User’s Guide   
for the FHWA Connected Vehicle Research Platform

**Publication No. FHWA-HRT-15-xxx  
September 2015**



**FOREWORD**

The platform technology supporting the Cooperative Adaptive Cruise control (CACC) application consisted of a platoon of five (5) 2013 Cadillac SRX vehicles. Each vehicle is designed to be a complete platform for research in CACC and Advanced Driver Assist Systems (ASAS). Each research platform was outfitted with a set of custom Electronic Control Units (ECUs) to enable longitudinal control, a specialized real-time computing platform, an auxiliary pc-based computing platform, and an integrated localization solution. The purpose of this document is to provide introduction and guidance on the hardware and software systems of this research platform.

The following list of Acronyms is used throughout this document:

|  |  |
| --- | --- |
| ACC | Adaptive Cruise Control: Headway to leading Remote Vehicle is maintained through vehicle control system |
| CACC | Cooperative Adaptive Cruise Control: Headway to leading Remote Vehicle is maintained through vehicle control system using DSRC |
| CVRP | Connected Vehicle Research Platform: An open vehicle hardware and Software platform consisting of vehicle control, general purpose computing, and Positioning localization interfaces that enables Research Applications to control vehicle systems such as Brake and Throttle |
| DSRC | Dedicated Short Range Communication: Vehicle-to-Vehicle and Vehicle-to-Roadside communication Protocols |
| Host Vehicle | Subject vehicle, Vehicle is question |
| Lead vehicle | The first vehicle in the Platoon |
| Leading Remote Vehicle | The Remote Vehicle immediately in front of the Host Vehicle; The first Remote Vehicle in front of the Host Vehicle. |
| Remote Vehicle | Any vehicle within communication range of the Host vehicle |
| Research Application | Software Application using the Vehicle Research Platform to control vehicle brake and throttle |

**Notice**

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the use of the information contained in this document. This report does not constitute a standard, specification, or regulation.

The United States Government does not endorse products or manufacturers. Trademarks or manufacturers’ names appear in this report only because they are considered essential to the objective of the document.

**Quality Assurance Statement**

The Federal Highway Administration (FHWA) provides high-quality information to serve Government, industry, and the public in a manner that promotes public understanding. Standards and policies are used to ensure and maximize the quality, objectivity, utility, and integrity of its information. FHWA periodically reviews quality issues and adjusts its programs and processes to ensure continuous quality improvement.

# Technical Report Documentation Page

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Report No.  FHWA-HRT-15-xxx | 2. Government Accession No. | | | 3. Recipient’s Catalog No. | | |
| 4. Title and Subtitle  User’s Guide for the CACC Vehicle Research Platform | | | | 5. Report Date  September 2015 | | |
| 6. Performing Organization Code: | | |
| 7. Author(s) | | | | 8. Performing Organization Report No. | | |
| 9. Performing Organization Name and Address  Leidos, Inc.  11251 Roger Bacon Drive  Reston, VA 20190 | | | | 10. Work Unit No. | | |
| 11. Contract or Grant No.  DTFH61-12-D-00020 | | |
| 12. Sponsoring Agency Name and Address  Office of Operations Research and Development  Federal Highway Administration  6300 Georgetown Pike  McLean, VA 22101-2296 | | | | 13. Type of Report and Period Covered  User’s Manual | | |
| 14. Sponsoring Agency Code  HRDO-20 | | |
| 15. Supplementary Notes  FHWA Contracting Officer’s Technical Representative (COTR): Taylor Lochrane | | | | | | |
| 16. Abstract  The platform technology supporting the Cooperative Adaptive Cruise control (CACC) application consisted of a platoon of five (5) 2013 Cadillac SRX vehicles. Each vehicle is designed to be a complete platform for research in CACC and Advanced Driver Assist Systems (ASAS). Each research platform was outfitted with a set of custom Electronic Control Units (ECU) to enable longitudinal control, a specialized real-time computing platform, an auxiliary pc-based computing platform, and an integrated localization solution. The purpose of this document is to provide introduction and guidance on the hardware and software systems of this research platform. | | | | | | |
| 17. Key Words | | | 18. Distribution Statement  No restrictions. This document is available through the National Technical Information Service, Springfield, VA 22161. | | | |
| 19. Security Classif. (of this report)  Unclassified | | 20. Security Classif. (of this page)  Unclassified | | | 21. No. of Pages  26 | 22. Price  N/A |
| Form DOT F 1700.7 (8-72) | | Reproduction of completed page authorized | | | | |



**Table of Contents**

[1. Vehicle Research Platform 1](#_Toc431329656)

[1A. TORC Controllers - Connections into the Vehicle 1](#_Toc431329657)

[1Ai. TORC Longitudinal Controller 1](#_Toc431329658)

[1Aii. Control Details 1](#_Toc431329659)

[1Aiii. Safety Considerations 3](#_Toc431329660)

[1B. MicroAutoBox 4](#_Toc431329661)

[1C. Simulink 5](#_Toc431329662)

[1D. Linux PC 5](#_Toc431329663)

[1Di. Secondary PC 5](#_Toc431329664)

[*1Dii.* V2IServer *- Forwarding PinPoint and DSRC Data to MAB* 6](#_Toc431329665)

[1Diii. Object Fusion 7](#_Toc431329666)

[1E. DSRC 9](#_Toc431329667)

[1F. PinPoint™ Localization Solution 10](#_Toc431329668)

[2. Description of APIs - How to use the Platform 12](#_Toc431329669)

[2A. Accessing Data in the CarmaData Object 12](#_Toc431329670)

[2B. PinPoint™ Localization System 14](#_Toc431329671)

[2Bi. Data Elements 16](#_Toc431329672)

[2Bii. How to access Data 19](#_Toc431329673)

[2C. Vehicle CAN Bus 1 19](#_Toc431329674)

[2Ci. Data Elements 19](#_Toc431329675)

[2Cii. How to access Data 20](#_Toc431329676)

[2D. Vehicle CAN Bus 2 (Forward Object Bus) 21](#_Toc431329677)

[2Di. Data Elements 21](#_Toc431329678)

[2Dii. How to access Data 22](#_Toc431329679)

[2E. Researcher CAN Bus (Vehicle Brake and Throttle Control) 22](#_Toc431329680)

[2Ei. Data Elements 22](#_Toc431329681)

[2Eii. How to access Data 23](#_Toc431329682)

[2Eiii. How to send data/commands 23](#_Toc431329683)

[2F. Logging 25](#_Toc431329684)

[2Fi. Data Elements 25](#_Toc431329685)

[2Fii. How to access logged Data 26](#_Toc431329686)

[2Fiii. How to send Data to be Logged 26](#_Toc431329687)

[2G. Research BSM 27](#_Toc431329688)

[2Gi. 2.2.1 Data Elements 27](#_Toc431329689)

[2Gii. 2.2.2 How to access Data 35](#_Toc431329690)

**List of Figures**

[Figure 1: Speed Controller Response 2](#_Toc431329691)

[Figure 2: Safety Mode State Diagram 4](#_Toc431329692)

[Figure 3: dSPACE MicroAutoBox II 5](#_Toc431329693)

[Figure 4: Secondary processor 6](#_Toc431329694)

[Figure 5: Configurable AOI 9](#_Toc431329695)

[Figure 6: TORC PinPoint 10](#_Toc431329696)

[Figure 7: PinPoint™ filter implementation 11](#_Toc431329697)

[Figure 8: Simulink Bus Editor Showing Elements of the GlobalStoreBus Definition 12](#_Toc431329698)

[Figure 9: Contents of carma\_platform.mdl Library with Data Producer block highlighted 13](#_Toc431329699)

[Figure 10: Bus Selector Extracting Data Elements out of the CarmaData Object 14](#_Toc431329700)

[Figure 11: Vehicle Coordinate Frame 16](#_Toc431329701)

[Figure 12: Contents of carma\_platform.mdl Library with Control Input block highlighted 24](#_Toc431329702)

[Figure 13: Contents of carma\_platform.mdl Library with UDP Logger / HMI block highlighted 26](#_Toc431329703)

[Figure 14: Internals of the Triggered Subsystem in the UDP Logger / HMI block 27](#_Toc431329704)

[Figure 15: The Contents of the dsrc\_handler.mdl library 36](#_Toc431329705)

[Figure 16: Setting up a BSMStoreSignal read 36](#_Toc431329706)

[Figure 17: Contents of the dsrc\_transmitter.mdl library. 38](#_Toc431329707)

**List of Tables**

[Table 1: Configurable Options for Object Fusion Program 8](#_Toc431329708)

[Table 2: Data Elements based on data from PinPoint™ Localization System 16](#_Toc431329709)

[Table 3: Data Elements based on data from Vehicle CAN Bus 1 19](#_Toc431329710)

[Table 4: Data Elements based on data from Vehicle CAN Bus 2 21](#_Toc431329711)

[Table 5: Data Elements based on data from Researcher CAN Bus 22](#_Toc431329712)

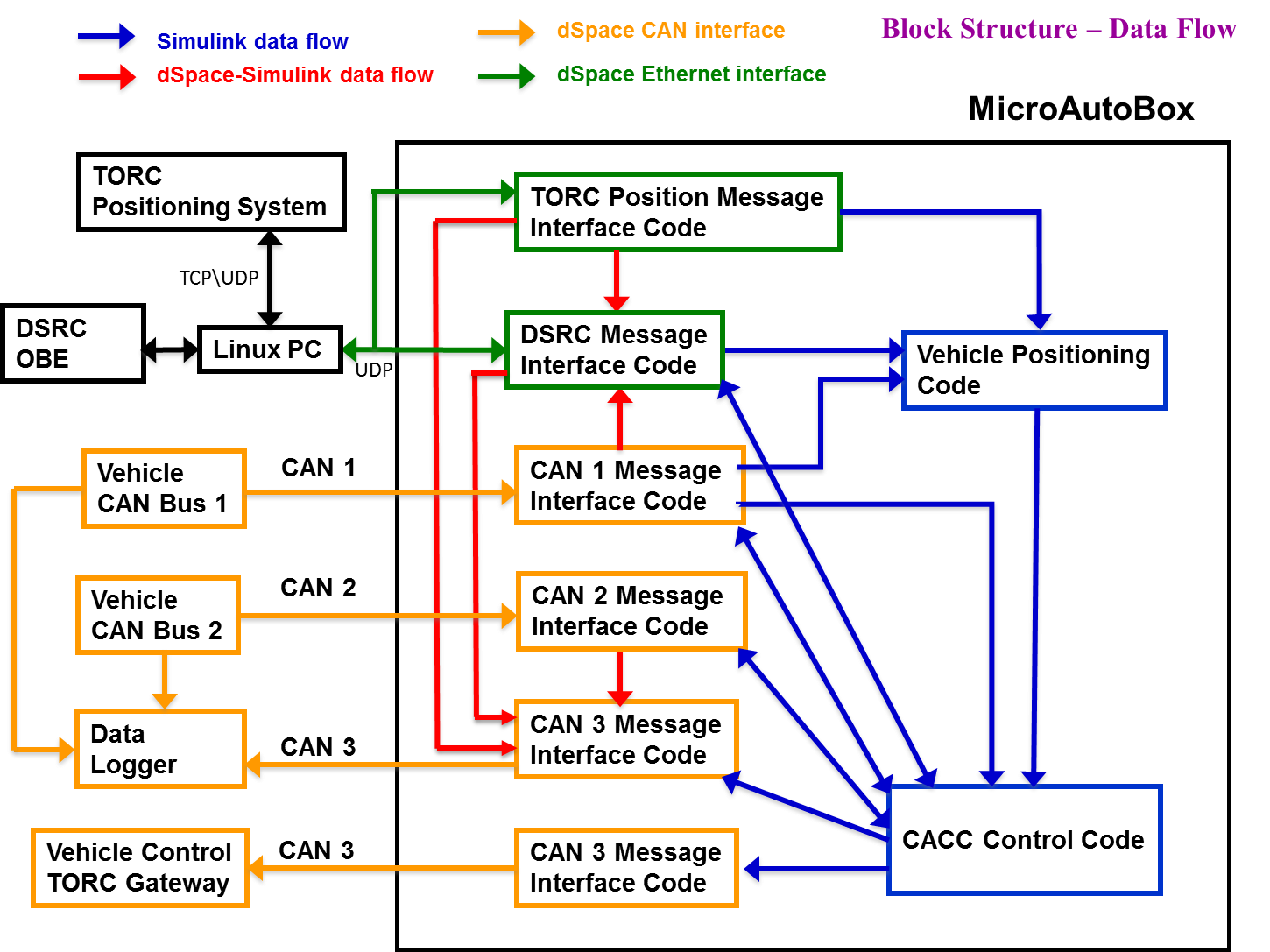
[Table 6: Longitudinal Controller Command Bus Definition 24](#_Toc431329713)

[Table 7: Data Elements of Research BSM 27](#_Toc431329714)

# Vehicle Research Platform

The platform technology supporting the CACC application consisted of a platoon of five (5) 2013 Cadillac SRX’s. Each vehicle is designed to be a complete platform for research in Cooperative Adaptive Cruise control (CACC) and Advanced Driver Assist Systems (ASAS). Each vehicle was outfitted with a set of custom Electronic Control Units (ECU) to enable longitudinal control, a specialized real-time computing platform, an axillary computing platform (Linux PC), and an integrated GPS localization solution.

Figure 1 is the system structure for the Connected Vehicle Research Platform.



**Figure 1 Overall System Structure for System Interfaces and Integration**

## TORC Controllers - Connections into the Vehicle

### TORC Longitudinal Controller

Longitudinal control of the vehicle is accomplished by the addition of a pair of TORC Automotive Interface Modules. These specially designed ECUs intercept CAN messages from the OEM Adaptive Cruise Control (ACC) system while simultaneously injecting new messages as commanded by the application\researcher. Researcher control is accomplished via a custom CAN protocol. Additionally the longitudinal control modules include a custom speed controller, allowing the researcher to select the desired speed and acceleration profile for the experiment.

The longitudinal control system integrates directly with OEM Adaptive Cruise Control (ACC) system. Rather than imitating all the outputs of the ACC system, the longitudinal control system injects new engine torque and braking deceleration commands in place of the commands sent by the ACC. Safety status and user input from the ACC system is preserved. This makes user control and system behavior nearly identical to the stock ACC system.

The TORC Automotive Interface Module is designed for automotive applications and consists of four CAN ports, analog I/O, digital I/O, and a power supply capable of operating on both 12V and 24V systems. Each ECU is responsible for reading in all CAN messages on one port while relaying all of the messages back out on a second CAN port with the modifications required for longitudinal control. The I/O capabilities of the ECUs are also responsible for driving the vehicle light bar.

### Control Details

The longitudinal controller implemented two separate methods of control: A percentage based wrench effort control and an integrated speed controller. The wrench effort control was meant to enable simple open loop control of the vehicle for a higher level control loop and present a near uniform acceleration along a single axis. As a side benefit, a single control axis simplified command parameter checking and command injection logic. To accomplish these design goals, a rough mapping was created to approximate vehicle brake and throttle commands along a 2.5 m/s2 to -2.5 m/s2 acceleration profile. This mapping was based on a simple model of vehicle response based on average vehicle weight, rolling resistance, and other measured parameters.

The speed control algorithm was built upon the wrench effort control to simplify the design. First, vehicle feedback, in the form of a calculated torque offset, was used to further normalize the open-loop control response at different speeds. A standard PID speed controller was used as the basic control algorithm with a user commanded maximum acceleration parameter as an output limiter. PID gains were all tunable. For this experiment they were tuned to try and match the user commanded speed as closely as possible with a minimal amount of oscillation. An example of the speed control profile is given in Figure 2.

Figure 2: Speed Controller Response

Due to the direct manipulation of the adaptive cruise control parameters, speed was controllable below the standard cruise control limit of 25 mph. Additionally, using the max acceleration parameter, the aggressiveness of acceleration to achieve a given set speed could be controlled.

### Safety Considerations

As the lowest level of vehicle control, safety was a primary design consideration of the Longitudinal Control System. Safety was enforced through the design of the control injection method, enforced safety checks, and physical hardware and system design. As previously stated, the longitudinal control system did not override the state of the OEM ACC controller; The ACC system itself controlled if longitudinal control was enabled or disabled. Therefore the primary method of interacting with the longitudinal control system was through the ACC system itself. Additionally, the longitudinal control could be disabled in software via an application command. Finally, all modules were physically disconnected from the vehicle bus if the emergency stop button was pressed.

The state diagram for the longitudinal control system is appended in Figure 2. This diagram outlines the different states to enable and disable Longitudinal Control. One important feature is the two stage entry system. Longitudinal control was only active when the controlling application actively issued commands and the driver had turned on the adaptive cruise control system. Any interruption in the status of the ACC system or the flow of application commands would cause longitudinal control to be disabled and the vehicle to return to manual control. Longitudinal control could also be disabled via any of the standard active cruise control commands or safety conditions, including:

* Tapping on the brake
* Turning the steering wheel more than one turn in any direction
* Canceling adaptive cruise control via steering wheel commands
* Any unsafe condition in the adaptive cruise control system itself

Additionally, longitudinal control can be disabled by the application by either not sending any messages or by sending a specially formatted message. Also, error checking and command counters were built in the protocol such that stale messages, messages with a bad checksum, and timeout errors all result in a return to manual mode.



Figure 3: Safety Mode State Diagram

## MicroAutoBox

The primary controller for the platform was the MicroAutobox II (MAB) from dSPACE. The MAB is a real-time prototyping system designed for rapidly deploying vehicle related applications. Programming the MAB was accomplished using dSPACE ControlDesk software along with MATLAB Simulink. The ControlDesk software communicated with the MAB through an Ethernet connection and allowed for simple visualization of the data being processed on the MAB for easy debugging. Once a program was loaded onto the MAB, it would run continuously (with or without a connection to the ControlDesk software) until the MAB was powered down or the program was explicitly stopped through the ControlDesk.

We used the dSPACE Control Desk software to load the code onto the MAB as well as control it during operation. The Control Desk software integrates tightly with the compiled C code and allows us to perform live introspection into the values and outputs that the program uses during runtime. This proved to be useful during testing because we could monitor the program in real time and watch for values that indicate aberrant behavior or sensor failure.

All code running on the MAB was written in Simulink (or in C using a Simulink-C code interface) and then compiled for the PowerPC architecture on the MAB using a toolkit provided by dSPACE. Interfaces to the hardware on the MAB (Ethernet and CAN) were supplied as Simulink Libraries containing the blocks needed to interact with that hardware.



Figure 3: dSPACE MicroAutoBox II

## Simulink

Simulink is a graphical programming tool developed by MathWorks and in many ways acts as a companion to their MATLAB computation software. Simulink’s visual nature enables rapid prototyping of algorithms or logic while making it easy to reason about relationships between functions and data flow. Simulink acted as the framework for much of the CACC software running on the MicroAutobox with both the algorithm as well as the vehicle interface code being developed in it. In cases where using Simulink alone would have made certain software components difficult to create, we were able to implement the logic in standard C code and integrate that code directly with the rest of the Simulink code. We used the Simulink Coder software to compile the Simulink model down to C code before using a cross-compiler to create the binary we could run on the MicroAutobox.

## Linux PC

### Secondary PC

In addition to the real-time MAB processor, a secondary processing unit was used to offload some of the computing responsibilities. The computing hardware in the secondary processing unit is an automotive style fan-less PC running a Linux-based operating system. For the current CACC experiment the secondary processor was responsible for controlling the Human Machine Interface (HMI), handling data interchange to the MAB ECU, object fusion processing, and experimental data logging. Further details on each of these individual functions are described below.



Figure 4: Secondary processor

### V2IServer *- Forwarding PinPoint and DSRC Data to MAB*

We created a stand-alone Java application, called v2iServer, that provides data transmission services for various devices. This includes moving data from the PinPoint™ localization device to the MAB, moving data bi-directionally between the DSRC radio and the MAB, and also providing the web-based human-machine interface (HMI) display and interactions on the external tablet device. The reason this application was necessary as a “middle man” between the PinPoint™ and DSRC devices and the MAB is that the MAB is limited in Ethernet interfaces and data contents the software is willing to accept.

In the case of PinPoint™ connection, the MAB is only capable of using UDP, and the PinPoint™ requires communication with both UDP and TCP. Therefore, the v2iServer sets up and manages the UDP/TCP connections with the PinPoint™, collects its data, then repackages it into UDP packets for forwarding on to the MAB.

For DSRC communications, the MAB is expecting incoming packets to be strictly limited to Wave Short Message Protocol (WSMP) messages wrapping BSMs of interest. The v2iServer monitors the data stream coming in from the DSRC radio, identifies the packets that contain BSMs, extracts the WSMP message, and throw away the transmission wrapper. It then forwards this WSMP message on to the MAB to minimize the unwrapping the MAB has to do. For outgoing BSMs, the v2iServer takes pre-formed WSMP messages created by the MAB on the same MAB UDP port, and forwards them verbatim to the DSRC radio on its own UDP port.

*Human Machine Interface (HMI)*

The v2iServer application is built with an embedded Tomcat web server such that it can serve up web interaction from a completely self-contained jar file. In addition to providing PinPoint™ and DSRC data to/from the MAB, this application also communicates with the MAB on another UDP port regarding human driver interactions with the vehicle. There are three HMI functions provided on a single web page. The first is a toggle switch by which the driver can specify the desired state of automation. It defaults to “Off”, but when the driver turns it “On” (by tapping the switch on the tablet) the v2iServer application sends a signal to the MAB indicating the driver’s desire to enter CACC automated control. The driver can use this switch to turn off automated control at any time. Unfortunately, due to the compressed schedule, this feature was not used, as the MAB logic was hard-coded to always enter automated control as soon as the driver turned on the factory ACC switch. The second HMI function is a display of the automation state, to give the driver a feel for what the control software is doing. The five states that can be indicated are: Manual, Factory ACC, PATH CC, PATH ACC, PATH CACC. This state is indicated in a data signal periodically received from the MAB. Due to compressed schedule, the values in this state data element did not fully reflect the actual software state in most cases. They did not distinguish among the various automated states. The final function is a visual and audible brake warning. This function is also indicated by a data element that may be received from the MAB, asynchronously. When received, the HMI will produce a warning to alert the driver that the automation software is incapable of providing adequate braking to keep the vehicle operation safe. Because of the schedule-induced shortcuts required in the CACC software, the HMI was only tangentially used during the tests.

### Object Fusion

The CACC algorithm required information about the vehicle directly in front of it to aid in its calculations. However, the object CAN bus from the vehicle (containing radar and camera information for objects around the vehicle) did not contain concise messages providing consistent tracking information for objects. It was therefore necessary to implement an object fusion algorithm that ran on the Secondary PC. This program consumed the radar data from the object CAN bus and injected its own CAN message containing filtered tracking information for objects in front of the vehicle. This CAN message was then forwarded on to the MAB where it was then passed to the CACC algorithm for processing.

The software reads in objects from the Forward Long Range Radar and the Left/Right Short Range Radar on CAN Bus 2. Initial analysis of the objects from the Short Range Radar showed that the limited range would not be suitable for objects over 14 meters away; and the objects from the Forward Vision system had ranges that where much noisier than those from the radar systems. Because of these two reasons, only objects from the Long Range Radar system for this application were used.

Objects are continuously read in from the CAN bus and saved in a list of objects. The objects are filtered using a set of configurable options including the Area of Interest (AOI) geometry, the minimum width of the object, and the number of objects to send. After filtering the list of objects, they are sorted by range in ascending order and published. This process is run at a configurable rate. The software will publish these objects as a series of Fusion Object Messages and the message ID can be configured. The Fusion Object Message definition is identical to the messages 0x511 – 0x519 defined in the Bus 3 symbol definitions file, however, we only pack the following fields:

* lat\_pos
* long\_pos
* rel\_lat\_vel
* rel\_lon\_vel
* message\_index
* rolling\_count

Additionally, a Fusion Header message, originally defined as message 0x510 in the Bus 3 symbol definition file, is published (also with a configurable ID) to indicate the number of valid objects published at each iteration.

The configurable options are saved in a file called “options.cfg” that should be saved in the same directory as the executable (/usr/local/torc). The configurable parameters are listed in the Table 1 below.

Table 1: Configurable Options for Object Fusion Program

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Units |
| angle\_of\_interest | Defines a forward looking cone area. Objects outside this range will not be considered. | [degrees] |
| fusion\_header\_message\_id | The CAN message id to give to the outgoing object header message. | [decimal] |
| fusion\_message\_id\_start | The starting CAN message id to give to the outgoing object fusion messages. | [decimal] |
| object\_timeout\_ms | The maximum time for an incoming object to live. After this time has elapsed, the objects are not considered. Prevents reporting stale data. | [ms] |
| publish\_delay\_ms | The delay between publishing the list of fusion object messages. A 100 ms delay will cause the program to output the list at about 10 Hz. | [ms] |
| num\_objects\_to\_send | The number of objects to send. The first will always be the best candidate, meaning it is in the AOI and is closest to the vehicle. If this value is greater than the number of valid objects, then the list will be padded with objects with an invalid range (0.0) and the Fusion Object Header message will have a “num\_val\_targets” value less than this parameter. |  |
| min\_object\_width\_m | The minimum object width to consider. Objects with less than this width will be thrown out. If the value is set to 0.0, the filter will not be active. | [m] |
| debug | If true, additional debug statements will be printed to the console. | [Boolean] |



Figure 5: Configurable AOI

The configurable AOI is shown in blue and the vehicle is shown in green. The rectangular area in front of the vehicle prevents close objects outside the radar cone but still in front of the vehicle being missed by the sensor.

To install the software, follow the steps below:

1. Copy the executable “ObjectFusion” and the configuration file, “options.cfg” to “/usr/local/torc”. Ensure ObjectFusion has executable permissions.
2. Configure the can0 interface in the “/etc/network/interfaces” file to have a bit rate of 500000
3. Modify the “/etc/rc.local” file to run the executable on startup
4. \* On some vehicles, it was discovered that the can0 interface needed to be started manually, so a call to “ifconfig can0 up” was added to the “/etc/rc.local” file along with a delay to start the ObjectFusion executable.

## DSRC

The DSRC hardware utilized in the CVRP Platform is an Arada Locomate Aftermarket Safety Devices. These units enabled the transmission and reception modified SAE Basic Safety Messages (BSMs) between all the vehicles. Using software developed by Arada to forward received BSMs to the Linux PC and to sign and broadcast BSM’s received from the Linux PC enabled communications between the CACC software running on all 5 vehicles. The BSMs themselves were modified from the way they are defined in the J2735 standard to include additional data specifically for the CACC application. This includes vehicle metrics such as radar sensor data, local-frame data from the PinPoint localization solution, and ACC system status data as well as algorithm specific parameters such as desired headway, desired speed, and desired maneuver. Complete detail on all the data added to the BSM can be found in the CVRP User’s Guide with definitions for formats and units of every value in the BSM. All of this data is transmitted and received by the CVRP Simulink library and presented to the CACC algorithm completely parsed and translated.

## PinPoint™ Localization Solution

The PinPoint™ localization system is a continuous positioning system for ground vehicles. PinPoint™ provides multi-sensor fusion of dual-GPS receivers, inertial sensors, and wheel speed sensors to provide real time position, orientation, velocity, and time information. All outputs are continuously updated regardless of GPS fixes, allowing operation during GPS degradation or complete signal loss.



Figure 6: TORC PinPoint



Figure 7: PinPoint™ filter implementation

PinPoint™ also served as the Network Time Protocol (NTP) time server on the vehicle allowing video logs to be time stamped and synced with GPS time.

Data synchronization between Ethernet and CAN data was also achieved through PinPoint™. The secondary CAN port on PinPoint™ reports GPS time and was recorded via the CAN Data Logger. This allowed correlation of the timestamps internal to the CAN Data Logger with GPS time.

# Description of APIs - How to use the Platform

## Accessing Data in the CarmaData Object

To make the vehicle and sensor data available to the application layer on the MAB a GlobalStoreBus was defined that contained all of the data elements needed by the application. Each element was combined with a timestamp (in floating-point seconds since the MAB program started) that indicated when the data were last updated so that an application using the data would know if it was stale. Busses were defined to combine a timestamp and value for each possible data type, such as TimestampedBool, TimestampedSingle, TimestampedDouble, etc., and each element of the GlobalStoreBus was one of these types. A globally accessible Simulink.Signal object was instantiated as a GlobalStoreBus named CarmaData and this object was used to write to and read from the shared memory object.



Figure 8: Simulink Bus Editor Showing Elements of the GlobalStoreBus Definition

Updating and accessing the data in the CarmaData object was accomplished by using the carma\_platform.mdl library. The Carma\_Platform block could be placed anywhere in the model, and had all of the logic for reading the hardware interfaces and populating the CarmaData object. The Data Producer block was used to pull the data out of the memory store at a configurable rate as one bus object. This object could then be passed into a Bus Selector block to remove certain elements from the bus that were needed. Figure 10 shows an example of how the data elements in the CarmaData object were extracted and sent to other parts of the model using Goto blocks. While this exampled extracted all elements from the object, any subset of the elements could also be exctracted using the Bus Selector block. Note that the CarmaData input 1 in Figure 10 would be connected to the output of the Data Producer block in the carma\_platform.mdl library.

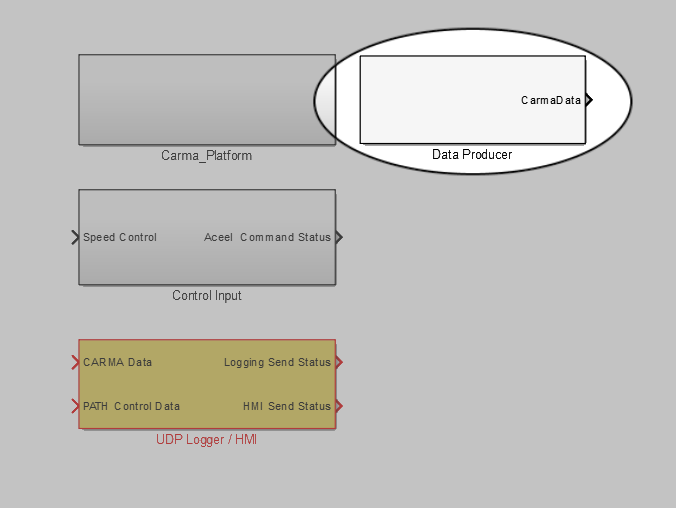


Figure 9: Contents of carma\_platform.mdl Library with Data Producer block highlighted

The GlobalStoreBus definition was saved in a .mat file (memory\_struct.mat for the CACC) such that it didn’t have to be re-defined when the project was opened on different computers. The .mat file was configured to be loaded automatically when the Simulink project was opened using project shortcuts.

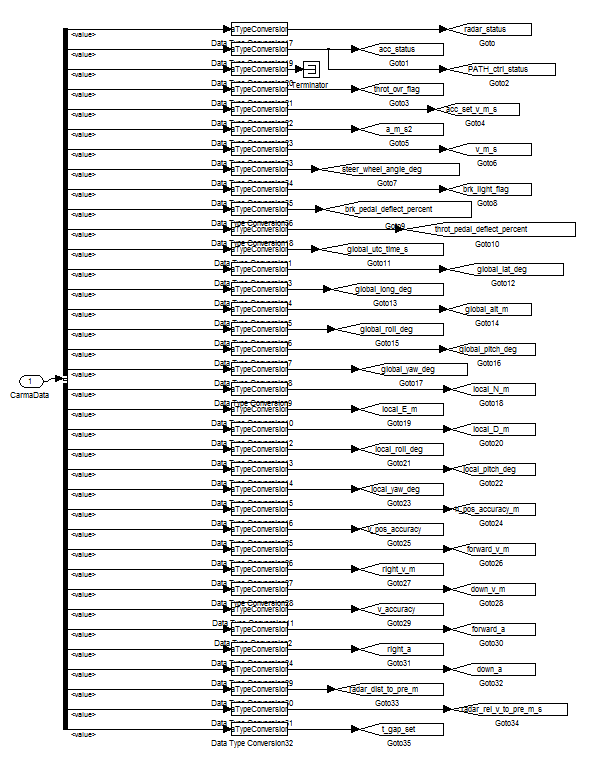


Figure 10: Bus Selector Extracting Data Elements out of the CarmaData Object

The following subsections describe the individual information sources that contribute to the CarmaData GlobalStoreBus and give detailed information on the data elements that come from each source.

## PinPoint™ Localization System

PinPoint™ provided data in three different coordinate frames. A global frame was used for representing where in the world the vehicle was located, a local frame was used for representing where the vehicle was relative to nearby objects, and a vehicle frame was used for representing vehicle velocities and accelerations.

The interaction between the local and global frames is analogous to rolling a plane on top of a sphere, where the contact point is the current vehicle position. Both the local position (the location of the contact point on the plane) and the global location (the location of the contact point on the sphere) change as the vehicle moves across the surface of the earth. The origin of the local plane is not fixed to any global coordinate, and any transforms between the two coordinate systems are done with respect to the current vehicle location. The vehicle frame provided a coordinate system that is referenced to the vehicle origin, unlike the local or global frames. The vehicle coordinate frame was used for reporting velocities and accelerations and is typically aligned with the vehicle’s primary direction of travel. These three coordinate frames are explained in more detail below.

#### Global Frame

PinPoint™ uses world geodetic (WGS-84) latitude, longitude, and height (LLH) above the ellipsoid to represent global position. Accuracy estimates of the global position are provided in meters.

#### Local Frame

PinPoint™ uses a floating, north-aligned local frame in north, east, down (NED) coordinates to represent local position. The local position provides a stable frame for local navigation that can be used to reference the positions of other vehicles. The local position is updated as the vehicle moves but never corrected; allowing calculations performed in local frame to be immune to GPS “pops”, where the solution quickly changes as satellites come into and go out of view.

#### Vehicle Frame

The vehicle specific parameters and the velocity / acceleration state outputs are represented in vehicle body frame. This frame is represented using forward, right, down (FRD) convention, with the associated rotations given as roll, pitch, yaw, respectively. The origin of the frame is located over the rear axle of the research platform.

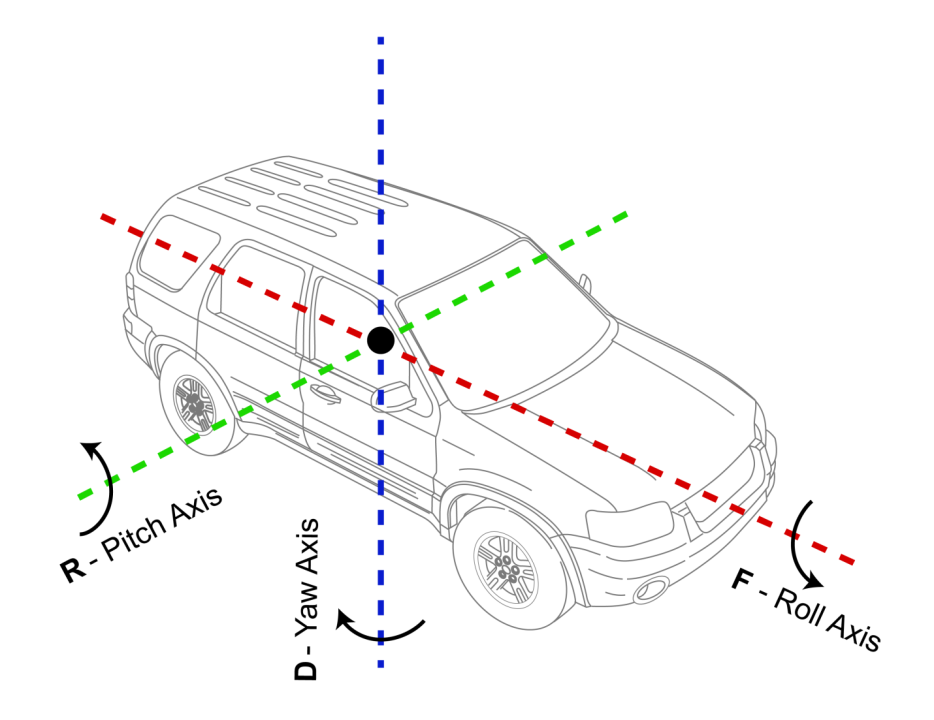


Figure 11: Vehicle Coordinate Frame

The orientation, given in Euler angles, represents the rotation from the local tangent plane to the vehicle body frame. The rotation order is roll, pitch, yaw if performed in the local NED frame, or yaw, pitch, roll if performed in the vehicle FRD frame.

### Data Elements

Table 2 shows the data elements from the PinPoint™ Localization System.

Table 2: Data Elements based on data from PinPoint™ Localization System

| Name in Data Store | Description | Format | Units | Range |
| --- | --- | --- | --- | --- |
| global\_utc\_time\_s | The current UTC time in Single precision floating point seconds (microsecond resolution) which was sent from the PinPoint™ over the researcher CAN bus to the MAB. Comparing the MAB timestamp to the UTC time in this data element allows for conversion between MAB time and UTC time. | Double precision float | s | [0, inf] |
| global\_lat\_deg | Latitude of the vehicle’s origin. | Double precision float | deg | [-inf, inf] |
| global\_long\_deg | Longitude of the vehicle’s origin. | Double precision float | deg | [-inf, inf] |
| global\_alt\_m | Altitude of the vehicle’s origin. | Double precision float | m | [-inf, inf] |
| global\_roll\_deg | Rotation about the Forward axis of the vehicle. Note this will be identical to the local\_roll\_deg. | Single precision float | deg | [-inf, inf] |
| global\_pitch\_deg | Rotation about the Right axis of the vehicle. Note this will be identical to the local\_pitch\_deg. | Single precision float | deg | [-inf, inf] |
| global\_yaw\_deg | Rotation about the Down axis of the vehicle. Note this will be identical to the local\_yaw\_deg. | Single precision float | deg | [-inf, inf] |
| local\_north\_m | Location of the vehicle’s origin in the North direction relative to the arbitrary local frame origin. | Double precision float | m | [-inf, inf] |
| local\_east\_m | Location of the vehicle’s origin in the East direction relative to the arbitrary local frame origin. | Double precision float | m | [-inf, inf] |
| local\_down\_m | Location of the vehicle’s origin in the Down direction relative to the arbitrary local frame origin. | Double precision float | m | [-inf, inf] |
| local\_roll\_deg | Rotation about the Forward axis of the vehicle. Note this will be identical to the global\_roll\_deg. | Single precision float | deg | [-inf, inf] |
| local\_pitch\_deg | Rotation about the Right axis of the vehicle. Note this will be identical to the global\_pitch\_deg. | Single precision float | deg | [-inf, inf] |
| local\_yaw\_deg | Rotation about the Down axis of the vehicle. Note this will be identical to the global\_yaw\_deg. | Single precision float | deg | [-inf, inf] |
| horizontal\_pos\_accuracy\_m | Horizontal position accuracy of the navigation solution. This is computed as the magnitude of the norm of the North and East accuracy vectors. The true position should lie within a circle centered on the current position value whose radius is equal to this horizontal accuracy. | Single precision float | m | [0, inf] |
| vertical\_pos\_accuracy\_m | Vertical position accuracy of the navigation solution. Computed as the magnitude of the Down accuracy vector. | Single precision float | m | [0, inf] |
| velocity\_fwd\_m\_s | Velocity of the vehicle in the Forward direction (vehicle’s frame). | Single precision float | m/s | [-inf, inf] |
| velocity\_right\_m\_s | Velocity of the vehicle in the Right direction (vehicle’s frame). | Single precision float | m/s | [-inf, inf] |
| velocity\_down\_m\_s | Velocity of the vehicle in the Down direction (vehicle’s frame). | Single precision float | m/s | [-inf, inf] |
| velocity\_accuracy\_m\_s | Accuracy of the velocity solution. This is computed as the magnitude of the norm of the Forward, Right, and Down accuracy vectors. | Single precision float | m/s | [0, inf] |
| accel\_fwd\_m\_s2 | Acceleration of the vehicle in the Forward direction (vehicle’s frame). | Single precision float | m/s2 | [-inf, inf] |
| accel\_right\_m\_s2 | Acceleration of the vehicle in the Right direction (vehicle’s frame). | Single precision float | m/s2 | [-inf, inf] |
| accel\_down\_m\_s2 | Acceleration of the vehicle in the Down direction (vehicle’s frame). | Single precision float | m/s2 | [-inf, inf] |

### How to access Data

The localization data could be accessed using the Data Producer block in the carma\_platform.mdl library as described in Section ***2A: Accessing Data in the CarmaData Object***.

## Vehicle CAN Bus 1

The Vehicle CAN Bus 1 contained general vehicle information including speed, acceleration, and pedal deflections. It also included information about the vehicle’s base Adaptive Cruise Control (ACC) system which could be used by the application to forward the base vehicle controls to the TORC Longitudinal Controllers through the Researcher Bus (CAN 3).

The Vehicle CAN Bus 1 was connected to both the MAB and the CAN Data Logger so that all raw CAN traffic was logged.

### Data Elements

Table 3 demonstrates the data elements from Vehicle CAN Bus 1.

Table 3: Data Elements based on data from Vehicle CAN Bus 1

| Name in Data Store | Description | Format | Units | Range |
| --- | --- | --- | --- | --- |
| acc\_status | Flag from the base vehicle indicating whether the ACC system is enabled (true) or not (false). | Boolean | True: ACC enabled,  False: ACC disabled | true, false |
| acc\_set\_speed\_m\_s | Current set speed of the base vehicle ACC system. | Single precision float | m/s | [0, 71.11] |
| accel\_m\_s2 | Vehicle acceleration measured by the base vehicle. | Single precision float | m/s2 | [-20.48, 20.47] |
| speed\_m\_s | Vehicle speed measured by the base vehicle. | Single precision float | m/s | [0, 142.22] |
| steering\_wheel\_angle\_deg | Angle of the steering wheel. | Single precision float | Deg | [-2047,2048] |
| brk\_light\_flag | Boolean value indicating whether the brake lights are on (true) or not (false). This can be used to tell if braking is being applied. | Boolean | True: Brake lights on,  False: Brake lights off | true, false |
| brk\_pedal\_deflect\_percent | Percent deflection of the brake pedal. Note that it is possible for this value to not reach 100 at full deflection. | Single precision float | % | [0, 100] |
| throttle\_pedal\_deflect\_percent | Percent deflection of the throttle pedal. Note that it is possible for this value to not reach 100 at full deflection. | Single precision float | % | [0, 100] |

### How to access Data

The Vehicle CAN bus 1 data could be accessed using the Data Producer block in the carma\_platform.mdl library as described in Section ***2A: Accessing Data in the CarmaData Object***.

## Vehicle CAN Bus 2 (Forward Object Bus)

The Vehicle CAN bus 2 (Forward Object Bus) contained information from the base vehicle’s front radar and camera object detection system. This CAN bus was also connected to the Secondary Linux PC where the Object Fusion program described in section ***1Diii: Object Fusion*** processed the long range radar data to track objects in front of the vehicle. These tracked objects were packaged into a series of CAN frames and also transmitted on the Forward Object Bus to the MAB.

### Data Elements

Table 4 shows the data elements from Vehicle CAN Bus 2.

Table 4: Data Elements based on data from Vehicle CAN Bus 2

| Name in Data Store | Description | Format | Units | Range |
| --- | --- | --- | --- | --- |
| radar\_status | Flag indicating whether the front radar is operating correctly. | Boolean | True: Radar OK,  False: Radar issue | true, false |
| preceding\_veh\_flag | Flag indicating whether or not there is a vehicle within the range of the radar in front of the current vehicle | Boolean | True: There is a preceding vehicle  False: There is no preceding vehicle in range | true, false |
| dist\_to\_preceding\_veh\_m | Distance from the front bumper of the current vehicle to the nearest vehicle in front of it. Undefined if there is no preceding vehicle. | Single precision float | M | [0, 255.875] |
| rel\_speed\_to\_preceding\_veh\_m\_s | Relative speed of the current vehicle and the nearest vehicle in front of it. Undefined if there is no preceding vehicle. | Single precision float | m/s | [-127,127.875] |
| object\_list\* | List of distances and relative speeds of four nearest objects to the vehicle that are being tracked. | ObjectList | - | - |

\* The object\_list was a bus object containing a timestamp and array of buses. Each bus in the array contained the distance to a tracked object in meters, and relative speed of a tracked object in m/s. There were four objects tracked in this array, and the first one (nearest) was used to populate the dist\_to\_preceding\_veh\_m and rel\_speed\_to\_preceding\_veh\_m\_s data elements in the CarmaData object. The data format, units and ranges for the objects in this array were identical to those of the dist\_to\_preceding\_veh\_m and rel\_speed\_to\_preceding\_veh\_m\_s data elements.

### How to access Data

The Forward Object Bus (CAN 2) data could be accessed using the Data Producer block in the carma\_platform.mdl library as described in Section ***2A: Accessing Data in the CarmaData Object***.

## Researcher CAN Bus (Vehicle Brake and Throttle Control)

The Researcher CAN bus connected the MAB with the Longitudinal Controllers described in section ***1A: TORC Controllers - Connections into the Vehicle***. This bus provided feedback about the state of the longitudinal control as well as providing an interface for the application to send commands to the controllers.

### Data Elements

Table 5 shows data elements from Researcher CAN Bus.

Table 5: Data Elements based on data from Researcher CAN Bus

| Name in Data Store | Description | Format | Units | Range |
| --- | --- | --- | --- | --- |
| path\_ctrl\_status\* | Flag indicating whether the TORC Longitudinal controller is active | Boolean | True: Longitudinal control active,  False: Longitudinal control inactive | true, false |
| throttle\_ovr\_flag | Flag indicating whether the ACC throttle command is being overridden by the throttle pedal | Boolean | True: Throttle pedal overriding ACC  False: Throttle pedal not overriding ACC | true, false |

\* The path\_ctrl\_status flag is indicating whether the TORC Longitudinal Controller is enabled and is not specific to PATH’s CACC application. The data element’s name is unfortunately misleading.

### How to access Data

The Researcher CAN bus data could be accessed using the Data Producer block in the carma\_platform.mdl library as described in Section ***2A: Accessing Data in the CarmaData Object*** above.

### How to send data/commands

Sending commands to the Longitudinal Controllers was accomplished using the Control Input block in the carma\_platform.mdl library (see Figure 12). The input to the Control Input block (Speed Control) was a bus object called the AccelCommandBus. The data elements of that bus object are defined in Table 6.



Figure 12: Contents of carma\_platform.mdl Library with Control Input block highlighted

The output of the Control Input block (Accel Command Status) was a zero or one that indicated whether the CAN message was successfully sent or not.

Table 6: Longitudinal Controller Command Bus Definition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Data Format | Unit | Range |
| max\_accel\_m\_s2 | The maximum acceleration allowed to achieve the desired speed | double-precision float | m/s2 | [0, 142.2] |
| speed\_command\_m\_s | The desired speed for the vehicle. This will be used when the command\_mode is 2 | double-precision float | m/s | [0, 142.2] |
| wrench\_effort\_command | Acceleration wrench effort command. This will be used when the command\_mode is 1 | double-precision float | % | [0, 100] |
| override\_enabled | Flag to enable robotic override of the ACC system | boolean | True: Enable robotic override,  False: Disable robotic override | true, false |
| command\_mode | Enum used to dictate how control should be applied. | Enum | 0: Disable ACC system (including base vehicle ACC)  1: robotic “wrench effort” control  2: robotic speed control | [0, 1, 2] |

## Logging

In order to log all the parameters that would be useful for post-processing, the MAB was configured to periodically send UDP packets to the Secondary PC with information that should be logged. The Secondary PC ran a simple Python script to listen for the logging messages, parse out the data, and write it to a CSV file that could be easily read by MATLAB.

The logging script was run by issuing the following command in its directory on the Secondary PC:

python carma\_udp\_logger.py

The log file was saved to a directory named “logs” (in the same directory as the python script) with a filename of “carma\_data\_log\_[DAY]\_[MONTH]\_[YEAR]\_[HOUR]\_[MIN]\_[SEC].csv” where the bracketed values are replaced with the corresponding value from the system time at log creation.

### Data Elements

The UDP packet contained all of the data elements listed in the above sections (2B: PinPoint™ Localization System, 2C: Vehicle CAN Bus 1, 2D: Vehicle CAN Bus 2 (Forward Object Bus), 2E: Researcher CAN Bus (Vehicle Brake and Throttle Control)) as well as a list of elements provided by the CACC application that were desired to be logged. The full object list from the Forward Object CAN Bus was not logged.

### How to access logged Data

The logged data was retrieved through the Secondary PC in the form of CSV files. The CSV files had column headings as entries on the first line so that they could be easily imported into MATLAB with useful variable names.

### How to send Data to be Logged

Logging data was accomplished by passing the CarmaData object into the UDP Logger / HMI block in the carma\_platform.mdl library (see Figure 13). There was also an input for the PATH Control Data which was just a vector of double-typed values that PATH wanted to log.



Figure 13: Contents of carma\_platform.mdl Library with UDP Logger / HMI block highlighted

The UDP packets were sent at a fixed time interval using an RTI Time-Triggered Task block to trigger the UDP Transmit block. The UDP Logger / HMI block also served as the interface to the HMI since the data being sent to the HMI was already part of the data that PATH wanted to log.

The UDP Transmit is very dependent on the type and number of data elements that need to be sent, so for a future application there will need to be significant modifications within the UDP Logger / HMI block. Specifically the DSEncode32 block inside the Triggered Subsystem needs to be adjusted so that the data type string exactly matches the types in the vector that is passed into it. The constant passed into the ETHERNET\_UDP\_TX\_BL1 defining the packet size also needs to be updated to be consistent with the data output by the DSEncode32 block. Finally, the maximum packet size in the menu for the ETHERNET\_UDP\_TX\_BL1 needs to be updated to match the packet size as well.



Figure 14: Internals of the Triggered Subsystem in the UDP Logger / HMI block

To retrieve the data using the same python script on the Secondary PC, it must also be updated to correctly parse the newly formatted UDP packet.

## Research BSM

### 2.2.1 Data Elements

Table 7 describes the format, units, and range of the values in the BSMStoreBus Simulink.Signal. This does not necessarily reflect their format, units, and range as they are transmitted over the air as they are processed upon being received. For a more complete description of the over-the-air format see file CACC\_R0.1.asn for a full technical definition.

Table 7: Data Elements of Research BSM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Format | Units | Range |
| caccFlagsBitMask | State of the ACC/CACC control system | Bitmask, for fields see CACC\_R0.1.asn CACCFlags definition | N/a | [0, 127] |
| setSpeed\_m\_s | Set speed for the ACC system | Single precision float | m/s | [-inf, +inf] |
| throtPos\_percent | Percentage throttle application | Single precision float | Percent | [0, 100] |
| lclPN\_mm | Position in the north direction of the local frame (initialized on vehicle startup) | Single preceision float | mm | [-inf, +inf] |
| lclPE\_mm | Position in the east direction of the local frame (initialized on vehicle startup) | Single precision float | mm | [-inf, +inf] |
| lclPD\_mm | Position in the down direction of the local frame (initialized on vehicle startup) | Single precision float | mm | [-inf, +inf] |
| roll\_deg | Vehicle roll in the global frame | Single precision float | deg | [-180, 180] |
| pitch\_deg | Vehicle pitch in the global frame | Single precision float | deg | [-180, 180] |
| yaw\_deg | Vehicle yaw in the global frame | Single precision float | deg | [0, 360] |
| hPosAcry\_m | Standard deviation of the horizontal positioning solution | Single precision float | m | [0, +inf] |
| vPosAcry\_m | Standard deviation of the vertical positioning solution | Single precision float | m | [0, +inf] |
| fwrdVel\_m\_s | Forward component of the vehicle’s velocity | Single precision float | m/s | [-inf, +inf] |
| rightVel\_m\_s | Right component of the vehicle’s velocity | Single precision float | m/s | [-inf, +inf] |
| downVel\_m\_s | Down component of the vehicle’s velocity | Single precision float | m/s | [-inf, +inf] |
| velAcc\_m\_s | Standard deviation of the velocity measurement | Single precision float | m/s | [0, +inf] |
| frwdAcc\_mm\_s\_s | Vehicle’s forward acceleration | 16-bit signed integer | mm/s2 | [-32768, 32767] |
| rightAcc\_mm\_s\_s | Vehicle’s right acceleration | 16-bit signed integer | mm/s2 | [-32768, 32767] |
| dwnAcc\_mm\_s\_s | Vehicle’s down acceleration | 16-bit signed integer | mm/s2 | [-32768, 32767] |
| grpID | ID number of CACC group | 8-bit unsigned integer | N/a | [0, 7] |
| grpSize | Size of CACC group | 8-bit unsigned integer | N/a | [0, 15] |
| grpMode | Mode of CACC group | 8-bit unsigned integer | N/a | [0, 7] |
| grpManDes | Desired maneuver of CACC group | 8-bit unsigned integer | N/a | [0, 127] |
| grpManID | Current maneuver of CACC group | 8-bit unsigned integer | N/a | [0, 127] |
| vehID | Unique ID of vehicle, Corresponds to last 3 digits of license plate by default | 8-bit unsigned integer | N/a | [0, 255] |
| frntCutIn | If there is a non-CACC vehicle cut into the platoon in front of the vehicle | 8-bit unsigned integer | N/a | [0, 7] |
| vehGrpPos | Vehicle’s position in its group | 8-bit unsigned integer | N/a | [0, 15] |
| vehFltMode | Vehicle’s fault mode. | 8-bit unsigned integer | N/a | [0, 15] |
| vehManDes | Vehicle’s desired maneuver | 8-bit unsigned integer | N/a | [0, 127] |
| vehManID | Vehicle’s current maneuver | 8-bit unsigned integer | N/a | [0, 127] |
| distToPVeh\_m | Distance to preceeding vehicle | 8-bit unsigned integer | m | [0, 127] |
| relSpdPVeh\_m\_s | Relative speed to preceeding vehicle | 8-bit unsigned integer | m/s | [-40, 40] |
| distToLVeh\_m | Distance to lead vehicle | 8-bit unsigned integer | m | [0, 255] |
| relSpdLVeh\_m\_s | Relative speed to lead vehicle | 8-bit unsigned integer | m/s | [-40, 40] |
| desTGapPVeh\_s | Desired time-gap to preceeding vehicle | Single precision float | s | [0, 3] |
| desTGapLVeh\_s | Desired time-gap to lead vehicle | Single precision float | s | [0, 3] |
| estDisPVeh\_m | Estimated distance gap to preceeding vehicle | 8-bit unsigned integer | m | [0, 150] |
| estDisLVeh\_m | Estiamted distance gap to lead vehicle | 8-bit unsigned integer | m | [0, 150] |
| desSpeed\_m\_s | Desired speed of vehicle | 8-bit unsigned integer | m/s | [0, 35] |
| desTrq\_N\_m | Desired torque of vehicle | 16-bit unsigned integer | N/m | [0, 2500] |
| msgID | DSRC ID of received message | 8-bit unsigned integer | N/a | [2,2] |
| msgCnt | Sequence number of received message | 8-bit unsigned integer | N/a | [0, 255] |
| id | Temporary ID of vehicle | 32-bit unsigned integer | N/a | [0, 4294967295] |
| secMark\_ms | Milleseconds elapsed in current minute | 16-bit unsigned integer | ms | [0, 65535] |
| lat\_deg | Vehicle’s latitude | Single precision float | deg | [-180, 180] |
| lon\_deg | Vehicle’s longitude | Single precision float | deg | [0, 360] |
| elev\_m | Vehicle’s altitude relative to the reference ellipsoid | Single precision float | m | [-409.5, 6143.9] |
| semi\_major\_accuracy\_m | Standard deviation of the positioning solution along the semi-major axis | Single precision float | m | [0, inf] |
| semi\_minor\_accuracy\_m | Standard deviation of the positioning solution along the semi-minor axis | Single precision float | m | [0, inf] |
| semi\_major\_accuracy\_orientation\_deg | Orientation of the semi-major axis of the accuracy value in the global frame. | Single precision float | deg | [0, 360] |
| speed\_m\_s | Vehicle speed | Single precision float | m/s | [0, 163.8] |
| transmission\_enum | Vehicle transmission state | 8-bit unsigned integer | 0 – Neutral  1 – Park  2 – Forward  3 – Reverse  4 – 6 Reserved  7 - Unavailable | [0, 7] |
| heading\_deg | Vehicle heading | Single precision float | deg | [0, 360] |
| angle\_deg | Vehicle steering wheel angle | Single precision float | deg | [-127, 127] |
| vertical\_acceleration\_g | Vehicle vertical acceleration | Single precision float | G | [-3.4, 1.54] |
| lat\_acceleration\_m\_s\_s | Vehicle lateral acceleration | Single precision float | m/s2 | [-inf, inf] |
| long\_acceleration\_m\_s\_s | Vehicle longitudinal acceleration | Single precision float | m/s2 | [-inf, inf] |
| yaw\_rate\_deg\_s | Vehicle yaw rate | Single precision float | deg/s | [-inf, inf] |
| wheelBrakes\_bitmask | Vehicle wheel brake status | 8-bit unsigned integer | Bitmask:  1 – Left front  2 – Left rear  4 – Right front  8 – Right rear | [0, 15] |
| wheelBrakesUnavailable | Vehicle wheel brake sensor status | Boolean | True = unavailable, False = available | {True, False} |
| tractionControlState\_enum | Vehicle traction control system state | 8-bit unsigned integer | 0 – Unavailable  1 – Off  2 – On  3 – Engaged | [0, 3] |
| anti\_lock\_brake\_status\_enum | Vehicle anti-lock-brake system status | 8-bit unsigned integer | 0 – Unavailable  1 – Off  2 – On  3 – Engaged | [0, 3] |
| stability\_control\_status\_enum | Vehicle stability control system state | 8-bit unsigned integer | 0 – Unavailable  1 – Off  2 – On | [0, 2] |
| brake\_boost\_enum | Vehicle brake boost system status | 8-bit unsigned integer | 0 – Unavailable  1 – Off  2 – On | [0, 2] |
| auxiliary\_brake\_status\_enum | Vehicle auxiliary brake status | 8-bit unsigned integer | 0 – Unavailable  1 – Off  2 – On  3 – Reserved | [0, 3] |
| vehicle\_height\_cm | Vehicle length, preserved for backwards compatibility | 32-bit unsigned integer | cm | [0, 4294967295] |
| vehicle\_width\_cm | Vehicle width | 32-bit unsigned integer | cm | [0, 4294967295] |
| userDE1 | User defined element 1 | 8-bit unsigned integer | N/a | [0, 255] |
| userDE2 | User defined element 2 | 8-bit unsigned integer | N/a | [0, 255] |
| userDE3 | User defined element 3 | 8-bit unsigned integer | N/a | [0, 255] |
| userDE4 | User defined element 4 | 8-bit unsigned integer | N/a | [0, 255] |
| userDE5 | User defined element 5 | 8-bit unsigned integer | N/a | [0, 255] |

### 2.2.2 How to access Data

To configure the BSM reception module in the software it is necessary to make use of the dsrc\_handler.mdl Simulink library, as shown in Figure 15. This library contains the DSRC Adapter block. This block should be connected to the UDP socket block that corresponds to where the Linux PC will forward DSRC data. Once connected, this block will take care of decoding the BSMs and updating the BSM storage in memory.

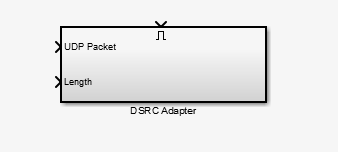


Figure 15: The Contents of the dsrc\_handler.mdl library

The research BSM data is stored in a globally available Simulink.Signal object. Once imported into the Simulink workspace, this data is accessible in any code that needs to integrate with it by way of a Data Store Read block. Incoming DSRC data is stored in a sub-bus corresponding to the vehicle it came from (determined by the id field in the BSM). Veh1BSM contains data from the Green Cadillac, Veh2BSM contains data from the Silver Cadillac, Veh3BSM contains data from the Gray Cadillac, Veh4BSM contains data from the White Cadillac, and Veh5BSM contains data from the Black Cadillac. One of the sub-busses corresponds to the current vehicle and as such is not populated by incoming BSM data. Instead, it is populated by the software for broadcast usage. Data in the current vehicle’s sub-bus are read, converted, and encoded into a Research BSM before being sent to the Linux PC and Arada OBU for broadcast. Figure 16 demonstrates how to set up a BSMStoreSignal read.

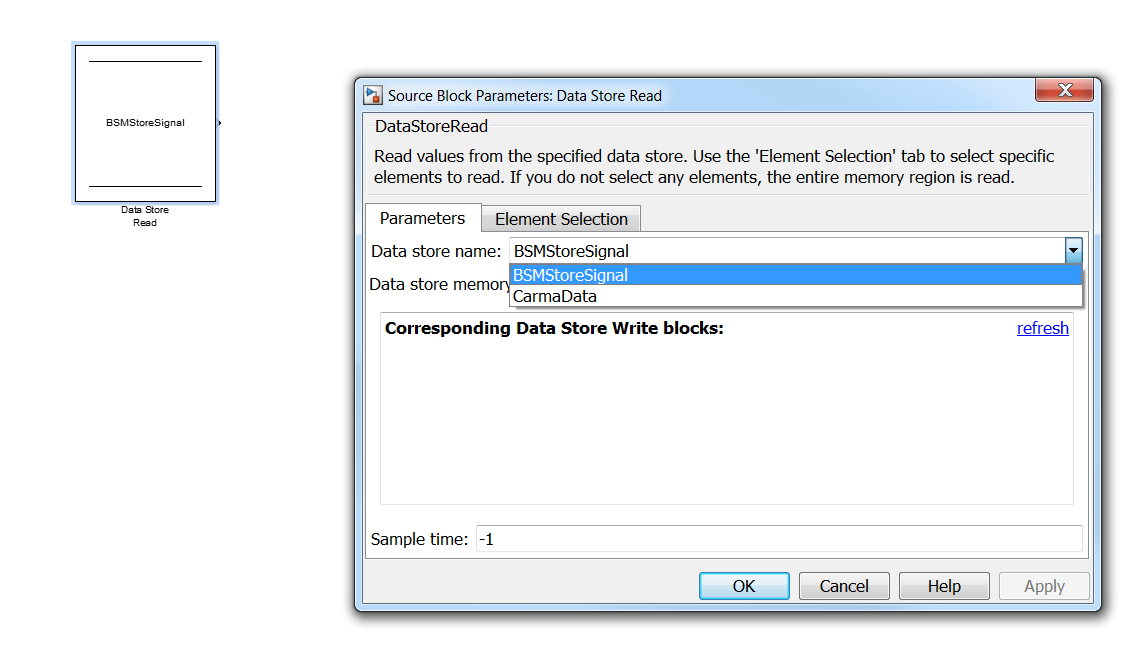


Figure 16: Setting up a BSMStoreSignal read

To broadcast Research BSMs it is necessary to make use of the dsrc\_transmitter.mdl Simulink Library, as shown in Figure 17. This library contains blocks to update the BSM memory associated with the current vehicle (based on stored vehicle ID) with both experimental data, update the memory with vehicle data, and trigger a Research BSM broadcast to occur. Once invoked, the BSM broadcast reoccurs at a 20Hz frequency to broadcast whatever data is contained in the current vehicle’s BSM storage.

The Update Current BSM Data block is used to create a Simulink.Bus object that contains the experimental data to be broadcast in the BSM. Due to constraints with the way the real-time processing on the Micro Autobox works it is necessary that this block only output a bus, instead of writing directly to the data store, to avoid race conditions with data modification. Also included in the library is the Write CACC Data block (which should be added to the model outside any real-time timer task) that will write the data from the Update Current BSM Data block into the correct data store.

The BSM Broadcast block is used to trigger a BSM broadcast at a fixed 20Hz frequency. When invoked it reads from the GlobalDataStore to get needed values, converts the data into the appropriate units, formats, and data types, and then writes it to the correct BSM data store. This data store is then converted into an ASN.1 encoded Research BSM and sent over UDP to the Linux PC.

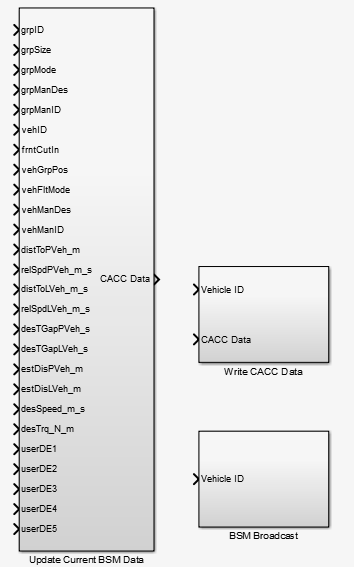


Figure 17: Contents of the dsrc\_transmitter.mdl library.