

# Code Generation with Anthropic Claude-1.5-Sonnet and Aider-chat

---

## SETUP

### 1. Clone starter repo from github

```
>git clone https://github.com/dmh2000/ai-gen.git
>git status
>git branch snc
>git switch snc
```

---

### 2. Setup Conda Virtual Environment

Conda is a command line utility that lets you set up 'virtual environments' that include specific software dependencies that you can for a particular application. Its mostly used with Python. If you are developing with python, you probably should be using conda or its big brother Anaconda.

Its not the same as docker. Its not a container. Its just setups up a separate directory with the dev tools needed for a particular project. Then it points the environment at that directory. It gives you a repeatable dev tool environment that can be duplicated.

- <https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html>

```
# conda create -n <venv-name> python=3
conda init
conda create -n aider python=3
conda activate aider
pip install aider-chat
```

---

### 3. Load VCAN Module

Linux 'virtual CAN' creates a local CAN bus that can be used to test offline software using a CAN bus

- CAN is a bus architecture
- There are not addresses, only messages
- each message has an id
- various clients can send and/or receive messages
- CAN has a cool way of priotizing messages by message id

- the 'ip' command is used for creating virtual network interfaces
- <https://linuxconfig.org/configuring-virtual-network-interfaces-in-linux>
- <https://www.pragmaticlinux.com/2021/10/how-to-create-a-virtual-can-interface-on-linux/>

```
#!/bin/sh

# check that the virtual CAN interface is loaded and ready
# ifconfig vcan0
# Load the kernel module.
sudo modprobe vcan
# Create the virtual CAN interface.
sudo ip link add dev vcan0 type vcan
# Bring the virtual CAN interface online.
sudo ip link set up vcan0
# Check its ok
sudo ip link show vcan0
ifconfig vcan0
```

---

## 4. Configure Initial Directory Structure

Set up an initial directory framework. Something simple.

```
#!/bin/bash

# install aider-chat
# pip install aider-chat
aider --version

# setup directory structure
mkdir c
mkdir can
mkdir g
cd g && go mod init can
mkdir ts
tree
```

---

## 5. Start of Aider conversation

```
aider
# Loaded /home/dmh2000/.env (api keys. DO NOT ADD TO GIT!)
# Aider v0.46.1
# Models: claude-3-5-sonnet-20240620 with diff edit format, weak model
claude-3-haiku-20240307
```

## create a library source for interfacing to VCAN

using the C language, implement functions to access a CAN bus using network sockets. The file should include functions that open, write, read and close a CAN socket. Write the file to can/can.c

### ERROR fix : struct ifreq ifr not defined

Took a couple of minutes to look this up. There is a linux specific file required.

add the file 'linux/if.h to can/can.c

### create the include file for can

create a file can/can.h that exports the functions from can/can.c

### MANUAL FIX

I didn't like having to include the system file can.h in client code because it can limit portability. So manually modify it to have its own definition of can\_frame.

using the updated file can/can.h as a model, modify can/can.c to match the function prototypes defined there

### build the library

create a Makefile in can that compiles can/can.c and builds a static library

### check the Makefile works

- cd can
- Make

### WARNING type mismatch

```
canlib.c:71:16: warning: comparison of integer expressions of different
signedness: 'int' and 'long unsigned int' [-Wsign-compare]
  71 |         if (nbytes < sizeof(struct can_frame)) {
      |                ^
```

in can/can.c change line 71 to add a cast (int)sizeof(struct can\_frame)

Rerun the Makefile

- make (should build without warnings)

create a can bus client in directory 'c'.

in directory 'c' create a file named sender.c. create a main function that opens a can socket. add a 32 bit signed integer variable named 'count' initialized to 0. add a loop that sends the contents of the variable count to the can bus socket at 10 Hz. increment count by 1 on each loop iteration

### build the client

add a makefile to directory c that compiles c/sender.c and includes the library libcan.a from directory can

Test the Makefile

### WARNING 'usleep undefined'

usleep is deprecated. use nanosleep instead. nanosleep requires a newer version of POSIX source than the default

in file sender.c change the call to usleep to use nanosleep. add an argument to c/Makefile CFLAGS to define the macro `\_POSIX_C_SOURCE=200000`

### Other fixes

in file sender.c include string.h to fix undefined memcpy

### See if 'sender' runs

```
./sender
```

In separate shell

```
candump vcan0
```

Manually update .gitignore to exclude build artifacts

### IN DIRECTORY 'g'

#### create go file

create a main.go file in directory 'g' and add an empty main function

#### add code to receive can messages

in the file g/main.go, add code to receive from a can socket using the library can/can.h. use can id 0x123. use 'cgo' for this

#### get rid of unsafe pointers

in g/main.go, instead of using an unsafe pointer to the can frame data, assume the value received is a 32 bit signed integer. convert the bytes to a 32 bit signed integer using shifts and adds.

WORKING!

## Conclusion

### Answers to the Questions

- does it save time?
  - yes it helps by generating boilerplate
- does it work first time?
  - sometimes
- can it fix errors
  - yes but not consistently
  - it helps if the dev knows what is going on
- does it require manual intervention
  - yes
  - sometimes it is just easier to edit the code than prompt
  - aider always re-reads the affected files so it sees any manual changes