

Software Requirements Specification (SRS)

Team TeraByte

Onboard Diagnostics System

Authors: Jonathan Rietveld, Kyle Bartush, Yuanchun Zhao, Brandon Dorazio

Customer: Dr. Ed Nelson, Ford Motor Company, Innovation Center

Instructor: Dr. Cheng

1 Introduction

This section provides a brief introduction to the whole document. It includes the purpose, scope and organization of the SRS document; as well as its glossary.

1.1 Purpose

The purpose of this document is to present a complete description of the behavior of the Onboard Diagnostics (OBD) System. It will illustrate the function and performance of the system, the constraints under which it must operate, and its external interface with system hardware, other programs, and users. This document is intended for both the stakeholders and the developers of the system.

1.2 Scope

Team TeraByte will create an OBD System for road vehicle applications based on the widely used Controller Area Network (CAN) architecture. The system will be designed to enhance the safety of the driver by monitoring the status of software modules in the vehicle, and facilitate the repair process by offering technicians access to read the health information for the vehicles modules.

This system will focus on diagnosing two modules; the camera and radar. The camera is used for backing up and parking assistance. The radar is used to detect oncoming objects. This system will warn the driver using an error message and indication light when a problem is detected with either of the modules. The system allows a technician to easily identify and repair malfunctions within the vehicle by offering real time data; as well as a standardized series of diagnostic trouble codes (DTCs).

1.3 Definitions, acronyms, and abbreviations

Table 1 defines all terms, acronyms and abbreviations needed to understand the SRS.

Term	Acronym / Abbreviation	Definition
Controller Area Network	CAN, CAN bus	A vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer
CAN ID		The ID specific to that message. This ID determines the type of message. Each module has two IDs, one for transmitting message and one for receiving message.
Checksum		A computed value from the message for the purpose of detecting accidental errors that may have been introduced during its transmission on the CAN bus
Diagnostic Trouble Code	DTC	An electronic signal stored in an automotive system indicating the presence of a fault detected by that system
Light-Emitting Diode	LED	A semiconductor light source
Module		A subsystem of the vehicle with both hardware and software, such as radar or camera
Onboard Diagnostic	OBD	OBD, in an automotive context, is a generic term referring to a vehicle's self-diagnostic and reporting capability
Operator		The driver of the vehicle
Software Requirement Specification	SRS	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate.
Stakeholder		Any person with an interest in the project who is not a developer
Tell Tales		An indicator, signal, or sign that conveys the status of a situation, mechanism, or system
User		The driver of the vehicle or a repair technician

Table 1: Glossary

1.4 Organization

Section two gives an overall description of the software, including the perspective and functions of the software, as well as constraints that affect the software and its requirements. Sections three and four offer the developers a detailed specification of the software requirements. Section five provides a prototype of the software.

2 Overall Description

This section contains the specific implementation details of the system. It includes an overall system description, the functions the system will serve, the constraints of the system, the requirements of the system, and what is assumed and expected from the operator and technician when they are using the system.

2.1 Product Perspective

The OBD System is a system that detects errors with modules used in the vehicle, reports those errors to the operator, and stores those errors so that a technician can read them. This system is implemented in a vehicle to prevent the operator from relying on a broken module and to report any errors that occur with a module. For example, the Onboard Diagnostic System will prevent the operator from using the adaptive cruise control if the radar module is out of specification or has an internal error. The Onboard Diagnostic System is built into each module and interacts with other modules by transmitting and receiving messages over the CAN bus.

2.2 Product Functions

The OBD System has several major functions. To perform these functions the system logs errors, transmits and receives messages, and runs automated self-tests. The message system in the vehicle is what allows us to communicate with the operator and with other modules. The diagnostic system uses the CAN messaging system. It consists of several modules all connected on a bus. Each module is responsible for running diagnostic tests on its sensors and to broadcast the appropriate messages out onto the CAN bus. When other modules receive the message, they determine what to do with the message based on the CAN ID of the message. The message's format is shown in **Figure 1** below. The checksum is used to test if there was any corruption of the message while being transmitted over the bus. If a checksum fails, the module that received the corrupted message will prompt the module that transmitted the message to retransmit.

Byte	Data
1	CAN ID
2	Type
3	Number
4	Data
5	Reserved (default to 0x00)
6	Checksum

Figure 1: Format for CAN messages

To be able to inform the operator in real time the modules must run self-tests. They are run at a specific interval that is determined from a safety standard. If an error is found the module must perform two actions. First it must send a message out on the CAN bus, then store a log entry of the error it has detected in a local memory store that is persistent across power loss. Due to memory constraints, the log will consist of a fixed number of messages. Depending on the specific CAN implementation used, there are bandwidth constraints which could cause modules to queue their messages if there is too much activity on the bus.

The diagnostic system can communicate with the operator or a technician. There are two ways in which the diagnostic system interacts with the operator. The first is a LED text display that will give errors in text form to the driver. The driver has the option to clear these messages. The second form is tell tales, which are LED lights on the display that cannot be turned off by the operator. The only time these error indicators will go away is when the error has been taken care of. For a technician to communicate with the diagnostic system a diagnostic tool is required. This tool can be plugged into the CAN bus by the technician. It will have three main purposes: to ask the modules to dump their error logs, to ask the modules to run diagnostic self-tests, and to ask the modules for their software version for updating. All these tasks are accomplished by exchanging messages over the CAN bus. This is shown below in **Figure 2**. This figure shows the different functions that the diagnostic system will serve to its two possible users, the technician and car operator.

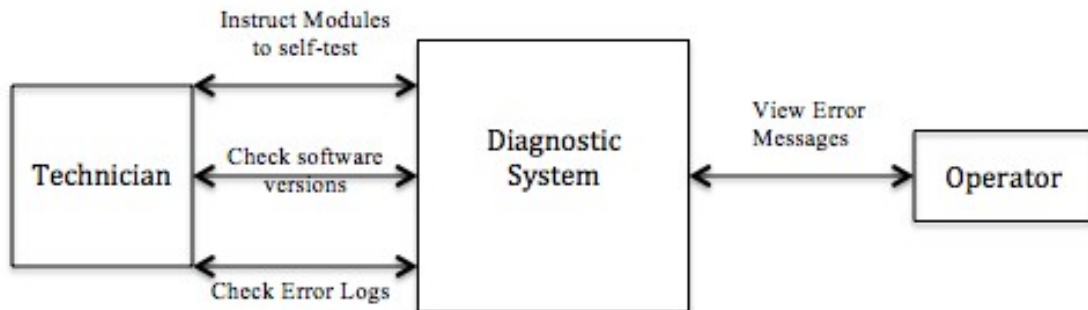


Figure 2: High-Level Diagram of system functions.

2.3 User Characteristics

It is important to know what to expect from the users of the diagnostic system. The operator is a user that will only be aware when a malfunction occurs in the Onboard Diagnostic System. The operator may have knowledge of the diagnostic system, but most likely will not know how it operates. The operator may rely on the diagnostic system for driving assistance features and the OBD system is designed to notify the operator with a text error message and a tell tale light if necessary for an error in the system. The operator does not need to take immediate action on an error message but it is in the operator's best interest to take the vehicle to a technician to get the error checked out. The technician will be trained and have a technical understanding of the diagnostic system. They have

the ability to check the meaning of certain fault types and error codes. The technician will communicate with the diagnostic system over the CAN bus by using a diagnostic tool. It should not be assumed that they know exactly how the diagnostic system operates or how the CAN protocol is implemented.

2.4 Constraints

There are some constraints that are associated with the diagnostic system. One is that the number of error messages allowed to be stored on each module is fixed. This number will be determined by the amount of storage space on the module. Another constraint is that for the diagnostic system to work properly the vehicle must be turned on and supply power to the modules.

There are some safety-critical properties involved in the OBD System. First, the diagnostic system has to be able to shut off any driver assistance features related to a module if some errors are detected within this module. Otherwise, an accident may happen since the user relies on a faulty system. Second, the diagnostic system must also be able to self test its modules to check whether there is any problems.

2.5 Assumptions and Dependencies

The operator does not interact with the diagnostic system directly, but is expected to take action on error messages. The hardware in the diagnostic system is assumed to be composed of a sensor(s) and a controller that has a diagnostic layer that interacts over the CAN bus with other modules and their controllers. The technician is assumed to have been trained and capable to properly interact and communicate over the CAN bus by using a diagnostic tool.

2.6 Apportioning of Requirements

This section contains information on requirements that are out of the current scope of this project. However, these features could be implemented in new versions. A requirement that we have determined to be out of the scope of this project is the ability to update software on modules.

3 Specific Requirements

This section contains a hierarchical list of the requirements of the OBD System for Ford Motor Company. It contains detailed requirements for the components of the CAN bus system.

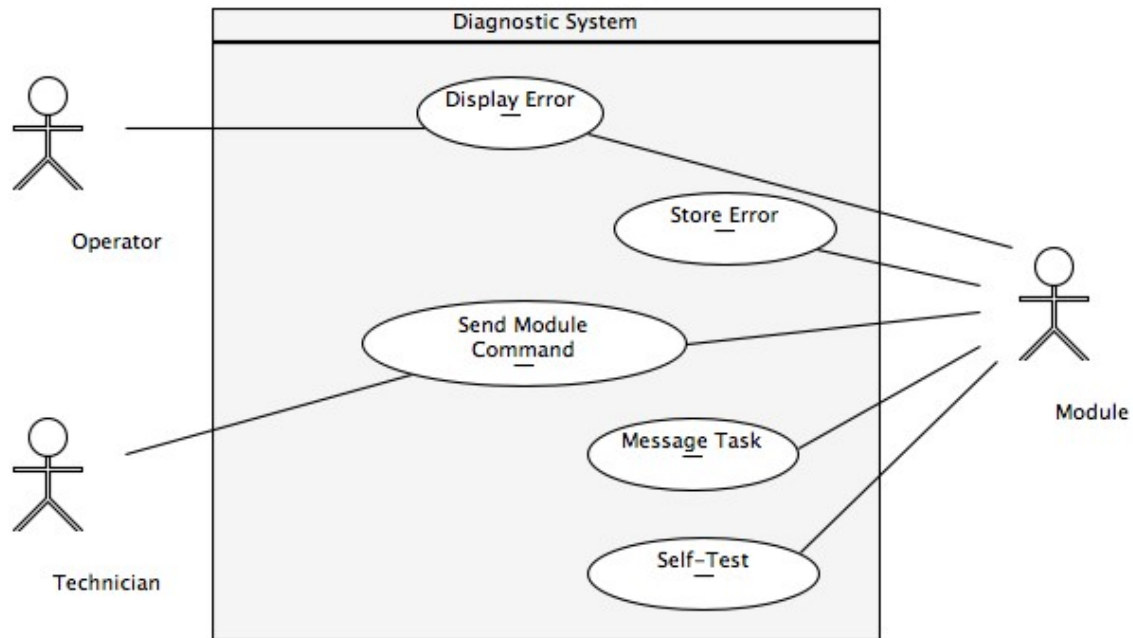
1. Module
 - 1.1. A module must be able to self diagnose and detect if there is an error within the module.
 - 1.1.1. The self diagnose test must be performed on startup and at regular intervals specified by a safety standard during the operation of the vehicle
 - 1.2. A module must be able to act on messages that contain a required action

- 1.3. A module must store a log of error messages
 - 1.3.1. An error message must contain a timestamp and the error type (id)
 - 1.3.2. If the log is full, the oldest message is deleted before saving the new log
 - 1.3.3. Repeated instances of the same error are not logged
- 1.4. A module must be able to dump the log of error messages when requested by the diagnostic tool
- 1.5. A module must be able to detect if a message received is corrupt
 - 1.5.1. This will be detected by verifying the checksum
- 1.6. A module must set an internal flag if a checksum is detected as corrupt
 - 1.6.1. The checksum is determined by use of a cyclic redundancy check
- 1.7. A module must be capable of sending and receiving messages
- 1.8. A module must be able to check the current software version
- 1.9. A module must accept a message if the CAN ID is the ID of 0x250, 0x350, 0x260, or 0x360
- 1.10. A module must be able to go into power down mode when an error has occurred and the action required for the error is to go into power down mode
2. Dashboard
 - 2.1. The dashboard should determine if an error message needs to be displayed to the operator
 - 2.2. The dashboard should display an error message to the driver if it has detected an error the operator needs to be made aware of
 - 2.2.1. The dashboard should display a LED text message if necessary for the error message
 - 2.2.2. The dashboard should display a tell tale LED if necessary for the error message
3. Messages
 - 3.1. A message should contain a unique CAN ID
 - 3.2. The general format of a received diagnostic message and that of a transmitted diagnostic message are different
 - 3.2.1. A received diagnostic message contains a one byte CAN ID, a one byte diagnostic command, a one byte diagnostic sub-command, a one byte reserved data, a one byte reserved data, and a one byte checksum
 - 3.2.2. A transmitted diagnostic message contains a one byte CAN ID, a one byte fault type, a one byte fault number, a one byte fault data, a one byte reserved data, and a one byte checksum
4. CAN Bus
 - 4.1. The CAN bus must transmit messages across the bus to everything attached to the CAN bus
 - 4.2. The CAN bus must determine message priority
 - 4.3. The CAN bus must send a message with the highest priority if more than one message needs to be sent across the CAN bus at the same time

4 Modeling Requirements

This section consists of diagrams and tables that give a high-level or low-level explanation of the Onboard Diagnostic System. From these diagrams and tables you should be able to get a good understanding of how the system works and how it fits in the larger picture of the overall system. A high-level diagram in **Figure 3** contains use cases for the diagnostic system.

Use Case Diagrams:



Use Case:	Display Error
Actors:	Operator, Module
Description:	A module can detect an error that needs to be displayed to the operator. It will tell the operator by displaying an error message on a display that will indicate what is wrong with the system.
Type:	Primary
Includes:	<N/A>
Extends:	<N/A>
Cross-refs:	2.1, 2.2
Use Cases:	<N/A>

Use Case:	Message Task
Actors:	Module
Description:	The modules can send and receive messages to each other. Each message has a unique CAN ID for identification. The messages sent to modules can contain actions the modules must take. The structure for received and transmitted messages differs from each other. Data corruption can be detected in messages using a checksum.
Type:	Primary
Includes:	<N/A>
Extends:	<N/A>
Cross-refs:	1.2, 1.5, 1.6, 1.7, 1.9, 1.10, 3.1, 3.2, 4.1, 4.2, 4.3
Use Cases:	Store Errors

Use Case:	Self Test
Actors:	Module
Description:	On the initial start up of the vehicle all modules must self-test and make sure they have no errors. At a determined interval all modules must run self-test to make sure they have not gone out of specification or have an internal error.
Type:	Primary
Includes:	<N/A>
Extends:	<N/A>
Cross-refs:	1.1
Use Cases:	<N/A>

Use Case:	Send Module Command
Actors:	Technician, Module
Description:	Allows a technician to view the errors stored on a particular module. Allows the technician to send commands that a module must take. The technician can check the software version of all the modules connected to the CAN bus.
Type:	Primary
Includes:	<N/A>
Extends:	<N/A>
Cross-refs:	1.4, 1.8
Use Cases:	<N/A>

Use Case:	Store Error
Actors:	Module
Description:	If any module detects an error in itself, it will store the error internally for later reading by a technician.
Type:	Primary
Includes:	<N/A>
Extends:	<N/A>
Cross-refs:	1.3
Use Cases:	Send Module Command

The section below is a domain diagram in the form of a class diagram. It contains the classes and their relationships that make up the diagnostic system. A class diagram can be used to get a lower-level understanding of how the system should work. Below in **Figure 4** is the class diagram for the diagnostic system.

Domain Diagram:

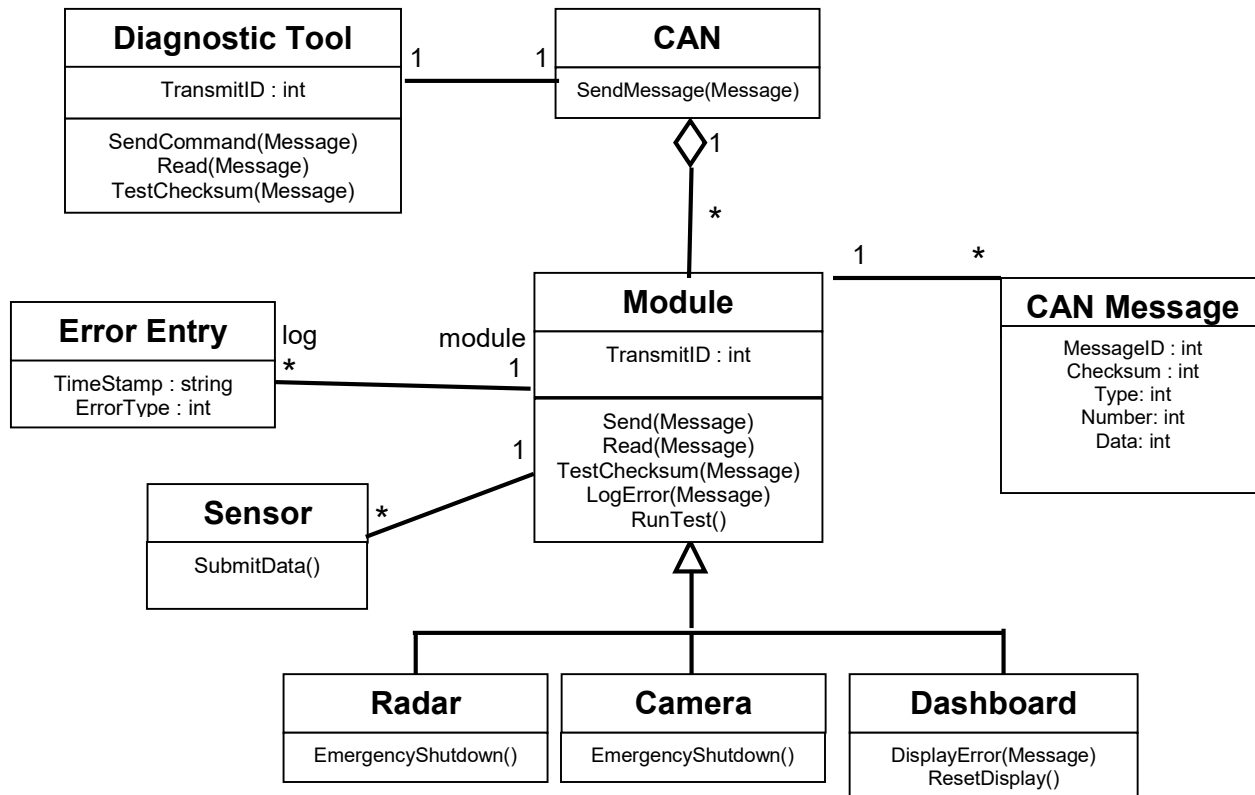


Figure 4: Class diagram

The following diagrams contain a data dictionary for each class in the class diagram above in figure 4. The data dictionary contains the class name, relationships, attributes, and the operations.

Data Dictionary:

Class Name:	Camera
Relationships:	- This inherits its functionality from the module class and is a more specific type of module that relies on data from a sensor
Attributes:	<N/A>
Operations:	
EmergencyShutdown()	- Gives command for the camera to shut down

Class Name:	CAN
Relationships:	- This is the container in which all of the modules are connected. It allows the capability of communication between modules
Attributes:	<N/A>
Operations:	
SendMessage()	- Takes a received message and retransmits it to all modules connected on the CAN bus

Class Name:	CAN Message
Relationships:	- This is a formatted item sent between modules
Attributes:	
MessageID : int	- Indicates the ID of the message
Checksum : int	- Indicates the checksum of the message
Type: int	- Indicates the type of the message
Number: int	- Indicates the number of the message
Data: int	- Indicates the data of the message
Operations:	<N/A>

Class Name:	Dashboard
Relationships:	- This is the device in which modules send errors to for displaying to the user
Attributes:	<N/A>
Operations:	
DisplayError(Message)	- Displays the message either through text or indicator lights (or both concurrently)
ResetDisplay()	- Gives command for the text display to stop displaying current error message

Class Name:	Diagnostic Tool
Relationships:	-Used by technician for diagnostic testing on modules attached to the CAN bus -Sends message commands to individual modules -It can request an error log from a specific module
Attributes:	
TransmitID : int	- Indicates the transmit message id of the diagnostic tool
Operations:	
SendCommand(Message)	- Sends a properly formatted message to the CAN bus
Read(Message)	- Reads a received message and then takes the proper action
TestChecksum(Message)	- Receives message and checks for data transmission errors

Class Name:	Error Entry
Relationships:	- This is a formatted item which is stored within a modules error log
Attributes:	
TimeStamp : string	- Indicates the time in which the error occurred
ErrorType : int	- Indicates the type of error that occurred
Operations:	<N/A>

Class Name:	Module
Relationships:	<ul style="list-style-type: none"> - All modules are connected to one-another through the CAN bus - The messages are sent and received one another - When a message is received, the module takes proper action to handle it internally
Attributes:	
TransmitID : int	- Indicates the transmit message id of the module
Operations:	
Send(Message)	- Sends a properly formatted message to the CAN bus
Read(Message)	- Reads a received message and then takes the proper action
Checksum(Message)	- Receives message and checks for data transmission errors
LogError(Message)	- Stores the error message into the log for later referencing
RunTest()	<ul style="list-style-type: none"> - Continually runs tests on modules to detect errors - Once an error is detected, it will log the error and take proper action to send a message to the other modules

Class Name:	Radar
Relationships:	- This inherits its functionality from the module class and is a more specific type of module that relies on data from a sensor
Attributes:	<N/A>
Operations:	
EmergencyShutdown()	- Gives command for the radar to shut down

Class Name:	Sensor
Relationships:	<ul style="list-style-type: none"> - The sensors report their data to the module while the module runs tests - If the sensor has a problem it will be detected by the modules RunTest() function
Attributes:	<N/A>
Operations:	
SubmitData()	- Supplies the module with data it needs while it runs its diagnostic tests

The next section contains sequence diagrams of possible scenarios in the diagnostic system. This is shown through the calls that classes would perform or call to execute the required task of the diagnostic system.

Sequence Diagrams:

Scenario 1a: A diagnostic testing unit is attached to the CAN bus in order to diagnose a general failure in one of the modules. As it queries the specific module, the module must read the message and send what the diagnostic testing unit requested back to it in a message. See **Figure 5** below.

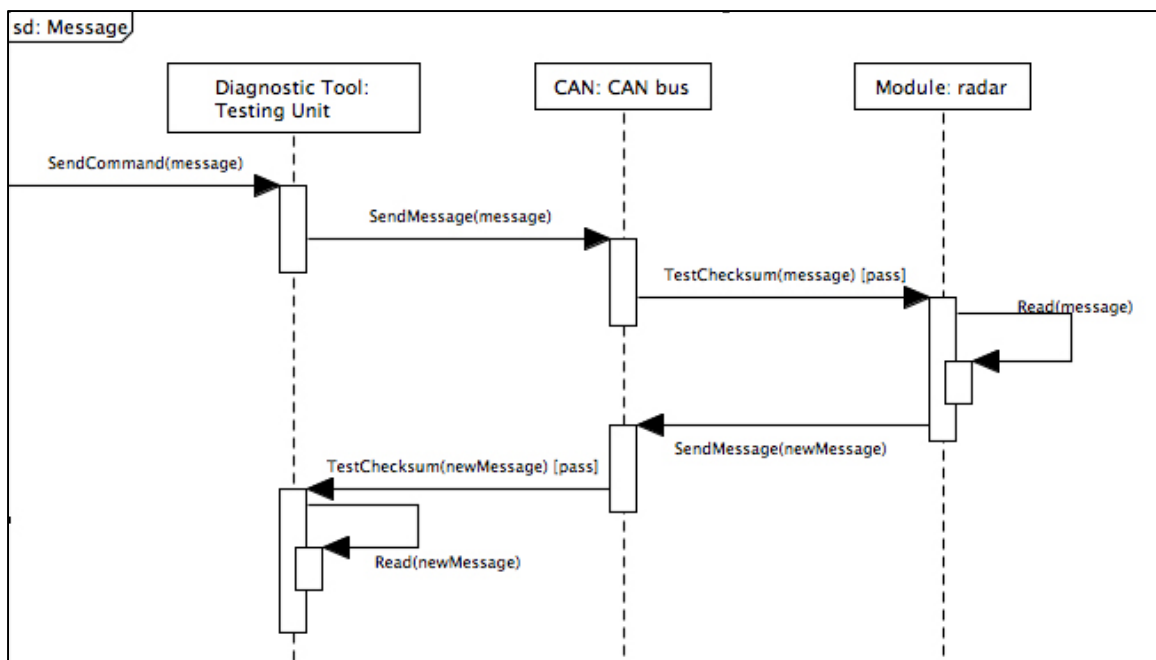


Figure 5: Sequence diagram for scenario 1a

Scenario 1b: A diagnostic testing unit is attached to the CAN bus in order to diagnose a general failure in one of the modules. As it queries the specific module, its data becomes corrupted along the way due to a collision with another CAN frame on the bus. The affected module must determine that an error has occurred, and log the failure internally. See **Figure 6** below.

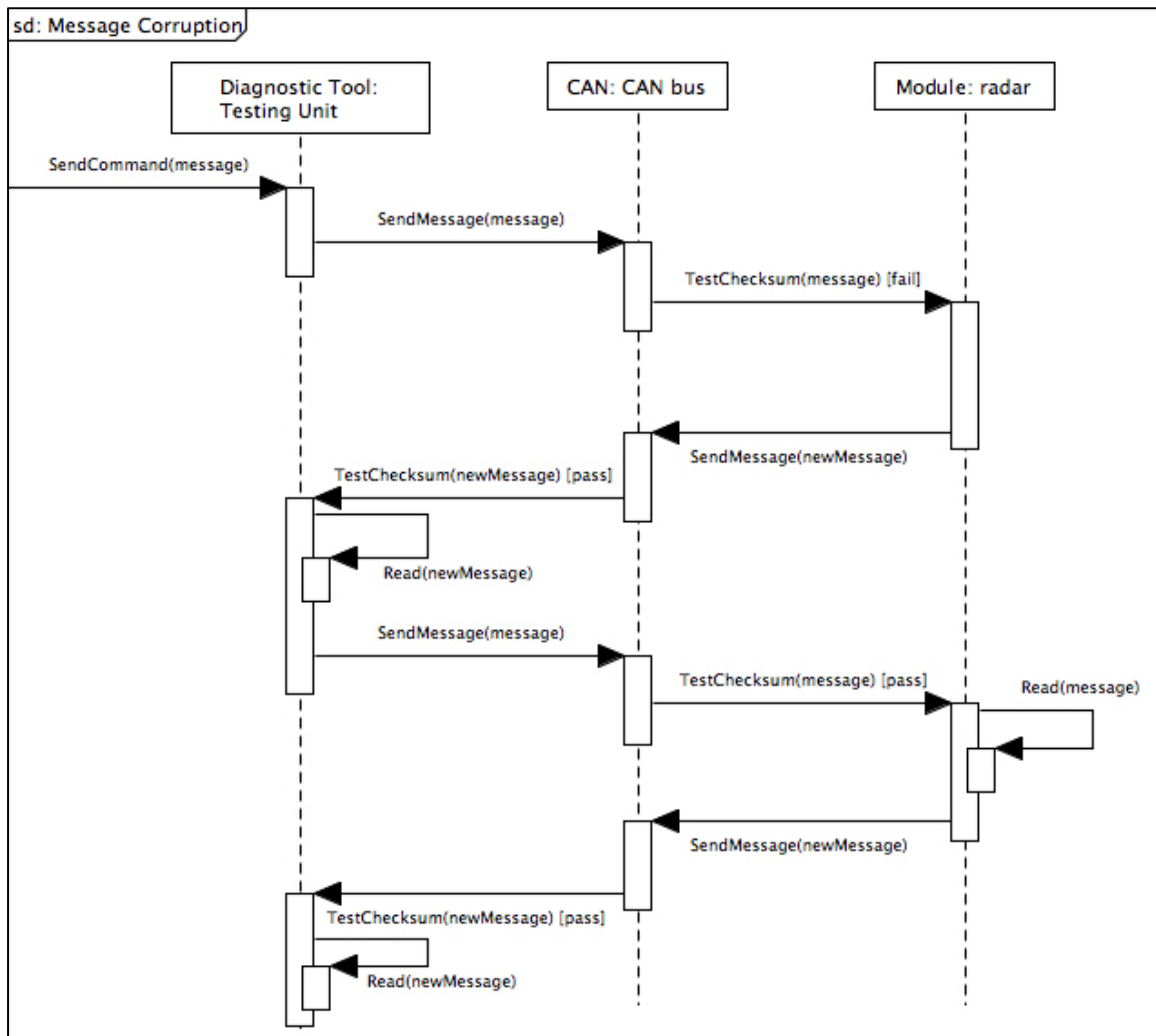


Figure 6: Sequence diagram for scenario 1b

Scenario 2: The camera system detects an error with its image processing unit, and needs to go into an emergency shutdown mode. In doing so, it needs to first notify the vehicle's central processing unit, and be able to log an internal error for later reading by a service technician. See **Figure 7** below.

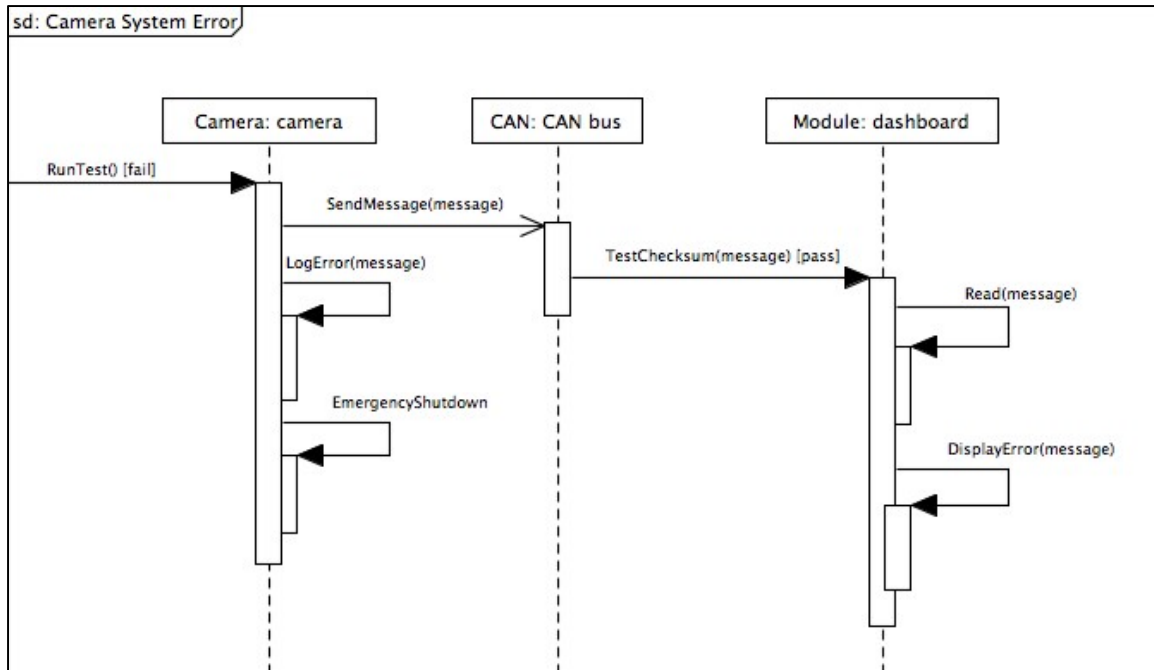


Figure 7: Sequence diagram for scenario 2

The following diagrams are behavioral diagrams. They are all used to show the changing states of different components in the diagnostic system. These changes in states are driven by events that would happen in the diagnostic system. The following state diagrams use states and transition events to show a high-level functionality of different components.

State Diagram:

Figure 8 describes the states and events involved in the module component of the diagnostic system.

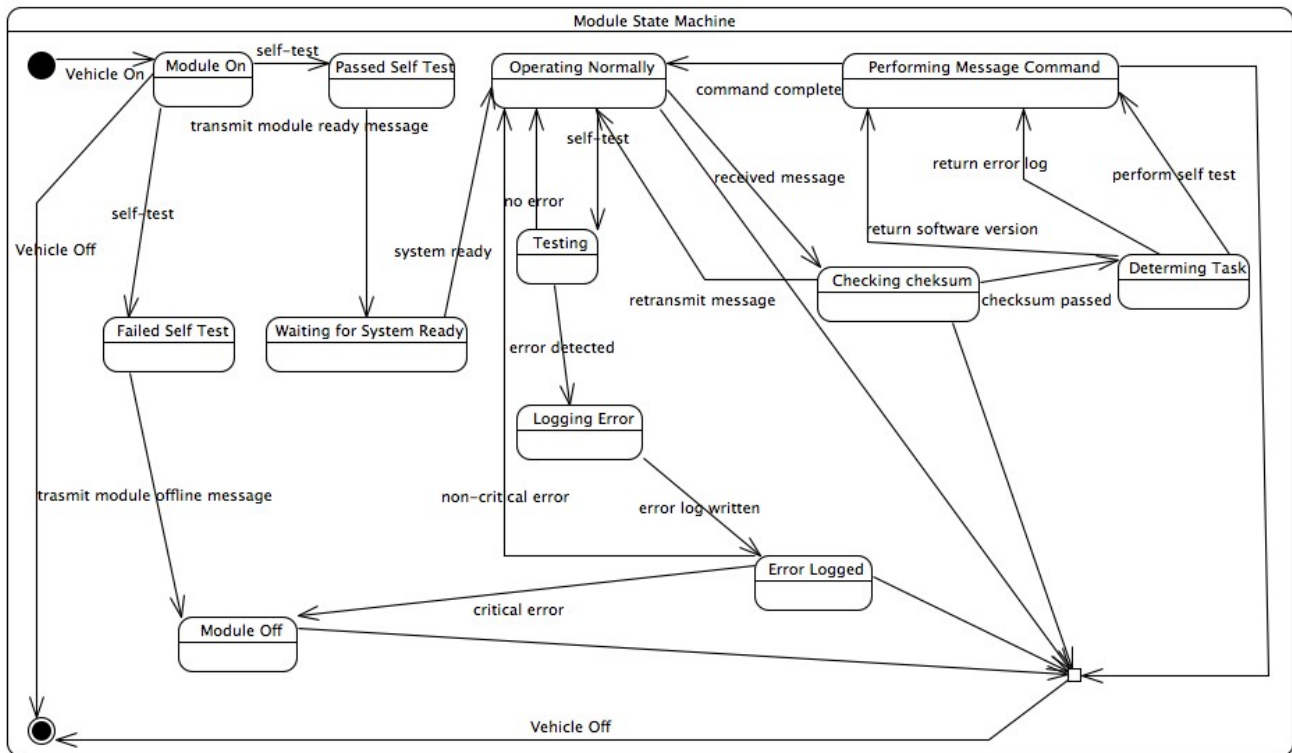


Figure 8: Module state diagram

Figure 9 shows the events and transitions involved in the dashboard interacting with the operator.

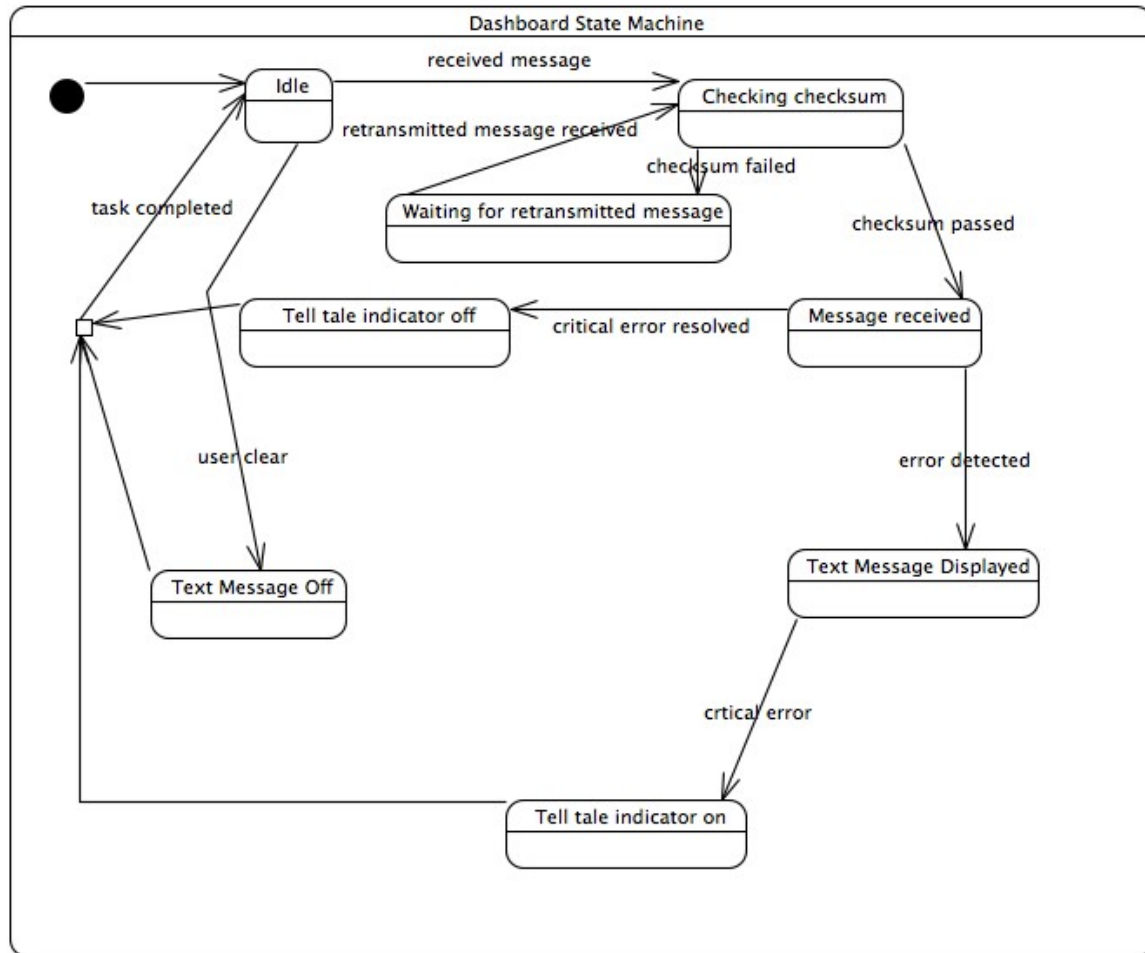


Figure 9: Dashboard state diagram

5 Prototype

The prototype is meant to be a fully functioning representation of how the system would work when implemented. It will allow the user to simulate how the system would react during certain events. These events include technician actions, hardware malfunctions, and checksum fails. When an event is selected by the user the following will be displayed: the message sent onto the CAN bus, the message received by each module, and the action that each module will consequently perform. Each module will also display its entire error log.

5.1 How to Run Prototype

To run the prototype it is only required that the user have any modern web browser with JavaScript enabled. For each module there is a drop-down list of possible actions that may be taken by or on the module. There is also a reset button for wiping the prototype back to its original configuration. To use the prototype simply direct your browser to the following link for version one: http://www.cse.msu.edu/~435diag2/prototype.php?page_id=1 and for version two direct your browser here: http://www.cse.msu.edu/~435diag2/prototype.php?page_id=2.

5.2 Sample Scenarios

One way a user would use the prototype is to see what would happen within the diagnostic system when the technician queries the modules for their diagnostic logs. Each interacting member of the CAN bus has a list of events you may choose from. Simply go to the technician member, and select the “Dump Logs” event from the drop down list. The prototype will then simulate how the system would react by filling in the proper information for each module. In **Figure 9** the system is sitting idle waiting for an event. After selecting the dump logs option under the diagnostic tool, the modules will dump all their logs to the tool and the appropriate information will be displayed to the screen as seen in **Figure 10**.

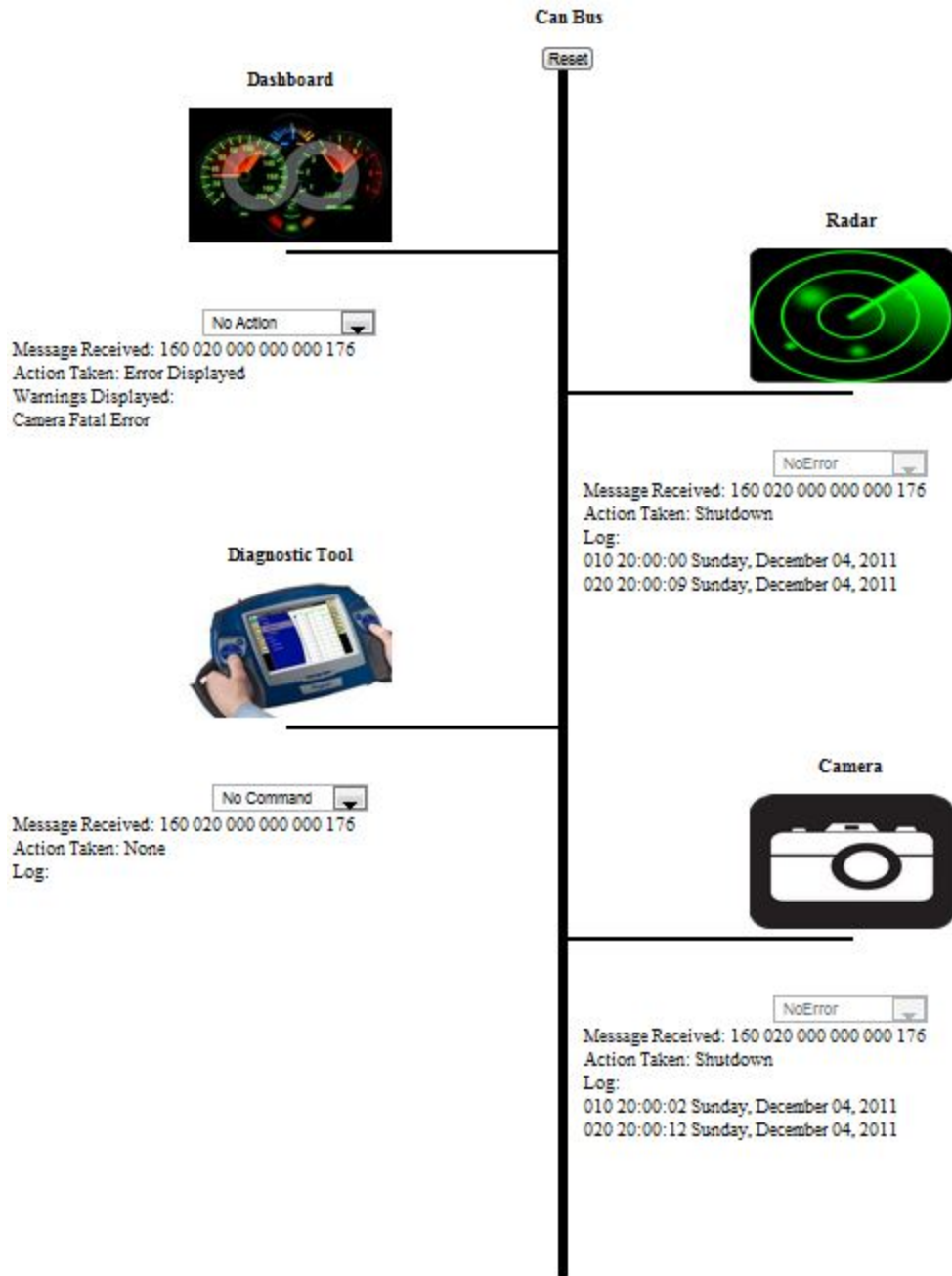


Figure 9: Prototype before any actions are taken.

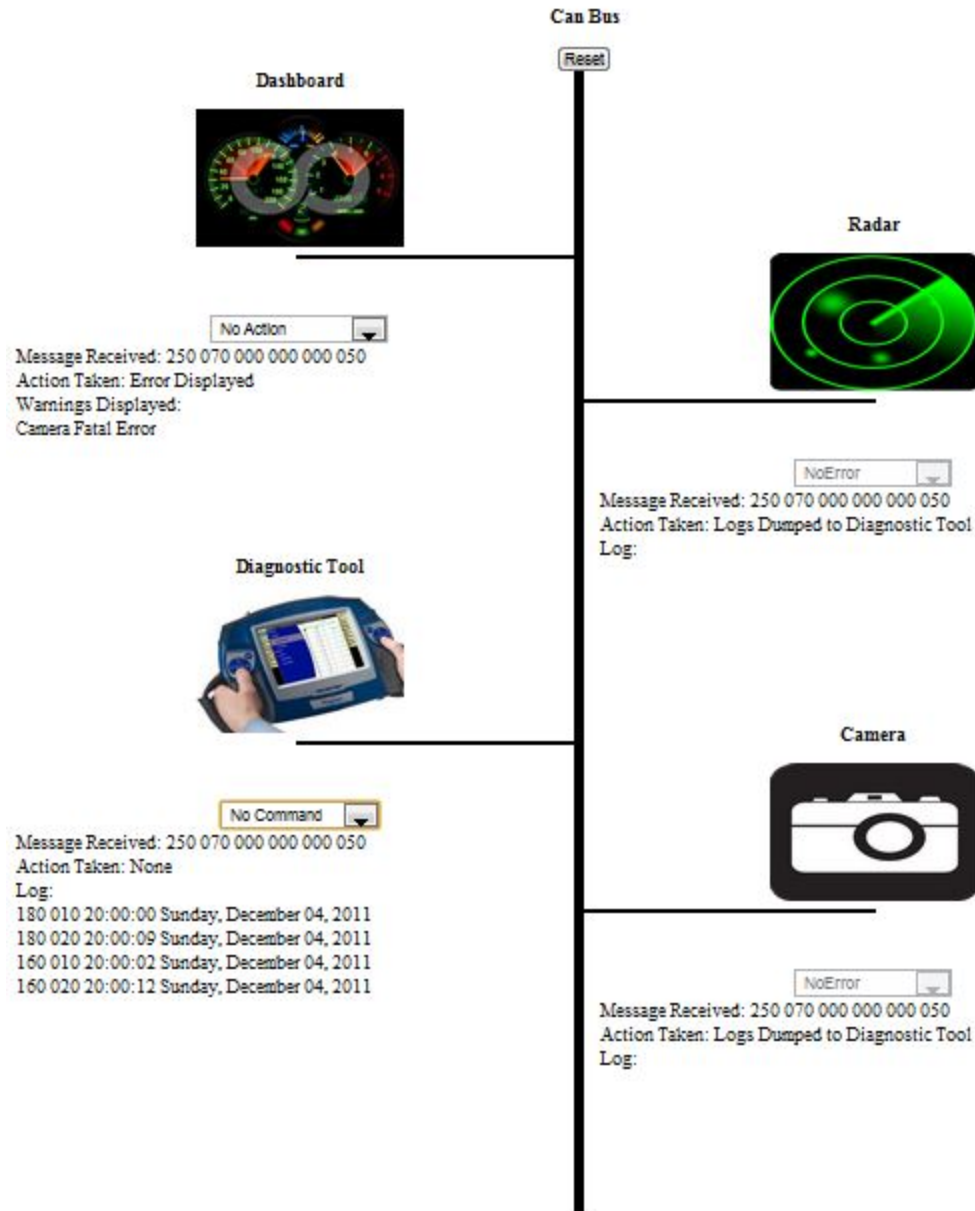


Figure 10: Prototype after the dump logs action has been selected.

6 References

- [1] Mannisto, Daniel, and Mark Dawson. "An Overview of Controller Area Network (CAN) Technology." *mBus*. N.p., 12 Nov. 2003. Web. 18 Oct. 2011.
- [2] *CAN in Automation*. CAN in Automation, n.d. Web. 18 Oct. 2011. <<http://www.can-cia.de/index.php?id=161>>.
- [3] *OBD-II Background*. AutoTrap, 2011. Web. 18 Oct. 2011. <<http://www.obdii.com/background.html>>.

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Betty H.C. Cheng** at Michigan State University (chengb at cse.msu.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.