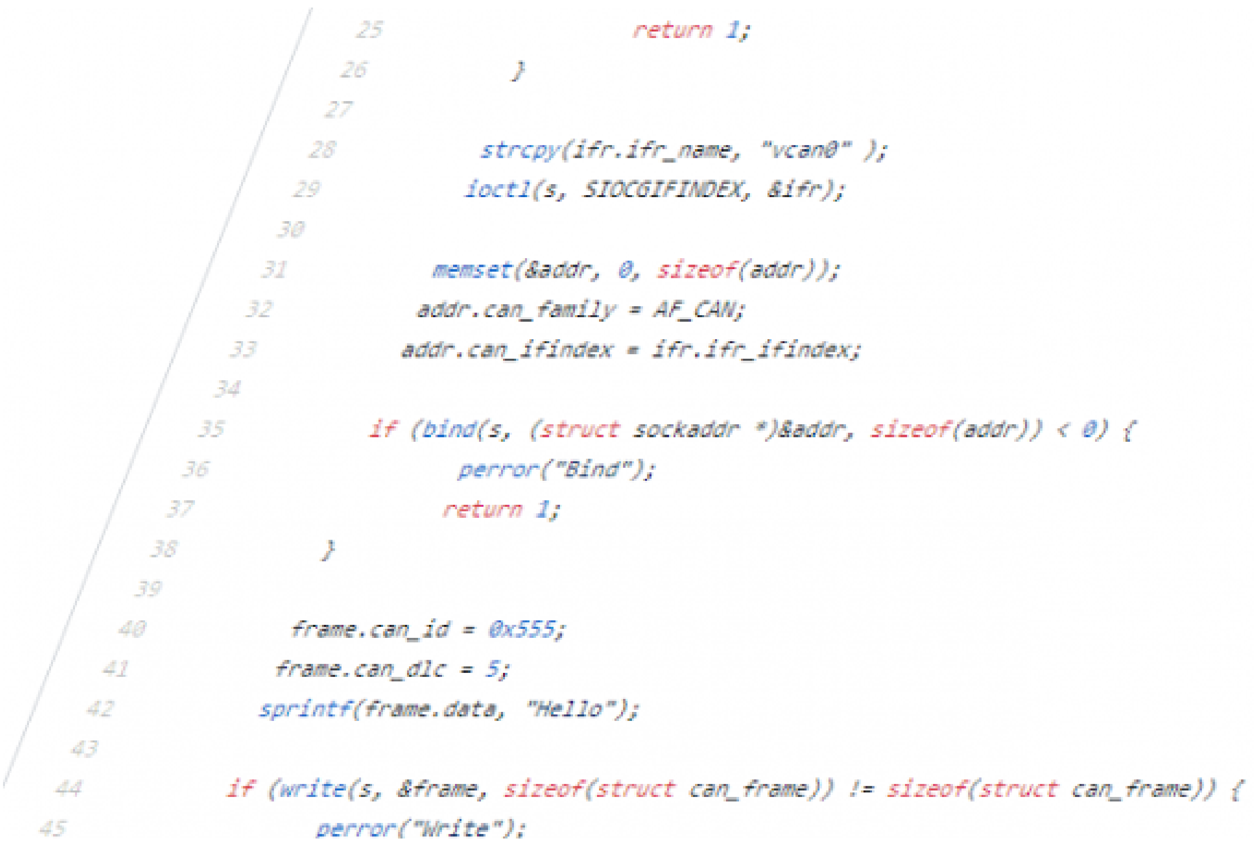
Example C SocketCAN Code



```
                return 1;
        }

        strcpy(ifr.ifr_name, "vcan0" );
        ioctl(s, SIOCGIFINDEX, &ifr);

        memset(&addr, 0, sizeof(addr));
        addr.can_family = AF_CAN;
        addr.can_ifindex = ifr.ifr_ifindex;

        if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
                perror("Bind");
                return 1;
        }

        frame.can_id = 0x555;
        frame.can_dlc = 5;
        sprintf(frame.data, "Hello");

        if (write(s, &frame, sizeof(struct can_frame)) != sizeof(struct can_frame)) {
                perror("Write");
```

Writing user space C code to talk to CAN devices via the Linux SocketCAN interface is relatively simple and efficient. SocketCAN uses the Berkeley socket API and hence is very similar to communicating with other network socket devices. Below is a simple guide to get you started reading, writing and filtering CAN packets.

Official documentation for the SocketCAN interface can be found at:
https://www.kernel.org/doc/Documentation/networking/can.txt

Complete code for the following examples can be found at the following GitHub repository:
https://github.com/craigpeacock/CAN-Examples

These examples do not include make files. To build a source file, you can simply use gcc. For example, to build cantransmit, execute:

```
gcc cantransmit.c -o cantransmit
```

Opening and binding to a CAN socket

The first step before doing anything is to create a **socket**. This function accepts three parameters – domain/protocol family (PF_CAN), type of socket (raw or datagram) and socket protocol. If successful, the function then returns a file descriptor.

```c
int s;

if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
   perror("Socket");
   return 1;
}
```

Next, we must retrieve the interface index for the interface name (can0, can1, vcan0 etc) we wish to use. To do this we send an **I/O control call and pass an ifreq** structure containing the interface name:

```c
struct ifreq ifr;

strcpy(ifr.ifr_name, "vcan0" );
ioctl(s, SIOCGIFINDEX, &ifr);
```

Alternatively, if you use zero as the interface index, you can retrieve packets from all CAN interfaces.

Armed with the interface index, we can now **bind** the socket to the CAN Interface:

```c
struct sockaddr_can addr;

memset(&addr, 0, sizeof(addr));
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
   perror("Bind");
   return 1;
}
```

Sending a frame

To send a CAN frame, one must initialise a *can_frame* structure and populate it with data. The basic *can_frame* structure is defined in include/linux/can.h and looks like:

```c
struct can_frame {
    canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8    can_dlc; /* frame payload length in byte (0 .. 8) */
    __u8    __pad;   /* padding */
    __u8    __res0;  /* reserved / padding */
    __u8    __res1;  /* reserved / padding */
    __u8    data[8] __attribute__((aligned(8)));
};
```

Below we initialise a CAN frame with an ID of 0x555, a payload of 5 bytes containing “hello” and send it using the write() system call:

```
struct can_frame frame;

frame.can_id = 0x555;
frame.can_dlc = 5;
sprintf(frame.data, "Hello");

if (write(s, &frame, sizeof(struct can_frame)) != sizeof(struct can_frame)) {
   perror("Write");
   return 1;
}
```

Reading a frame

To read a frame, initialise a *can_frame* and call the read() system call. This will block until a frame is available:

```
int nbytes;
struct can_frame frame;

nbytes = read(s, &frame, sizeof(struct can_frame));

if (nbytes < 0) {
   perror("Read");
   return 1;
}

printf("0x%03X [%d] ",frame.can_id, frame.can_dlc);

for (i = 0; i < frame.can_dlc; i++)
   printf("%02X ",frame.data[i]);

printf("\r\n");
```

In the example above, we display the ID, data length code (DLC) and payload.

Setting up a filter

In addition to reading, you may want to filter out CAN frames that are not relevant. This happens at the *driver level* and this can be more efficient that reading each frame in a user mode application.

(Most CAN controllers have acceptance filters and masks included in silicon (hardware). Unfortunately, the current architecture performs filtering in the kernel and is not as optimal, but still better than passing all frames up to the user mode app.)

To set up a filter, initialise a single can_filter structure or array of structures and populate the can_id and can_mask. The call setsockopt():

```
struct can_filter rfilter[1];

rfilter[0].can_id   = 0x550;
rfilter[0].can_mask = 0xFF0;
//rfilter[1].can_id   = 0x200;
```

```
//rfilter[1].can_mask = 0x700;

setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter));
```

Closing the socket

And finally, if there is no further need for the socket, close it:

```
if (close(s) < 0) {
   perror("Close");
   return 1;
}
```

Testing

Virtual CAN interface

While you can develop and test code on real hardware, Linux also provides a virtual CAN interface driver (vcan.ko) that acts like a loopback port. When used with **CAN-Utils**, it makes development and testing a breeze. No more trying to work out if your issue is hardware or software related.

To create a virtual CAN network interface called vcan0:

```
sudo ip link add dev vcan0 type vcan
sudo ifconfig vcan0 up
```

The examples supplied above are coded to work with the vcan0 interface straight out of the box (or github repository).

CAN-utils

CAN-utils is a collection of extremely useful debugging tools using the SocketCAN interface. It includes applications such as:

- **candump** – Dump can packets – display, filter and log to disk.
- **canplayer** – Replay CAN log files.
- **cansend** – Send a single frame.
- **cangen** – Generate random traffic.
- **canbusload** – Display the current CAN bus utilisation.
- ISO-TP – Tools for multiple frame transport protocol (**ISO 157650-2**)

Pre-compiled binaries can be installed using (*platform dependant*):

```
sudo apt-get install can-utils
```

Alternatively, the CAN-utils source code can be obtained from the GitHub repository: **https://github.com/linux-can/can-utils**