Research & Vehicle Technology
"Infotainment Systems Product Development"

Feature – Button Strategy

APIM Infotainment Subsystem Part Specific Specification (SPSS)

Version 1.6
UNCONTROLLED COPY IF PRINTED

Version Date: June 30, 2020

FORD CONFIDENTIAL

Revision History

Date	Version	Notes	
May 30, 2013	1.0	**Initial Release**	
October 15, 2013	1.1	**Updated Release**	
	BUTTON-GREQ-39750-3-BCP message structure usage		Updated button example
December 8, 2014	1.2	**Updated Release**	**Initial release of LIN BUTTON strategy**
April 24, 2015	1.3		
	BUTTON-SR-REQ-096736/C-LongEvent		<jmyslin2 / hzubert> changed long press from 2.0 to 1.5 seconds
	BUTTONv3-CLD-REQ-132978/B-Button Input Client (Button Transmitter) - I2C		<jmyslin2 / hzubert> added I2C over LVDS button strategy to the Button SPSS
March 8, 2016	1.4		
	BUTTON-SR-REQ-096742/C-Tx SetRotarySteps (CAN)+		<Stefan Berg_Jason Myslinski> Updated to add a configuration item if necessary to support setRotaryTune for fast browse/accelerated tune
February 2, 2018	1.5		
	BUTTON-SR-REQ-014704/C-Cancelling RBAP (TcSE ROIN-39789-1)+		<jmyslin2> working level before release
	BUTTON-SR-REQ-014704/D-Cancelling RBAP (TcSE ROIN-39789-1)+		<jmyslin2> working level before release
	BUTTON-SR-REQ-014704/E-Cancelling RBAP (TcSE ROIN-39789-1)		<jmyslin2> added volume as an example, referenced volume requirement for cancelling volume button RBAP
	MD-REQ-275443/A-BCP_Button_Press		<jmyslin2> Put interface table description in MD form. Not requirement content change and only a clarification and formatting update
	MD-REQ-275444/A-SetVolume+		<jmyslin2> Put interface table description in MD form. Not requirement content change and only a clarification and formatting update
	MD-REQ-275444/B-SetVolume		<jmyslin2> Updated signal description
	MD-REQ-275445/A-SetPointVolume+		<jmyslin2> Put interface table description in MD form. Not requirement content change and only a clarification and formatting update
	MD-REQ-275445/B-SetPointVolume		<jmyslin2> updated to include comment regarding SetVol_Level
	MD-REQ-275446/A-SetRotarySteps		<jmyslin2> Updated signal description
	BUTTON-SR-REQ-293306/A-Button Press reaction time (LIN)		<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN effected it on the CAN implementation guide part so had to create a duplicate requirement
	BUTTON-TMR-REQ-293307/A-T_button_unstuck (LIN)		<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
	BUTTON-SR-REQ-293490/A-Receivers of Held Button Presses (LIN)		<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
	BUTTON-SR-REQ-293493/A-Button Presses with no Held function or Combination with another button press (LIN)		<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
	BUTTON-SR-REQ-293491/A-Buttons with Held Function (LIN)		<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version

	for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293492/A-Receivers of combination type button presses (LIN)	<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293494/A-Combination Type button press operation (LIN)	<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293495/A-Cancelling RBAP (LIN)	<jmyslin2> added volume as an example. Made the requirement LIN specific since impacts the implementation guide for CAN.
BUTTON-SR-REQ-293496/A-Cancelling RBAP when change to Standby (LIN)	<jmyslin2> created a LIN specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the LIN section and in the implementation guide changing it on LIN affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293497/A-Button Press reaction time (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTONv3-CLD-REQ-153566/B-Button Input Server (Button Receiver) - I2C	<jmyslin2> updated to include I2C specific duplicate requirements
STR-496896/A-Button Type Definition (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
STR-496899/A-Button Activation - Receiver Timing Requirements (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293502/A-Receiver Button Activation Process (RBAP) Timing (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293503/A-Receivers of Button Presses follow RBAP (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293504/A-Receivers of Held Button Presses (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293505/A-Button Presses with no Held function or Combination with another button press (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293506/A-Buttons with Held Function (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293507/A-Buttons with Held Function (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
BUTTON-SR-REQ-293508/A-Combination Type button press operation (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement

	BUTTON-SR-REQ-293509/A-Cancelling RBAP (I2C)	<jmyslin2> created an I2C specific requirement. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
	BUTTON-SR-REQ-293510/A-Cancelling RBAP when change to Standby (I2C)	<jmyslin2> created an I2C specific requirement. THERE IS NO CONTENT CHANGE. This requirement used to use CAN version for the I2C section and in the implementation guide changing it on I2C affected it on the CAN implementation guide part so had to create a duplicate requirement
▨	▨	▨
June 30, 2020	**1.6**	
	STR-174872/B-Button Interface Requirements - LIN	<hzubert> added description for rolling counters, indicators, new illumination strategy and new error codes for new generartion ICPs
	BUTTONv2-IIR-REQ-096644/D-LIN BCP Button Press - Button Interface Requirements	<hzubert> added hint for interfaces having less button slots; corrected 0x255 to 0xFF
	BUTTON-IIR-REQ-372239/A-LIN - ApplicationInformation0_NewGeneration	<hzubert> initial release
	BUTTON-IIR-REQ-372241/A-LIN - ApplicationInformation1_NewGeneration	<hzubert> initial release
	BUTTON-IIR-REQ-372242/A-LIN - ApplicationInformation2_NewGeneration	<hzubert> initial release
	BUTTON-IIR-REQ-372243/A-LIN - ApplicationInformation3_NewGeneration	<hzubert> initial release
	BUTTON-IIR-REQ-107294/C-LIN - LINStatus	<hzubert> revised description to be more precise
	BUTTON-IIR-REQ-366705/A-LIN - ICP_RC	<hzubert> initial release
	BUTTON-IIR-REQ-366707/A-LIN - DSPL_RC	<hzubert> initial release
	BUTTON-IIR-REQ-366710/A-LIN - DSPLIlluIndPTS	<hzubert> initial release
	BUTTON-IIR-REQ-366711/A-LIN - DSPLIlluIndX	<hzubert> initial release
	BUTTON-IIR-REQ-366718/A-LIN - DSPLIlluBtnPTS	<hzubert> initial release
	BUTTON-IIR-REQ-366719/A-LIN - DSPLIlluBtnX	<hzubert> initial release
	BUTTON-IIR-REQ-366725/A-LIN - DSPLIlluVolKnob	<hzubert> initial release
	BUTTON-IIR-REQ-366726/A-LIN - DSPLIlluBtnChrome	<hzubert> initial release
	BUTTON-IIR-REQ-366729/A-LIN - DSPLIlluPWMIndicatorTargetPTS	<hzubert> initial release
	BUTTON-IIR-REQ-366730/A-LIN - DSPLIlluPWMBacklightTargetPTS	<hzubert> initial release
	BUTTON-IIR-REQ-366731/A-LIN - DSPLIlluPWMBacklightTarget	<hzubert> initial release
	BUTTON-IIR-REQ-366732/A-LIN - DSPLIlluPWMIndicatorTarget	<hzubert> initial release
	BUTTON-IIR-REQ-366733/A-LIN - DSPLIlluPWMTimerUp	<hzubert> initial release
	BUTTON-IIR-REQ-366736/A-LIN - DSPLIlluPWMTimerDown	<hzubert> initial release
	BUTTON-IIR-REQ-366738/A-LIN - DSPLIlluDimmingCurveType	<hzubert> initial release
	BUTTON-IIR-REQ-367005/A-LIN - IlluIndAllocX	<hzubert> initial release
	BUTTON-IIR-REQ-367006/A-LIN - IlluBtnAllocX	<hzubert> initial release
	BUTTON-IIR-REQ-367267/A-LIN - ICPIlluSmoothDimmSupp	<hzubert> initial release
	BUTTON-IIR-REQ-366723/A-LIN - ICP_CRC8	<hzubert> initial release
	BUTTON-IIR-REQ-366724/A-LIN - DSPL_CRC8	<hzubert> initial release
	STR-193581/B-General Requirements	<hzubert> added description for rolling counter
	BUTTON-SR-REQ-107308/B-LIN Scheduler turnaround frequency for standard interface	<hzubert> renamed to "LIN Scheduler turnaround frequency for standard interface"; no content change
	BUTTON-SR-REQ-116454/B-LIN Scheduler turnaround default values for standard interface	<hzubert> renamed to "LIN Scheduler turnaround default values for standard interface"; no content change
	BUTTON-SR-REQ-366885/A-Rolling Counter behavior	<hzubert> initial release
	BUTTON-SR-REQ-366886/A-Rolling Counter default value if CRC is supported	<hzubert> initial release
	BUTTON-SR-REQ-366706/A-Rolling Counter default value if CRC not supported	<hzubert> initial release
	BUTTON-SR-REQ-366887/A-Indicator position allocation	<hzubert> initial release

	BUTTON-SR-REQ-366888/A-Button position allocation	<hzubert> initial release
	BUTTON-SR-REQ-383144/A-Illumination configuration	<hzubert> initial release
	BUTTON-SR-REQ-383145/A-Setup illumination allocation	<hzubert> initial release
	BUTTON-SR-REQ-366708/A-CRC8 algorithm if CRC is supported	<hzubert> initial release
	BUTTON-SR-REQ-366709/A-CRC8 notation	<hzubert> initial release
	BUTTON-SR-REQ-366712/A-CRC8 initialization	<hzubert> initial release
	BUTTON-SR-REQ-366713/A-CRC8 computation	<hzubert> initial release
	BUTTON-SR-REQ-366714/A-CRC8 calculation	<hzubert> initial release
	BUTTON-SR-REQ-366715/A-CRC8 unused bits	<hzubert> initial release
	BUTTON-SR-REQ-366716/A-CRC8 unused bytes	<hzubert> initial release
	BUTTON-SR-REQ-366720/A-CRC8 data stream	<hzubert> initial release
	BUTTON-SR-REQ-366721/A-CRC8 algorithm if CRC is not supported	<hzubert> initial release
	BUTTON-SR-REQ-366722/A-decision of interface variant	<hzubert> initial release
	BUTTON-SR-REQ-372268/A-standard interface	<hzubert> initial release
	BUTTON-SR-REQ-372270/A-new generation interface	<hzubert> initial release
	BUTTONv2-SR-REQ-096645/C-LIN BCP message structure	<hzubert> added hint for interfaces having less button slots
	BUTTONv2-SR-REQ-096646/C-LIN BCP message structure usage	<hzubert> added hint for interfaces having less button slots
	BUTTONv2-SR-REQ-096647/C-LIN Multiple stuck buttons in BCP message structure	<hzubert> added hint for interfaces having less button slots

Table of Contents

1 Architectural Design - CAN Interface

1.1 Button Interface Requirements - CAN

1.1.1 MD-REQ-275443/A-BCP_Button_Press

Message Type: Status

Signals from the Button Input Client to the receiving modules indicating the button ID of the button pressed and Pressed/Not Pressed state of the button.

Logical Signal Name	Literals	Value	Description
ButtonANameID	See Input Translation Matrix	0x0 – 0xFF	
ButtonBNameID	See Input Translation Matrix	0x0 – 0xFF	
ButtonCNameID	See Input Translation Matrix	0x0 – 0xFF	
ButtonDNameID	See Input Translation Matrix	0x0 – 0xFF	
ButtonAActivationState	Not_Pressed	0x0	
	Pressed	0x1	
ButtonBActivationState	Not_Pressed	0x0	
	Pressed	0x1	
ButtonCActivationState	Not_Pressed	0x0	
	Pressed	0x1	
ButtonDActivationState	Not_Pressed	0x0	
	Pressed	0x1	

1.1.2 MD-REQ-275444/B-SetVolume

Message Type: Request

Signal for incrementing / decrementing volume (used with a rotary volume knob)

Logical Signal Name	Literals	Value	Description
SetVolume	-30 steps	0x0	
	-29 steps	0x1	
	-28 steps	0x2	
	…continued		
	-2 steps	0x1C	
	-1 step	0x1D	Decrements volume
	Not Pressed / Inactive	0x1E	
	+1 step	0x1F	Increments volume
	+2 steps	0x20	
	+3 steps	0x21	
	…continued		
	+30 steps	0x3C	

1.1.3 MD-REQ-275445/B-SetPointVolume

Message Type: Request

Signal for selecting a particular volume level

Logical Signal Name	Literals	Value	Description
SetPointVolume	Not Pressed / Inactive	0x0	Note: also called SetVol_Level in some SPSS features
	No Volume	0x1	
	Vol_Step1	0x2	
	Vol_Step2	0x3	
	Vol_Step3	0x4	
	Continued		
	Vol_Step30	0x1F	

1.1.4 MD-REQ-275446/A-SetRotarySteps

Message Type: Request

Signal for incrementing / decrementing Tune/Rotary Browsing (used with a rotary knob)

Logical Signal Name	Literals	Value	Description
SetRotarySteps	-7 steps	0x0	
	-6 steps	0x1	
	-5 steps	0x2	
	-4 steps	0x3	
	-3 steps	0x4	
	-2 steps	0x5	
	-1 step	0x6	
	Not Pressed / Inactive	0x7	
	+1 step	0x8	
	+2 steps	0x9	
	+3 steps	0xA	
	+4 steps	0xB	
	+5 steps	0xC	
	+6 steps	0xD	
	+7 steps	0xE	

1.2 BUTTON-FUN-REQ-014675/B-CAN BCP (Button Control Panel) Button Press Message Structure (TcSE ROIN-39745-1)

1.2.1 CAN - Infotainment Button Press / Rotary Knob Message Structure

1.2.1.1 BUTTON-SR-REQ-014676/B-BCP message structure (TcSE ROIN-39749-2)

Up to 4 infotainment button press / rotary knob events can be activated simultaneously by the BCP plus a volume setting using either of the signals SetVolume.Rq or SetPointVolume.Rq. The button press messages are event-periodic. The basic structure of these messages is shown below.

Button ID	Button Encoding
Button A Activation State	0x0 Not Pressed
	0x1 Pressed
	0x2 Reserved
	0x3 Reserved
	0x4 Reserved
	0x5 Reserved
Button B Activation State	0x0 Not Pressed
	0x1 Pressed
	0x2 Reserved
	0x3 Reserved
	0x4 Reserved
	0x5 Reserved
Button C Activation State	0x0 Not Pressed
	0x1 Pressed
	0x2 Reserved
	0x3 Reserved
	0x4 Reserved
	0x5 Reserved
Button D Activation State	0x0 Not Pressed
	0x1 Pressed
	0x2 Reserved
	0x3 Reserved
	0x4 Reserved
	0x5 Reserved

Infotainment Button Activation State Coding

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5		Byte 6		Byte 7	Byte 8
Button A Name ID	Button B Name ID	Button C Name ID	Button D Name ID	Button A Activation State	Button B Activation State	Button C Activation State	Button D Activation State	SetVolume	SetPointVolume

Infotainment Button / Rotary Knob Message Structure

1.2.1.2 BUTTON-SR-REQ-014677/A-BCP message structure usage (TcSE ROIN-39750-3)

When a button is activated it shall encode Button A first and if that position is already being used (ie pressed) shall move to the next Button position in the message. If all 4 buttons (A – D) are being used then any new button inputs shall be ignored until one of the 4 buttons are released.

Once a button press is active for a button ID (A – D) the BCP shall not move that same button press to another button ID location. For example if seek is being pressed and held and is assigned to 'Button C Name ID' it shall not change to 'Button A Name ID' while still being pressed.

Reference INPUT_SWITCH-GREQ-112565-1-Input Translation Matrix for Buttons A – D Name ID (bytes 1-4) Infotainment Button Activation mapping.

The default to set Button Activation States A – D (bytes 5 – 6) is 'Not Pressed' unless there is a button activation event. Note this message structure applies to the '360 Degree Rotary Knob' but does not apply to the 'Absolute Position Rotary Knob'.

Example:
1. Button Module powers up and bus awake
2. Button Module sending: Button A Name ID = Inactive AND Button Activation State = Not Pressed
3. User presses button X
4. Button Module sending: Button A Name ID = X AND Button Activation State = Pressed
5. User releases button X
6. Button Module sends: Button A Name ID = X AND Button Activation State = Not Pressed
7. Button Module continues to send (if periodic) Button A Name ID = X AND Button Activation State = Not Pressed until the next button press or sends Button A Name ID = Inactive AND Button Activation State = Not Pressed. Any new button press can change Button A Name ID from X to the new button value.

1.2.1.3 BUTTON-SR-REQ-014678/A-Multiple stuck buttons in BCP message structure (TcSE ROIN-39751-1)

If all 4 BCP buttons A – D encoded in the BCP_Button_Press message are determined to be 'stuck' (not including Volume signals – Byte 7 and 8) then one of the 4 button bytes (Button A – D) shall be released so that other buttons can be activated.

1.2.2 Volume Signals

1.2.2.1 BUTTON-SR-REQ-014679/A-Stuck volume buttons in BCP message structure (TcSE ROIN-39754-2)

The Volume buttons being stuck shall not prevent any other infotainment button from being used.

1.2.3 Climate Control Button Press

1.2.3.1 BUTTON-SR-REQ-014680/A-Climate stuck buttons in BCP message structure (TcSE ROIN-39757-2)

All BCP climate control buttons can be supported simultaneously such that if there is stuck Climate Control button(s) the other unstuck climate control buttons can still be activated.

Reference the applicable Climate Control documents for Climate Button Activation mapping.

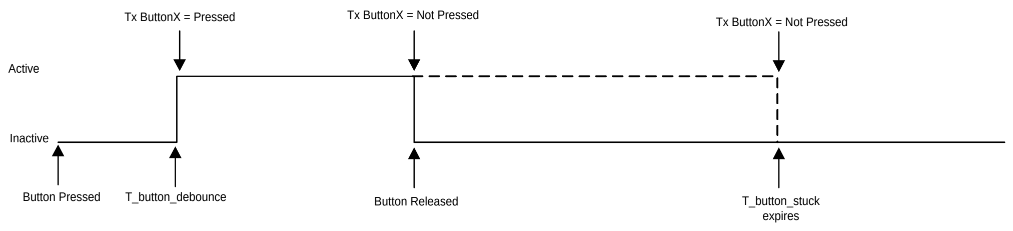
1.3 BUTTON-CLD-REQ-014681/B-Button Input Client (Button Transmitter) - CAN (TcSE ROIN-39759-1)

The following sections define the Button Activation Strategy from the Transmitters perspective.

1.3.1 Push Button Activation - Transmitter Timing Requirements

1.3.1.1 BUTTON-SR-REQ-014682/A-Transmitter Button Activation Process timing (TcSE ROIN-39761-2)

The TBAP timing figure below will always remain true for any button activation event. The exception to this rule is for "Rotary Knobs" which is covered in the next section.



Transmitter Button Activation Process (TBAP) Timing

1.3.1.2 BUTTON-TMR-REQ-014683/C-T_reaction_time (TcSE ROIN-39775-1)

Name	Description	Units	Range	Resolution	Default
T_reaction_time	The maximum transmitter reaction time from when a push button switch is closed until the push button message is put on the CAN bus. Note: use the default value	msec	0-1000	10	70

1.3.2 Push / Touch Button Activation - Transmitter Functional Requirements

1.3.2.1 BUTTON-SR-REQ-014684/C-Button Pressed / Not Pressed transimission (TcSE ROIN-39763-1)

Once a button is pressed and debounced on the transmitter the button message will be sent to the receiver (ex. MFD, Cluster…) with the button signal coding set to "Pressed" (or SetVolume/SetPointVolume/SetRotarySteps equal to press equivalent).

Once the button is released the transmitter will set the button coding as "Not Pressed".

1.3.2.2 BUTTON-SR-REQ-014685/C-Button Press reaction time (TcSE ROIN-39764-1)

The transmitter (ex.EFP, ICP, SWC, SDM) reaction time from when a push button switch is closed until the push button message is put on the bus shall not exceed T_reaction_time.

Note: this does not apply to the touch sense buttons. Reference applicable specifications for touch sense debounce requirements.

1.3.2.3 BUTTON-SR-REQ-014686/B-Transmitter Stuck Button (TcSE ROIN-39765-2)

When the transmitter determines a button to be stuck:
-- The transmitter will set a stuck button DTC as defined in the transmitter component specification and/or Infotainment Diagnostic Specification.
-- When a button is determined to be stuck the button encoding will be set to 'Not Pressed'.
-- The transmitter will keep the button encoding status as 'Not Pressed' as long as the button is stuck. Upon a new ignition cycle the button encoding shall remain as 'Not Pressed' until the button is determined to be operational.
-- Once a button becomes unstuck after previously being stuck then after remaining unstuck for T_button_unstuck the button shall become operational again.

Note: Reference applicable Climate Control specifications for additional CC Stuck Button requirements and exceptions.

1.3.2.4 BUTTON-TMR-REQ-014687/B-T_button_unstuck (TcSE ROIN-60370-1)

Name	Description	Units	Range	Resolution	Default
T_button_unstuck	Once a button is determined to be stuck then T_button_unstuck is the time the button has to be unstuck before the button can be operational again. Note: always use the default value	sec	0-100	1	10

1.3.3 Rotary Knob Transmitter Requirements

The BCP can support two types of rotary knobs, the '360 Degree Rotary Knob' and 'Absolute Position Rotary Knob'.

1.3.3.1 Transmitter - "360 Degree Rotary Knob"

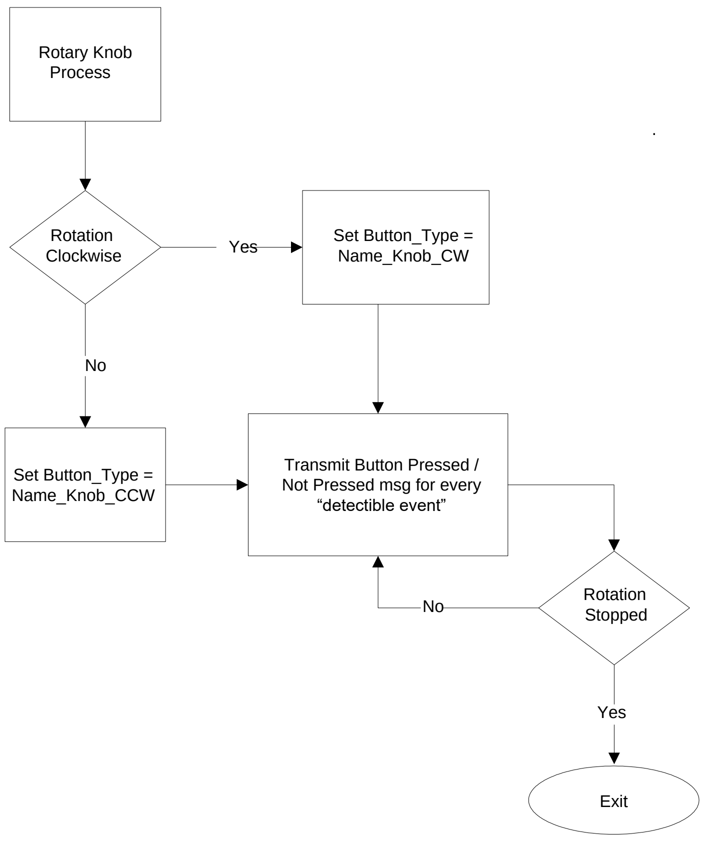
A "360 Degree Rotary Knob" typically sends a pulse train to a processing device (ex. CPU) whenever the knob is being rotated. The processing device will then use this information to determine in what direction (clock-wise vs. counter-clock-wise) the knob is being rotated and at what rate it is being rotated. A knob that utilizes physical detents for customer feedback may be designed such that as the knob is rotated each detected detent sends a pulse to the processing device. Therefore the number of detents rotated can be directly mapped to the number of pulses sent to the processing device. For example, if the knob is rotated through 10 detents then 10 separate pulses would get sent to the processing device.

Transmitter – "360 Degree Rotary Knob" Timing Requirements

Ideally, there should be a "one-to-one" mapping of one detected detent to one Info-CAN button press message being transmitted to a peripheral. The "one-to-one" mapping is difficult to achieve for a "Rotary Knob" which is rotated quickly. This difficulty is due to the time required to detect and process the detent while rotated, place the corresponding Info-CAN button press message in the transmit queue, transmit the message on the bus, retrieve the Info-CAN message from the receive queue, and process the message. Therefore, the rate at which detents are detected and button press messages are transmitted needs to be limited by a worst case time stack up for the entire transmit and receive process.

1.3.3.1.1 <u>BUTTON-SR-REQ-014691/A-360 Degree Rotary Knob (TcSE ROIN-39769-1)</u>

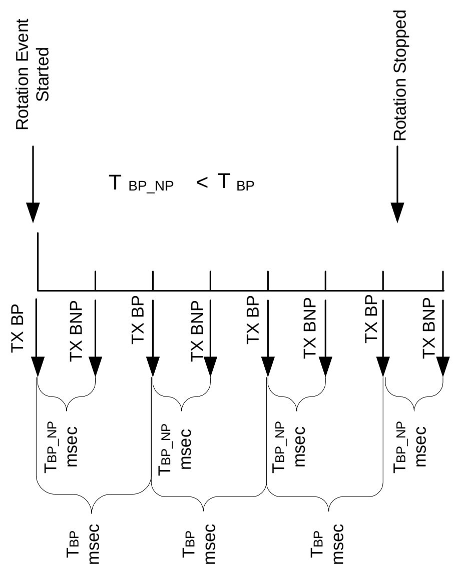
The figure below defines the functional requirements that a module shall follow for transmitting button press messages for a "360 Degree Rotary Knob".


Rotary Knob Process
Rotation Clockwise
Yes
Set Button_Type = Name_Knob_CW
No
Set Button_Type = Name_Knob_CCW
Transmit Button Pressed / Not Pressed msg for every "detectible event"
Rotation Stopped
No
Yes
Exit

Transmitter "360 Degree Rotary Knob"

1.3.3.1.2 <u>BUTTON-SR-REQ-014688/A-Rotary Knob Tx timing values (TcSE ROIN-39771-1)</u>

The figure below is an example of the timing involved with mapping a "detectible event" (i.e. detent) to a button_press message.



"360 Degree Rotary Knob" Timing Figure

For the 360 Degree Rotary Knob the T_{BP_NP} and T_{BP} timing values shall be met and be able to support the rotary knob turn process.

1.3.3.1.3 BUTTON-TMR-REQ-014689/C-Tbp_np (Time between button Press and Not Press) (TcSE ROIN-39772-3)

Name	Description	Units	Range	Resolution	Default
Tbp_np (Time between button Press and Not Press)	Nominal time allowed between button press and button not pressed messages on rotary knobs. Use the default value with a tolerance +/- 10%. Note: use the default value	msec	0-1000	10	20

1.3.3.1.4 BUTTON-TMR-REQ-014690/C-Tbp (T Button Press) (TcSE ROIN-39773-3)

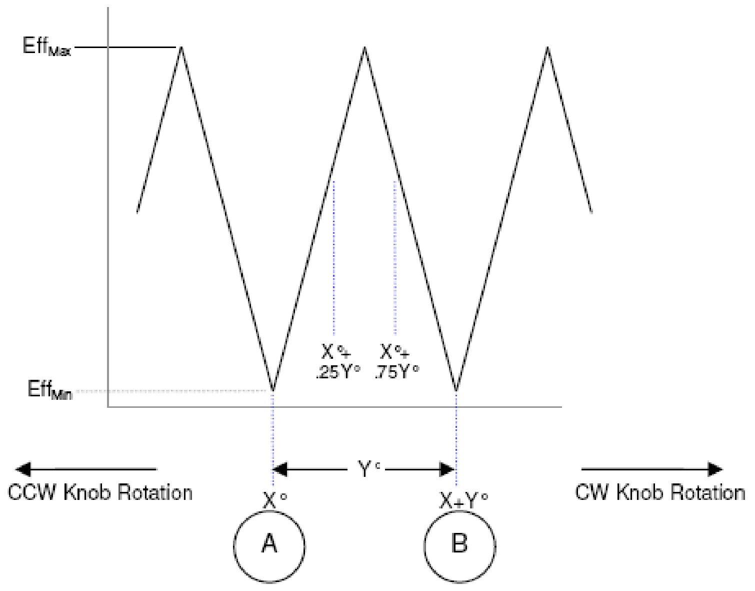
Name	Description	Units	Range	Resolution	Default
Tbp (T Button Press)	Nominal time allowed between button press sequences on rotary knobs. Use the default value with a tolerance +/- 10%. Note: use the default value	msec	0-1000	10	40

1.3.3.2 Transmitter - "Absolute Position Rotary Knob" Requirements

The "Absolute position Rotary Knob" will provide absolute position in degrees of clockwise rotation from the full counter clockwise knob mechanical stop position (using detent or non-detent move interval as calibrated & filtered in the EFP software per applications).

1.3.3.2.1 BUTTON-SR-REQ-014692/A-Absolute Position Rotary Knob (TcSE ROIN-39777-1)

An absolute position rotary knob is at rest at position "A" which corresponds to X° of rotation from the full CCW position. The Info-CAN message shall contain the angular information for X. This information shall continue to be transmitted, as the knob is rotated from position "A" to "B" until the angular rotation exceeds 75% of the angular rotation between detent positions A and B. Once the angular position exceeds the X°+ .75Y° +/- .05Y° (or 1°, whichever is greater) point the Info-CAN data shall transmit the angular information which corresponds to position B. The only valid angular data shall be that which corresponds to position A or B. The module transmitting this angular position shall not transmit Info-CAN data for intermediate positions. The strategy is maintained in a similar fashion for rotation in the opposite direction (i.e. CCW). If the knob is resting at position B the transmitting module shall transmit the total angular sum of X°+Y°. This information shall be transmitted until the angular position of the knob exceeds X° +.25 Y° +/- .05Y° (or 1°, whichever is greater).



Rotary Knob Rotation Multiplex Message Format

1.3.4 Volume Signals

1.3.4.1 <u>BUTTON-SR-REQ-014693/C-Tx SetVolume (TcSE ROIN-39753-3)</u>

The SetVolume signal shall be used to increment / decrement the volume level. Each volume level step encoding shall be treated as a press event in the remainder of this document.

After a press event the SetVolume signal shall set the 'Not Pressed' encoding before another press event can occur. Therefore a periodic SetVolume signal status update will not be treated as a new volume press event and will be ignored by the Volume Settings Server/Button Input Server (Button Receiver).

The Volume Button Input Client (Button Transmitter) or Volume Settings Client shall increment the SetVolume signal by 1 for every detected volume up step (clockwise direction if have volume knob).

The Volume Button Input Client (Button Transmitter) or Volume Settings Client shall decrement the SetVolume signal by 1 for every detected volume down step (counter-clockwise direction if have volume knob).

The Volume Button Input Client (Button Transmitter) shall send out the delta counts accumulated since the last SetVolume signal sent out on the infotainment network bus.

1.3.4.2 <u>BUTTON-SR-REQ-014694/A-Tx SetPointVolume (TcSE ROIN-60429-1)</u>

The SetPointVolume/MFD_SetVol_Level signal shall be used to set the volume to a pre-defined level. Each volume set point encoding shall be treated as a press event in the remainder of this document.

After a press event the SetPointVolume/MFD_SetVol_Level signal shall set the 'Not Pressed' encoding before another press event can occur. Therefore a periodic SetPointVolume signal status update will not change the volume.

1.3.5 setRotarySteps signal

1.3.5.1 <u>BUTTON-SR-REQ-096742/D-Tx SetRotarySteps (CAN)</u>

The setRotarySteps signal can be used for example to increment / decrement the for the tune function or for fast browse. Reference the applicable SPSS section for details. Each setRotarySteps step encoding shall be treated as a press event in the remainder of this document.

After a press event the setRotarySteps signal shall set the 'Not Pressed' encoding before another press event can occur. Therefore a periodic setRotarySteps signal status update will not be treated as a new setTune press event and will be ignored by the Button Input Server (Button Receiver).

The Button Input Client (Button Transmitter) shall increment the setRotarySteps signal by 1 for every detected rotary knob detent in the clockwise direction.

- For example if the Button Transmitter detects 3 detents in the clockwise direction could send +3 (fast turn) or three +1 button press/not press messages (slower turn) as long as no information is lost.

The Button Input Client (Button Transmitter) shall decrement the setRotarySteps signal by 1 for every detected rotary knob detent in the counter-clockwise direction.

- For example if the Button Transmitter detects 3 detents in the counter clockwise direction could send -3 (fast turn) or three -1 button press/not press messages (slower turn) as long as no information is lost.

The Button Input Client (Button Transmitter) shall send out the delta counts accumulated since the last setRotarySteps signal sent out on the bus.

Configuration of Button Transmitter for Rotary Tune knob on the Button ECP:
For Rotary Tune Knob if the fast tune/accelerated input feature is supported on the vehicle then the ECP will need to send setRotarySteps.

If fast tune/accelerated input feature is not supported on a vehicle (legacy vehicles) then the ECP will use the BCP button structure for tune Press / Not Pressed (example Button(A/B/C/D)NameId / Button(A/B/C/D)ActivationState).

If the ECP software needs to support both methods (ECP paired with modules that support Fast Tune and modules that do not) then a configuration parameter will need to be implemented on the ECP.

1.4 BUTTON-CLD-REQ-014695/C-Button Input Server (Button Receiver) - CAN (TcSE ROIN-39778-1)

The following sections define the Button Activation Strategy from the Receivers perspective.

1.4.1 Button Type Definition

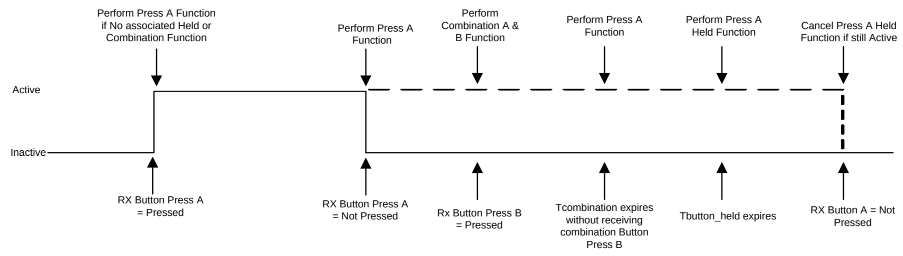
Singe Function – Single Function Buttons are buttons that can only map to a single outcome.

Combination – Combination Buttons are buttons that have a single outcome based on two simultaneous button presses.

1.4.2 Button Activation - Receiver Timing Requirements

1.4.2.1 *BUTTON-SR-REQ-014696/A-Receiver Button Activation Process (RBAP) Timing (TcSE ROIN-39781-2)*

All component receivers of button press information shall implement the Receiver Button Activation Process (RBAP) timing defined in the figure below.


Perform Press A Function if No associated Held or Combination Function
Perform Press A Function
Perform Combination A & B Function
Perform Press A Function
Perform Press A Held Function
Cancel Press A Held Function if still Active
Active
Inactive
RX Button Press A = Pressed
RX Button Press A = Not Pressed
Rx Button Press B = Pressed
Tcombination expires without receiving combination Button Press B
Tbutton_held expires
RX Button A = Not Pressed

Receiver Button Activation Process (RBAP) Timing

Exception: If a particular button supports press and hold function there may be times where a functional requirement may require the function to first be performed on the press (not wait for not pressed) and then take additional action when Tbutton_held expires and the buttons are determined to be held. This should only be performed if explicitly called out in a functional requirement otherwise the function shall be performed only on a Not Press or when Tbutton_held expires as shown in the RBAP above.

1.4.3 Button Activation - Receiver Functional Requirements

1.4.3.1 *BUTTON-SR-REQ-014697/A-Receivers of Button Presses follow RBAP (TcSE ROIN-39783-2)*

All receivers of button press information shall activate the RBAP upon receipt of the button press message.

Note:
If the button press network messages are event-periodic with the event messages being transmitted "OnChange", the receiver modules for a particular signal that had "Press" sent on event shall not treat subsequent periodic "Press" encodings as a new RBAP event.

Put another way after the Button Receiver receives a Press for a particular button it shall wait for a Not Press for the same button before acting upon another button "Press" of the same button. After a Not Press then the Press of the same button can be acted upon again initiating a new RBAP event.

1.4.3.2 *BUTTON-SR-REQ-014698/A-Button Receiver Sampling Rate (TcSE ROIN-52599-1)*

The sampling rate used by the receiver to read incoming button "Pressed" and "Not Pressed" messages shall be fast enough to read the multiple incoming messages (ex. could read multiple incoming rotary knob pressed / not pressed messages).

For rotary knob button presses the button receiver sampling rate to support reference requirements:
BUTTON-GREQ-39777-1-Absolute Position Rotary Knob and
BUTTON-GREQ-39772-2-Tbp_np (Time between button Press and Not Press) and
BUTTON-GREQ-39773-2-Tbp (T Button Press)

1.4.3.3 BUTTON-SR-REQ-014699/A-Receivers of Held Button Presses (TcSE ROIN-39784-1)

The receiver shall determine whether the specific button has an associated held function.

1.4.3.4 BUTTON-SR-REQ-014700/B-Button Presses with no Held function or Combination with another button press (TcSE ROIN-39785-1)

If the button does not have an associated held or combination type function, then the receiver shall perform the function associated with a button press immediately upon receipt of the button press message and the RBAP process shall be exited.

1.4.3.5 BUTTON-SR-REQ-014701/B-Buttons with Held Function (TcSE ROIN-39786-1)

If the button does have an associated held function, the receiver shall start a hold timer (Tbutton_held) and wait for button 'not pressed' information.

-- If a button 'not pressed' message is not received prior to the expiration of Tbutton_held, the receiver shall perform the associated held function for that button press.

-- If a button 'not pressed' message is received prior to the expiration of Tbutton_held, then the receiver shall perform the associated press function.

1.4.3.6 BUTTON-SR-REQ-014702/A-Receivers of combination type button presses (TcSE ROIN-39787-1)

The receiver shall determine whether the button press has an associated combination type button press.

1.4.3.7 BUTTON-SR-REQ-014703/B-Combination Type button press operation (TcSE ROIN-39788-1)

When the receiver detects a button press (A) that has an associated combination the receiver must wait Tcombination before executing the button press function associated with the single button press (A).

-- If a second button (B) is received prior to Tcombination expiring, the receiving module must now determine if this (B) is part of a valid combination with (A).
- If the combination is valid, then the resulting combination (A + B) can be performed.
- If the combination is invalid, then no combination function is performed and the button presses (A) & (B) can be processed independently if allowed by the receiving module.

-- If Tcombination expires, then button press (A) is now valid and the (A) function can be performed.

1.4.3.8 BUTTON-SR-REQ-014704/E-Cancelling RBAP (TcSE ROIN-39789-1)

The receiver shall cancel the RBAP (Receiver Button Activation Process) upon:

- Receipt of the button 'Not Pressed' message.
- If the receiver does not receive a button 'Not Pressed' message within T_RBAP_Timeout
  - Unless noted otherwise all buttons shall timeout at some point in case of a stuck button
  - T_RBAP_Timeout may vary depending on how the button is used for a particular feature. The T_RBAP_Timeout would be part of error handling for a particular feature unless a T_RBAP_Timeout value is explicitly noted in a feature SPSS.
    - For example: if a feature does not have a press and hold or combination button press associated with it then that button might time out quickly. Other features like Seek/Volume have press and hold features associated with them and would not time out for a longer period of time. Follow up with the Ford HMI or Ford feature team on what make sense for timeout values.

Reference:
- For cancelling Volume Button RBAP reference requirements "VOL-TMR_REQ-292290-T_Vol_RBAP_Timeout" and "VOL-REQ-292289-Volume Press and Hold Timeout".

1.4.3.9 BUTTON-SR-REQ-014705/A-Cancelling RBAP when change to Standby (TcSE ROIN-39790-2)

For Infotainment receivers any event that shall cause a transition from Functional mode to Standby mode shall cancel a RBAP unless noted otherwise. The receiver shall cancel the operation, if active, and perform the required actions to enter Standby State.

1.4.3.10 BUTTON-SR-REQ-014706/A-Steering Wheel Control Update Bit Button Presses (TcSE ROIN-39791-1)

When the Gateway module sends to the Infotainment components a Steering Wheel Control Button Press Update Bit indicating the message from the SWCM to the Gateway module is missing (ie unknown) as called out in "ERRMGNT-GREQ-149906-1-Infotainment Update Bit handling" then the infotainment components shall treat the button press as a "Not Press" until the update bits indicates that the data is fresh again.

1.4.3.11 BUTTON-SR-REQ-014707/B-Receivers of SetVolume Button presses (CAN) (TcSE ROIN-110929-1)

The Button Input Server (Volume Setting Server) shall increment/decrement the volume based on the delta count of the volume steps received since the last SetVolume signal update.

After the Button Input Server receives a press event (volume step) for the event-periodic SetVolume signal from the Button Input Client it shall wait for 'SetVolume' to equal 'Not Pressed' before it can react to another SetVolume press event. This is to avoid acting on a periodic SetVolume status update and only act on SetVolume Press events.

1.4.3.12 BUTTON-SR-REQ-096761/B-Receivers of SetRotarySteps Button presses (CAN)

The Button Input Server (setRotarySteps Server) shall increment/decrement the tune/fast browse (as defined in other SPSS specifications) based on the delta count of the setRotarySteps steps received since the last SetRotarySteps signal update.

After the Button Input Server receives a press event (setRotarySteps step) for the event-periodic SetRotarySteps signal from the Button Input Client it shall wait for 'SetRotarySteps' to equal 'Not Pressed' before it can react to another SetRotarySteps press event. This is to avoid acting on a periodic SetRotarySteps status update and only act on SetRotarySteps Press events.

1.4.3.13 BUTTON-SR-REQ-014708/A-Receivers of SetPointVolume and MFD_SetVol_Level Button Presses (TcSE ROIN-110930-1)

The Button Input Server (Volume Setting Server) shall sets its volume based on the SetPointVolume signal value it receives from the Button Input Client.

After the Button Input Server (Volume Settings Server) receives a press event (set point volume level) for the event-periodic SetPointVolume signal (also applies to MFD_SetVol_Level in Volume section of SPSS) from the Button Input Client it shall wait for 'SetPointVolume' to equal 'Not Pressed' before it can react to another SetPointVolume press event. This is to avoid acting on a periodic SetPointVolume status update and only act on SetPointVolume Press events.

1.4.3.14 BUTTON-SR-REQ-014709/C-ICI Button Coding State Normalization (TcSE ROIN-194455-1)

The implementation of the Input Control Interface (ICI) method for button coding state (ICI_Coding_BtnID_X) utilizes seven unique states.

ICI_Coding_BtnID_X	State
0x0	Inactive
0x1	Active
0x2	ShortEvent
0x3	ShortElapsed
0x4	LongEvent
0x5	Stuck
0xF	Idle

Due to system deployment a receiver may be assigned to the ICI interface but will need to normalize the button states into the updated normalized interface. The following is the normalization table from the seven-state ICI to the normalized button state.

7-State ICI ICI_Coding_BtnID_X	Normalized Button State
(0x0) Inactive	Not_Pressed
(0x1) Active	Pressed
(0x2) ShortEvent	Pressed
(0x3) ShortElapsed	Pressed
(0x4) LongEvent	Pressed
(0x5) Stuck	Not_Pressed
(0xF) Idle	Not_Pressed

1.4.3.15 BUTTON-REQ-014710/A-Missing Button Press Message (TcSE ROIN-281257-1)

When Ignition_Status does not equal Run (ex Accessory, OFF) and the button press message is missing from the bus for more than 5 seconds then the Button Input Server (Button Receiver) shall assume that the buttons are set to Not Pressed. When ignition_Status = Run and the button press message is missing for more than 5 seconds then the Button Input Server (Button Receiver) shall follow the error handling as called out in the IDS (infotainment diagnostic specificati on).

2 Architectural Design - LIN Interface

2.1 Button Interface Requirements - LIN

2.1.1 BUTTONv2-IIR-REQ-096644/D-LIN BCP Button Press - Button Interface Requirements

BCP_Button_Press (ICPBtnStateRotary, ICPBtnState…)	Method from the Button Input Client to the receiving modules.	ButtonANameID (ICPBtnID_A) 0x0 - 0xFF see Input Translation Matrix ButtonAActivationState (ICPBtnCoding_A) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 ShortEvent / Pressed 0x3 ShortElapsed / Pressed 0x4 LongEvent / Pressed 0x5 Stuck 0xF Idle / Not_Pressed ButtonBNameID (ICPBtnID_B) 0x0 – 0xFF see Input Translation Matrix ButtonBActivationState (ICPBtnCoding_B) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 ShortEvent / Pressed 0x3 ShortElapsed / Pressed 0x4 LongEvent / Pressed 0x5 Stuck 0xF Idle / Not_Pressed ButtonCNameID (ICPBtnID_C) 0x0 – 0xFF see Input Translation Matrix ButtonCActivationState (ICPBtnCoding_C) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 ShortEvent / Pressed 0x3 ShortElapsed / Pressed 0x4 LongEvent / Pressed 0x5 Stuck 0xF Idle / Not_Pressed ButtonDNameID (ICPBtnID_D) 0x0 – 0xFF see Input Translation Matrix ButtonDActivationState (ICPBtnCoding_D) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 ShortEvent / Pressed 0x3 ShortElapsed / Pressed 0x4 LongEvent / Pressed 0x5 Stuck 0xF Idle / Not_Pressed

Hint: maybe new interfaces like e.g. new generation interface, contains less button slots (including related ID and coding slots) e.g. 2. In this case only first number of slots e.g. A and B shall be considered).

2.1.2 BUTTON-IIR-REQ-107291/A-LIN setVolume - Button Interface Requirements

SetVolume (ICPVolumeCmd)	Method for incrementing / decrementing Volume (always used with a rotary knob)	0x0 -30 steps 0x1 -29 steps 0x2 -28 steps … 0x1B -3 steps 0x1C -2 steps 0x1D -1 step (decrement volume) 0x1E Not Pressed / Inactive 0x1F +1 step (increment volume) 0x20 +2 steps 0x21 +3 steps … 0x3A +28 steps 0x3B +29 steps 0x3C +30 steps

2.1.3 BUTTONv2-IIR-REQ-096643/C-LIN setVolume 2 - Button Interface Requirements

SetVolume (ICPVolumeCmd2)	Method for incrementing / decrementing Volume (always used with a rotary knob)	0x0 - 7 steps 0x1 - 6 steps 0x2 - 5 steps 0x3 -4 steps 0x4 -3 steps 0x5 -2 steps 0x6 -1 step (decrement volume) 0x7 Not Pressed / Inactive 0x8 +1 step (increment volume) 0x9 +2 steps 0xA +3 steps 0xB +4 steps 0xC +5 steps 0xD +6 steps 0xE +7 steps

2.1.4 BUTTON-IIR-REQ-095295/E-LIN setRotarySteps - Button Interface Requirements

SetRotarySteps (ICPRotary)	Method for incrementing / decrementing Tune/Rotary Browsing (always used with a rotary knob)	0x0 - 7 steps 0x1 - 6 steps 0x2 - 5 steps 0x3 -4 steps 0x4 -3 steps 0x5 -2 steps 0x6 -1 step 0x7 Not Pressed / Inactive 0x8 +1 step 0x9 +2 steps 0xA +3 steps 0xB +4 steps 0xC +5 steps 0xD +6 steps 0xE +7 steps

2.1.5 BUTTON-IIR-REQ-153850/A-LIN - ApplicationInformation0

ApplicationInformation0 (APINFO0)	Method for error reporting for the volume knob	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.6 BUTTON-IIR-REQ-153851/A-LIN - ApplicationInformation1

ApplicationInformation1 (APINFO1)	Method for error reporting for the rotary knob	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.7 BUTTON-IIR-REQ-153852/B-LIN - ApplicationInformation2

ApplicationInformation2 (APINFO2)	Method for error reporting for an under voltage error	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.8 BUTTON-IIR-REQ-153853/A-LIN - ApplicationInformation3

ApplicationInformation3 (APINFO3)	Method for error reporting for an over voltage error	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.9 BUTTON-IIR-REQ-153854/A-LIN - ApplicationInformation4

ApplicationInformation4 (APINFO4)	Method for reporting that the slave needs to be configured*	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

*not used for now and is always set to "No Configuration Needed"

2.1.10 BUTTON-IIR-REQ-372239/A-LIN - ApplicationInformation0_NewGeneration

ApplicationInformation0_NewGeneration (APINFO_0_NG)	Method for error reporting for knob faults in general. E.g. covers both, volume knob and rotary, if available	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.11 BUTTON-IIR-REQ-372241/A-LIN - ApplicationInformation1_NewGeneration

ApplicationInformation1_NewGeneration (APINFO_1_NG)	Method for error reporting for hardwired output fault. E.g. of hardwired buttons	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.12 BUTTON-IIR-REQ-372242/A-LIN - ApplicationInformation2_NewGeneration

ApplicationInformation2_NewGeneration (APINFO_2_NG)	Method for error reporting for an indicator fault. E.g. LED error of indicators	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.13 BUTTON-IIR-REQ-372243/A-LIN - ApplicationInformation3_NewGeneration

ApplicationInformation3_NewGeneration (APINFO_3_NG)	Method for error reporting for a circuit fault. E.g. short to ground, short to battery, open circuit	See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting")

2.1.14 BUTTON-IIR-REQ-107294/C-LIN - LINStatus

LINStatus (ICPLINStatus)	Method for additional error reporting	0x0 No detected fault 0x1 Reset (module start after reset) 0x2 Reserved (not used) 0x3 Reserved (not used) 0x4 Data Error 0x5 Checksum Error 0x6 Byte Field Framing Error 0x7 ID Parity Error See LIN Data Link and Physical Layer for further information (Chapter "Ford Standard Error Reporting", respectively SAE J2602 Chapter "Error Field Definition")

2.1.15 BUTTON-IIR-REQ-107295/A-LIN - IlluminationZone

IlluminationZone (DSPIlluZone)	Method for the active zones to illuminate	See LIN Illumination Specifications for further information

2.1.16 BUTTON-IIR-REQ-107296/A-LIN - IlluminationLevel1

IlluminationLevel1 (DSPIlluLvl1)	Method for Value for the 8-bit button backlight PWM generator.	See LIN Illumination Specifications for further information

2.1.17 BUTTON-IIR-REQ-107297/A-LIN - IlluminationLevel2

IlluminationLevel2 (DSPIlluLvl2)	Method for Value for the 8-bit knob backlight PWM generator.	See LIN Illumination specifications for further information

2.1.18 BUTTON-IIR-REQ-107298/A-LIN - PartNumberXxxxx

PartNumberXxxxx (ICPPartNumXxxxx) Ex. ICPPartNumIndex, ICPPartNumData3,…	Method for transferring part number. See LIN Data Link and Physical Layer specification for further information.	See LIN Data Link and Physical Layer specification for further information.

2.1.19 BUTTON-IIR-REQ-107299/B-LIN - ConfigDataXxxx

ConfigDataXxxx (DSPLConfigXxxx) Ex. DSPLConfigIndex, DSPLConfigData2,…	Method for sending configuration data to the LIN slave. NOT USED.	See LIN Data Link and Physical Layer specification for further information.

2.1.20 BUTTON-IIR-REQ-117484/A-LIN - SerialNumberXxxx

SerialNumberXxxx (ICPSrNrDigitXX) Ex. ICPSrNrDigit00, ICPSrNrDigit00, ICPSrNrDigit12,…	Method for transferring serial number of the slave to the master for standard readout. Note: this shall be used equivalent to part number readout.	See LIN Bezel Diagnostics SPSS for further information.

2.1.21 BUTTON-IIR-REQ-366705/A-LIN - ICP_RC

ICP_RC	Method for implementing a rolling counter for having the possibility to detect possible transmission faults of this frame. Note: see "BUTTON-SR-REQ-366885-Rolling Counter behavior" for further information	always 0x0: not supported 0x1 - 0xF

2.1.22 BUTTON-IIR-REQ-366707/A-LIN - DSPL_RC

DSPL_RC	Method for implementing a rolling counter for having the possibility to detect possible transmission faults of this frame. Note: see "BUTTON-SR-REQ-366885-Rolling Counter behavior" for further information	always 0x0: not supported 0x1 - 0xF

2.1.23 BUTTON-IIR-REQ-366710/A-LIN - DSPLIlluIndPTS

DSPLIlluIndPTS	Method for setting status of “push-To-start” indicator. Note: see illumination specification for further information	0x0 Off 0x1 On 0x2 Blinking 0x3 Reserved

2.1.24 BUTTON-IIR-REQ-366711/A-LIN - DSPLIlluIndX

DSPLIlluIndX	Method for setting status of indicator on position X. Note: see illumination specification for further information	0x0 Off 0x1 On 0x2 Blinking 0x3 Reserved

2.1.25 BUTTON-IIR-REQ-366718/A-LIN - DSPLIlluBtnPTS

DSPLIlluBtnPTS	Method for setting illumination status of “push-To-start” button Note: see Illumination specification for further information	0x0 Off 0x1 On

2.1.26 BUTTON-IIR-REQ-366719/A-LIN - DSPLIlluBtnX

DSPLIlluBtnX	Method for setting illumination status of button on position X. Note: see illumination specification for further information	0x0 Off 0x1 On

2.1.27 BUTTON-IIR-REQ-366725/A-LIN - DSPLIlluVolKnob

DSPLIlluVolKnob	Method for setting illumination status of knob inside volume ring. In fact, it is On/Off button. Note: see Illumination specification for further information	0x0 Off 0x1 On

2.1.28 BUTTON-IIR-REQ-366726/A-LIN - DSPLIlluBtnChrome

DSPLIlluBtnChrome	Method for setting illumination status of chrome button(s). Note: see Illumination specification for further information	0x0 Off 0x1 On

2.1.29 BUTTON-IIR-REQ-366729/A-LIN - DSPLIlluPWMIndicatorTargetPTS

DSPLIlluPWMIndicatorTargetPTS	Method for setting brightness target value of “push-to-start” indicator. Note: see illumination specification for further information	10 Bit PWM value (linear or logarithmic) see illumination specification for how to use

2.1.30 BUTTON-IIR-REQ-366730/A-LIN - DSPLIlluPWMBacklightTargetPTS

DSPLIlluPWMBacklightTargetPTS	Method for setting brightness target value of “push-to-start” button backlight. Note: see illumination specification for further information	10 Bit PWM value (linear or logarithmic) see illumination specification for how to use

2.1.31 BUTTON-IIR-REQ-366731/A-LIN - DSPLIlluPWMBacklightTarget

DSPLIlluPWMBacklightTarget	Method for setting brightness target value of button area backlight. Note: see illumination specification for further information	10 Bit PWM value (linear or logarithmic) see illumination specification for how to use

2.1.32 BUTTON-IIR-REQ-366732/A-LIN - DSPLIlluPWMIndicatorTarget

DSPLIlluPWMIndicatorTarget	Method for setting brightness target value of indicator area. Note: see illumination specification for further information	10 Bit PWM value (linear or logarithmic) see illumination specification for how to use

2.1.33 BUTTON-IIR-REQ-366733/A-LIN - DSPLIlluPWMTimerUp

DSPLIlluPWMTimerUp	Method for setting value for timer to reach target brightness value for smooth dimming direction upwards. Note: see illumination specification for further information	4 Bit value see illumination specification for how to use

2.1.34 BUTTON-IIR-REQ-366736/A-LIN - DSPLIlluPWMTimerDown

DSPLIlluPWMTimerDown	Method for setting value for timer to reach target brightness value for smooth dimming direction downwards. Note: see illumination specification for further information	4 Bit value see illumination specification for how to use

2.1.35 BUTTON-IIR-REQ-366738/A-LIN - DSPLIlluDimmingCurveType

DSPLIlluDimmingCurveType	Method for setting how PWM values shall be interpreted). Note: see Illumination specification for further information	0x0 linear 0x1 exponential (e-curve)

2.1.36 BUTTON-IIR-REQ-367005/A-LIN - IlluIndAllocX

IlluIndAllocX (X = 1 to 7)	Method for transferring allocation of available indicators. Means: it contains button ID of related indicator at this position. If none is available at this place, “ICPBtnID_Idle” is used. Hint: That is needed for LIN Master to know which indicators are available on connected LIN slave and on which position to find. This is needed e.g. for setting status of indicators via DSPLIlluIndX. Example: Position 1 is “MAX Defrost” Position 2 is no indicator Position 3 is “Traction Control” Position 4 is no indicator Position 5 is no indicator Position 6 is no indicator Example: this would result in: DSPLIlluInd1 = ICPBtnID_MAXDefr DSPLIlluInd2 = ICPBtnID_Idle DSPLIlluInd3 = ICPBtnID_TracContr	ButtonNameID 0x00 – 0xFF see encoding types of ICPBtnID in LDF

	DSPLIlluInd4 = ICPBtnID_Idle
	DSPLIlluInd5 = ICPBtnID_Idle
	DSPLIlluInd6 = ICPBtnID_Idle	

2.1.37 BUTTON-IIR-REQ-367006/A-LIN - IlluBtnAllocX

IlluBtnAllocX (X = 1 to 8)	Method for transferring allocation of available buttons. Means: it contains button ID of related button at this position. If none is available at this plasce, "ICPBtnID_Idle" is used. Hint: That is needed for LIN Master to know which buttons are available on connected LIN slave and on which position to find. This is needed e.g. for setting status of buttons via DSPLIlluBtnX. Example: Position 1 is "MAX Defrost" Position 2 is no button Position 3 is "Traction Control" Position 4 is no button Position 5 is no button Position 6 is no button Example: this would result in: DSPLIlluBtn1 = ICPBtnID_MAXDefr DSPLIlluBtn2 = ICPBtnID_Idle DSPLIlluBtn3 = ICPBtnID_TracContr DSPLIlluBtn4 = ICPBtnID_Idle DSPLIlluBtn5 = ICPBtnID_Idle DSPLIlluBtn6 = ICPBtnID_Idle	ButtonNameID 0x00 – 0xFF see encoding types of ICPBtnID in LDF

2.1.38 BUTTON-IIR-REQ-367267/A-LIN - ICPIlluSmoothDimmSupp

ICPIlluSmoothDimmSupp	Method for transferring if LIN slave supports smooth dimming strategy. Note: For further information on how smooth dimming works, please refer to illumination specification.	0x0 No 0x1 Yes

2.1.39 BUTTON-IIR-REQ-366723/A-LIN - ICP_CRC8

ICP_CRC8	Method for carrying CRC8 from ICP to master Note: see CRC8 requirements for further information	0x00..0xFF

2.1.40 BUTTON-IIR-REQ-366724/A-LIN - DSPL_CRC8

DSPL_CRC8	Method for carrying CRC8 from master to slave. Note: see CRC8 requirements for further information	0x00..0xFF

2.2 General Requirements

2.2.1 <u>BUTTON-SR-REQ-107308/B-LIN Scheduler turnaround frequency for standard interface</u>

It shall be configurable (e.g. via diagnosis) to send each scheduler a specified number of times (e.g. X, Y, …) and then the next one a specified number of times (e.g. Y). After that it shall begin from start.
X, Y shall be possible to set from 0x0 to 0xF.

Note: This not applies for the configuration scheduler(s).

An example for clarification:
E.g. LIN11: X=0, LIN12: Y=1, LIN13: Z=5
This will result in: LIN11 is not sent, LIN12 will be sent 1 times and then LIN13 will be sent 5 times. After that it begins from start.

LIN11: X=0												
LIN12: Y=1	▨						▨					
LIN13: Z=5		▨	▨	▨	▨	▨		▨	▨	▨	▨	▨

2.2.2 <u>BUTTON-SR-REQ-116454/B-LIN Scheduler turnaround default values for standard interface</u>

LIN11: X=0,
LIN12: Y=1,
LIN13: Z=0

2.2.3 <u>BUTTON-SR-REQ-116455/A-Reset ButtonID after button release</u>

If a button is "released" after "pressed" and an "inactive/Not_Pressed " was sent at least once then the corresponding ICPBtnID/ButtonNameID Byte will be changed to "Idle". If the ButtonID-Code is "Idle" then the appropriate ICPBtnCoding/ButtonActivationState Nibble is a don't care and should always be "Inactive/Not_Pressed".

2.2.4 <u>BUTTON-SR-REQ-107300/A-LimpHome state (Button Transmitter)</u>

If a LIN Server (ex. LIN ICP) reaches the limp-home state it should:
- Activate battery saver mode
- Switch off illumination (if available)

2.2.5 <u>BUTTON-SR-REQ-366885/A-Rolling Counter behavior</u>

The rolling counter shall be incremented by one, in each consecutive corresponding frame. As well, if all values of this frame are the same.
If the rolling counter reaches its maximum value, it shall be reset and begin with default value, again.

Hint: Please keep in mind that 0x0 is never used if CRC is supported! (See "BUTTON-SR-REQ-366886-Rolling Counter default value if CRC is supported")

Hint: Please keep in mind that a value of 0x0 shows that CRC is not supported (BUTTON-SR-REQ-366706/A-Rolling Counter default value if CRC not supported)

Example of a rolling counter supporting CRC:

Startup rolling counter is 1. So, first frame carries 1 in RC. Next frame will have 2, and so on until RC is 15. For next frame the rolling counter is reset to default value. So, it will be 1, again.

Hint: This example assumes “BUTTON-SR-REQ-366886-Rolling Counter default value if CRC is supported” defines default value to 1 and size is four bits.

2.2.6 BUTTON-SR-REQ-366886/A-Rolling Counter default value if CRC is supported

If CRC is supported, default value for rolling counter is 0x1.

2.2.7 BUTTON-SR-REQ-366706/A-Rolling Counter default value if CRC not supported

If CRC is NOT supported, default value for rolling counter is 0x0.

2.2.8 BUTTON-SR-REQ-366887/A-Indicator position allocation

This method allows LIN master to get information about position of each indicator. This is needed since, later on, it only sets status of position on LIN slave.
If master requests “IlluIndAllocX” (X will be e.g. 1 to 6) it shall get related button IDs of each available indicator sorted by position.

The position shall not change.

This can be requested every time master needs this information.

For further information of “IlluIndAllocX” see “BUTTON-IIR-REQ-366711-LIN – DSPLIlluIndX”

2.2.9 BUTTON-SR-REQ-366888/A-Button position allocation

This method allows LIN master to get information about position of each button. This is needed since, later on, it only sets status of position on LIN slave.
If master requests “IlluBtnAllocX” (X will be e.g. 1 to 6) it shall get button IDs of each available button sorted by position.

The position shall not change.

This can be requested every time master needs this information.

For further information on “IlluBtnAllocX” see “BUTTON-IIR-REQ-367006-LIN – IlluBtnAllocX”

2.2.10 BUTTON-SR-REQ-383144/A-Illumination configuration

To dynamically configure illumination allocation (means: which buttons are located at which position for background illumination and which indicators are located at which position) the master shall use “BUTTON-SR-REQ-366888-Button position allocation” and “BUTTON-SR-REQ-366887-Indicator position allocation”.

2.2.11 BUTTON-SR-REQ-383145/A-Setup illumination allocation

A master shall read illumination allocation only once, if internal information is not available anymore.

This is usually the case after cold boot (e.g. startup from battery reconnect) or warm boot (e.g. reset) and store this information internally (e.g. in volatile memory like RAM).

Hint: This saves transmission time and means, if LIN slave is changed, master needs to be rebooted, to guarantee that illumination allocation is correct.
However, in any other unintentional case of losing this information, master can request this information at any time.

2.2.12 BUTTON-SR-REQ-366708/A-CRC8 algorithm if CRC is supported

Following look up table is used for 0x83 CRC algorithm:

```
uint8 CalcCRC8(uint8 data[], uint8 len)
```

```
{
uint8 crc = 5; // CRC is non-zero CRC when all data is zero
uint8 tmp;
uint8 i = 0;
// CRC Lookup Table for #0x83 = x^8 +x^2 +x +1 (0x107) <=> (0xe0; 0x1c1)
static uint8 CRC_table_0x83[256] = { // so array is not allocated on stack
0x00, 0x07, 0x0E, 0x09, 0x1C, 0x1B, 0x12, 0x15, 0x38, 0x3F, 0x36, 0x31, 0x24, 0x23, 0x2A, 0x2D,
0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65, 0x48, 0x4F, 0x46, 0x41, 0x54, 0x53, 0x5A, 0x5D,
0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5, 0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85, 0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2, 0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2, 0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 0x9A,
0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32, 0x1F, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0D, 0x0A,
0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42, 0x6F, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A,
0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C, 0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC, 0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 0xD4,
0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B, 0x7C, 0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44,
0x19, 0x1E, 0x17, 0x10, 0x05, 0x02, 0x0B, 0x0C, 0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34,
0x4E, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5C, 0x5B, 0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63,
0x3E, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B, 0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14, 0x13,
0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB, 0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB, 0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3};


while (i <> len)
{
// XOR datat byte into CRC
tmp = (data[i] ^ crc);
// fetch CRC value from table
crc = CRC_table_0x83[tmp]);
}
return crc;
```

Hint: have a look in “BUTTON-SR-REQ-366886-Rolling Counter default value if CRC is supported”

2.2.13 BUTTON-SR-REQ-366709/A-CRC8 notation

CRC algorithm shall use x^8 +x^2 +x +1 (0x83 in “Koopman” notation) to calculate the 8-bit CRC of the data byte set.
It has Hamming Distance of four (HD=4) for 119 data bits.

2.2.14 BUTTON-SR-REQ-366712/A-CRC8 initialization

The CRC shall not return zero when the data set is all zero by initializing the CRC algorithm CRC to five (`uint8 crc = 5;--` as shown in the above algorithm)

2.2.15 BUTTON-SR-REQ-366713/A-CRC8 computation

The CRC shall be computed whenever:

- Any of the data is updated
- OR – whenever the message is transmitted. This option has less CPU load

2.2.16 BUTTON-SR-REQ-366714/A-CRC8 calculation

CRC calculation shall use all bytes before CRC byte (includes rolling counter, as well!)

2.2.17 BUTTON-SR-REQ-366715/A-CRC8 unused bits

Unused bits in the message frame shall be set to 0.

2.2.18 BUTTON-SR-REQ-366716/A-CRC8 unused bytes

Bytes with no signal/data (0x0) shall be used in CRC calculation. I.e. All 7 bytes are used in all CRC calculations.

2.2.19 BUTTON-SR-REQ-366720/A-CRC8 data stream

The data stream fed into the CRC shall be composed of a stream of single byte values.

2.2.20 BUTTON-SR-REQ-366721/A-CRC8 algorithm if CRC is not supported

If CRC is not supported whole CRC8 shall be set to 0x00.

Hint: have a look in "BUTTON-SR-REQ-366706-Rolling Counter default value if CRC not supported"

2.2.21 BUTTON-SR-REQ-366722/A-decision of interface variant

A master can decide on an available response/reaction of "IlluIndAllocX" and "IlluBtnAllocX" if a slave supports new generation interface or standard interface.

Means: a slave supporting only standard interface will give no response/reaction on "IlluIndAllocX" and "IlluBtnAllocX" that are in frames "DSPLIlluIndicationAllocation" and "DSPLIlluButtonAllocation" that are unknown, too.

2.2.22 BUTTON-SR-REQ-372268/A-standard interface

The standard interface covers following:

Schedulers:
LIN11,
LIN12,
LIN13,
LINConfig1,
MRF_schedule,
SRF_schedule

Frames:
ICPBtnState,
DSPLSendSignals,
ICPBtnStateRotary,
DSPLConfigCalibrate,
ICPPartNum,
ICPSerialNum,
MasterReq,
SlaveResp

Signals:
ICP_APINFO_0,
ICP_APINFO_1,
ICP_APINFO_2,
ICP_APINFO_3,
ICP_APINFO_4,
ICPBtnCoding_A,
ICPBtnCoding_B,
ICPBtnCoding_C,
ICPBtnCoding_D,
ICPLINStatus,
ICPRotaryCmd,
ICPVolumeCmd,
ICPVolumeCmd2,
ICPBtnID_A,
ICPBtnID_B,
ICPBtnID_C,
ICPBtnID_D,
ICPPartNumIndex,
ICPPartNumData0,
ICPPartNumData1,

ICPPartNumData2,
ICPPartNumData3,
ICPPartNumData4,
ICPPartNumData5,
ICPSrNrDigit00,
ICPSrNrDigit01,
ICPSrNrDigit02,
ICPSrNrDigit03,
ICPSrNrDigit04,
ICPSrNrDigit05,
ICPSrNrDigit06,
ICPSrNrDigit07,
ICPSrNrDigit08,
ICPSrNrDigit09,
ICPSrNrDigit10,
ICPSrNrDigit11,
ICPSrNrDigit12,
ICPSrNrDigit13,
DSPLIlluZone,
DSPLDimmLvl1,
DSPLDimmLvl2,
DSPLConfigIndex,
DSPLConfigData0,
DSPLConfigData1,
DSPLConfigData2,
DSPLConfigData3,
DSPLConfigData4,
DSPLConfigData5,
DSPLConfigData6,
MasterReqB0,
MasterReqB1,
MasterReqB2,
MasterReqB3,
MasterReqB4,
MasterReqB5,
MasterReqB6,
MasterReqB7,
SlaveRespB0,
SlaveRespB1,
SlaveRespB2,
SlaveRespB3,
SlaveRespB4,
SlaveRespB5,
SlaveRespB6,
SlaveRespB7

2.2.23 BUTTON-SR-REQ-372270/A-new generation interface

The new generation interface covers following:

Schedulers:
LINBtnIndIllu,
LINIlluConfig,
LINConfig1NG,
MRF_schedule,
SRF_schedule

Frames:

ICPBtnStateTwoRC,
DSPLIlluBlocks,
DSPLPWMs,
DSPLIlluIndicationAllocation,
DSPLIlluButtonAllocation,
DSPLConfigCalibrate,
ICPPartNumNG,
ICPSerialNumNG,
MasterReq,
SlaveResp

Signals:
ICP_APINFO_0_NG,
ICP_APINFO_1_NG,
ICP_APINFO_2_NG,
ICP_APINFO_3_NG,
ICP_APINFO_4,
ICPBtnCoding_A,
ICPBtnCoding_B,
ICPLINStatus,
ICPRotaryCmd,
ICPVolumeCmd2,
ICPBtnID_A,
ICPBtnID_B,
ICPIlluSmoothDimmSupp,
Reserved3BitIPC,
ICP_RC,
ICP_CRC8,
DSPLIlluIndPTS,
DSPLIlluInd1,
DSPLIlluInd2,
DSPLIlluInd3,
DSPLIlluInd4,
DSPLIlluInd5,
DSPLIlluInd6,
DSPLIlluInd7,
DSPLIlluBtnPTS,
DSPLIlluBtn1,
DSPLIlluBtn2,
DSPLIlluBtn3,
DSPLIlluBtn4,
DSPLIlluBtn5,
DSPLIlluBtn6,
DSPLIlluBtn7,
DSPLIlluBtn8,
DSPLIlluVolKnob,
DSPLIlluBtnChrome,
Reserved1BitDSPL,
DSPL_RC,
DSPL_CRC8,
DSPLIlluPWMIndicatorTarget,
DSPLIlluPWMBacklightTarget,
DSPLIlluPWMIndicatorTargetPTS,
DSPLIlluPWMBacklightTargetPTS,
DSPLIlluPWMTimerUp,
DSPLIlluPWMTimerDown,
DSPLIlluDimmingCurveType,
Reserved7BitDSPL,
IlluIndAlloc1,

IlluIndAlloc2,
IlluIndAlloc3,
IlluIndAlloc4,
IlluIndAlloc5,
IlluIndAlloc6,
IlluIndAlloc7,
IlluBtnAlloc1,
IlluBtnAlloc2,
IlluBtnAlloc3,
IlluBtnAlloc4,
IlluBtnAlloc5,
IlluBtnAlloc6,
IlluBtnAlloc7,
IlluBtnAlloc8,
ICPPartNumIndex,
ICPPartNumData0,
ICPPartNumData1,
ICPPartNumData2,
ICPPartNumData3,
ICPPartNumData4,
ICPPartNumData5,
ICPSrNrDigit00,
ICPSrNrDigit01,
ICPSrNrDigit02,
ICPSrNrDigit03,
ICPSrNrDigit04,
ICPSrNrDigit05,
ICPSrNrDigit06,
ICPSrNrDigit07,
ICPSrNrDigit08,
ICPSrNrDigit09,
ICPSrNrDigit10,
ICPSrNrDigit11,
ICPSrNrDigit12,
ICPSrNrDigit13,
DSPLConfigIndex,
DSPLConfigData0,
DSPLConfigData1,
DSPLConfigData2,
DSPLConfigData3,
DSPLConfigData4,
DSPLConfigData5,
DSPLConfigData6,
MasterReqB0,
MasterReqB1,
MasterReqB2,
MasterReqB3,
MasterReqB4,
MasterReqB5,
MasterReqB6,
MasterReqB7,
SlaveRespB0,
SlaveRespB1,
SlaveRespB2,
SlaveRespB3,
SlaveRespB4,
SlaveRespB5,
SlaveRespB6,
SlaveRespB7

2.3 BUTTONv2-FUN-REQ-095292/A-LIN Message Structure

2.3.1 LIN - BCP (Button Control Panel) Button Press Message Structure

2.3.1.1 LIN - Infotainment Button Press

2.3.1.1.1 BUTTONv2-SR-REQ-096645/C-LIN BCP message structure

Up to 4 infotainment push button press events can be activated simultaneously by the BCP (**hint**: maybe new interfaces like e.g. new generation interface, contains less button slots (including related ID and coding slots) e.g. 2. In this case only first number of slots e.g. A and B shall be considered). The LIN button press messages are periodic. The basic structure of these messages is shown below.

Note: The LIN ICP will always send the LIN Button Encoding (ex 0x1 Active, 0x2 ShortEvent…). For the receiving module of the LIN ICP button presses whether to use the LIN Button Encoding or the Button Encoding Normalization column depends on how a feature is specified. If the feature SPSS or HMI spec (ex. press and hold timer) uses pressed/not pressed then the Button Encoding Normalization column shall be used. If the SPSS or HMI spec (ex. press and hold timer) uses the LIN Button Encoding (ex 0x1 Active, 0x2 ShortEvent…) then the LIN Button Encoding values shall be used.

Button ID	LIN Button Encoding	Button Encoding Normalization
Button A Activation State	0x0 Inactive	0x0 Not Pressed
	0x1 Active	0x1 Pressed
	0x2 ShortEvent	0x2 Pressed
	0x3 ShortElapsed	0x3 Pressed
	0x4 LongEvent	0x4 Pressed
	0x5 Stuck	0x5 Stuck
Button B Activation State	0x0 Inactive	0x0 Not Pressed
	0x1 Active	0x1 Pressed
	0x2 ShortEvent	0x2 Pressed
	0x3 ShortElapsed	0x3 Pressed
	0x4 LongEvent	0x4 Pressed
	0x5 Stuck	0x5 Stuck
Button C Activation State	0x0 Inactive	0x0 Not Pressed
	0x1 Active	0x1 Pressed
	0x2 ShortEvent	0x2 Pressed
	0x3 ShortElapsed	0x3 Pressed
	0x4 LongEvent	0x4 Pressed
	0x5 Stuck	0x5 Stuck
Button D Activation State	0x0 Inactive	0x0 Not Pressed
	0x1 Active	0x1 Pressed
	0x2 ShortEvent	0x2 Pressed
	0x3 ShortElapsed	0x3 Pressed
	0x4 LongEvent	0x4 Pressed
	0x5 Stuck	0x5 Stuck

Infotainment Button Activation State Coding

BYTE X	BYTE X + 1	BYTE X + 2	BYTE X + 3	BYTE X + 4	BYTE X + 5

ICPBtnID_A (ie Button A Name ID)	ICPBtnID_B (ie Button B Name ID)	ICPBtnID_C (ie Button C Name ID)	ICPBtnID_D (ie Button D Name ID)	ICPBtnCoding_A (ie Button A Activation State)	ICPBtnCoding_B (ie Button B Activation State)	ICPBtnCoding_C (ie Button C Activation State)	ICPBtnCoding_D (ie Button D Activation State)

Note: position of x may vary from scheduler frame to scheduler frame

Infotainment Button Message Structure

2.3.1.1.2 BUTTONv2-SR-REQ-096646/C-LIN BCP message structure usage

When a button is activated it shall encode Button A first and if that position is already being used (ie pressed) shall move to the next Button position in the message. If all 4 buttons (A – D) are being used then any new button inputs shall be ignored until one of the 4 buttons are released.

Once a button press is active for a button ID (A – D) the BCP shall not move that same button press to another button ID location. For example if seek is being pressed and held and is assigned to 'Button C Name ID' it shall not change to 'Button A Name ID' while still being pressed.

The default to set Button Activation States A – D is 'Not Pressed / Inactive' unless there is a button activation event.

Example:
1. Button Module powers up and bus awake
2. Button Module sending: Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed
3. User presses button X
4. Button Module sending: Button A Name ID = X AND Button Activation State = Pressed
5. User releases button X
6. Button Module sends: Button A Name ID = X AND Button Activation State = Not Pressed at least once.
7. Button Module sends: Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed
8. Button Module continues to send (periodically) Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed until the next button press. Any new button press changes Button A Name ID from Inactive (Idle) to the new button value.

Hint: maybe new interfaces like e.g. new generation interface contains less button slots (including related ID and coding slots) e.g. 2. In this case only first number of slots e.g. A and B shall be considered.

2.3.1.1.3 BUTTONv2-SR-REQ-096647/C-LIN Multiple stuck buttons in BCP message structure

If all 4 BCP buttons A – D encoded in the BCP_Button_Press message are determined to be 'stuck' (not including setVolume / setRotarySteps signals – Byte 7 and 8) then one of the 4 button bytes (Button A – D) shall be released so that other buttons can be activated.

Hint: maybe new interfaces like e.g. new generation interface contains less button slots (including related ID and coding slots) e.g. 2. In this case only first number of slots e.g. A and B shall be considered.

2.3.1.2 LIN Signal Functionality (not normalized to Pressed / Not Pressed)

LIN signals are sent out periodically at a pre-defined set of time defined in the LDF. The receiving module may choose to utilize the LIN specific signals such as ShortEvent, ShortElapsed, LongEvent, Stuck to reduce the variability of not knowing when a button was pressed since button press could have occurred +/- (LIN periodic rate) msec from the event.

Also the normalization values of Press/Not_Pressed for encodings can be used (unless noted otherwise) but there could be variability of +/- (LIN periodic rate) msec.

2.3.1.2.1 BUTTON-SR-REQ-096734/B-Active

The Active encoding is set when a LIN button is first pressed.

2.3.1.2.2 BUTTON-SR-REQ-096735/A-ShortElapsed

The ShortElapsed button encoding is set 250 msec after the LIN button press event.

2.3.1.2.3 BUTTON-SR-REQ-096736/C-LongEvent

The LongEvent is set 1.5 seconds after the LIN button press event.

2.3.1.2.4 BUTTON-SR-REQ-096650/B-ShortEvent

The ShortEvent encoding is set if button presses are pressed quicker than ShortElapsed time. This value shall never be overwritten before it is sent out on the bus.

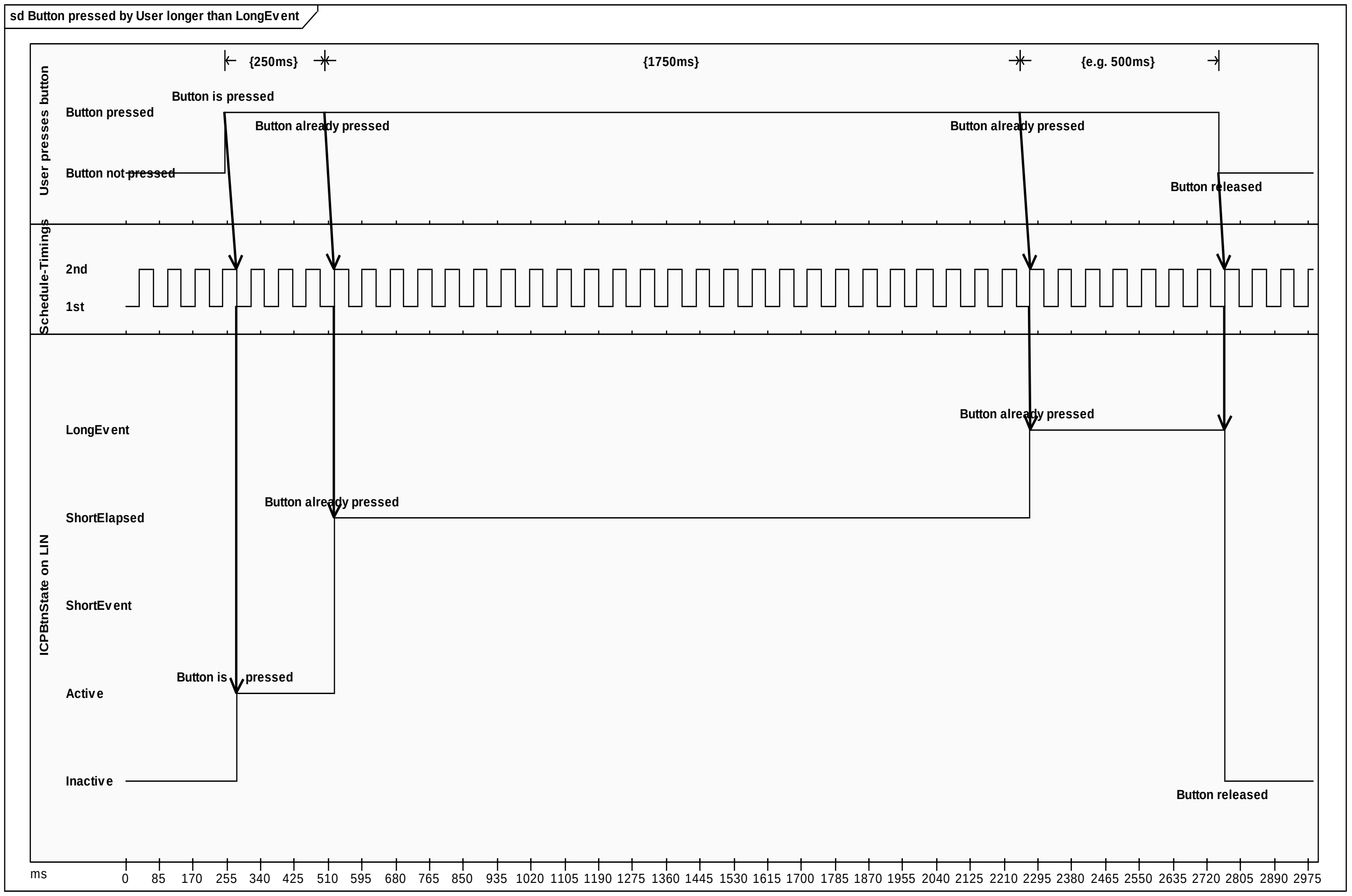
2.3.1.2.5 BUTTON-SR-REQ-096737/B-Stuck

The Stuck encoding is set 120 seconds after a press of a LIN button with no release.

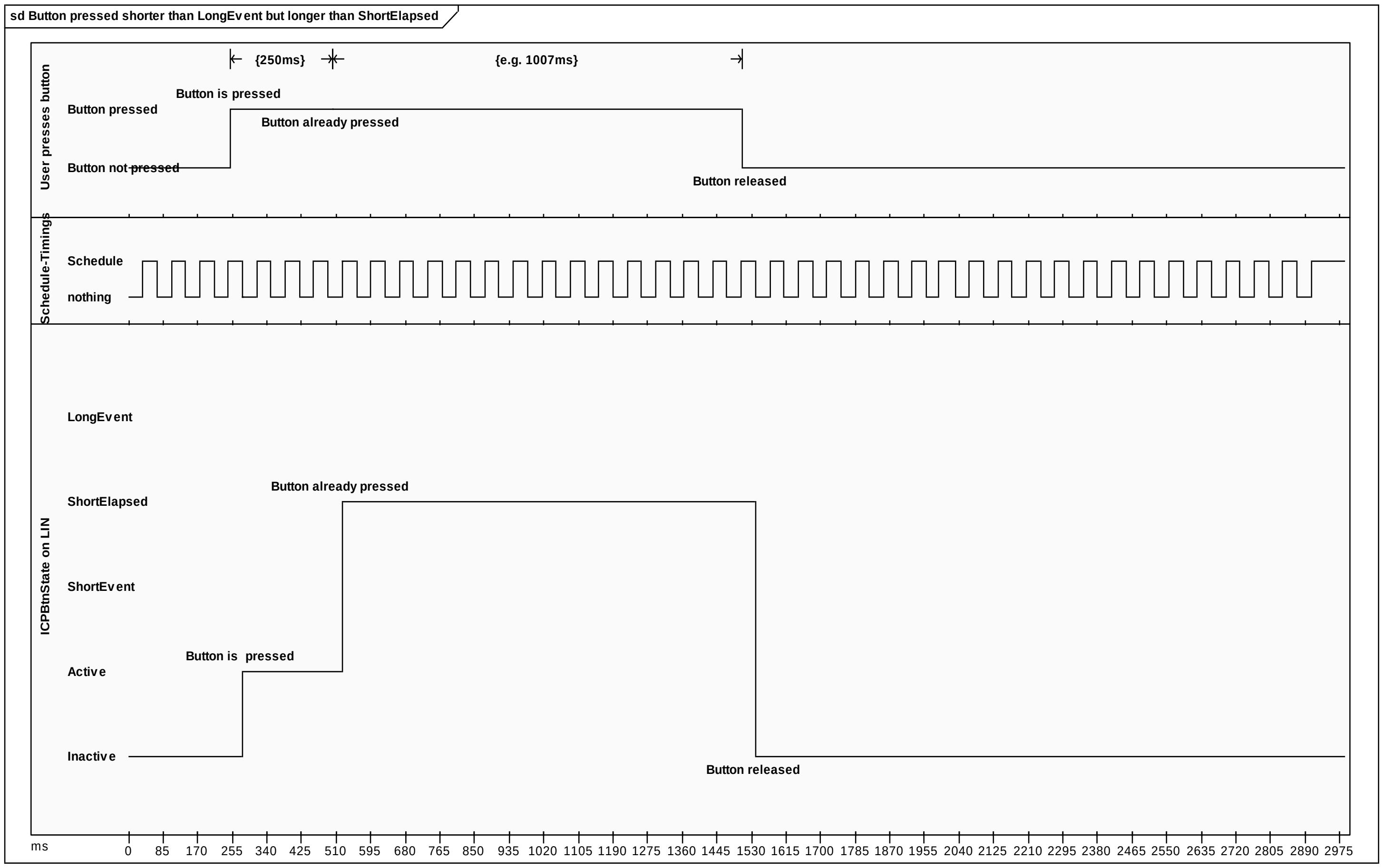
Note: The receiving module of the LIN button press is the module responsible for setting the LIN stuck button DTC as defined in the IDS (infotainment diagnostic specification).

2.3.1.2.6 Button Sequences

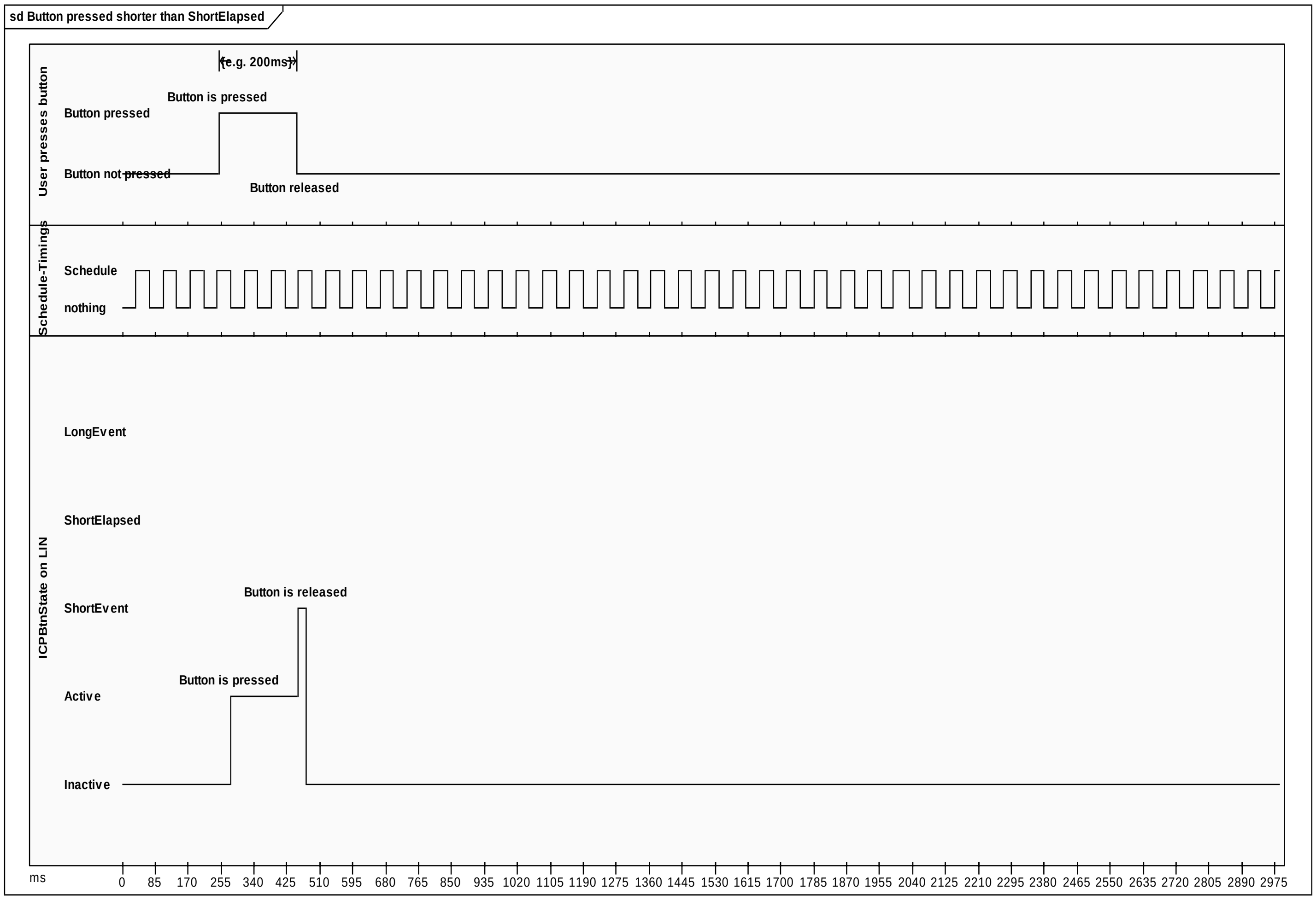
2.3.1.2.6.1 BUTTON-SR-REQ-107301/A-Button pressed by User longer than LongEvent - Button Sequence



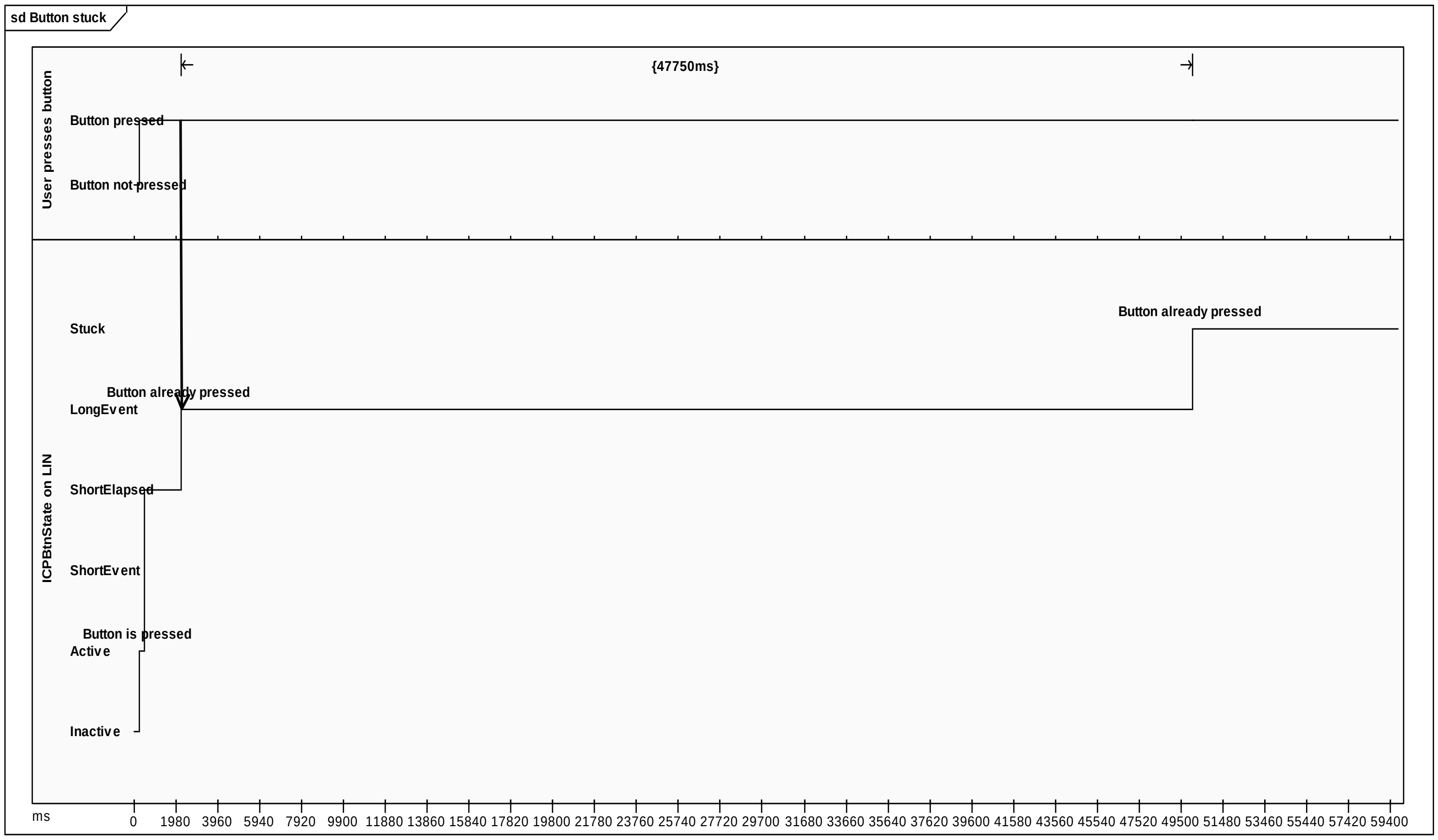
2.3.1.2.6.2 BUTTON-SR-REQ-107302/A-Button Pressed shorter than LongEvent but longer than ShortElapsed - Button Sequence



2.3.1.2.6.3 BUTTON-SR-REQ-107303/A-Button Pressed shorter than ShortElapsed - Button Sequence



2.3.1.2.6.4 BUTTON-SR-REQ-107304/B-Button Stuck - Button Sequence



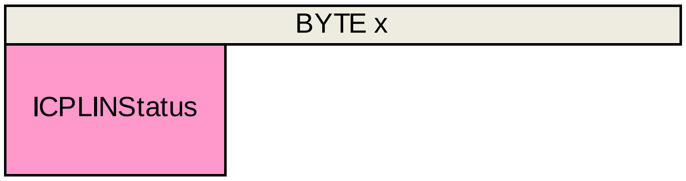
Note: this is an example only and reference the other parts of the SPSS for stuck button timer value to be used if the diagram above does not match what is called out elsewhere in the SPSS. Currently 120 seconds is used for stuck button timer.

2.3.2 ICP Status Transferrring

2.3.2.1 BUTTON-SR-REQ-115765/A-LIN ICP Status Structure

This sub chapter describes how status information is transferred from the ICP to the Master.
ICPLINStatus shows states or errors that are possible to detect but not supported in another way.

An example of the basic structure of this part of message is shown below:
(keep in mind this byte is not entire filled here)



2.3.3 LIN Rotary Structure

2.3.3.1 BUTTON-SR-REQ-116456/A-LIN Rotary Structure

This requirement describes how rotary information is transferred from the ICP to the Master. Only delta counts from frame to frame will be sent.

An example of the basic structure of these parts of messages is shown below:

BYTE x	
ICPVolumeCmd	ICPRotary

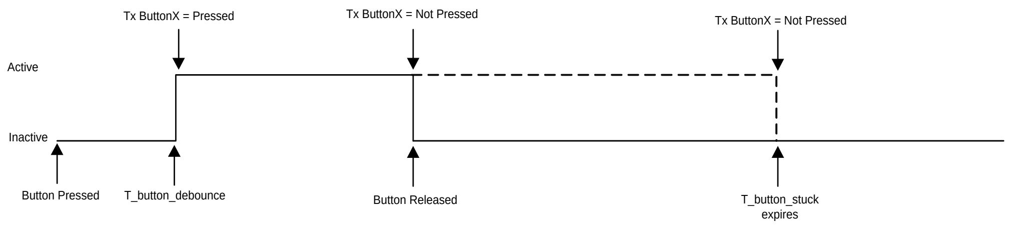
2.4 BUTTONv2-CLD-REQ-095293/A-Button Input Client (Button Transmitter) - LIN

The following sections define the Button Activation Strategy from the Transmitters perspective.

2.4.1 Push Button Activation - Transmitter Timing Requirements

2.4.1.1 <u>BUTTONv2-SR-REQ-107305/A-Transmitter Button Activation Process timing (LIN)</u>

The TBAP timing figure below will always remain true for any button activation event using the Pressed/Not Pressed normalization (ie not using LIN button sequences). The exception to this rule is for "Rotary Knobs" which is covered in the next section.



Transmitter Button Activation Process (TBAP) Timing

2.4.1.2 BUTTONv2-TMR-REQ-096739/B-T_reaction_time (LIN)

Name	Description	Units	Range	Resolution	Default
T_reaction_time (LIN)	The maximum transmitter reaction time from when a push button switch is closed until the push button message is put on the LIN bus.	msec	0-1000	10	70

2.4.2 Push / Touch Button Activation - Transmitter Functional Requirements

2.4.2.1 <u>BUTTONv2-SR-REQ-107306/A-Button Pressed / Not Pressed transimission (LIN)</u>

Once a button is pressed and debounced on the transmitter the button message will be sent to the receiver with the button signal coding set to "Pressed" (ie LongEvent, shortEvent…).

Once the button is released the transmitter will set the button coding as "Not Pressed / Inactive".

2.4.2.2 <u>BUTTON-SR-REQ-293306/A-Button Press reaction time (LIN)</u>

The transmitter (ex.EFP, ICP, SWC, SDM) reaction time from when a push button switch is closed until the push button message is put on the bus shall not exceed T_reaction_time.

Note: this does not apply to the touch sense buttons. Reference applicable specifications for touch sense debounce requirements.

2.4.2.3 <u>BUTTONv2-SR-REQ-103693/A-Transmitter Stuck Button (LIN)</u>

When the transmitter determines a button to be stuck:
-- The transmitter will keep the button encoding status as 'Stuck' as long as the button is stuck. Upon a new ignition cycle the button encoding shall remain as 'Stuck' until the button is determined to be operational.
-- Once a button becomes unstuck after previously being stuck then after remaining unstuck for T_button_unstuck the button shall become operational again.

2.4.2.4 BUTTON-TMR-REQ-293307/A-T_button_unstuck (LIN)

Name	Description	Units	Range	Resolution	Default

T_button_unstuck (LIN)	Once a button is determined to be stuck then T_button_unstuck is the time the button has to be unstuck before the button can be operational again. Note: always use the default value	sec	0-100	1	10

2.4.3 setRotarySteps signal (LIN)

2.4.3.1 <u>BUTTON-SR-REQ-095296/D-Tx SetRotarySteps (LIN)</u>

The setRotarySteps signal could possibly be used to increment / decrement for the tune or fast browse function for example. Reference the applicable SPSS section for details. Each setRotarySteps step encoding shall be treated as a press event in the remainder of this document.

The Button Input Client (Button Transmitter) shall increment the setRotarySteps signal by 1 for every detected rotary knob detent in the clockwise direction.

- For example if the Button Transmitter detects 3 detents in the clockwise direction could send +3 (fast turn) or three +1 button press messages (slower turn) as long as no information is lost. Note for LIN do not need to see the Not_Press to act on one of the volume steps.

The Button Input Client (Button Transmitter) shall decrement the setRotarySteps signal by 1 for every detected rotary knob detent in the counter-clockwise direction.

- For example if the Button Transmitter detects 3 detents in the counter clockwise direction could send -3 (fast turn) or three -1 button press messages (slower turn) as long as no information is lost. Note for LIN do not need to see the Not_Press to act on one of the volume steps.

The Button Input Client (Button Transmitter) shall send out the delta counts accumulated since the last setRotarySteps signal sent out on the bus.

If delta counts are 0 then “Not Pressed / Inactive” shall be sent out.

2.4.4 setVolume signal (LIN)

2.4.4.1 <u>BUTTON-SR-REQ-096743/B-Tx SetVolume (LIN)</u>

The setVolume signal can be used to increment / decrement the Volume for a volume rotary knob for example. Reference the applicable SPSS section for details. Each setVolume step encoding shall be treated as a press event in the remainder of this document.

The Button Input Client (Button Transmitter) shall increment the setVolume signal by 1 for every detected rotary knob detent in the clockwise direction.

- For example if the Button Transmitter detects 3 detents in the clockwise direction could send +3 (fast turn) or three +1 button press messages (slower turn) as long as no information is lost. Note for LIN do not need to see the Not_Press to act on one of the volume steps.

The Button Input Client (Button Transmitter) shall decrement the setVolume signal by 1 for every detected rotary knob detent in the counter-clockwise direction.

- For example if the Button Transmitter detects 3 detents in the counter clockwise direction could send -3 (fast turn) or three -1 button press messages (slower turn) as long as no information is lost. Note for LIN do not need to see the Not_Press to act on one of the volume steps.

The Button Input Client (Button Transmitter) shall send out the delta counts accumulated since the last setVolume signal sent out on the bus.

If delta counts are 0 then “Not Pressed / Inactive” shall be sent out.

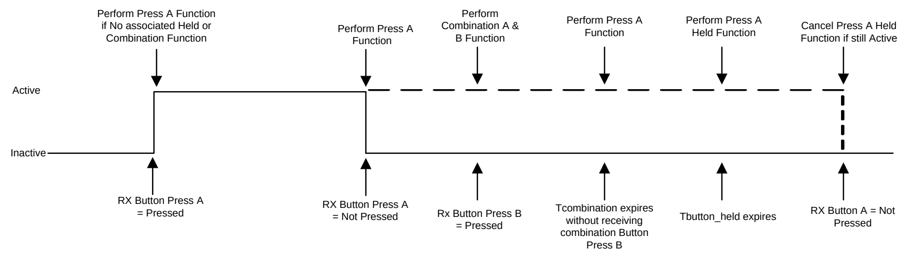
2.5 BUTTONv2-CLD-REQ-095294/A-Button Input Server (Button Receiver) - LIN

The following sections define the Button Activation Strategy from the Receivers perspective.

2.5.1 Button Activation - Receiver Timing Requirements (LIN)

2.5.1.1 BUTTONv2-SR-REQ-096745/B-Receiver Button Activation Process (RBAP) Timing (LIN)

All component receivers of button press information shall implement the Receiver Button Activation Process (RBAP) timing defined in the figure below.


Perform Press A Function if No associated Held or Combination Function
Perform Press A Function
Perform Combination A & B Function
Perform Press A Function
Perform Press A Held Function
Cancel Press A Held Function if still Active
Active
Inactive
RX Button Press A = Pressed
RX Button Press A = Not Pressed
Rx Button Press B = Pressed
Tcombination expires without receiving combination Button Press B
Tbutton_held expires
RX Button A = Not Pressed

Receiver Button Activation Process (RBAP) Timing

Exception: If a particular button supports press and hold function there may be times where a functional requirement may require the function to first be performed on the press (not wait for not pressed) and then take additional action when Tbutton_held expires and the buttons are determined to be held. This should only be performed if explicitly called out in a functional requirement otherwise the function shall be performed only on a Not Press or when Tbutton_held expires as shown in the RBAP above.

Exception 2: Rotary Knobs signals do not need to wait for the not pressed (ex setVolume, setRotarySteps).

2.5.2 Button Activation - Receiver Functional Requirements (LIN)

2.5.2.1 BUTTONv2-SR-REQ-096746/A-Receivers of Button Presses follow RBAP (LIN)

All receivers of button press information shall activate the RBAP upon receipt of the button press message.

2.5.2.2 BUTTONv2-SR-REQ-096747/B-Button Receiver Sampling Rate (LIN)

The sampling rate used by the receiver to read incoming button information messages shall be fast enough to read the multiple incoming messages.

2.5.2.3 BUTTON-SR-REQ-293490/A-Receivers of Held Button Presses (LIN)

The receiver shall determine whether the specific button has an associated held function.

2.5.2.4 BUTTON-SR-REQ-293493/A-Button Presses with no Held function or Combination with another button press (LIN)

If the button does not have an associated held or combination type function, then the receiver shall perform the function associated with a button press immediately upon receipt of the button press message and the RBAP process shall be exited.

2.5.2.5 BUTTON-SR-REQ-293491/A-Buttons with Held Function (LIN)

If the button does have an associated held function, the receiver shall start a hold timer (Tbutton_held) and wait for button 'not pressed' information.

-- If a button 'not pressed' message is not received prior to the expiration of Tbutton_held, the receiver shall perform the associated held function for that button press.

-- If a button 'not pressed' message is received prior to the expiration of Tbutton_held, then the receiver shall perform the associated press function.

2.5.2.6 <u>*BUTTON-SR-REQ-293492/A-Receivers of combination type button presses (LIN)*</u>

The receiver shall determine whether the button press has an associated combination type button press.

2.5.2.7 <u>*BUTTON-SR-REQ-293494/A-Combination Type button press operation (LIN)*</u>

When the receiver detects a button press (A) that has an associated combination the receiver must wait Tcombination before executing the button press function associated with the single button press (A).

-- If a second button (B) is received prior to Tcombination expiring, the receiving module must now determine if this (B) is part of a valid combination with (A).
- If the combination is valid, then the resulting combination (A + B) can be performed.
- If the combination is invalid, then no combination function is performed and the button presses (A) & (B) can be processed independently if allowed by the receiving module.

-- If Tcombination expires, then button press (A) is now valid and the (A) function can be performed.

2.5.2.8 <u>*BUTTON-SR-REQ-293495/A-Cancelling RBAP (LIN)*</u>

The receiver shall cancel the RBAP (Receiver Button Activation Process) upon:

- Receipt of the button 'Not Pressed' message.

- If the receiver does not receive a button 'Not Pressed' message within T_RBAP_Timeout
  - Unless noted otherwise all buttons shall timeout at some point in case of a stuck button
  - T_RBAP_Timeout may vary depending on how the button is used for a particular feature. The T_RBAP_Timeout would be part of error handling for a particular feature unless a T_RBAP_Timeout value is explicitly noted in a feature SPSS.
    - For example: if a feature does not have a press and hold or combination button press associated with it then that button might time out quickly. Other features like Seek/Volume have press and hold features associated with them and would not time out for a longer period of time. Follow up with the Ford HMI or Ford feature team on what make sense for timeout values.

<u>Reference</u>:

- For cancelling Volume Button RBAP reference requirements "<u>VOL-TMR_REQ-292290-T_Vol_RBAP_Timeout</u>" and "<u>VOL-REQ-292289-Volume Press and Hold Timeout</u>".

2.5.2.9 <u>*BUTTON-SR-REQ-293496/A-Cancelling RBAP when change to Standby (LIN)*</u>

For Infotainment receivers any event that shall cause a transition from Functional mode to Standby mode shall cancel a RBAP unless noted otherwise. The receiver shall cancel the operation, if active, and perform the required actions to enter Standby State.

2.5.2.10 <u>*BUTTONv2-SR-REQ-096749/A-Receivers of SetVolume Button presses (LIN)*</u>

The Button Input Server (Volume Setting Server) shall increment/decrement the volume based on the delta count of the volume steps received since the last SetVolume signal update.

2.5.2.11 <u>*BUTTON-SR-REQ-107307/A-Receivers of SetRotarySteps Button Presses (LIN)*</u>

The Button Input Server (Rotary Setting Server) shall increment/decrement the steps based on the delta count of the steps received since the last SetRotarySteps signal update.

3 Architecture Design - I2C over LVDS Interface

3.1 Button Interface Requirements - I2C

3.1.1 BUTTONv3-IIR-REQ-132973/A-I2C BCP Button Press - Button Interface Requirements

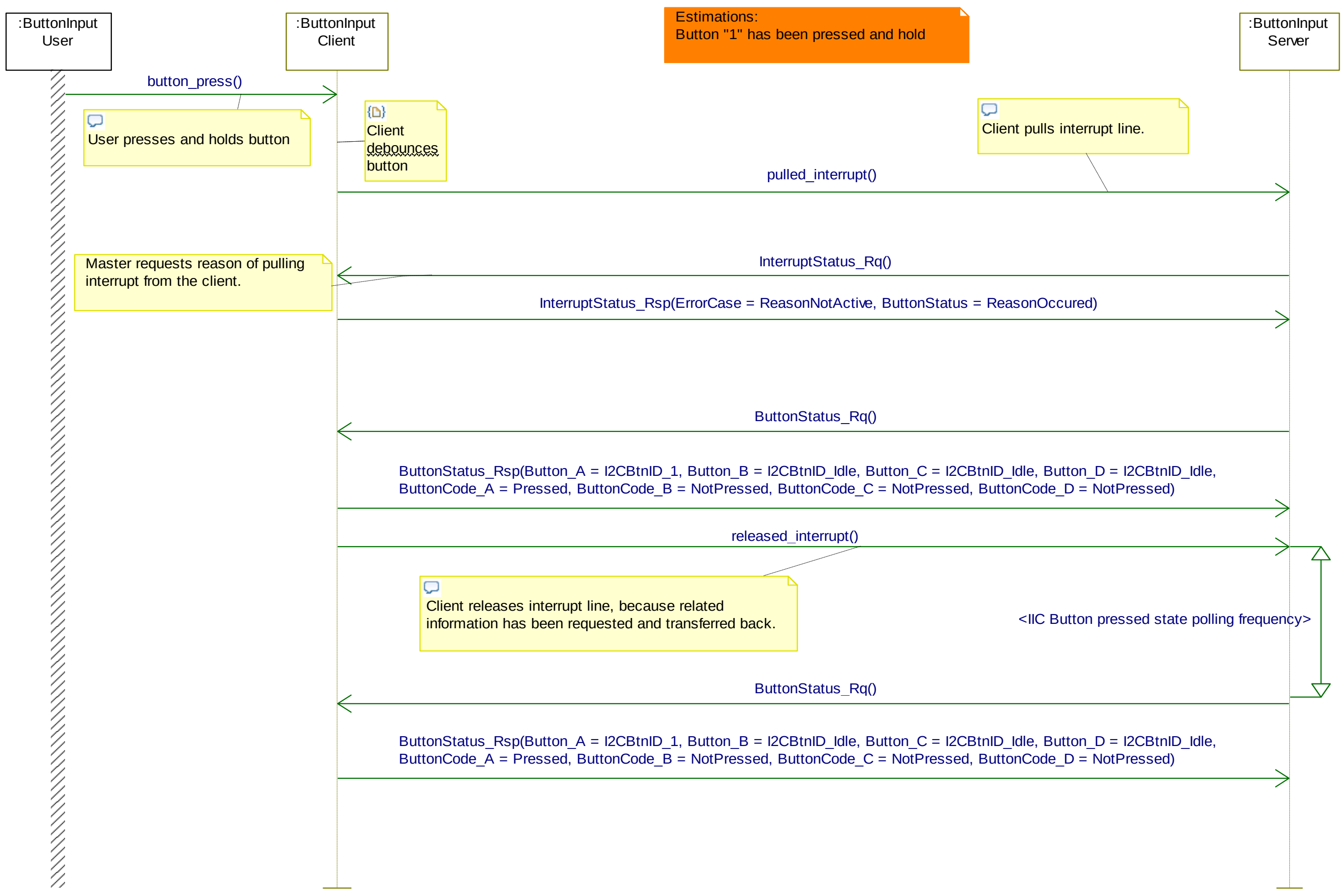
BCP_Button_Press (see Button Status Message)	Method from the Button Input Client to the receiving modules.	ButtonANameID (ButtonID_A) 0x0 - 0x255 see Input Translation Matrix ButtonAActivationState (ButtonCoding_A) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 reserved 0x3 reserved ButtonBNameID (ButtonID_B) 0x0 – 0x255 see Input Translation Matrix ButtonBActivationState (ButtonCoding_B) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 reserved 0x3 reserved ButtonCNameID (ButtonID_C) 0x0 – 0x255 see Input Translation Matrix ButtonCActivationState (ButtonCoding_C) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 reserved 0x3 reserved ButtonDNameID (ButtonID_D) 0x0 – 0x255 see Input Translation Matrix ButtonDActivationState (ButtonCoding_D) 0x0 Inactive / Not_Pressed 0x1 Active / Pressed 0x2 reserved 0x3 reserved See LVDS Communication Protocol specification for further information

3.2 General Requirements

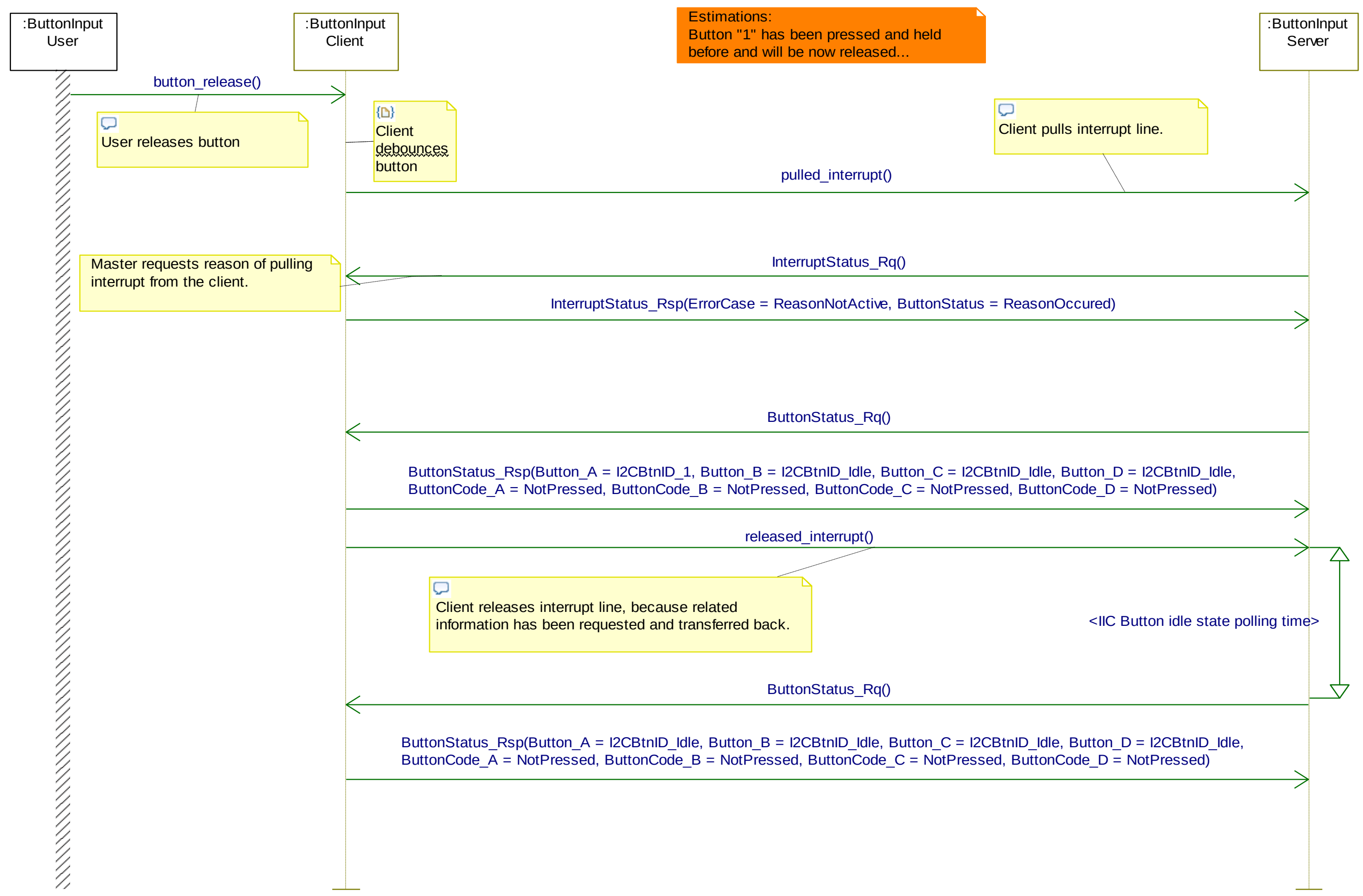
3.2.1 BUTTON-SR-REQ-141168/A-I2C Physical button state change sequence

If a button has been physically pressed or released and it has been de-bounced the Button Input Client has to pull the interrupt. Now the Button Input Server needs to request the reason of the interrupt. In this case the answer of the Button Input Server will be “button reason”. After that, the Button Input Server must request button status information.

Example 1:


:ButtonInput User
:ButtonInput Client
Estimations:
Button "1" has been pressed and hold
:ButtonInput Server
button_press()
User presses and holds button
Client debounces button
Client pulls interrupt line.
pulled_interrupt()
Master requests reason of pulling interrupt from the client.
InterruptStatus_Rq()
InterruptStatus_Rsp(ErrorCase = ReasonNotActive, ButtonStatus = ReasonOccured)
ButtonStatus_Rq()
ButtonStatus_Rsp(Button_A = I2CBtnID_1, Button_B = I2CBtnID_Idle, Button_C = I2CBtnID_Idle, Button_D = I2CBtnID_Idle, ButtonCode_A = Pressed, ButtonCode_B = NotPressed, ButtonCode_C = NotPressed, ButtonCode_D = NotPressed)
released_interrupt()
Client releases interrupt line, because related information has been requested and transferred back.
<IIC Button pressed state polling frequency>
ButtonStatus_Rq()
ButtonStatus_Rsp(Button_A = I2CBtnID_1, Button_B = I2CBtnID_Idle, Button_C = I2CBtnID_Idle, Button_D = I2CBtnID_Idle, ButtonCode_A = Pressed, ButtonCode_B = NotPressed, ButtonCode_C = NotPressed, ButtonCode_D = NotPressed)

Example 2:


:ButtonInput User
:ButtonInput Client
Estimations: Button "1" has been pressed and held before and will be now released...
:ButtonInput Server
button_release()
User releases button
Client debounces button
Client pulls interrupt line.
pulled_interrupt()
Master requests reason of pulling interrupt from the client.
InterruptStatus_Rq()
InterruptStatus_Rsp(ErrorCase = ReasonNotActive, ButtonStatus = ReasonOccured)
ButtonStatus_Rq()
ButtonStatus_Rsp(Button_A = I2CBtnID_1, Button_B = I2CBtnID_Idle, Button_C = I2CBtnID_Idle, Button_D = I2CBtnID_Idle, ButtonCode_A = NotPressed, ButtonCode_B = NotPressed, ButtonCode_C = NotPressed, ButtonCode_D = NotPressed)
released_interrupt()
Client releases interrupt line, because related information has been requested and transferred back.
<IIC Button idle state polling time>
ButtonStatus_Rq()
ButtonStatus_Rsp(Button_A = I2CBtnID_Idle, Button_B = I2CBtnID_Idle, Button_C = I2CBtnID_Idle, Button_D = I2CBtnID_Idle, ButtonCode_A = NotPressed, ButtonCode_B = NotPressed, ButtonCode_C = NotPressed, ButtonCode_D = NotPressed)

3.2.2 BUTTON-SR-REQ-141225/A-I2C Periodic button pressed state request

While at least one button is in "pressed state" up to the last button is released the Button Input Server has to poll button state with a frequency of defined in requirement "BUTTON-SR-REQ-141230-I2C Button Pressed state polling frequency".

3.2.3 BUTTON-SR-REQ-141230/A-I2C Button pressed state polling frequency

Button pressed state polling frequency has to be set to 5Hz (200ms).

3.2.4 BUTTON-SR-REQ-141232/A-I2C Button idle state polling time

Button idle state polling frequency has to be set to 1Hz (1000ms).

3.2.5 BUTTON-SR-REQ-141264/A-I2C Periodic button idle state request

While all buttons are in released state the Button Input Server has to poll button state with a frequency of defined in requirement "BUTTON-SR-REQ-141232-I2C Button Idle state polling time".

3.2.6 BUTTON-SR-REQ-141271/A-I2C Missing button status request

If a button status has been changed and the Button Input Server does not send an appropriate read request the Button Input Client must keep this information up to a timeout defined in requirement "BUTTON-SR-REQ-141277-I2C Button status change timeout". If the Button Input Server sends a read request after this time it will not get this information.

3.2.7 BUTTON-SR-REQ-141277/A-I2C Button status change timeout

The timeout must be 5 seconds

3.2.8 BUTTON-SR-REQ-142088/A-I2C Additional button status change while timeout

If another button status change occurs it will overwrite the old one if it relates to the same button.

Example 1:
Button B has been pressed (and held) but released before timeout occurs and Button Input Server reads after release, it just gets a "not pressed" of Button B.

Example 2:
Button B has been pressed (and held) but released before timeout occurs and Button C, too, and Button Input Server reads after release of both buttons, it just gets a "not pressed" of Button B and C.

3.2.9 BUTTON-SR-REQ-143036/A-I2C Button status change data to be kept while timeout

If a button status changes and the Button Input Server is not sending a read request it just needs to keep "pressed" and "release". The duration how long a button has been pressed does not need to be kept.

This information does not need to be kept after shutdown.

3.2.10 BUTTON-SR-REQ-132976/B-Reset ButtonID after button release (I2C)

If a button is "released" after "pressed" and an "inactive/Not_Pressed " was sent at least once then the corresponding ButtonID /ButtonNameID Byte will be changed to "Idle". If the ButtonID-Code is "Idle" then the appropriate ButtonCoding /ButtonActivationState Nibble is a don't care and should always be "Inactive/Not_Pressed".

3.2.11 BUTTONv3-SR-REQ-132977/B-I2C Message Structure Usage

When a button is activated it shall encode Button A first and if that position is already being used (ie pressed) shall move to the next Button position in the message. If all 4 buttons (A – D) are being used then any new button inputs shall be ignored until one of the 4 buttons are released.

Once a button press is active for a button ID (A – D) the BCP shall not move that same button press to another button ID location. For example if seek is being pressed and held and is assigned to 'Button C Name ID' it shall not change to 'Button A Name ID' while still being pressed.

The default to set Button Activation States A – D is 'Not Pressed / Inactive' unless there is a button activation event.

Example:
1. Button Module powers up and bus awake
2. Button Module sending: Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed
3. User presses button X
4. Button Module sending: Button A Name ID = X AND Button Activation State = Pressed
5. User releases button X
6. Button Module sends: Button A Name ID = X AND Button Activation State = Not Pressed at least once.
7. Button Module sends: Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed

8. Button Module continues to send (periodically) Button A Name ID = Inactive (Idle) AND Button Activation State = Not Pressed until the next button press. Any new button press changes Button A Name ID from Inactive (Idle) to the new button value.

3.2.12 BUTTON-SR-REQ-153567/A-I2C Button Architecture

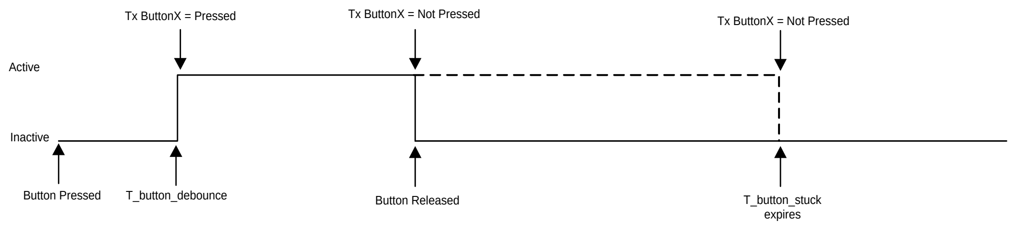
Button Input Server is part of LVDS Source module.
Button Input Client is part of LVDS Client module.

3.3 BUTTONv3-CLD-REQ-132978/B-Button Input Client (Button Transmitter) - I2C

The following sections define the Button Activation Strategy from the Transmitters perspective.

3.3.1 BUTTONv3-SR-REQ-132979/B-Transmitter Button Activation Process Timing (I2C)

The TBAP timing figure below will always remain true for any button activation event using the Pressed/Not Pressed normalization.



Transmitter Button Activation Process (TBAP) Timing

3.3.2 BUTTONv3-TMR-REQ-133003/B-T_reaction_time (I2C)

Name	Description	Units	Range	Resolution	Default
T_reaction_time (I2C)	The maximum transmitter reaction time from when a push button switch is closed until the push button message can be put on the I2C bus.	msec	0-1000	10	70

3.3.3 Push / Touch Button Activation - Transmitter Functional Requirements

3.3.3.1 BUTTONv3-SR-REQ-133004/B-Button Pressed / Not Pressed transmission (I2C)

Once a button is pressed and debounced on the transmitter the button message will be sent to the receiver with the button signal coding set to "Pressed".

Once the button is released the transmitter will set the button coding as "Not Pressed / Inactive".

3.3.3.2 BUTTON-SR-REQ-293497/A-Button Press reaction time (I2C)

The transmitter (ex.EFP, ICP, SWC, SDM) reaction time from when a push button switch is closed until the push button message is put on the bus shall not exceed T_reaction_time.

Note: this does not apply to the touch sense buttons. Reference applicable specifications for touch sense debounce requirements.

3.4 BUTTONv3-CLD-REQ-153566/B-Button Input Server (Button Receiver) - I2C

3.4.1 Button Type Definition (I2C)

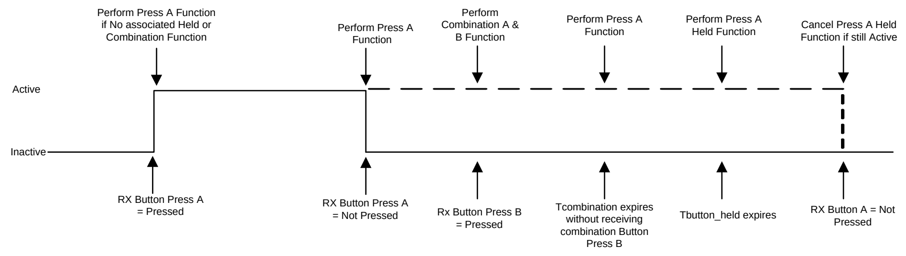
Singe Function – Single Function Buttons are buttons that can only map to a single outcome.

Combination – Combination Buttons are buttons that have a single outcome based on two simultaneous button presses.

3.4.2 Button Activation - Receiver Timing Requirements (I2C)

3.4.2.1 BUTTON-SR-REQ-293502/A-Receiver Button Activation Process (RBAP) Timing (I2C)

All component receivers of button press information shall implement the Receiver Button Activation Process (RBAP) timing defined in the figure below.


Perform Press A Function if No associated Held or Combination Function
Perform Press A Function
Perform Combination A & B Function
Perform Press A Function
Perform Press A Held Function
Cancel Press A Held Function if still Active
Active
Inactive
RX Button Press A = Pressed
RX Button Press A = Not Pressed
Rx Button Press B = Pressed
Tcombination expires without receiving combination Button Press B
Tbutton_held expires
RX Button A = Not Pressed

Receiver Button Activation Process (RBAP) Timing

Exception: If a particular button supports press and hold function there may be times where a functional requirement may require the function to first be performed on the press (not wait for not pressed) and then take additional action when Tbutton_held expires and the buttons are determined to be held. This should only be performed if explicitly called out in a functional requirement otherwise the function shall be performed only on a Not Press or when Tbutton_held expires as shown in the RBAP above.

3.4.3 BUTTON-SR-REQ-153568/A-I2C Evaluation of button press durations

If a button has been pressed for a short, long or stuck time must be evaluated by the Button Input Server.

3.4.4 Button Activation - Receiver Functional Requirements

3.4.4.1 BUTTON-SR-REQ-293503/A-Receivers of Button Presses follow RBAP (I2C)

All receivers of button press information shall activate the RBAP upon receipt of the button press message.

Note:
If the button press network messages are event-periodic with the event messages being transmitted "OnChange", the receiver modules for a particular signal that had "Press" sent on event shall not treat subsequent periodic "Press" encodings as a new RBAP event.

Put another way after the Button Receiver receives a Press for a particular button it shall wait for a Not Press for the same button before acting upon another button "Press" of the same button. After a Not Press then the Press of the same button can be acted upon again initiating a new RBAP event.

3.4.4.2 BUTTON-SR-REQ-153576/A-Button Receiver Sampling Rate - I2C

The sampling rate used by the receiver to read incoming button "Pressed" and "Not Pressed" messages shall be fast enough to read the multiple incoming messages (ex. could read multiple incoming pressed / not pressed messages).

3.4.4.3 *BUTTON-SR-REQ-293504/A-Receivers of Held Button Presses (I2C)*

The receiver shall determine whether the specific button has an associated held function.

3.4.4.4 *BUTTON-SR-REQ-293505/A-Button Presses with no Held function or Combination with another button press (I2C)*

If the button does not have an associated held or combination type function, then the receiver shall perform the function associated with a button press immediately upon receipt of the button press message and the RBAP process shall be exited.

3.4.4.5 *BUTTON-SR-REQ-293506/A-Buttons with Held Function (I2C)*

If the button does have an associated held function, the receiver shall start a hold timer (Tbutton_held) and wait for button 'not pressed' information.

-- If a button 'not pressed' message is not received prior to the expiration of Tbutton_held, the receiver shall perform the associated held function for that button press.

-- If a button 'not pressed' message is received prior to the expiration of Tbutton_held, then the receiver shall perform the associated press function.

3.4.4.6 *BUTTON-SR-REQ-293507/A-Receivers of combination type button presses (I2C)*

The receiver shall determine whether the button press has an associated combination type button press.

3.4.4.7 *BUTTON-SR-REQ-293508/A-Combination Type button press operation (I2C)*

When the receiver detects a button press (A) that has an associated combination the receiver must wait Tcombination before executing the button press function associated with the single button press (A).

-- If a second button (B) is received prior to Tcombination expiring, the receiving module must now determine if this (B) is part of a valid combination with (A).
- If the combination is valid, then the resulting combination (A + B) can be performed.
- If the combination is invalid, then no combination function is performed and the button presses (A) & (B) can be processed independently if allowed by the receiving module.

-- If Tcombination expires, then button press (A) is now valid and the (A) function can be performed.

3.4.4.8 *BUTTON-SR-REQ-293509/A-Cancelling RBAP (I2C)*

The receiver shall cancel the RBAP (Receiver Button Activation Process) upon:

- Receipt of the button 'Not Pressed' message.

- If the receiver does not receive a button 'Not Pressed' message within T_RBAP_Timeout
  - Unless noted otherwise all buttons shall timeout at some point in case of a stuck button
  - T_RBAP_Timeout may vary depending on how the button is used for a particular feature. The T_RBAP_Timeout would be part of error handling for a particular feature unless a T_RBAP_Timeout value is explicitly noted in a feature SPSS.
    - For example: if a feature does not have a press and hold or combination button press associated with it then that button might time out quickly. Other features like Seek/Volume have press and hold features associated with them and would not time out for a longer period of time. Follow up with the Ford HMI or Ford feature team on what make sense for timeout values.

Reference:
- For cancelling Volume Button RBAP reference requirements "VOL-TMR_REQ-292290-T_Vol_RBAP_Timeout" and "VOL-REQ-292289-Volume Press and Hold Timeout".

3.4.4.9 BUTTON-SR-REQ-293510/A-Cancelling RBAP when change to Standby (I2C)

For Infotainment receivers any event that shall cause a transition from Functional mode to Standby mode shall cancel a RBAP unless noted otherwise. The receiver shall cancel the operation, if active, and perform the required actions to enter Standby State.

4 Appendix: Reference Documents

Reference #	Document Title
1	Input Translation Matrix
2	LIN Specifications: LIN DVM specifications LIN Data LINK and Physical Layer specifications LIN Physical Layer Approved Components SAE J2602-3 SAE J2602-2 SAE J2602-1 LIN Database File (LDF) LIN Illumination Specification LIN Netcom SOW
3	I2C over LVDS Specifications I2C over LVDS Communication Protocol specification I2C over LVDS Illumination Specification I2C over LVDS Bezel Diagnostic SPSS specification
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	