Distance Measurement Controller Project Report

Document ID:	**Distance Measurement Controller Project Report**
Origin Date:	**Mar 16, 2022**
Applicable to:	**ECE-554 Embedded Systems Winter 2022**
Student Name:	**Luis Castaneda-Trejo**

Revision History:

Version No.	Date	Details of Change	Modified by
1.0	29 Mar 2022	Initial Version.	Luis Castaneda-Trejo

Tasks for a future implementation

Move to Confluence page in Jira later.

#	Feature	Source
1		
2		
3		
4		

Table of Contents

Introduction 5
1.1 Concept 5
1.2 Block Diagram 5
1.3 Scope 6
1.4 Goals 6
1.5 Status of this document ..... **Error! Bookmark not defined.**
1.6 Definitions, acronyms and abbreviations ..... **Error! Bookmark not defined.**
1.7 References 7
Overall description ..... **Error! Bookmark not defined.**
2.1 Hardware -BCM 7
2.2 Hardware -NI 8
2.3 Software -BCM 9
2.4 Software -NI 9
2.5 Control of BCM I/O by LabVIEW ..... **Error! Bookmark not defined.**
2.6 Limits of This Specification 10
Specific Requirements 11
3.1 High Level Requirements 11
3.2 High Level GUI 12
BCM I/O Connections to NI cDAQ 13
3.2.1 CAN diagnostics services and message ID 14
3.2.2 Generic UDS command message structure (more than 8 bytes) 15
3.2.3 Generic UDS command message structure (8 bytes or less) 15
3.2.4 Positive UDS command reply: 15
3.2.5 Negative UDS command reply 15
3.2.6 Generic Negative UDS response codes and explanations 15
Appendix A List Negative Response Codes (NRC) and meaning 16

Introduction

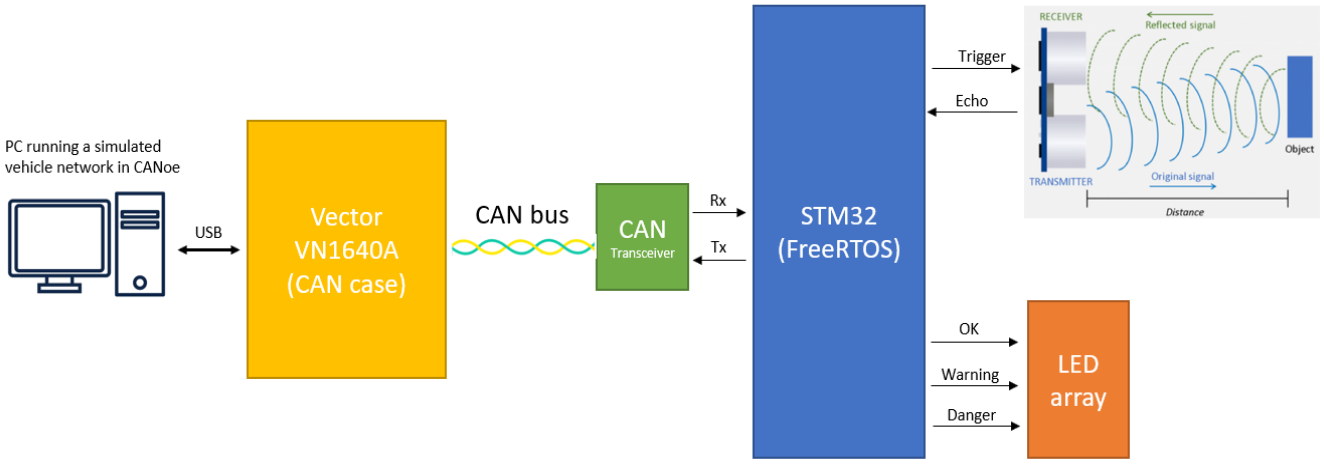
Autonomous mobility technology is becoming more and more sophisticated every year and OEMs are investing heavily in this area. Not so long ago vehicles didn't have any equipment installed that could notify the driver that a crash can occur if the vehicle was moving at a certain speed or an object was too close in front or behind that could impact the car. Now days, almost all new models from all OEMs have basic safety features included in their most basic package. In Advanced Driver Assistance Systems (ADAS), LIDAR sensors are used to determine the proximity of objects from the vehicle. The sensors are usually located on top of the vehicle and they scan the surrounding area proving distance information to objects to one of the ADAS ECUs. Most automotive LiDAR sensors have a motor that rotates 360 deg to provide distance information but there are also LiDAR sensors that are unidirectional.

Similar to the LiDAR unidirectional sensors there are ultrasonic sensors like the HS-SR04 that work in a similar way. Rather than using reflective light they use ultrasonic sound to measure the distance to an object. They work OK with materials that are able to reflect sound and the best environment where to use them is indoors with no other noises in similar frequencies that could interfere with them.

1.1 Concept

The purpose of this project is emulating the functionality of an automotive distance measurement controller. Automobiles use LIDAR sensors but since this project is a Proof of Concept an ultrasound sensor can provide similar results in a smaller scale. The measured distance will be categorized as Safe, Warning or Danger and depending on each category a visual alarm will be triggered.

1.2 Block Diagram



1.3 Scope

The distance measurement controller provides the ability to configure and set new distance categories via CAN

~~The scope of this document includes the BCM software, the modes of I/O activity and the command interface with external test tools. This also includes external GUI on LabView. However, no a detailed design of LabView VIs, DLLs and internals are beyond the scope of this document~~.

1.4 Goals

The overall goals of the Distance Measurement Controller are:

- ~~As the name states, the goal is to make a Simple Software Tester, with the emphasis on Simplicity.~~
- ~~That is, this will be no harder than flipping switches and observing LEDs at the physical bench.~~
- ~~Provide a clean interface that can be easily extended and modified across the life of the BCM product. (Rationale: Maintainability, Modularity, Scalability).~~

1.5 References

Table 1

Ref	Document Name	Version[1]	Current version and Location[1]
[1]	SISTER design, HW etc		DG-061502 > Project Documents > 110 Engineering > 040 Software Engineering > 120 SW Tools > 040 Special Tools >
[2]			
[3]			
[4]			
[5]			
[6]			
[7]			

Project Elements

This section describes the parts of the Distance Measurement Controller, including interconnection of I/O between the simulated ECU (microcontroller), CAN transceiver, Vector interface and LED for alarm visualization.

2.1 Hardware -Simulated ECU

The ECU hardware consists of an STM32 (Nucleo-G431KB) microcontroller. The selected board has an Arm 32-bit Cortex-M4, 128 Kbytes of Flash and 32 Kbytes of RAM. The microcontroller has 1 FDCAN controller supporting flexible data rate. The FDCAN interface is configured as CAN High Speed (HS) only because the distance measurement application does not require more than 8 bytes for payload.

To communicate with a CAN network the TJA1441AT CAN transceiver from NXP was used. This transceiver supports up to 5 Mbit/s in FD mode. The configured speed for the CAN controller is 1 Mbit/s.

The ultrasonic HC-SR04 sensor was used to measure the distance to an object. With a short 10uS pulse to the trigger the module will send out 8 cycle burst of ultrasound at 40kHz and raise its echo. The echo is a pulse width proportional to the measured distance to the object.

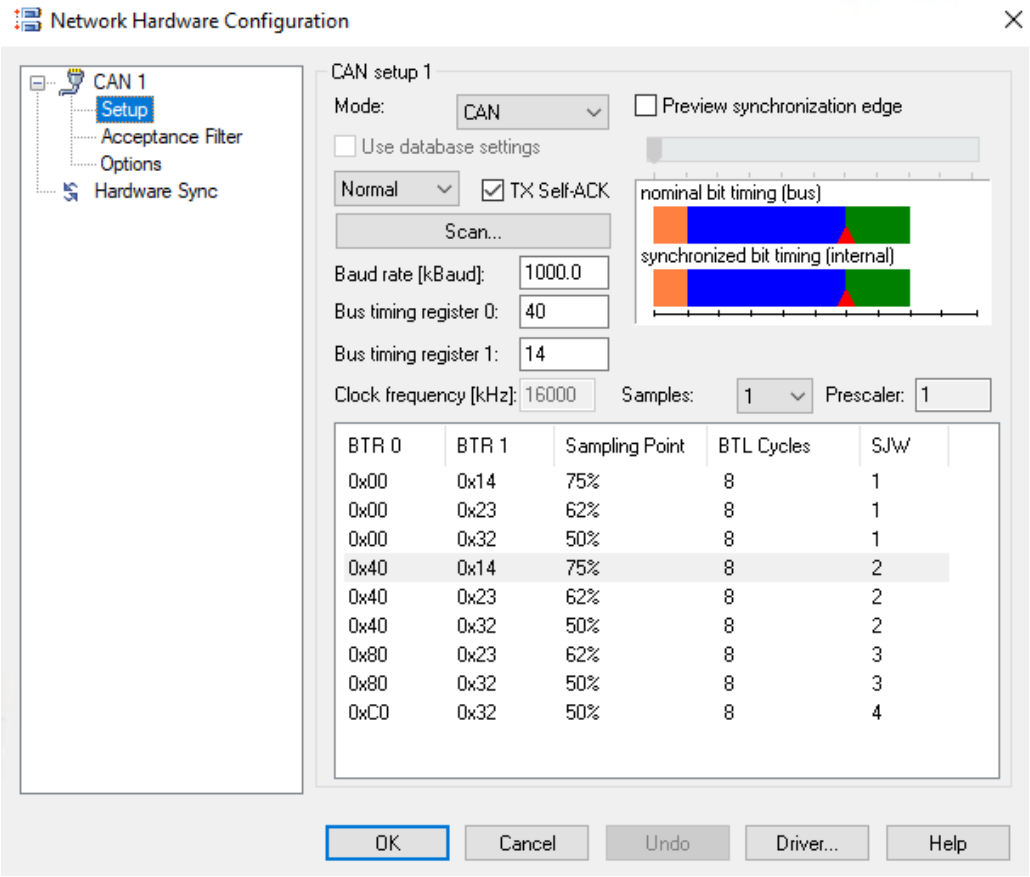
For an easy distance category visualization, an array of 3 LEDs was used. The GREEN LED shows any distance greater than 20 cm. The YELLOW LED shows any distance greater than 10 but less than 20 cm. The RED LED shows any distance less than 10 cm.

The following table summarizes the distance thresholds in the controller:

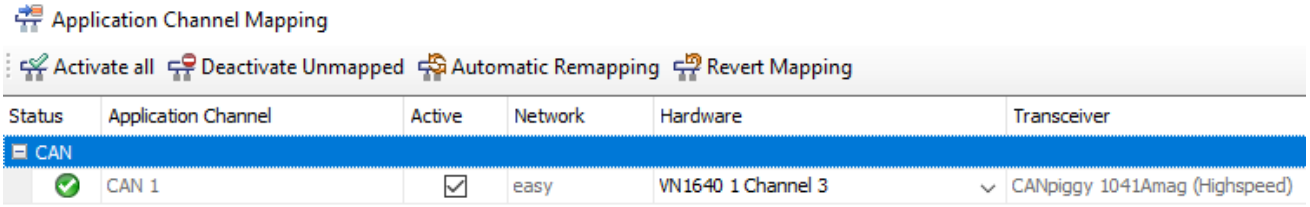
Distance Category	Threshold	Color
Danger	Less than 10 cm	RED
Warning	Greater than 10 cm but less than 20 cm	YELLOW
Safe	More than 20 cm	GREEN

2.2 Hardware -CAN Network

The simulated CAN network provides the right environment to test the ECU. A VN1640A CAN case from Vector was used to interface the ECU to a real CAN network. The VN1640A is a modular interface that supports CAN and LIN interfaces. CAN 3 channel was used as the CAN interface. The following setup was applied to achieve a 1Mbit/s speed network:

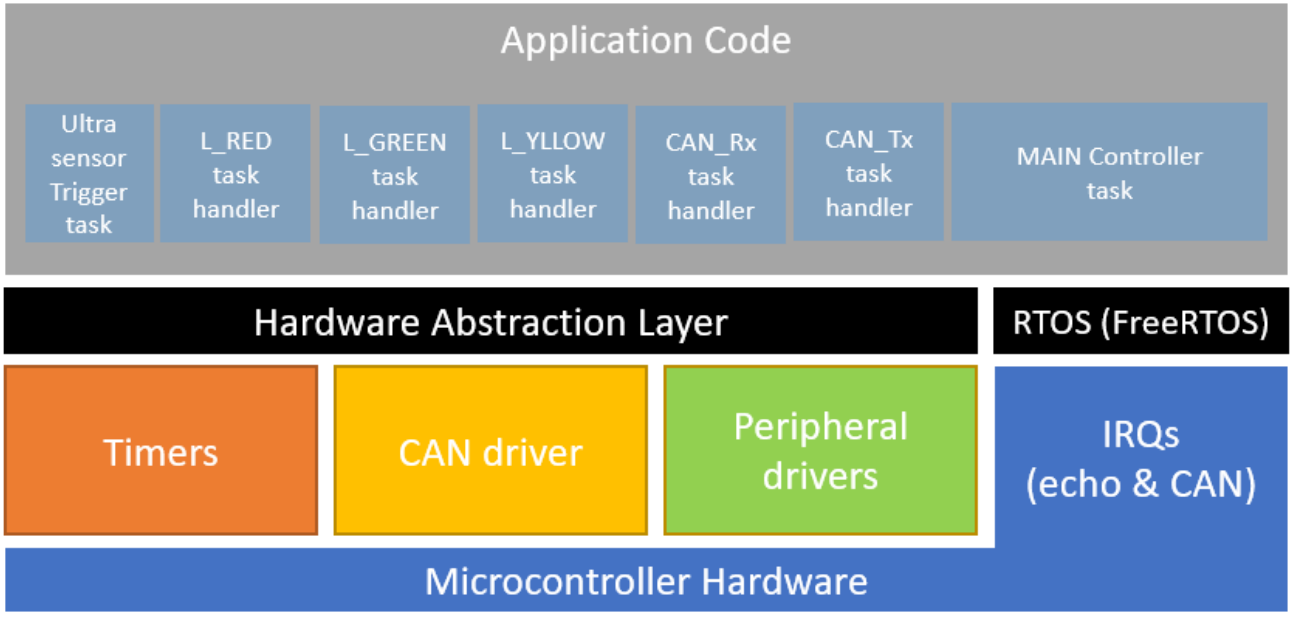


The following image shows the mapping of the CAN channel number used to interface with the ECU.



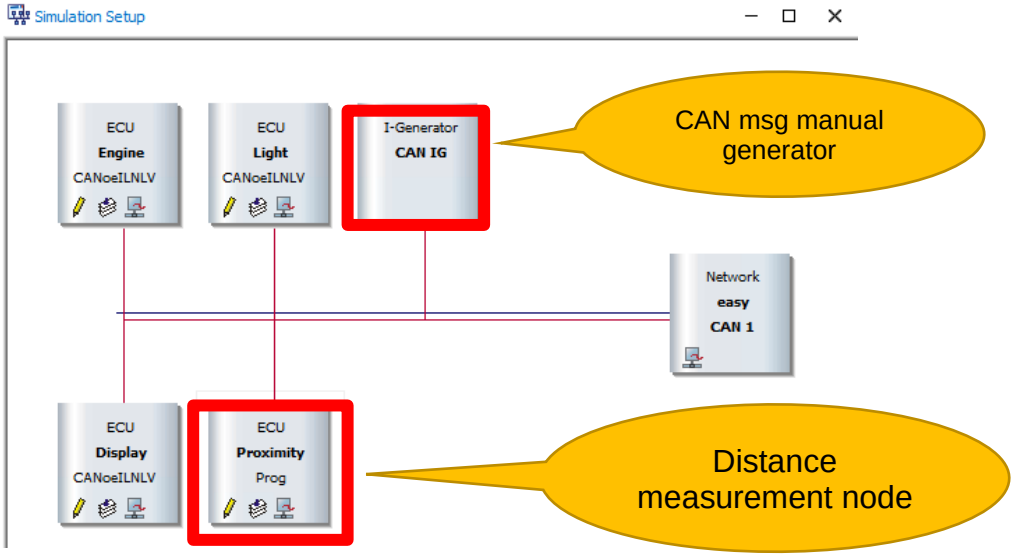
2.3 Software -Simulated ECU

Software in the ECU uses a Real-Time Operative System (FreeRTOS) to handle the tasks of the project. The following figure shows the main software architecture.



2.4 Software -Simulated CAN Network

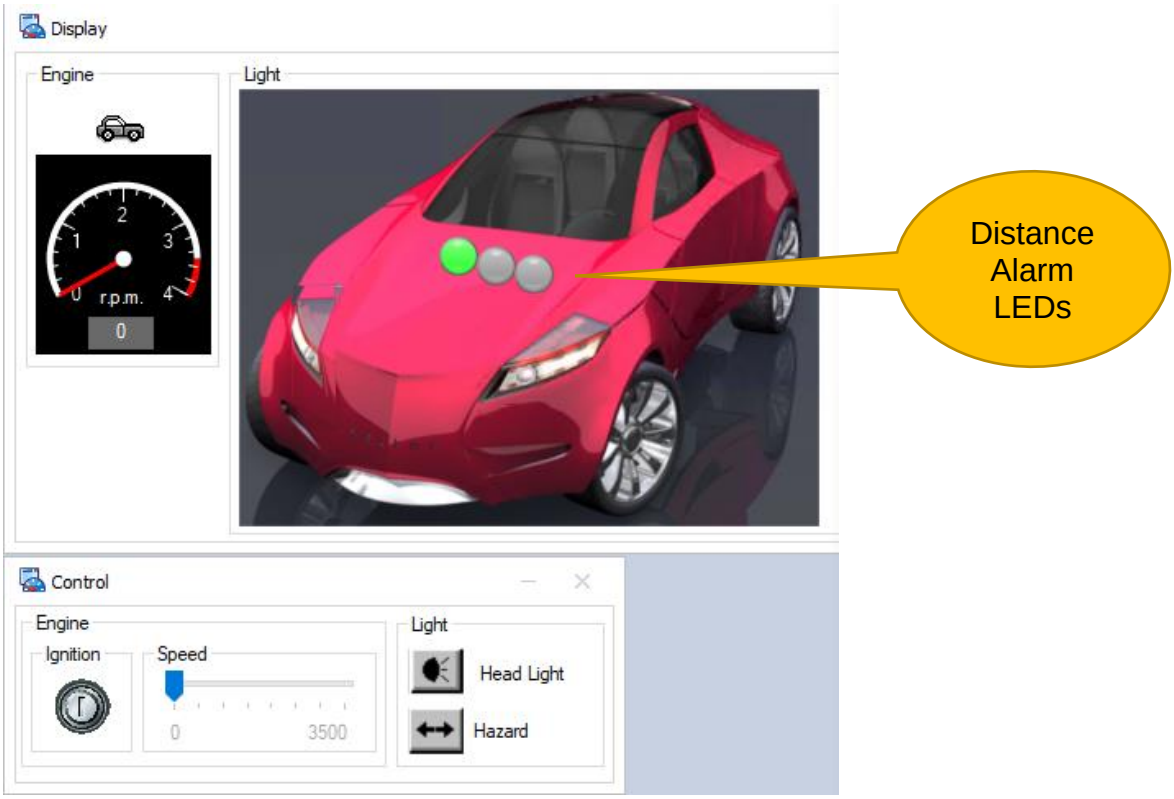
The simulated CAN network was implemented using Vector CANoe. CANoe is a commercial off the shelf software tool to develop, test and analyze individual ECUs and entire ECU networks. It comes preloaded with examples to quickly start analyzing automotive networks. The following CAN network was implemented based on one of the examples that came with the tool and modified to add the proximity ECU in the network. The following figure is the complete CAN network.



An interactive node was added to the network to act as a CAN message manual generator. This node is useful to manually change the distance thresholds mentioned in section 2.1. The CAN message structure follows the diagnostics UDS standard described in section zzz.

2.5 Software -Simulated CAN Network Panel

To visualize the distance measurement sent by the ECU, the main control panel of the CANoe example was modified to include the distance values. The following picture shows the main panel including the 3 LEDs that represent the distance categories. In real modern vehicles the distance alarm is usually located inside the cluster or above it.



The following

2.6 Limits of This Specification

This document does not specify or describe:

- Operation of the BCM Application except for details relevant to DV operation.
- Operation of external test tools and fixtures.

- Testing limits of the BCM inputs and outputs.
- Use of UDS diagnostics other than DV mode for product test purposes.
- Operation of BCM with HW and SW other than the ones referenced above.

Specific Requirements

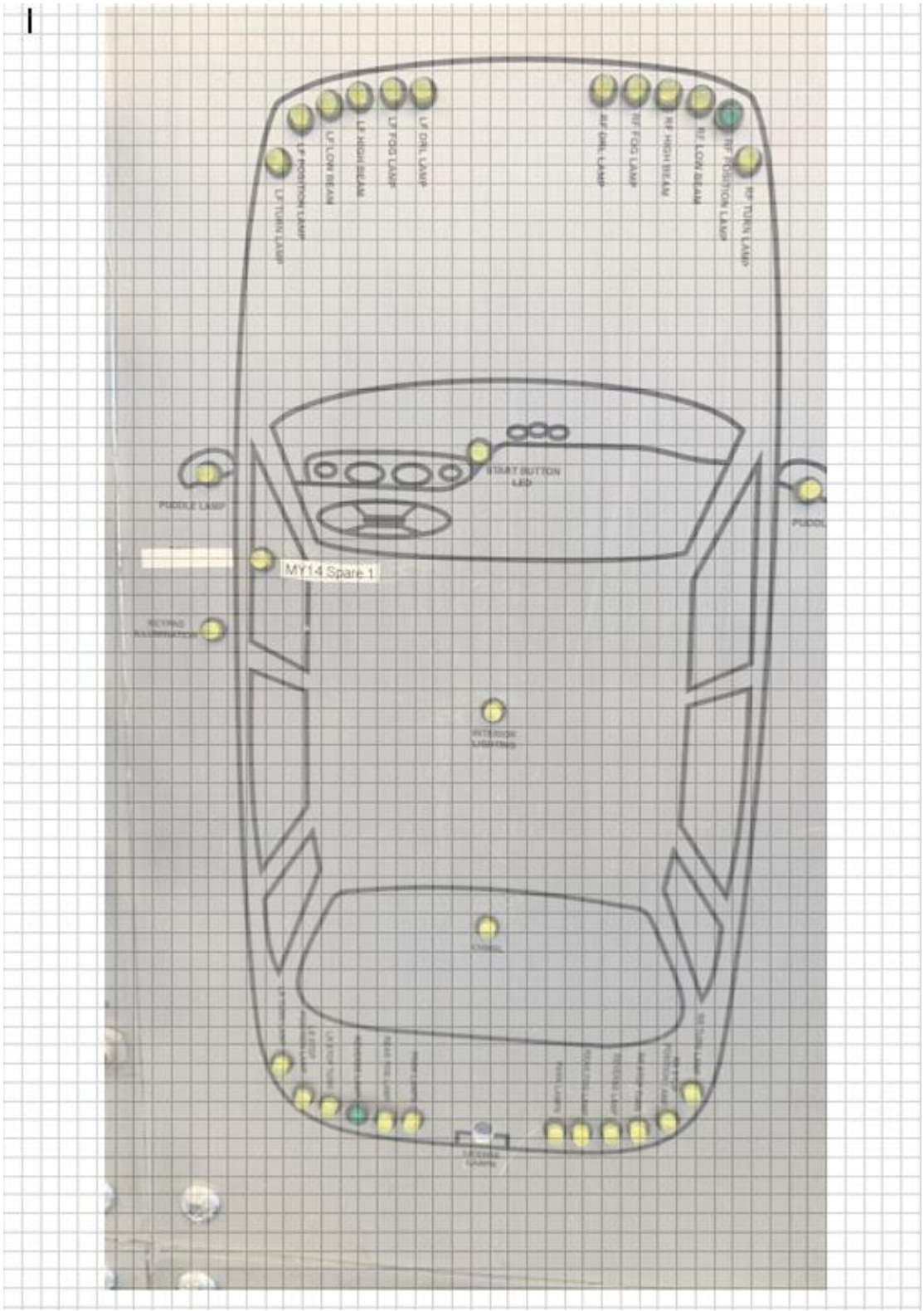
3.1 High Level Requirements

1. This section defines all high-level requirements for LabVIEW code and some BCM DV code. These will be implemented in 2 phases -initial and final.
2. Thus document shall be the master requirements document for all LabVIEW and previous documentation, or discussions is obsolete from now on.
3. LabVIEW Requests/Responses shall utilize the existing CAN communication links and Diagnostics CAN format messages. Preferred method is via NI-XNET CAN module. Use of Vector CAN, can be used initially.
4. Final method of control shall be without LabVIEW GUI but via a browser using JavaScript and DLLs for NI modules.

5. The Cybersecurity aspects of BCM will be the same as that implemented by the Application. This includes key learning for RKE, OTA and other Diagnostic commands. All of these need to be implemented in final version of code using browser.

3.2 High Level GUI

For the initial phase, we need to construct this GUI in Grids and as a Vector rendering such that most grids are clickable. We will add/modify as we develop.



BCM I/O Connections to NI cDAQ

Each BCM Input is carefully matched and connected with a corresponding NI module Output and Vice versa. Similarly BCM's Analog Inputs are also matched with NI's Analog outputs. An initial I/O selection is given below. More up to date details area at: DG-061502 > Project Documents > 110 Engineering > 040 Software Engineering > 120 SW Tools > 040 Special Tools > SISTER wire mapping.xlsx

BCM I/O type	BCM I/O count (per schematic)	BCM I/O count after allocation	NI I/O type	NI I/O count	Remaining NI (BCM) I/O	NI I/O Module (#qty)	Notes
AIH (Pulldown in DUT)	7	4	AO	4	0	NI 9269 (#1)	
AIL (Pullup in DUT)	10	4	AO	4	0	NI 9269 (#2)	
AO	0	0	AI	8	8	NI 9221 (#1)	
DIB (Pullup or down in DUT)	2	0			0	NI 9476 (#1)	
DIH (Pulldown in DUT)	6	11	DO-Source	32	21	NI 9476 (#1)	
DIL (Pullup in DUT)	32	42	DO-Sink	64	22	NI 9477 (#1,#2a)	
DIL (Pullup in DUT (Timer))	2	0	DO-Sink	0	0	NI 9477 (#2b)	
DOH (High Side)	59	63	DI-Sink	64	1	NI 9425 (#1,2a)	
DOL (Low Side)	17	24	DI-Source	32	8	NI 9426 (#1a)	
DOB (Push-Pull)	7	0	DI-Sink + Source	1	1	NI 9425 (#2b), NI 9426 (#1b)	
LIN	16	16	CAN + LIN	1	(15)	NI 9860 (#1a)	Using Vector CANcases.
K-Line (UART)	1	1		0	(1)		Using Vector CANcases.
CAN FD	1	1	CAN + LIN	1	0	NI 9860 (#1b)	
Ethernet	1	1		0	(1)		Use PC's ethernet port.
Antennas (DO to Z load)	14	14		0	(14)		Use real Antennas.
Power In (Vbatt/12V)	4	4			(4)		Use SW switcheable Power supply.
Power Out (VCC/5V)	1	1			(1)		Use SW switcheable Power supply.
GND	5	5		0	(5)		GND monitoring not in scope.
JTAG (DEBUG)	14	14		0	(14)		Use iSystem Debugger.

3.2.1 CAN diagnostics services and message ID

Wherever possible, the intention is to control all the I/O using Application DIDs rather than DV DIDs, though this involves more configs and control. All the DV CAN communications shall use the CAN Message Identifiers as specified in section 3.2.2 Generic UDS command message structure (more than 8 bytes). The CDD is the binding document for all IDs, baud rates and diagnostic info. DV SW Component uses all the same UDS service IDs (SID) which are found in CDD.

SID #	Function	Description
$10	Session Change	This service is used to switch to Extended Diagnostic Session along with Security unlock ($27), to enter any Diagnostic mode including DV diagnostic commands.
$11	Reset	This service is used to reset the BCM in Hard Reset Mode. Tester has to start afresh with Session change, Security unlock etc after a reset.
$14	Clear DTCs	This service must be used to clear all DTCs *before* and *after* the DV test.
$19	Read DTCs	The Read Diagnostic Trouble Codes by Status can be used if any DTC codes are set during DV test.
$22	Read DIDs	This service is used for reading software revision, software part number etc using DIDs. Refer to CDD for a list of DIDs that can be read.
$27	Security	This service is used unlocking the BCM for accessing I/O via Diagnostic commands. This will get more complex as we get closer to production such as getting the keys from Ford Server.
$2E	Write DIDs	This service is used for writing configurations such as Antenna thresholds using DIDs. Refer to CDD for a list of DIDs that can be written.
$2F	I/O Control	This service is used for using changing outputs, monitoring inputs for change etc. Refer to CDD for a list of DIDs that can be controlled. Also see Appendix H for DV specific DIDs.
$31	Routine Control	This service is used for complex controls, monitoring inputs for change etc. Refer to CDD for a list of DIDs that can be controlled. Also see Appendix H for DV specific DIDs.
$3E	Tester Present	A frequent tester present message (every <5s) keeps the BCM in Diagnostic (and DV) mode. Alternatively, any Diagnostic command within this interval will also keep the BCM in Diagnostic mode.

3.2.2 Generic UDS command message structure (more than 8 bytes)

Byte #	Request (0x726 for BCM) → (Tester to BCM)	Data/ range (Hex)	← Response (0x72E for BCM) (BCM to Tester)	Data/range (Hex)
1	UDS Protocol Control Info (PCI) # of Bytes Sent by Tester (ex. XYZ = 0x123 = 291 bytes, excluding these 2 bytes)	1X	Same as Request (These Request/Response bytes 1,2 are visible in CANalyzer, not CANoe)	1X
2		YZ		YZ
3	UDS Service ID (SID) (ex. 2F = Write DID command)	2F	Positive response to SID per UDS (Request + 0x40)	6F
4	DID number (See Appendix for list) (ex. 0xFE01 = Digital Inputs)	FE	Same as Request	FE
5		01		01
6	UDS Command for Short Term Adjustment (inputOutputControlParameter (CSR_IOCP))	03	Same as Request (last byte of mandatory UDS protocol)	03
7	Beginning of Payload Request. [1]	0-FF	Beginning of Payload Response. [1]	0-FF
8	Payload Request	0-FF	Payload Response	0-FF
9	Payload Request	0-FF	Payload Response	0-FF
XYZ-1	Payload Request	0-FF	Payload Response	0-FF
XYZ	Payload Request	0-FF	Payload Response	0-FF
XYZ+1	Payload Request	0-FF	Payload Response	0-FF
XYZ+2	Payload Request	0-FF	Payload Response	0-FF

[1] Beginning in this color, is the Data [0] passed to BCM Application by Vector layer (ex. DataServices_DID_FE0A_External_Memory_Integrity_Test_ReadData(…)). That is, subtract 7 from here onwards to access Payload.

3.2.3 Generic UDS command message structure (8 bytes or less)

Byte #	Request (0x726 for BCM) → (Tester to BCM)	Data/ range (Hex)	← Response (0x72E for BCM) (BCM to Tester)	Data/ range (Hex)
1	# of Bytes Sent by Tester (ex. 0Z= 0x07 (max) bytes, excluding this byte) (UDS Protocol Control Info / PCI)	0Z	Same as Request (This Request/Response byte 1 is visible in CANalyzer, not CANoe)	0Z
2	UDS Service ID (SID) (ex. 2F = Write DID command)	2F	Positive response to SID per UDS (Request + 0x40)	$6F
3	DID number (See Appendix for list) (ex. 0xFE01 = Digital Inputs)	$FE	Same as Request	$FE
4		$01		$01
5	UDS Command for Short Term Adjustment (inputOutputControlParameter (CSR_IOCP))	03	Same as Request (last byte of mandatory UDS protocol)	03
0Z-1	Payload Request	0-FF	Payload Response	0-FF
0Z	Payload Request	0-FF	Payload Response	0-FF
0Z+1	Payload Request	0-FF	Payload Response	0-FF

3.2.4 Positive UDS command reply:

All commands area acknowledged with a positive or negative response. Results (pass / fail results etc.) are usually sent after a second (stop) command.

3.2.5 Negative UDS command reply

Negative responses are usually due to incompatible CDD versions or commanding before security unlock. Out of sequence commands such as Stop before Start may also generate negative responses.

3.2.6 Generic Negative UDS response codes and explanations

These codes are listed in Appendix/

Appendix A **List Negative Response Codes (NRC) and meaning**

Table A.1 defines all negative response codes used within this standard. Each diagnostic service specifies applicable negative response codes. The diagnostic service implementation in the server may also utilize additional and applicable negative response codes specified in this as defined by the vehicle manufacturer. The negative response code range 0x00 – 0xFF is divided into three ranges:

— **0x00:** positiveResponse parameter value for server internal implementation,

— **0x01 to 0x7F:** communication related negative response codes,

— **0x80 to 0xFF:** negative response codes for specific conditions that are not correct at the point in time the request is received by the server. These response codes may be utilized whenever response code 0x22 (conditionsNotCorrect) is listed as valid in order to report more specifically why the requested action can not be taken.

Table A.1 — Negative Response Code (NRC) definition and values

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x00	**positiveResponse** This NRC shall not be used in a negative response message. This positiveResponse parameter value is reserved for server internal implementation. Refer to 7.5.5.	PR
0x01 – 0x0F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x10	**generalReject** This NRC indicates that the requested action has been rejected by the server. The generalReject response code shall only be implemented in the server if none of the negative response codes defined in this document meet the needs of the implementation. At no means shall this NRC be a general replacement for the response codes defined in this document.	GR
0x11	**serviceNotSupported** This NRC indicates that the requested action will not be taken because the server does not support the requested service. The server shall send this NRC in case the client has sent a request message with a service identifier which is unknown, not supported by the server, or is specified as a response service identifier. Therefore this negative response code is not shown in the list of negative response codes to be supported for a diagnostic service, because this negative response code is not applicable for supported services.	SNS
0x12	**sub-functionNotSupported** This NRC indicates that the requested action will not be taken because the server does not support the service specific parameters of the request message. The server shall send this NRC in case the client has sent a request message with a known and supported service identifier but with "sub-function“ which is either unknown or not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified service or the format of the parameters do not match the prescribed format for the specified service.	IMLOIF
0x14	**responseTooLong** This NRC shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer. This could occur if the response message exceeds the maximum size allowed by the underlying transport protocol or if the response message exceeds the server buffer size allocated for that purpose. EXAMPLE This problem may occur when several DIDs at a time are requested and the combination of all DIDs in the response exceeds the limit of the underlying transport protocol.	RTL
0x15 – 0x20	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x21	**busyRepeatRequest** This NRC indicates that the server is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical	BRR

Byte value	Negative Response Code (NRC) definition	Mnemonic
	request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents. EXAMPLE In a multi-client environment the diagnostic request of one client might be blocked temporarily by a NRC 0x21 while a different client finishes a diagnostic task. If the server is able to perform the diagnostic task but needs additional time to finish the task and prepare the response, the NRC 0x78 shall be used instead of NRC 0x21. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met.	CNC
0x23	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x24	**requestSequenceError** This NRC indicates that the requested action will not be taken because the server expects a different sequence of request messages or message as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order. EXAMPLE A successful SecurityAccess service specifies a sequence of requestSeed and sendKey as sub-functions in the request messages. If the sequence is sent different by the client the server shall send a negative response message with the negative response code 0x24 requestSequenceError	RSE
0x25	**noResponseFromSubnetComponent** This NRC indicates that the server has received the request but the requested action could not be performed by the server as a subnet component which is necessary to supply the requested information did not respond within the specified time. The noResponseFromSubnetComponent negative response shall be implemented by gateways in electronic systems which contain electronic subnet components and which do not directly respond to the client's request. The gateway may receive the request for the subnet component and then request the necessary information from the subnet component. If the subnet component fails to respond, the server shall use this negative response to inform the client about the failure of the subnet component. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	NRFSC
0x26	**FailurePreventsExecutionOfRequestedAction** This NRC indicates that the requested action will not be taken because a failure condition, identified by a DTC (with at least one DTC status bit for TestFailed, Pending, Confirmed or TestFailedSinceLastClear set to 1), has occurred and that this failure condition prevents the server from performing the requested action. This NRC can, for example, direct the technician to read DTCs in order to identify and fix the problem. NOTE This implies that diagnostic services used to access DTCs shall not implement this NRC as an external test tool may check for the above NRC and automatically request DTCs whenever the above NRC has been received. This NRC is in general supported by each diagnostic service (except the services mentioned above), as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	FPEORA
0x27 - 0x30	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x31	**requestOutOfRange** This NRC indicates that the requested action will not be taken because the server has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a dataIdentifier/routineIdentifer that is not supported or not supported in active session. This NRC shall be implemented for all services, which allow the client to read data, write data or adjust functions by data in the server.	ROOR
0x32	**ISOSAEReserved**	ISOSAERESRVD

Byte value	Negative Response Code (NRC) definition	Mnemonic
	This range of values is reserved by this document for future definition.	
0x33	**securityAccessDenied** This NRC indicates that the requested action will not be taken because the server's security strategy has not been satisfied by the client. The server shall send this NRC if one of the following cases occur: — the test conditions of the server are not met, — the required message sequence e.g. DiagnosticSessionControl, securityAccess is not met, — the client has sent a request message which requires an unlocked server. Beside the mandatory use of this negative response code as specified in the applicable services within this standard, this negative response code can also be used for any case where security is required and is not yet granted to perform the required service.	SAD
0x34	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x35	**invalidKey** This NRC indicates that the server has not given security access because the key sent by the client did not match with the key in the server's memory. This counts as an attempt to gain security. The server shall remain locked and increment its internal securityAccessFailed counter.	IK
0x36	**exceedNumberOfAttempts** This NRC indicates that the requested action will not be taken because the client has unsuccessfully attempted to gain security access more times than the server's security strategy will allow.	ENOA
0x37	**requiredTimeDelayNotExpired** This NRC indicates that the requested action will not be taken because the client's latest attempt to gain security access was initiated before the server's required timeout period had elapsed.	RTDNE
0x38 – 0x4F	**reservedByExtendedDataLinkSecurityDocument** This range of values is reserved by extended data link security.	RBEDLSD
0x50 – 0x6F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x70	**uploadDownloadNotAccepted** This NRC indicates that an attempt to upload/download to a server's memory cannot be accomplished due to some fault conditions.	UDNA
0x71	**transferDataSuspended** This NRC indicates that a data transfer operation was halted due to some fault. The active transferData sequence shall be aborted.	TDS
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory).	GPF
0x73	**wrongBlockSequenceCounter** This NRC indicates that the server detected an error in the sequence of blockSequenceCounter values. Note that the repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server.	WBSC
0x74 – 0x77	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x78	**requestCorrectlyReceived-ResponsePending** This NRC indicates that the request message was received correctly, and that all parameters in the request message were valid, but the action to be performed is not yet completed and the server is not yet ready to receive another request. As soon as the requested service has been completed, the server shall send a positive response message or negative response message with a response code different from this. The negative response message with this NRC may be repeated by the server until the requested service is completed and the final response message is sent. This NRC might impact the application layer timing parameter values. The detailed specification shall be included in the data link specific implementation document. This NRC shall only be used in a negative response message if the server will not be able to receive further request messages from the client while completing the requested diagnostic service. When this NRC is used, the server shall always send a final response (positive or negative) independent of the suppressPosRspMsgIndicationBit value or the suppress requirement for responses with NRCs SNS, SFNS, SNSIAS, SFNSIAS	RCRRP

Byte value	Negative Response Code (NRC) definition	Mnemonic
	and ROOR on functionally addressed requests. A typical example where this NRC may be used is when the client has sent a request message, which includes data to be programmed or erased in flash memory of the server. If the programming/erasing routine (usually executed out of RAM) is not able to support serial communication while writing to the flash memory the server shall send a negative response message with this response code. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	
0x79 – 0x7D	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x7E	**sub-functionNotSupportedInActiveSession** This NRC indicates that the requested action will not be taken because the server does not support the requested sub-function in the session currently active. This NRC shall only be used when the requested sub-function is known to be supported in another session, otherwise response code SFNS (sub-functionNotSupported) shall be used (e.g., servers executing the boot software generally do not know which subfunctions are supported in the application (and vice versa) and therefore may need to respond with NRC 0x12 instead). This NRC shall be supported by each diagnostic service with a sub-function parameter, if not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	SFNSIAS
0x7F	**serviceNotSupportedInActiveSession** This NRC indicates that the requested action will not be taken because the server does not support the requested service in the session currently active. This NRC shall only be used when the requested service is known to be supported in another session, otherwise response code SNS (serviceNotSupported) shall be used (e.g., servers executing the boot software generally do not know which services are supported in the application (and vice versa) and therefore may need to respond with NRC 0x11 instead). This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	SNSIAS
0x80	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x81	**rpmTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is above a preprogrammed maximum threshold).	RPMTH
0x82	**rpmTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is below a preprogrammed minimum threshold).	RPMTL
0x83	**engineIsRunning** This NRC is required for those actuator tests which cannot be actuated while the Engine is running. This is different from RPM too high negative response, and needs to be allowed.	EIR
0x84	**engineIsNotRunning** This NRC is required for those actuator tests which cannot be actuated unless the Engine is running. This is different from RPM too low negative response, and needs to be allowed.	EINR
0x85	**engineRunTimeTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for engine run time is not met (current engine run time is below a pre-programmed limit).	ERTTL
0x86	**temperatureTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is above a pre-programmed maximum threshold).	TEMPTH
0x87	**temperatureTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is below a	TEMPTL

Byte value	Negative Response Code (NRC) definition	Mnemonic
	pre-programmed minimum threshold).	
0x88	**vehicleSpeedTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is above a preprogrammed maximum threshold).	VSTH
0x89	**vehicleSpeedTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is below a preprogrammed minimum threshold).	VSTL
0x8A	**throttle/PedalTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current TP/APP is above a pre-programmed maximum threshold).	TPTH
0x8B	**throttle/PedalTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current TP/APP is below a pre-programmed minimum threshold).	TPTL
0x8C	**transmissionRangeNotInNeutral** This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in neutral is not met (current transmission range is not in neutral).	TRNIN
0x8D	**transmissionRangeNotInGear** This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in gear is not met (current transmission range is not in gear).	TRNIG
0x8E	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x8F	**brakeSwitch(es)NotClosed (Brake Pedal not pressed or not applied)** This NRC indicates that for safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.	BSNC
0x90	**shifterLeverNotInPark** This NRC indicates that for safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.	SLNIP
0x91	**torqueConverterClutchLocked** This NRC indicates that the requested action will not be taken because the server prerequisite condition for torque converter clutch is not met (current TCC status above a pre-programmed limit or locked).	TCCL
0x92	**voltageTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is above a pre-programmed maximum threshold).	VTH
0x93	**voltageTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is below a pre-programmed maximum threshold).	VTL
0x94 – 0xEF	**reservedForSpecificConditionsNotCorrect** This range of values is reserved by this document for future definition.	RFSCNC
0xF0 – 0xFE	**vehicleManufacturerSpecificConditionsNotCorrect** This range of values is reserved for vehicle manufacturer specific condition not correct scenarios.	VMSCNC
0xFF	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD