl_cut_first_primitive — Low Level Routine to Cut First Node from Tree

**Synopsis**

```c
gavl_node_t * gavl_cut_first_primitive (gavl_node_t ** root_nodep);
```
Arguments

root_nodep
    pointer to pointer to GAVL tree root node

Description
This enables fast delete of the first node without tree balancing. The resulting tree is
degraded but height differences are kept consistent. Use of this function can result in
height of tree maximally one greater the tree managed by optimal AVL functions.

Return Value
returns the first node or NULL if the tree is empty

gsa_struct_init

Name
gsa_struct_init — Initialize GSA Structure

Synopsis

void gsa_struct_init (gsa_array_t * array, int key_offs, gsa_cmp_fnc_t *
cmp_fnc);

Arguments

array
    pointer to the array structure declared through GSA_ARRAY_FOR
key_offs
    offset to the order controlling field obtained by UL_OFFSETOF
cmp_fnc
    function defining order of items by comparing fields at offset key_offs.
gsa_delete_all

Name

gsa_delete_all — Delete Pointers to the All Items in the Array

Synopsis

void gsa_delete_all (gsa_array_t * array);

Arguments

array

pointer to the array structure declared through GSA_ARRAY_FOR

Description

This function releases all internally allocated memory, but does not release memory of the array structure

gsa_bsearch_index

Name

gsa_bsearch_index — Search for Item or Place for Item by Key

Synopsis

int gsa_bsearch_index (gsa_array_t * array, void * key, int key_offs,
gsa_cmp_fnc_t * cmp_fnc, int mode, unsigned * indx);

Arguments

array

pointer to the array structure declared through GSA_ARRAY_FOR

key

key value searched for

key_offs

offset to the order controlling field obtained by UL_OFFSETOF
**cmp_fnc**
function defining order of items by comparing fields

**mode**
mode of the search operation

**indx**
pointer to place, where store value of found item array index or index where new item should be inserted

**Description**
Core search routine for GSA arrays binary searches for item with field at offset `key_off` equal to `key` value. Values are compared by function pointed by `*cmp_fnc` field in the array structure `array`. Integer `mode` modifies search algorithm: `GSA_FANY`.. finds item with field value `*key`, `GSA_FFIRST`.. finds the first item with field value `*key`, `GSA_FAFTER` .. index points after last item with `*key` value, reworded - index points at first item with higher value of field or after last item

**Return Value**
Return of nonzero value indicates match found.

### gsa_find

**Name**
gsa_find — Find Item for Provided Key

**Synopsis**

```c
void * gsa_find (gsa_array_t * array, void * key);
```

**Arguments**

`array`
pointer to the array structure declared through `GSA_ARRAY_FOR`

`key`
key value searched for
Return Value
pointer to item associated to key value or NULL.

gsa_find_first

Name
gsa_find_first — Find the First Item for Provided Key

Synopsis

void * gsa_find_first (gsa_array_t * array, void * key);

Arguments

array
    pointer to the array structure declared through GSA_ARRAY_FOR
key
    key value searched for

Description
same as above, but first matching item is found.

Return Value
pointer to the first item associated to key value or NULL.

gsa_find_indx

Name
gsa_find_indx — Find the First Item with Key Value and Return Its Index

Synopsis

void * gsa_find_indx (gsa_array_t * array, void * key, int * indx);
Arguments

*array*

pointer to the array structure declared through `GSA_ARRAY_FOR`

*key*

key value searched for

*indx*

pointer to place for index, at which new item should be inserted

Description

same as above, but additionally stores item index value.

Return Value

pointer to the first item associated to key value or `NULL`.

gsa_insert_at

Name

gsa_insert_at — Insert Existing Item to the Specified Array Index

Synopsis

```c
int gsa_insert_at (gsa_array_t * array, void * item, unsigned where);
```

Arguments

*array*

pointer to the array structure declared through `GSA_ARRAY_FOR`

*item*

pointer to inserted Item

*where*

at which index should be `item` inserted
Return Value
positive or zero value informs about success

gsa_insert

Name
gsa_insert — Insert Existing into Ordered Item Array

Synopsis

int gsa_insert (gsa_array_t * array, void * item, int mode);

Arguments

array
  pointer to the array structure declared through GSA_ARRAY_FOR
item
  pointer to inserted Item
mode
  if mode is GSA_FAFTER, multiple items with same key can be stored into array,
  else strict ordering is required

Return Value
positive or zero value informs about success

gsa_delete_at

Name
gsa_delete_at — Delete Item from the Specified Array Index

Synopsis

int gsa_delete_at (gsa_array_t * array, unsigned indx);
Arguments

array
    pointer to the array structure declared through GSA_ARRAY_FOR

indx
    index of deleted item

Return Value
positive or zero value informs about success

gsa_delete

Name
gsa_delete — Delete Item from the Array

Synopsis

int gsa_delete (gsa_array_t * array, void * item);

Arguments

array
    pointer to the array structure declared through GSA_ARRAY_FOR

item
    pointer to deleted Item

Return Value
positive or zero value informs about success

gsa_resort_buble

Name
gsa_resort_buble — Sort Again Array If Sorting Criteria Are Changed
Synopsis

int gsa_resort_buble (gsa_array_t * array, int key_offs, gsa_cmp_fnc_t * cmp_fnc);

Arguments

array
pointer to the array structure declared through GSA_ARRAY_FOR
key_offs
offset to the order controlling field obtained by UL_OFFSETOF
cmp_fnc
function defining order of items by comparing fields

Return Value

non-zero value informs, that resorting changed order

gsa_setsort

Name
gsa_setsort — Change Array Sorting Criterion

Synopsis

int gsa_setsort (gsa_array_t * array, int key_offs, gsa_cmp_fnc_t * cmp_fnc);

Arguments

array
pointer to the array structure declared through GSA_ARRAY_FOR
key_offs
new value of offset to the order controlling field
cmp_fnc
new function defining order of items by comparing fields at offset key_offs
Return Value
non-zero value informs, that resorting changed order

..../utils/ulut/ul_gsacust.c

Name
..../utils/ulut/ul_gsacust.c  — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ..../utils/ulut/ul_gsacust.c at this point, but none was found. This dummy section is inserted to allow generation to continue.

..../utils/ulut/ul_gsacust.h

Name
..../utils/ulut/ul_gsacust.h  — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ..../utils/ulut/ul_gsacust.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

struct gsa_array_field_t

Name
struct gsa_array_field_t — Structure Representing Anchor of custom GSA Array
Synopsis

struct gsa_array_field_t {
    void ** items;
    unsigned count;
    unsigned alloc_count;
};

Members

items
    pointer to array of pointers to individual items

count
    number of items in the sorted array

alloc_count
    allocated pointer array capacity

../../utils/ulut/ul_hptree.c

Name

../../utils/ulut/ul_hptree.c — Document generation inconsistency

Oops

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The template for this document tried to insert the structured comment from the file ../../utils/ulut/ul_hptree.c at this point, but none was found. This dummy section is inserted to allow generation to continue.</td>
</tr>
</tbody>
</table>

../../utils/ulut/ul_hptree.h

Name

../../utils/ulut/ul_hptree.h — Document generation inconsistency
..../utils/ulut/ul_htimbase.c

Name

..../utils/ulut/ul_htimbase.c — Document generation inconsistency

Oops

..../utils/ulut/ul_htimdefs.h

Name

..../utils/ulut/ul_htimdefs.h — Document generation inconsistency

Oops
struct ul_htim_node

Name

struct ul_htim_node — Timer queue entry base structure

Synopsis

```c
struct ul_htim_node {
    #ifndef UL_HTIMER_WITH_HPTREE
        ul_hpt_node_t node;
    #else
        ul_hpt_node_t node;
    #endif
    ul_htim_time_t expires;
};
```

Members

node

    regular GAVL node structure for insertion into

node

    regular GAVL node structure for insertion into

expires

    time to trigger timer in &ul_htim_time_t type defined resolution
Description
This is basic type useful to define more complete timer types

struct ul_htim_queue

Name
struct ul_htim_queue — Timer queue head/root base structure

Synopsis
struct ul_htim_queue {
    #ifndef UL_HTIMER_WITH_HPTREE
        ul_hpt_root_field_t timers;
    #else
        ul_hpt_root_field_t timers;
    #endif
    int first_changed;
};

Members

    timers
        root of FLES GAVL tree of timer entries

    timers
        root of FLES GAVL tree of timer entries

    first_changed
        flag, which is set after each add, detach operation which concerning of firsts scheduled timer

Description
This is basic type useful to define more complete timer queues types

struct ul_htimer

Name
struct ul_htimer — Standard timer entry with callback function
Synopsis

```
struct ul_htimer {
    ul_htim_node_t htim;
    ul_htimer_fnc_t * function;
    unsigned long data;
};
```

Members

htim

basic timer queue entry

function

user provided function to call at trigger time

data

user selected data

Description

This is standard timer type, which requires data casting in many cases. The type of function field has to be declared in "ul_htimdefs.h" header file.

struct ul_htimer_queue

Name

```
struct ul_htimer_queue — Standard timer queue
```

Synopsis

```
struct ul_htimer_queue {
    ul_htim_queue_t htim_queue;
};
```

Members

htim_queue

the structure wraps &ul_htim_queue structure
Description
This is standard timer type, which requires data casting in many cases

../../utils/ulut/ul_htimmstime.c

Name
../../utils/ulut/ul_htimmstime.c  — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../../utils/ulut/ul_htimmstime.c at this point, but none was found. This dummy section is inserted to allow generation to continue.

../../utils/ulut/ul_itbase.h

Name
../../utils/ulut/ul_itbase.h  — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../../utils/ulut/ul_itbase.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

list_add

Name
list_add — add a new entry
Synopsis

```c
void list_add (struct list_head * new, struct list_head * head);
```

Arguments

- `new`:
  new entry to be added

- `head`:
  list head to add it after

Description

Insert a new entry after the specified head. This is good for implementing stacks.

---

**list_add_tail**

Name

- `list_add_tail — add a new entry`

Synopsis

```c
void list_add_tail (struct list_head * new, struct list_head * head);
```

Arguments

- `new`:
  new entry to be added

- `head`:
  list head to add it before

Description

Insert a new entry before the specified head. This is useful for implementing queues.
list_del

**Name**
list_del — deletes entry from list.

**Synopsis**

```c
void list_del (struct list_head * entry);
```

**Arguments**

`entry`
the element to delete from the list.

**Note**
list_empty on entry does not return true after this, the entry is in an undefined state.

list_del_init

**Name**
list_del_init — deletes entry from list and reinitialize it.

**Synopsis**

```c
void list_del_init (struct list_head * entry);
```

**Arguments**

`entry`
the element to delete from the list.
list_move

Name
list_move — delete from one list and add as another’s head

Synopsis

void list_move (struct list_head * list, struct list_head * head);

Arguments

list
    the entry to move

head
    the head that will precede our entry

list_move_tail

Name
list_move_tail — delete from one list and add as another’s tail

Synopsis

void list_move_tail (struct list_head * list, struct list_head * head);

Arguments

list
    the entry to move

head
    the head that will follow our entry
list_empty

Name
list_empty — tests whether a list is empty

Synopsis

```c
int list_empty (struct list_head * head);
```

Arguments

`head`
the list to test.

list_splice

Name
list_splice — join two lists

Synopsis

```c
void list_splice (struct list_head * list, struct list_head * head);
```

Arguments

`list`
the new list to add.

`head`
the place to add it in the first list.
list_splice_init

Name
list_splice_init — join two lists and reinitialise the emptied list.

Synopsis

void list_splice_init (struct list_head * list, struct list_head * head);

Arguments

list
the new list to add.

head
the place to add it in the first list.

Description
The list at list is reinitialised

list_entry

Name
list_entry — get the struct for this entry

Synopsis

list_entry ( ptr, type, member);

Arguments

ptr
the &struct list_head pointer.

type
the type of the struct this is embedded in.
list_for_each

Name
list_for_each — iterate over a list

Synopsis

list_for_each (pos, head);

Arguments

pos
the &struct list_head to use as a loop counter.

head
the head for your list.

__list_for_each

Name
__list_for_each — iterate over a list

Synopsis

__list_for_each (pos, head);

Arguments

pos
the &struct list_head to use as a loop counter.
**head**

the head for your list.

**Description**

This variant differs from `list_for_each` in that it’s the simplest possible list iteration code, no prefetching is done. Use this for code that knows the list to be very short (empty or 1 entry) most of the time.

**list_for_each_prev**

**Name**

`list_for_each_prev` — iterate over a list backwards

**Synopsis**

```c
list_for_each_prev (pos, head);
```

**Arguments**

- `pos`  
  the &struct list_head to use as a loop counter.
- `head`  
  the head for your list.

**list_for_each_safe**

**Name**

`list_for_each_safe` — iterate over a list safe against removal of list entry

**Synopsis**

```c
list_for_each_safe (pos, n, head);
```
Arguments

pos
the &struct list_head to use as a loop counter.

n
another &struct list_head to use as temporary storage

head
the head for your list.

list_for_each_entry

Name
list_for_each_entry — iterate over list of given type

Synopsis

list_for_each_entry ( pos, head, member);

Arguments

pos
the type * to use as a loop counter.

head
the head for your list.

member
the name of the list_struct within the struct.

list_for_each_entry_reverse

Name
list_for_each_entry_reverse — iterate backwards over list of given type.
**Synopsis**

```
list_for_each_entry_reverse ( pos, head, member);
```

**Arguments**

- `pos`  
  the type * to use as a loop counter.
- `head`  
  the head for your list.
- `member`  
  the name of the list_struct within the struct.

**list_for_each_entry_safe**

**Name**

`list_for_each_entry_safe` — iterate over list of given type safe against removal of list entry

**Synopsis**

```
list_for_each_entry_safe ( pos, n, head, member);
```

**Arguments**

- `pos`  
  the type * to use as a loop counter.
- `n`  
  another type * to use as temporary storage
- `head`  
  the head for your list.
- `member`  
  the name of the list_struct within the struct.
Chapter 1. CAN/CANopen user guide

../..utils/ulut/ul_list.h

Name

../..utils/ulut/ul_list.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file ../..utils/ulut/ul_list.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

../..utils/ulut/ul_utdefs.h

Name

../..utils/ulut/ul_utdefs.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file ../..utils/ulut/ul_utdefs.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

../..utils/ulut/ul_utexport.h

Name

../..utils/ulut/ul_utexport.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file ../..utils/ulut/ul_utexport.h at this point, but none was found. This dummy section is inserted to allow generation to continue.
..../utils/ulut/ul_utmalloc.h

Name

..../utils/ulut/ul_utmalloc.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file ..../utils/ulut/ul_utmalloc.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

libsuiut API

..../utils/suiut/sui_dievc.h

Name

..../utils/suiut/sui_dievc.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file ..../utils/suiut/sui_dievc.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

sui_dinfo_inc_refcnt

Name

sui_dinfo_inc_refcnt — Increase reference count of DINFO
Synopsis

```c
void sui_dinfo_inc_refcnt (sui_dinfo_t * datai);
```

Arguments

`datai`

Pointer to dinfo structure.

File

`sui_dinfo.c`

`sui_dinfo_dec_refcnt`

Name

`sui_dinfo_dec_refcnt` — Decrease reference count of DINFO

Synopsis

```c
void sui_dinfo_dec_refcnt (sui_dinfo_t * datai);
```

Arguments

`datai`

Pointer to dinfo structure.

Description

If the reference count reaches zero, DINFO starts to be destroyed. The event `SUEV_COMMAND` with command `SUCM_DONE` is sent to dinfo, next event `SUEV_FREE` is emmitted or direct `free` is called the `SUEV_FREE` is disabled.

File

`sui_dinfo.c`
sui_create_dinfo

Name
sui_create_dinfo — Creates new dynamic DINFO

Synopsis

sui_dinfo_t * sui_create_dinfo (void * adata, int afdig, long amin, longamax, long ainfo, sui_datai_rdfnc_t * rd, sui_datai_wrfnc_t * wr);

Arguments

adata
  DINFO type specific pointer to the data

afdig
  Number of fractional digits if the fixed decimal point format is used

amin
  The minimal allowed value

amax
  The maximal allowed value

ainfo
  DINFO type specific pointer

rd
  Pointer to the read processing function

wr
  Pointer to the write processing function

Return Value

Pointer to newly created DINFO.

File
sui_dinfo.c
sui_create_dinfo_int

Name
sui_create_dinfo_int — Creates DINFO for signed integer or fixed point data

Synopsis

sui_dinfo_t * sui_create_dinfo_int (void * adata, long aidxsize, int asize);

Arguments

adata
Pointer to the signed char, short, int, long or fixed point data

aidxsize
Allowed range of indexes form 0 to aidxsize-1, if zero, then no check

asize
The size of the integer type representation returned by sizeof

Return Value
Pointer to newly created DINFO.

File
sui_dinfo.c

sui_create_dinfo_uint

Name
sui_create_dinfo_uint — Creates DINFO for unsigned integer or fixed point data

Synopsis

sui_dinfo_t * sui_create_dinfo_uint (void * adata, long aidxsize, int asize);
Arguments

adata
   Pointer to the unsigned char, short, int, long or fixed point data

aidxsiz
   Allowed range of indexes form 0 to aidxsiz-1, if zero, then no check

asiz
   The size of the integer type representation returned by sizeof

Return Value

Pointer to newly created DINFO.

File

sui_dinfo.c

sui_rd_long

Name

sui_rd_long — Reads long integer data from specified DINFO

Synopsis

int sui_rd_long (sui_dinfo_t * datai, long idx, long * buf);

Arguments

datai
   Pointer to the DIONFO

idx
   Index of read data inside DINFO.

buf
   Pointer to where the read value is stored
Return Value
Operation result code, SUDI_DATA_OK in the case of success.

File
sui_dinfo.c

sui_wr_long

Name
sui_wr_long — Writes long integer data to specifies DINFO

Synopsis

int sui_wr_long (sui_dinfo_t * datai, long idx, const long * buf);

Arguments

datai
	Pointer to the DIONFO
idx
	Index of read data inside DINFO.
buf
	Pointer to the new data value

Return Value
Operation result code, SUDI_DATA_OK in the case of success.

File
sui_dinfo.c
**dinfo_scale_proxy**

**Name**
dinfo_scale_proxy — Creates value scale proxy DINFO

**Synopsis**

sui_dinfo_t * dinfo_scale_proxy (sui_dinfo_t * dfrom, long ainfo, long amultiply, long adivide);

**Arguments**

- **dfrom**
  Pointer to the underlying DINFO
- **ainfo**
  The local DINFO specific parameter
- **amultiply**
  Multiply factor
- **adivide**
  Divide factor

**Description**

Creates scaling proxy DINFO. Read value is multiplied by amultiply factor and then divided by adivide factor. The long integer overflow is not checked. If the full checking is required use sui_lintrans_proxy instead which works with wider numbers representations and checks for all overflow cases.

**Return Value**

Pointer to newly created DINFO.

**File**

sui_dinfo.c
**dinfo_simple_proxy**

**Name**

dinfo_simple_proxy — Creates simple proxy DINFO

**Synopsis**

```c
sui_dinfo_t * dinfo_simple_proxy (sui_dinfo_t * dfrom, long ainfo);
```

**Arguments**

dfrom

Pointer to the underlying DINFO

ainfo

The local DINFO specific parameter which specifies index value for calling of underlying DINFO

**Return Value**

Pointer to newly created DINFO.

**File**

sui_dinfo.c

**../../utils/suiut/sui_dinfochk.c**

**Name**

../../utils/suiut/sui_dinfochk.c — Document generation inconsistency

**Oops**

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The template for this document tried to insert the structured comment from the file ../../utils/suiut/sui_dinfochk.c at this point, but none was found. This dummy section is inserted to allow generation to continue.</td>
</tr>
</tbody>
</table>
Name

`../../utils/suiut/sui_dinfo_dbuff.c` — Document generation inconsistency

Oops

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The template for this document tried to insert the structured comment from the file <code>../../utils/suiut/sui_dinfo_dbuff.c</code> at this point, but none was found. This dummy section is inserted to allow generation to continue.</td>
</tr>
</tbody>
</table>

`sui_dinfo_dbuff_create`

Name

`sui_dinfo_dbuff_create` — Creates DINFO for `ul_dbuff` structure

Synopsis

```c
sui_dinfo_t * sui_dinfo_dbuff_create (ul_dbuff_t * db, long aidxsize);
```

Arguments

- `db`
  Pointer to the dbuff

- `aidxsize`
  Allowed range of indexes form 0 to `aidxsize`-1, if zero then no check

Returns

Pointer to newly created DINFO.

File

`sui_dinfo_dbuff.c`
**sui_dinfo_dbuff_rd_dbuff**

**Name**

`sui_dinfo_dbuff_rd_dbuff` — Reads ul_dbuff data from specified DINFO

**Synopsis**

```c
int sui_dinfo_dbuff_rd_dbuff (sui_dinfo_t * di, long idx, ul_dbuff_t * dbuf);
```

**Arguments**

- `di`
  Pointer to the DIONFO
- `idx`
  Index of read data inside DINFO.
- `dbuf`
  Pointer to where the read value is stored

**Return Value**

Operation result code, `SUDI_DATA_OK` in the case of success.

**File**

`sui_dinfo_dbuff.c`

---

**sui_dinfo_dbuff_wr_dbuff**

**Name**

`sui_dinfo_dbuff_wr_dbuff` — Writes ul_dbuff data to specifies DINFO

**Synopsis**

```c
int sui_dinfo_dbuff_wr_dbuff (sui_dinfo_t * di, long idx, const ul_dbuff_t * dbuf);
```
Arguments

di
   Pointer to the DIONFO

idx
   Index of read data inside DINFO.

dbuf
   Pointer to the dbuff

Return Value
Operation result code, SUDI_DATA_OK in the case of success.

File
sui_dinfo_dbuff.c

**sui_dinfo_dbuff_rd_long**

Name
sui_dinfo_dbuff_rd_long — Reads long integer data from specified dbuff DINFO

Synopsis

```c
int sui_dinfo_dbuff_rd_long (sui_dinfo_t * di, long idx, long * buf);
```

Arguments

di
   Pointer to the DIONFO

idx
   Index of read data inside DINFO.

buf
   Pointer to the dbuff
Return Value
Operation result code, SUDI_DATA_OK in the case of success.

File
sui_dinfo_dbuff.c

sui_dinfo_dbuff_wr_long

Name
sui_dinfo_dbuff_wr_long — Writes long integer data to specified dbuff
DINFO

Synopsis

int sui_dinfo_dbuff_wr_long (sui_dinfo_t * di, long idx, const long * buf);

Arguments

di
    Pointer to the DINFO

idx
    Index of read data inside DINFO.

buf
    Pointer to the dbuff

Return Value
Operation result code, SUDI_DATA_OK in the case of success.

File
sui_dinfo_dbuff.c
Chapter 1. CAN/CANopen user guide

../..utils/suiut/sui_dinfo.h

Name

../..utils/suiut/sui_dinfo.h — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../..utils/suiut/sui_dinfo.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

../..utils/suiut/sui_dtrans.c

Name

../..utils/suiut/sui_dtrans.c — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../..utils/suiut/sui_dtrans.c at this point, but none was found. This dummy section is inserted to allow generation to continue.

../..utils/suiut/sui_dtrans.h

Name

../..utils/suiut/sui_dtrans.h — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../..utils/suiut/sui_dtrans.h at this point, but none was found. This dummy section is inserted to allow generation to continue.
sui_dtree_lookup

Name
sui_dtree_lookup — Find dinfo in the named dinfo database

Synopsis

```c
int sui_dtree_lookup (sui_dtree_dir_t * from_dir, const char * path,
                     sui_dtree_dir_t ** found_dir, sui_dinfo_t ** datai);
```

Arguments

from_dir
  the directory to start from
path
  path from directory to dinfo or directory
found_dir
  the optional pointer to space that would hold pointer to directory of found dinfo
datai
  optional pointer to store the found dinfo

Return Value

SUI_DTREE_FOUND,  SUI_DTREE_DIR,  SUI_DTREE_NOPATH,
SUI_DTREE_ERROR

File
sui_dtree.c

../..../utils/suiut/sui_dtree.h

Name
../..../utils/suiut/sui_dtree.h — Document generation inconsistency
Oops

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The template for this document tried to insert the structured comment from the file ../../utils/suiut/sui_dtree.h at this point, but none was found. This dummy section is inserted to allow generation to continue.</td>
</tr>
</tbody>
</table>

sui_dtree_mem_lookup

**Name**
sui_dtree_mem_lookup — Find dinfo in the named dinfo database

**Synopsis**

```c
int sui_dtree_mem_lookup (sui_dtree_dir_t * from_dir, const char * path, int * consumed, sui_dtree_dir_t ** found_dir, sui_dinfo_t ** datai);
```

**Arguments**

- `from_dir`
  - the directory to start from
- `path`
  - path from directory to dinfo or directory
- `consumed`
  - pointer to location for number of consumed characters from path
- `found_dir`
  - the optional pointer to space that would hold pointer to directory of found dinfo
- `datai`
  - optional pointer to store the found dinfo

**Return Value**

- SUI_DTREE_FOUND
- SUI_DTREE_DIR
- SUI_DTREE_NOPATH
- SUI_DTREE_ERROR
struct sui_dtree_memdir_t

Name

struct sui_dtree_memdir_t — Ancestor of sui_dtree_dir_t which containing
sui_dtree_memnode_t GAVL list.

Synopsis

struct sui_dtree_memdir_t {
    sui_dtree_dir_t dir;
    gavl_cust_root_field_t name_root;
};

Members

dir
    base struct (Container_of technology). Containing dir needs it.

name_root
    GAVL with children of type &sui_dtree_memnode_t

Header

sui_dtreemem.h

struct sui_dtree_memnode_t

Name

struct sui_dtree_memnode_t — structure representing single node in
memtree.

Synopsis

struct sui_dtree_memnode_t {
    char * name;
    int node_type;
    gavl_node_t name_node;
    union ptr;
    void * dll_handle;
};
Members

name
structure neccessary for storing node in GAVL tree, is NULL for subindicies

dnode_type
type of node contents (dir or dinfo)

name_node
the structure can be stored in GAVL tree thanks to that field

ptr
pointer to dinfo or directory that this node contains.

dll_handle
if memnode is one imported from DLL, DLLs handle is stored here. (else it is 0)

Description
Node can contain dinfo or directory (&sui_dtree_dir_t).

Header
sui_dtreemem.h

struct sui_event

Name
struct sui_event — Common suitk event structure

Synopsis
struct sui_event {
    unsigned short what;
};

Members

what
Code of event.(See ‘event_code’ enum with ‘SUEV_’ prefix)
enum event_code

Name

enum event_code — Code of SUITK events [‘SUEV_’ prefix]

Synopsis

```c
enum event_code {  
  SUEV_MDOWN,  
  SUEV_MUP,  
  SUEV_MMOVE,  
  SUEV_MAUTO,  
  SUEV_KDOWN,  
  SUEV_KUP,  
  SUEV_DRAW,  
  SUEV_REDRAW,  
  SUEV_COMMAND,  
  SUEV_BROADCAST,  
  SUEV_SIGNAL,  
  SUEV_GLOBAL,  
  SUEV_FREE,  
  SUEV_NOTHING,  
  SUEV_MOUSE,  
  SUEV_KEYBOARD,  
  SUEV_MESSAGE,  
  SUEV_DEFMASK,  
  SUEV_GRPMASK  
};
```

Constants

SUEV_MDOWN
  Mouse button is down.

SUEV_MUP
  Mouse button is up.

SUEV_MMOVE
  Mouse is in move.

SUEV_MAUTO

SUEV_KDOWN
  Key is down.
SUEV_KUP
   Key is up.

SUEV_DRAW
   Draw widget.

SUEV_REDRAW
   Redraw widget.

SUEV_COMMAND
   Command event.

SUEV_BROADCAST
   Broadcast event.

SUEV_SIGNAL
   ?

SUEV_GLOBAL
   ?

SUEV_FREE
   ?

SUEV_NOTHING
   ?

SUEV_MOUSE
   ?

SUEV_KEYBOARD
   ?

SUEV_MESSAGE
   ?

SUEV_DEFMASK
   ?

SUEV_GRPMAK
   ?

File

sui_base.h
enum command_event

Name

enum command_event — Command codes for command event ['SUCM_' prefix]

Synopsis

enum command_event {
   SUCM_VALID,
   SUCM_QUIT,
   SUCM_ERROR,
   SUCM_MENU,
   SUCM_CLOSE,
   SUCM_ZOOM,
   SUCM_RESIZE,
   SUCM_NEXT,
   SUCM_PREV,
   SUCM_HELP,
   SUCM_OK,
   SUCM_CANCEL,
   SUCM_YES,
   SUCM_NO,
   SUCM_DEFAULT,
   SUCM_FOCUASK,
   SUCM_FOCUSSET,
   SUCM_FOCUSREL,
   SUCM_INIT,
   SUCM_DONE,
   SUCM_NEWDISPLAY,
   SUCM_DISPNUMB,
   SUCM_CHANGE_STBAR,
   SUCM_NEXT_GROUP,
   SUCM_PREV_GROUP,
   SUCM_EVC_LINK_TO }

Constants

SUCM_VALID
   VALID command event.

SUCM_QUIT
   QUIT command event.

SUCM_ERROR
   ERROR command event.

SUCM_MENU
   MENU command event. Open, select, close, ... menu.

SUCM_CLOSE
   CLOSE command event.

SUCM_ZOOM
   ZOOM command event.
SUCM_RESIZE
    RESIZE command event.

SUCM_NEXT
    NEXT command event. Mainly for change widget focus by pressing TAB key.

SUCM_PREV
    PREV command event. Mainly for change widget focus by pressing SHIFT+TAB key.

SUCM_HELP
    HELP command event.

SUCM_OK
    OK button pressed.

SUCM_CANCEL
    CANCEL button pressed.

SUCM_YES
    YES button pressed.

SUCM_NO
    NO button pressed.

SUCM_DEFAULT
    DEFAULT button pressed.

SUCM_FOCUSASK
    Which widget has focus?

SUCM_FOCUSET
    Set focus to the widget.

SUCM_FOCUSREL
    Release focus from the widget.

SUCM_INIT
    Initialize widget.

SUCM_DONE
    Done widget - decrement reference counter, deallocate widget data.

SUCM_NEWDISPLAY
    Create new screen from pointer to screen.

SUCM_DISPNUMB
    Create new screen from number to screen.

SUCM_CHANGE_STBAR
    Status bar is changed.
SUCM_NEXT_GROUP
  Change focus between groups (like as ALT+TAB in Windows).

SUCM_PREV_GROUP
  Change focus between groups.

SUCM_EVC_LINK_TO
  Delegate connection of the EVC link to the proxy or target object

File
sui_base.h

../../utils/suiut/sui_internal.h

Name
  ..../../utils/suiut/sui_internal.h — Document generation inconsistency

Oops

Warning
The template for this document tried to insert the structured comment from the file ../../utils/suiut/sui_internal.h at this point, but none was found. This dummy section is inserted to allow generation to continue.

Notes