Understanding the CAN Bus

A Team-wide Simplified CAN Training Module

The purpose of this document is to provide a consolidated training module for those throughout the team that would like to gain a basic-to-intermediate level understanding of CAN.

Introduction

The **Controller Area Network Bus**, or **CAN Bus**, is the internal communications network used to allow *microcontrollers and other devices* to send information to one another. Every modern vehicle built after 1996 uses this network to send data across it.

Modern cars and trucks contain many of these **microcontrollers**, or small computers, to *monitor, control, or adjust settings* for different vehicle modules such as the engine, transmission, and motor. The automotive industry term for a microcontroller is an **electronic control unit** or **ECU**. Today, almost every device inside a car that uses or interacts with electricity has an ECU attached to it. In cars sold within the last five years (2017-2021), it is not uncommon to find *anywhere from 70 to 150 total ECUs*.

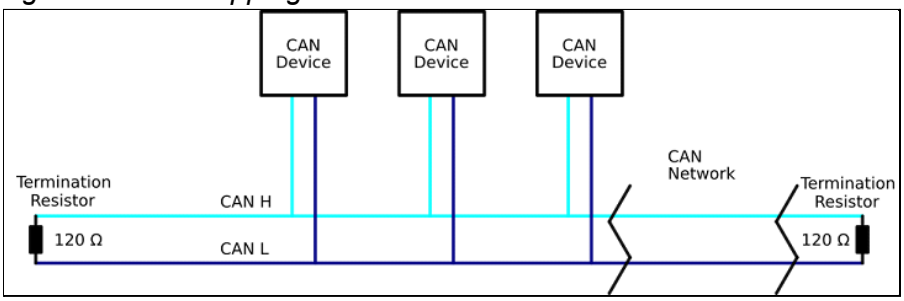
The Problem

Early cars with this technology contained only a few ECUs; only one data wire had to be connected between them and messages could easily be sent to each module. However, as the number of ECUs increased, so did the number of wires. Every ECU has to communicate with every other ECU in the vehicle, so if each connection between each ECU needs one wire, the number of wires could become exponentially large, like a *giant spider-web*. Running wires through the vehicle is not only very challenging but at this number could cause a decrease in vehicle weight efficiency and even possible signal losses.

The Solution

The CAN Bus solves this problem by **multiplexing** or allowing multiple signals from many ECUs to *share the same communication wires*. Instead of having a “giant spider-web” of interconnected wires between each ECU, CAN allows all communication between all ECUs to happen with *just two wires*. Once each ECU in the vehicle taps into this two-wire harness, it can communicate seamlessly with all other ECUs in the vehicle.

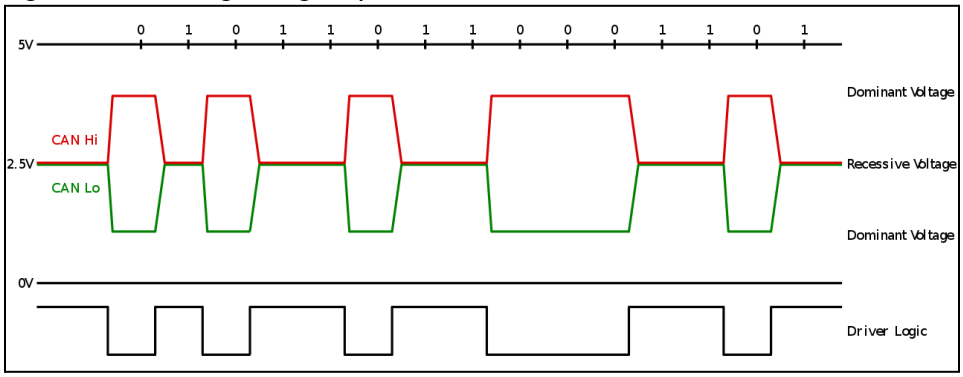
Figure 1: ECUs tapping into a CAN Network



How Signals are Sent over CAN

CAN signals sent across the vehicle via these two wires are made up of *binary 0s and 1s*. By sending these 0s and 1s in a particular order, a *binary message* can be sent and decoded by other ECUs on the network. The CAN Bus does this by using the voltage between its two wires. One of these CAN wires is labeled as **CAN High** while the other is labeled as **CAN Low**. To send a *binary 1*, both the CAN High and CAN Low wires stay at 2.5 V. To send a *binary 0*, the CAN High wire is increased to 5 V while the CAN Low wire is decreased to 0 V. Both wires will send the same binary message at the same time.

Figure 2: CAN Signaling Representation



CAN Bus Physical Design

The reason for sending the same signal over two wires instead of one is to *reduce noise* on the network. All ECUs contain individual CAN Controllers which compare these two simultaneous signals and filter out any electromagnetic interference that may have distorted the signal. To further eliminate noise, the two CAN wires are run in a *twisted pair* configuration with the CAN High and Low wires braided around each other. In areas of extreme electromagnetic interference such as near a hybrid/electric vehicle's inverter or motor, additional *braided metal shielding* may be required to eliminate noise.

Figure 3: Twisted Pair Wire

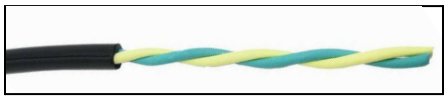
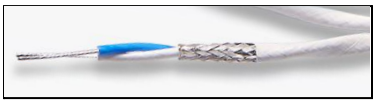


Figure 4: Shielded CAN Wire



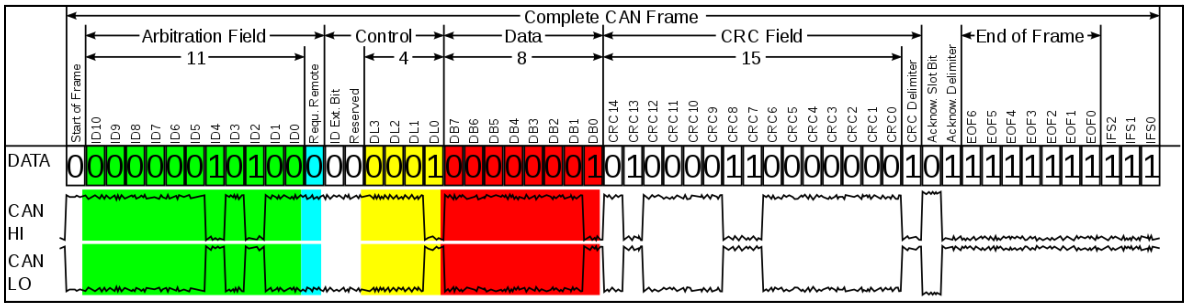
Since signals are being transmitted across relatively large distances in vehicles, **Transmission Line Effects** need to be accounted for. Without going into the specifics of such a complex topic, all twisted-pair CAN wires are designed with 120 Ω of something called **characteristic impedance**, which allows wires to send data over great distances without losing any power. However, because wires with characteristic impedance tend to reflect signals off of wire ends and can therefore cause signal losses, every CAN wire must be *terminated by two 120 Ω resistors on either end of the wire*. These resistors act to both absorb the reflections and mark the termination point of any given CAN wire. Note the "Termination Resistors" in *Figure 1*.

How ECUs Talk over CAN

If CAN signals were just a constant stream of binary data, some problems would emerge. First, there needs to be a way to tell when an old message has ended and a new message has begun. There also needs to be a way to identify which ECU the message was sent from and where the message is going. The CAN Network Protocol solves these problems by organizing the 0s and 1s, or **bits**, into 49- to 131-bit long messages called **CAN Frames**. Each frame represents a single message organized further into **fields** that can be used to *transmit data, requests, delays, or error information*. Since this can get very complex, described below are the important fields which correspond to *Figure 5*:

- **Start of Frame (SOF) -** Designates the start of the frame
- **11-bit Identifier (ID) -** A unique 11-bit identifier that represents the message's priority
- **Remote Transmission Request (RTR) -** If the frame is a data or remote-request frame
- **Data Length Code (DLC) -** Number of bytes of data (0-8 bytes)
- **Data Field -** Data to be transmitted
- **End of Frame (EOF) -** Designates the end of the frame

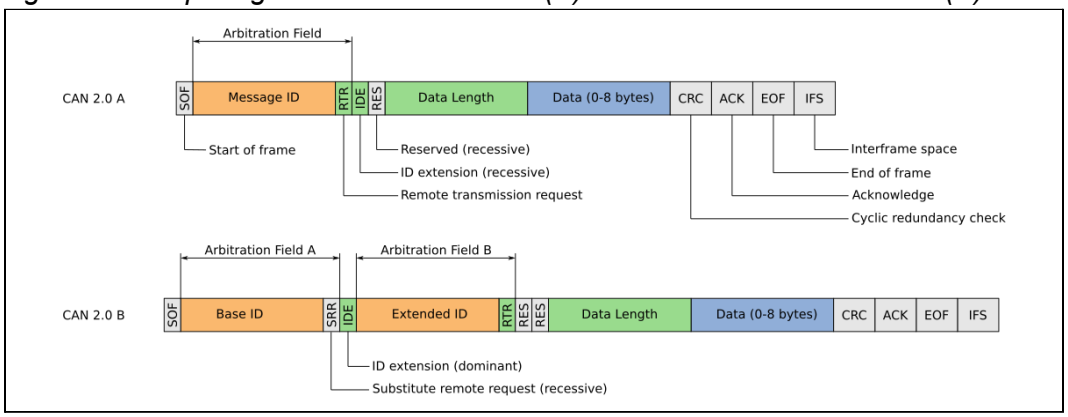
Figure 5: Complete CAN Frame



CAN Frames also have an additional **Extended Frame Format** for Identifiers (IDs) that are longer than 11 bits; these frames will contain a second 18-bit Identifier field in addition to the 11-bit field that we already have, creating a longer 29-bit Identifier to be used when larger ID numbers are needed. Note the "Extended ID" in *Figure 6-B*.

Although ECUs can easily decipher and create these CAN Frames, they are nearly unreadable to humans in this binary form. Using an external module that can connect to a PC called an **interface** or **logger**, these messages can be *translated into English sentences*.

Figure 6: Comparing Base Frame Format (A) & Extended Frame Format (B)



CAN Frame Example

Let's say you would like to know the Diagnostic ID of the Battery Control Module (BCM). Using an interface, you could send a CAN Frame to tell the Central Gateway Module (CGM) to request the Diagnostic ID from the BCM. The CGM will then send the following frame encoded in binary:

Binary

0 0000010100 1 0 0 1000 10000110110111111111011110100000000110010001011110001101100000100000011000000001 01 1111111

Hexadecimal

14 1 8 10DB FE F4 03 22 F1 B0

English

Remote-request (request for data) *message with ID #*20 *and a data field of* 8 *bytes long:*
Functional Request 10DB for every module FE from the CGM F4; 3 bytes of data 03 in the form of a question 22 asking for the Diagnostic ID F1 B0.

CAN Message Priority

From the previous section's example, we can see that it was not an exaggeration to say that the CAN Bus allows modules to talk to one another. We could even describe the types of messages CAN is able to send and receive with English sentence-types.

To review, English sentence-types can be described by their functions: *declarative* for statements, *interrogative* for questions, *imperative* for commands, and *exclamative* for exclamations. The CAN frame example uses the RTR bit and Data field to ask the BCM for its Diagnostic ID, which matches the interrogative sentence structure. The CAN frame can actually use the RTR and Data fields to form interrogative, declarative, and some imperative sentences when it is sending or responding to a request or when it is just transmitting packets of data.

With just these "sentence-types" or **CAN Frame Structures**, about 90% of the functionality of the CAN protocol can be achieved. However, we still have a problem. Let's say a vehicle is about to crash and needs to send an "exclamatory-type" signal that is of utmost importance. As CAN signals are sent extremely quickly (around one million 0s and 1s per second), safety features such as *airbags* and *ABS braking* can be connected to the CAN Bus. Although signals

are usually sent *sequentially* (in order of creation), they can still collide with one another if more than one device transmits at the same time. When this happens, only one CAN frame out of the two gets sent. To ensure that this does not happen with airbag signals or any other signals of importance, the CAN Protocol has a *priority system*.

To summarize, the message with the higher Identifier field number will have the highest priority of transmission if more than one device transmits at the same time. This can also notify other modules to stop transmitting altogether for a short period of time. When the vehicle finally crashes, the airbags can activate and send a CAN signal in the range of hundreds of microseconds without being blocked by other lower priority signals.

Interfacing with CAN

Interfaces or loggers are used to read and translate binary CAN frames into human-readable signals. However, the messages contained within the data field of each CAN frame can mean *something different to different vehicles*. For example, reading a signal from a Chevrolet Blazer might display a message saying that the passenger-side door is open, while reading that same signal on another vehicle may display the message responsible for the status of the wiper blades. Therefore, each vehicle has its own **DBC File** for every CAN Network. DBC files or *CAN Database files* are text files that contain the information needed to translate raw CAN messages into human-readable values. When paired with an interface, a DBC file can read and display these CAN messages in real-time.

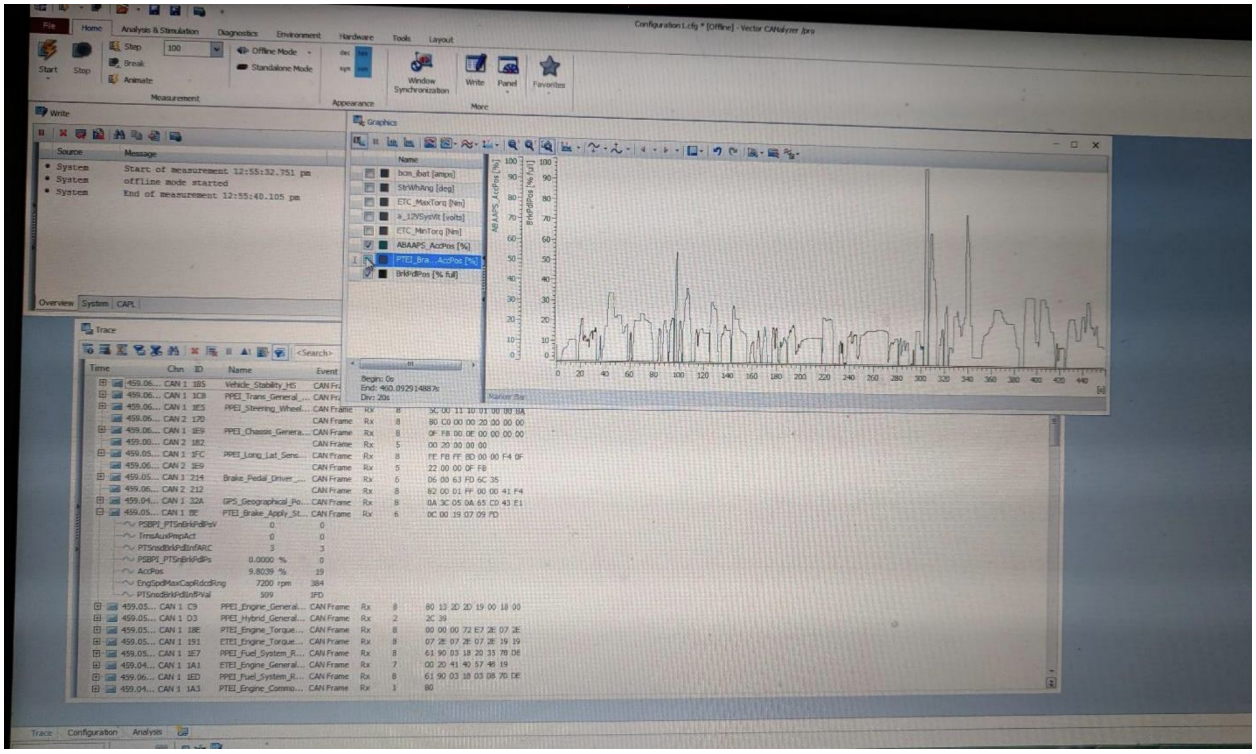
In EcoCAR, we use a Vector CAN/LIN Interface connected to a PC running CANalyzer software and employ the “Trace” and “Graphics” tools to show all messages received over a log period as well as dynamic information like brake and accelerator position (*Figure 10*). The PCM or Controls Swimlane is usually responsible for creating some of the CAN test harnesses that are used to connect to the interface. These cables will connect to the vehicle’s **OBD-II port** (*Figure 7*), which is a required standard port on all American and European vehicles as of 1996, and acts as an output for all stock CAN Networks in the vehicle. On the other end of the cable will usually be a **DB9 Connector** (*Figure 7*) which has 9 inputs, two of which (pins 2 and 7) are used for CAN High and Low; this DB9 Connector is what connects directly to the interface.

Figure 7: OBD-II Port

Figure 8: DB9 Connector

Figure 9: Vector CAN/LIN Interface

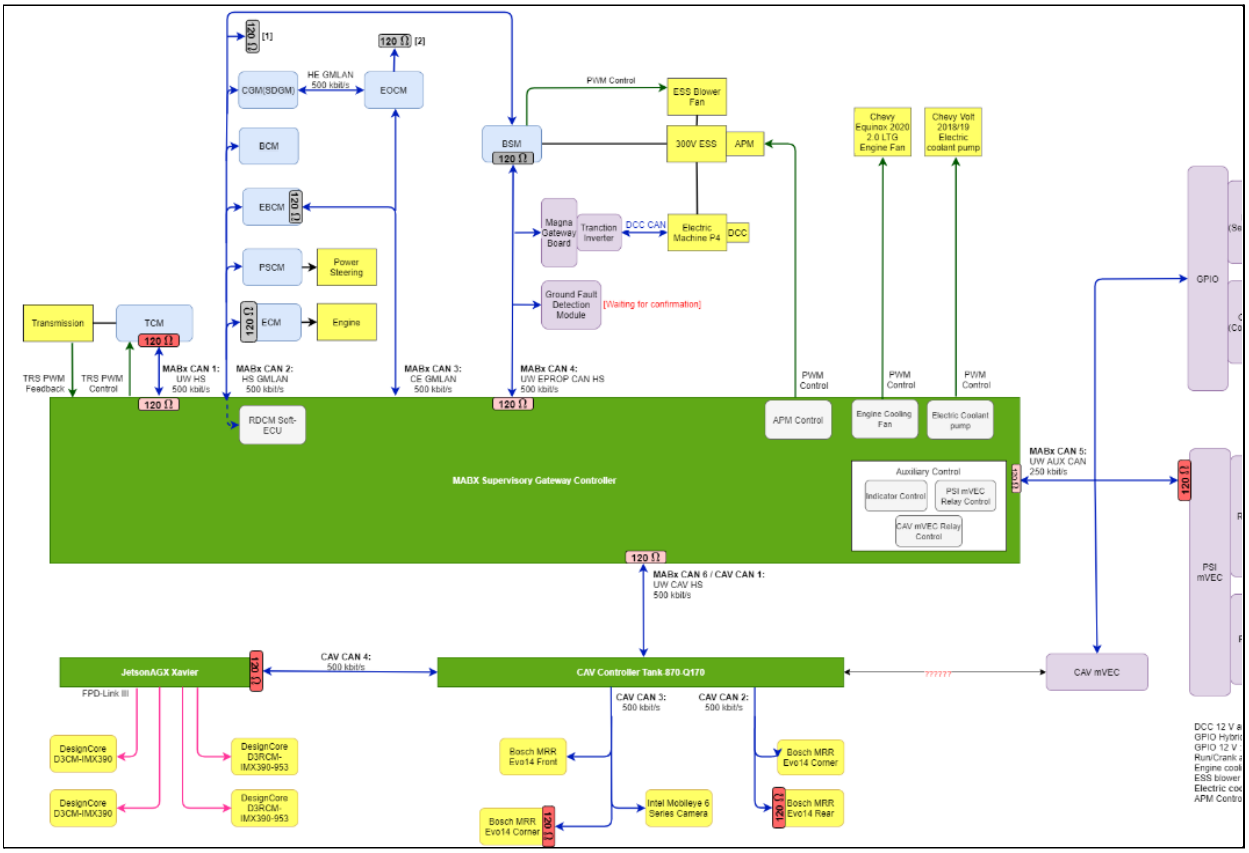
Figure 10: CANalyzer Software with “Trace” and “Graphics” Windows



Wiring a CAN Bus

While the PCM and CAVs swimlanes will usually only be responsible for creating a few test harnesses, the Low Voltage Project Team of the Propulsion System Integration (PSI) swimlane will likely be responsible for creating all team-added CAN Buses during the integration phase of the EcoCAR competition. A basic map of all six team-added CAN Buses and many more CAN-connected components from the Chevrolet Blazer is shown and linked in *Figure 11*. If you would like to zoom in to see more detail in this diagram, please click the image to be taken to the full document.

Figure 11: CAN Serial Network Diagram



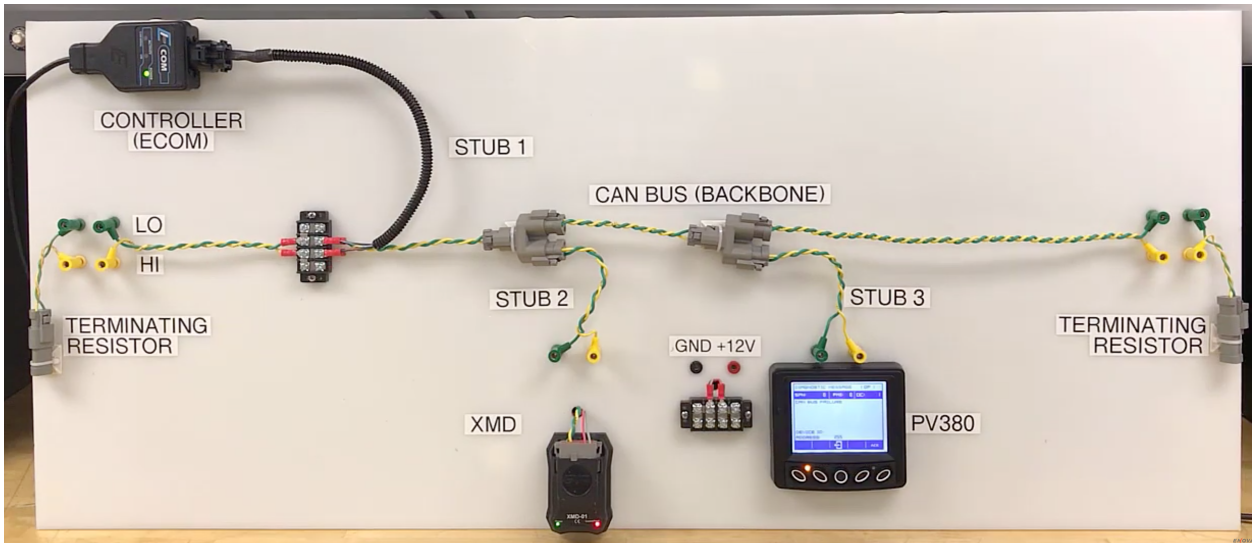
From *Figure 11*, we can see that each CAN Network is labeled with a different number *from CAN 1 to CAN 6*. We can also see *three green boxes*, which are the ECUs that send out CAN messages to the blue, yellow, and purple modules. The biggest green box is the **MABx Supervisory Gateway Controller** which sends out messages to every other ECU and module

in the car. The second biggest green box is the **CAV Controller** or **Intel Tank** which sends out messages to all of the sensors and cameras around the vehicle.

Each team-added CAN network is usually designated for a specific vehicle area or application, since having multiple CAN networks for different purposes makes them easier to test and troubleshoot. Looking back to *Figure 11*, we can see that CAN 5 is used primarily for ECUs on the controller shelf while CAN 6 is used for sensors across the vehicle. However, this diagram does not show what the networks are physically composed of.

Figure 12 is a training example of a team-added CAN Bus with grey **Deutsch/Amphenol connectors**, twisted CAN wire, and example ECUs. Focusing on just the wires and connectors, we can see that either end of the network is terminated by a grey 120 Ω resistor-connector. Additionally, looking towards the center of the bus, we can see two bigger **Y-type** grey connectors which split the CAN networks into two **stubs**, or short offshoots of CAN wire used to connect to an ECU. In this way, all team-added CAN networks are organized much like *a tree*, branching up and outward into stubs that connect to ECUs much like leaves at the ends of tree branches.

Figure 12: Example Team-added CAN Bus



Pinning Each Connector

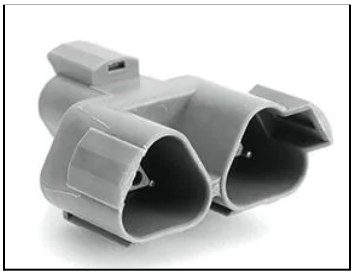
Since leads and experienced Low Voltage Project Team members can more-easily demonstrate how to pin the grey Deutsch/Amphenol connectors, this section will just give a brief introduction to the procedure.

After collecting 3-pin Deutsch/Amphenol Connectors with 3 Female & 3 Male Pins (*Figure 13*), 3-pin 120 Ω Deutsch/Amphenol Resistors (*Figure 14*), 3-pin Y-type Deutsh/Amphenol Connectors (*Figure 15*), CAN wire, and various other wiring, pinning, crimping, and soldering tools, we can start preparing the wires for pinning.

Figure 13: 3-pin Connectors & Pins

Figure 14: 3-pin Resistor

Figure 15: Y-type Connector



Preparing Wires

First, we cut some CAN wire about four inches over the desired length. We then strip about two inches of just the white plastic jacket covering to expose the shielding underneath, but not cut into the shielding. Holding the end of the wire towards us, we carefully insert the knife between the two wires and inside the shielding and cut the shielding such that it turns from a tube shape surrounding the wires to a flat, paper-like shape under the wires. Without cutting the shielding from the rest of the wire, we twist the wire shielding mesh to form a tertiary wire that electrically connects to the shielding throughout the rest of the wire. To make this easier later on, we solder a 20 AWG-size wire to this shielding and strip about a quarter of an inch. We then untwist the exposed twisted pair of wires and strip the high and low wires about a quarter of an inch.

Pinning & Crimping Wires

Next, we take a crimping tool as well as a few female pins and crimp the pins onto each individual CAN wire in addition to the newly made “shielding wire” by squeezing the tool around the pins around the exposed wires.

Inserting Wires into the Connector

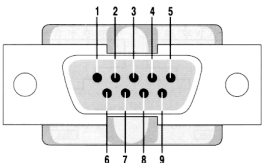
Finally, we insert each pinned wire into the connector such that it makes a clicking sound. CAN High goes into pin A, Low into B, and shielding into C. Once three distinct clicks are heard, a female wedge-lock is inserted into the other end of the connector to lock the pins in place.

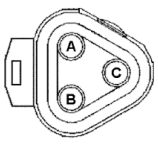
Summary

To reiterate and add to some important points from this training document:

- CAN uses two wires to send and receive messages: **CAN High** (CAN H) and **CAN Low** (CAN L).
- **120 Ω resistors** are needed to terminate CAN connections at either end of each network.
- The CAN Protocol sends messages **in binary**, with 0 as a dominant bit and 1 as recessive.
- CAN messages are sent sequentially, but **some messages have higher priority than others**.
- If more than one device transmits at the same time, **the highest priority message is sent first**.

UW Blazer CAN Conventions			
	DB-9	**Deutsch 3-pin**	**Cable Color**
CAN High	A	7	White-Blue
CAN Low	B	2	White
Shielding / Ground	C	3	Green





Works Cited

"CAN Bus." *Wikipedia*, Wikimedia Foundation, 18 Feb. 2021, en.wikipedia.org/wiki/CAN_bus.

"CAN DBC File Explained - A Simple Intro [+Editor Playground]." *CSS Electronics*, www.csselectronics.com/screen/page/can-dbc-file-database-intro/language/en.

"Can-Bus-Cable-10ft." *AndyMark*, Workarea, andymark-weblinc.netdna-ssl.com/product_images/can-bus-cable-10ft/5bd35a6661a10d295496434a/zoom.jpg?c=1540577894.

"db9-Male-Crimp-Connector-Kit." *ShowMeCables*, Infinite, showmecables-static.scdn3.secure.raxcdn.com/media/catalog/product/cache/0aa466c0d56d23d446bcd4aa4538d2f7/d/b/db9-male-crimp-connector-kit-plastic-1109-ck-m-1.jpg.

Hennessy, Bryan. "An Introduction to J1939 and DBC Files." *Kvaser*, 13 Sept. 2019, www.kvaser.com/developer-blog/an-introduction-j1939-and-dbc-files/.

JamesBM. "AST-CAN485 Hookup Guide." *SparkFun*, SparkFun Electronics, 14 Feb. 2019, learn.sparkfun.com/tutorials/ast-can485-hookup-guide/introduction-to-can-bus.

Kumparak, Greg. "Obdii." *Tech Crunch*, Verizon Media, 19 Apr. 2017, techcrunch.com/wp-content/uploads/2017/04/obdii.jpg.

McBride, Cooper. "CAN Bus Overview." UW EcoCAR, 2018.

"Shielded CAN Wire." *Gore*, W. L. Gore & Associates, Inc., www.gore.com/sites/g/files/ypyipe116/files/2019-12/THUMB-STP-DXN2602-Coiled1.jpg.