BOSCH

From	Our Reference	Tel	Koramangala
RBEI/EEG1	Aditya Barve	+91(80)6657-6422	05 Sept 2018

Draft Requirements Specification

.Recipient

Cc

Topic **Draft API Definition for CATS V2.2 DLL.**

1 Scope

The following API is intended to be used by any Automated Test Frame work to interact with Compact Automated Test System. The Test Cases for Automated Test Frame work must be able to call this API functions from their code. The functions are in C#.net, so that they can be provided as a dynamic link library, which should be accessible from other standard scripts.

2 *API Description*

2.1 *Functions for I/O Channels Access*

2.1.1 *Function for ADC Measurement:*

readADCvoltage:

Service name:	*readADCvoltage*	
Syntax:	byte **readADCvoltage** (byte channelId, ref short voltage)	
Parameters (in):	channelId	Identifier of the ADC input channel, according to connector panel labelling(0-3)
Parameters (inout):	none	----------
Parameters (out):	voltage	Voltage in respective analog input channel in milli volts.
Return value:	byte	**0x10** if everything was OK and voltage contains a valid value. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions

Description:	This function can be used to measure ADC input voltage

2.1.2 Function to Control the DAC Analog Output:

writeDACoutput:

Service name:	*writeDACoutput*	
Syntax:	byte **writeDACoutput** (byte channelId, short Outputvoltage)	
Parameters (in):	channelId Outputvoltage	Identifier of the DAC output channel, according to connector panel labelling(0 to 19) Respective DAC output voltage in milli volts. (Range 0 to 10,000 milli volts).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used control DAC Output voltage.	

2.1.3 Function to Measure the DAC Analog Output:

readDACoutput:

Service name:	*readDACoutput*	
Syntax:	byte **readDACoutput** (byte channelId, ref short Outputvoltage)	
Parameters (in):	channelId	Identifier of the DAC output channel, according to connector panel labelling(0 to 19)
Parameters (in-out):	none	----------
Parameters (out):	Outputvoltage	Respective DAC output voltage in milli volts. (Range 0 to 10,000 milli volts).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used read DAC Output voltage.	

2.1.4 Function to Control the DAC Locked Output:

writeLockDACoutput:

Service name:	*writeLockDACoutput*	
Syntax:	byte **writeLockDACoutput** (byte channelIdX, byte channelIdY, short ChXoutputvoltage, short ChYoutputvoltage)	
Parameters (in):	ChXoutputvoltage ChYoutputvoltage	DACX output voltage in milli volts. (Range 0 to 10,000 milli volts). DACY output voltage in milli volts. (Range 0 to 10,000 milli volts).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used control DACX and DACY output voltage in locked mode (Useful functionality to simulate Acc pedal signals). Note: X and Y can be one of the following pairs: 0&1, 2&3, 4&5, 6&7, 8&9.	

Note: Ensure that Ch X is lower than Ch Y in the valid pair formation, else the API considers the input as invalid. i.e. for 0&1 pair, Ch X is 0 and Ch Y is 1.

2.1.5 Function to Measure the DAC Locked Output:

readLockDACoutput:

Service name:	*readLockDACoutput*	
Syntax:	byte **readLockDACoutput** (byte channelIdX, byte channelIdY, ref short ChXoutputvoltage, ref short ChYoutputvoltage)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	ChXoutputvoltage ChYoutputvoltage	DACX output voltage in milli volts. (Range 0 to 10,000 milli volts). DACY output voltage in milli volts. (Range 0 to 10,000 milli volts).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read DACX and DACY output voltage values in DAC locked mode. Note: X and Y can be one of the following pairs: 0&1, 2&3, 4&5, 6&7, 8&9.	

2.1.6 Function for Digital Input Configuration:

writeDigitalinputPinconfiguration:

Service name:	*writeDigitalinputPinconfiguration*	
Syntax:	byte **writeDigitalinputPinconfiguration** (byte channelId, byte Pinconfiguration)	
Parameters (in):	channelId	Identifier of the digital input channel, according to connector panel labelling(1 to 8)
	Pinconfiguration	Respective Digital Channel Input Configuration (0 – pulled up, 1 – pulled down, 2 – Vref).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure Digital Input pin.	

2.1.7 Function for Digital Input Measurement:

readDigitalinput:

Service name:	*readDigitalinput*	
Syntax:	byte **readDigitalinput** (byte channelId, ref byte Pinconfiguration, ref byte Pinstatus)	
Parameters (in):	channelId	Identifier of the digital input channel, according to connector panel labelling(1 to 8)
Parameters (in-out):	none	----------
Parameters (out):	Pinconfiguration Pinstatus	Respective Digital Channel Input Configuration (0– pulled up, 1 – pulled down, 2 – Vref). Respective Digital Channel Input Pin Status (0 – Detected Low, 1 – Detected High).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read Digital Input Status	

2.1.8 Function to Control the Digital Output:

writeDigitaloutput:

Service name:	*writeDigitaloutput*	
Syntax:	byte **writeDigitaloutput** (byte channelId, byte Outputvoltage, byte Outputstatus)	
Parameters (in):	channelId	Identifier of the digital output channel, according to connector panel labelling(1 to 8)
	Outputvoltage	Digital Channel individual Output Configuration (0 – 5Volts, 1 - UBD)
	Outputstatus	Respective Digital Channel Output Pin Status (0 – Low/Off, 1 – High/On).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used control Digital Output Status	

2.1.9 Function for Digital Output Measurement:

***readDigitaloutput*:**

Service name:	*readDigitaloutput*	
Syntax:	byte **readDigitaloutput** (byte channelId, ref byte Outputvoltage, ref byte Outputstatus)	
Parameters (in):	channelId	Identifier of the digital output channel, according to connector panel labelling(1 to 8)
Parameters (in-out):	none	----------
Parameters (out):	Outputvoltage Outputstatus	Individual Digital Channel Output Configuration (0 – 5Volts, 1 - UBD). Respective Digital Channel Output Pin Status (0 – Low/Off, 1 – High/On).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used read Digital Output Status	

2.1.10 Function to Control the Relay Output:

***writeRelayoutput**:*

Service name:	*writeRelayoutput*	
Syntax:	byte **writeRelayoutput** (byte channelId, byte Outputstatus)	
Parameters (in):	channelId	Identifier of the Relay channel, according to connector panel labelling (0 to 4)
	Outputstatus	Respective Relay Channel Output Pin Status (0 – Off, 1 – On).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used control Relay Nodes	

2.1.11 *Function to Measure Relay Output Status:*

***readRelayoutput*:**

Service name:	*readRelayoutput*	
Syntax:	byte **readRelayoutput** (byte channelId, ref byte Outputstatus)	
Parameters (in):	channelId	Identifier of the Relay channel, according to connector panel labelling (0 to 4)
Parameters (in-out):	none	----------
Parameters (out):	Outputstatus	Respective Relay Channel Output Pin Status (0 – Off, 1 – On).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read Relay Nodes Status.	

2.1.12 Function for PWM Input Configuration:

***writePWMinputPinconfiguration**:*

Service name:	*writePWMinputPinconfiguration*	
Syntax:	byte **writePWMinputPinconfiguration** (byte channelId, byte Pinconfiguration)	
Parameters (in):	channelId Pinconfiguration	Identifier of the PWM input channel, according to connector panel labelling(1 to 8) Respective PWM Channel Input Configuration (0 – pulled up, 1 – pulled down, 2 – Vref).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure PWM Input pin.	

2.1.13 *Function for PWM Input Measurement:*

readPWMinput:

Service name:	*readPWMinput*	
Syntax:	byte **readPWMinput** (byte channelId, ref byte Inputconfiguration, ref short Frequency, ref byte Dutycycle)	
Parameters (in):	channelId	Identifier of the PWM Input channel, according to connector panel labelling(1 to 8)
Parameters (in-out):	none	----------
Parameters (out):	Inputconfiguration	Respective PWM Channel Input Configuration (0 – pulled up, 1 – pulled down, 2 – Vref).
	Frequency	Measured Frequency Value (20Hz – 10 KHz).
	Dutycycle	Measured Duty cycle Value (5% - 95%).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to measure PWM Input signals.	

2.1.14 *Function to Control the PWM Output:*

writePWMoutput:

Service name:	*writePWMoutput*	
Syntax:	byte **writePWMoutput** (byte channelId, byte Outputvoltage, short Outputfrequency, byte Outputdutycycle)	
Parameters (in):	channelId	Identifier of the PWM output channel, according to connector panel labelling(1 to 8)
	Outputvoltage	PWM Channel Individual Output Configuration (0 – 5Volts, 1 - UBD)
	Outputfrequency	Respective PWM Channel Output Frequency (10Hz to 10KHz).
	Outputdutycycle	Respective PWM Channel Output Dutycycle. (0% to 100%).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to control PWM Output Signals.	

2.1.15 *Function for PWM Output Measurement:*

<u>readPWMoutput</u>:

Service name:	*readPWMoutput*	
Syntax:	byte **readPWMoutput** (byte channelId, ref byte Outputvoltage, ref short Outputfrequency, ref byte Outputdutycycle)	
Parameters (in):	channelId	Identifier of the PWM output channel, according to connector panel labelling(1 to 8)
Parameters (in-out):	none	----------
Parameters (out):	Outputvoltage	Individual PWM Channel Output Configuration (0 – 5Volts, 1 - UBD).
	Outputfrequency	Respective PWM Channel Output Frequency in Hz
	Outputdutycycle	Respective PWM Channel Output Duty cycle in %
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read PWM Output Signal Parameters	

2.1.16 Function to Control the H-Bridge Output:

***writeHBridgeoutput*:**

Service name:	*writeHBridgeoutput*	
Syntax:	byte **writeHBridgeoutput** (byte channelId, short Outputfrequency, byte Outputdutycycle)	
Parameters (in):	channelId	Identifier of the H-Bridge output channel, according to connector panel labelling (HB1+, HB1-, HB2+, HB2-).
	Outputfrequency	Respective H Bridge Channel Output Frequency (10Hz to 10KHz).
	Outputdutycycle	Respective H Bridge Channel Output Dutycycle. (0% to 100%).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to control H-Bridge Output Signals.	

2.1.17 Function for H-Bridge Output Measurement:

readHBridgeoutput:

Service name:	*readHBridgeoutput*	
Syntax:	byte **readHBridgeoutput** (byte channelId, ref short Outputfrequency, ref byte Outputdutycycle)	
Parameters (in):	channelId	Identifier of the H-Bridge output channel, according to connector panel labelling (HB1+, HB1-, HB2+, HB2-).
Parameters (in-out):	none	----------
Parameters (out):	Outputfrequency Outputdutycycle	Respective H-Bridge Channel Output Frequency in Hz Respective H-Bridge Channel Output Duty cycle in %
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read H-Bridge Output Signal Parameters	

2.1.18 Function for Power Supply Measurement:

readPowersupply:

Service name:	*readPowersupply*	
Syntax:	byte **readPowersupply** (ref short Powersupplyvoltage)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	Powersupplyvoltage	Input Power supply voltage reading in milli volts.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to measure input power supply voltage.	

2.1.19 Function to Reset CATS Hardware:

***executeResetHardware*:**

Service name:	*executeResetHardware*
Syntax:	byte **executeResetHardware**()
Parameters (in):	none ----------
Parameters (in-out):	none ----------
Parameters (out):	none ----------
Return value:	byte **0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to reset the CATS unit.

2.1.20 *Function for Handshake with CATS Hardware:*

execute Handshake:

Service name:	*executeHandshake*	
Syntax:	byte **executeHandshake**()	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to execute handshake with CATS unit.	

2.1.21 Function to Read Firmware Version:

***readFirmwareversion**:*

Service name:	*readFirmwareversion*	
Syntax:	byte **readFirmwareversion** (ref string FirmwareID)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	FirmwareID	Firmware ID in string Format.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the firmware ID from CATS unit.	

2.1.22 Function to Configure Miscellaneous Inputs:

writeMiscellaneousinputPinconfiguration:

Service name:	*writeMiscellaneousinputPinconfiguration*	
Syntax:	byte **writeMiscellaneousinputPinconfiguration** (byte channelId, byte configuration)	
Parameters (in):	channelId configuration	Identifier of the Miscellaneous Input channel. 0 – Main Relay, 1 – Ignition, 2 – Injector. Miscellaneous Channel Input Pin Configuration 0 - Pull down/Active High. 1 - Pull up/Active Low.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure the miscellaneous input pins.	

2.1.23 Function to Measure Miscellaneous Inputs:

***readMiscellaneousinput**:*

Service name:	*readMiscellaneousinput*	
Syntax:	byte **readMiscellaneousinput** (byte channelId, ref byte configuration, ref byte inputstatus)	
Parameters (in):	channelId	Identifier of the miscellaneous input channel. 0 – Main Relay, 1 – Ignition, 2 – Injector.
Parameters (in-out):	none	----------
Parameters (out):	configuration	Miscellaneous Input Channel Pin Configuration 0 - Pull down/Active High. 1 - Pull up/Active Low.
	inputstatus	Miscellaneous Channel Input Pin Status 0 - Detected Logic Low. 1 - Detected Logic High.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to measure the miscellaneous input pin configuration setting and pin status.	

2.1.24 Function to Control Engine Speed RPM:

writeESMRPM:

Service name:	*writeESMRPM*	
Syntax:	byte **writeESMRPM** (short RPMValue)	
Parameters (in):	RPMValue	Engine speed signal RPM value.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to control RPM of Engine speed output.	

Note: User to take care that the RPM Value is within the limits 0-10000. An entry beyond these values might result in un-predictable behaviour.

2.1.25 *Function to Measure Engine Speed RPM:*

readESMRPM:

Service name:	*readESMRPM*	
Syntax:	byte **readESMRPM** (ref short RPMValue)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	RPMValue	Engine speed signal RPM value.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to measure RPM of Engine speed output.	

2.1.26 Function to Control CAM Phase:

***writeCAMPhaseshift*:**

Service name:	*writeCAMPhaseshift*	
Syntax:	byte **writeCAMPhaseshift** (double Phaseshiftvalue)	
Parameters (in):	Phaseshiftvalue	CAM Phase shift with respect to Crank.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to change CAM phase with respect to Crank.	

Note: The above API is applicable only for adjusting phase shift of CAM1.

2.1.27 Function to Measure CAM Phase:

***readCAMPhaseshift**:*

Service name:	*readCAMPhaseshift*	
Syntax:	byte **readCAMPhaseshift** (ref double Phaseshiftvalue)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	Phaseshiftvalue	CAM Phase shift with respect to Crank.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read CAM phase with respect to Crank.	

Note: An angle of -180° is equivalent to +180 ° hence only for the -180 phase shift the device returns +180 degrees. Reads phase shift of only CAM1 with respect to Crank.

2.1.28 Function for Selection of ESM Pattern:

***executeESMPattern*:**

Service name:	*executeESMPattern*	
Syntax:	byte **executeESMPattern** (byte PatternNo, ref string PatternName)	
Parameters (in):	PatternNo	Engine Speed Pattern number (1 to 6)
Parameters (in-out):	none	----------
Parameters (out):	PatternName	Pattern Name corresponding to selected pattern.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to select and generate the ESM Pattern stored inside memory of a CATS unit.	

2.1.29 Function for reading Selected ESM Pattern:

readexecutingESMPattern:

Service name:	*readexecutingESMPattern*	
Syntax:	byte **readexecutingESMPattern** (ref byte PatternNo, ref string Pattern)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	PatternNo Pattern	Pattern Number of currently running pattern Pattern Name corresponding to running pattern.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the details of a currently executing Engine speed pattern.	

2.1.30 Function for Selection of Crank Type:

***writeCrankType*:**

Service name:	*writeCrankType*	
Syntax:	byte **writeCrankType** (byte CrankMode, byte DG23iMode, byte DG23iDirection)	
Parameters (in):	CrankMode DG23iMode DG23iDirection	Crank Mode: 0 – Differential, 1 – TTL. DG23i Mode: 0 – Off, 1 – On. DG23i Direction: 0 – Forward, 1 – Backward (DG23i mode and direction Works only in TTL Mode).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to set the Crank type in engine speed signal.	

2.1.31 *Function for Reading Running Crank Type:*

***readCrankType**:*

Service name:	*readCrankType*	
Syntax:	byte **readCrankType** (ref byte CrankMode, ref byte DG23iMode, ref byte DG23iDirection)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	CrankMode DG23iMode DG23iDirection	Crank Mode: 0 – Differential, 1 – TTL. DG23i Mode: 0 – Off, 1 – On. DG23i Direction: 0 – Forward, 1 – Backward (DG23i mode and direction values are valid only in TTL Mode, In differential mode these are don't care values).
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the Crank type of running engine speed signal.	

2.1.32 Function for Selection of CAM Type:

writeCAMType:

Service name:	*writeCAMType*	
Syntax:	byte **writeCAMType** (byte CAM1mode, byte CAM2mode, byte CAM3mode, byte CAM4mode)	
Parameters (in):	CAM1mode CAM2mode CAM3mode CAM4mode	Mode of CAM1 to CAM4 Output. Mode: 0 – Differential, 1 - TTL.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to set the CAM type of engine speed signal Output.	

2.1.33 Function for Reading Running CAM Type:

***readCAMType**:*

Service name:	*readCAMType*
Syntax:	byte **readCAMType** (ref byte CAM1mode, ref byte CAM2mode, ref byte CAM3mode, ref byte CAM4mode)
Parameters (in):	none ----------
Parameters (in-out):	none ----------
Parameters (out):	CAM1mode CAM2mode CAM3mode CAM4mode Mode of running CAM1 to CAM4 Outputs. Mode: 0 – Differential, 1 - TTL.
Return value:	byte **0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the CAM type in current running engine speed signal.

2.1.34 Function for Changing Crank Voltage in Differential Mode:

writeCrankDifferentialvoltage:

Service name:	*writeCrankDifferentialvoltage*	
Syntax:	byte **writeCrankDifferentialvoltage** (short Voltage)	
Parameters (in):	Voltage	Crank Differential Output Voltage (0 to 10 Volts) (Will be represented in milli volts)
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to set the Crank output voltage in differential mode.	

2.1.35 Function for Reading back Crank Voltage in Differential Mode:

readCrankDifferentialvoltage:

Service name:	*readCrankDifferentialvoltage*	
Syntax:	byte **readCrankDifferentialvoltage** (ref short Voltage)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	Voltage	Crank Differential Output Voltage (0 to 10 Volts) (Will be represented in milli volts)
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the Crank output voltage in differential mode.	

2.1.36 Function for starting ESM Ramp:

***startESMRamp**:*

Service name:	*startESMRamp*	
Syntax:	byte **startESMRamp** (short startRPM, short endRPM, short voltage, short duration)	
Parameters (in):	startRPM endRPM voltage duration	Ramp mode ESM Start RPM Ramp mode ESM End RPM Ramp mode ESM signal voltage (Works only in Differential Mode) Ramping Time Duration
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to start ESM in Ramping mode.	

2.1.37 Function for stopping ESM Ramp:

***stopESMRamp**:*

Service name:	*stopESMRamp*	
Syntax:	byte **stopESMRamp** (ref short endRPM)	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	endRPM	ESM End RPM after stop in Ramping mode.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to stop ESM while in Ramping mode.	

2.1.38 Function for configuring CAN module:

***configureCANmodule**:*

Service name:	*configureCANmodule*	
Syntax:	byte **configureCANmodule** (short Baudrate)	
Parameters (in):	Baud rate	CAN baud rate parameter in kbps. Supported Baud rates are: 100 kbps, 125 kbps, 250 kbps, 500 kbps, 666kbps, 1000 kbps.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure CAN Baud rate.	

2.1.39 Function for sending CAN message:

***writeCANmessagecyclic**:*

Service name:	*writeCANmessagecyclic*	
Syntax:	byte **writeCANmessagecyclic**(short Datarate, byte IDType, byte Datalength, byte MsgObject, long MsgID, long Mask, byte Data1, byte Data2, byte Data3, byte Data4, byte Data5, byte Data6, byte Data7, byte Data8)	
Parameters (in):	Datarate	Message transmission interval (1-255 milli Sec).
	IDType	Identifier type (0- standard, 1- Extended).
	Datalength	Number of Total Data Bytes in the ID (1-8).
	MsgObject	CAN message object (0x01-0x0F).
	MsgID	Message ID.
	Mask	Message Mask.
	Data1 to Data8	CAN data pay load bytes.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to trigger CAN frames from CATS.	

2.1.40 *Function for reading Data from CAN objects:*

readCANmessageobject:

Service name:	*readCANmessageobject*	
Syntax:	byte **readCANmessageobject**(byte MsgObject, ref byte ObjectType, ref byte IDType,ref byte Datalength, ref long MsgID, ref long Mask, ref byte Data1, ref byte Data2, ref byte Data3, ref byte Data4, ref byte Data5, ref byte Data6, ref byte Data7, ref byte Data8)	
Parameters (in):	MsgObject	CAN message object (0x01-0x0F).
Parameters (in-out):	none	----------
Parameters (out):	ObjectType	CAN Object type (0- Transmit, 1- Receive).
	IDType	Identifier type (0- standard, 1- Extended).
	Datalength	Number of Total Data Bytes in the ID (1-8).
	MsgID	Message ID.
	Mask	Message Mask.
	Data1 to Data8	CAN data pay load bytes.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read CAN object configuration.	

Note: For the current release of the API, the parameters ObjectType, ID Type, DataLength, MsgId and Mask are not updated. Only Data1 – Data8 is updated as the DLC set during configuration. User has to maintain a copy of the 'not updated' parameter for their use.

2.1.41 Function for configure CAN Rx Filter:

writeCANRxFilterconfiguration:

Service name:	*writeCANRxFilterconfiguration*	
Syntax:	byte **writeCANRxFilterconfiguration** (byte MsgObject, long MsgID, long Mask, byte IDType, byte Datalength)	
Parameters (in):	MsgObject	CAN message object (0x01 - 0x0F).
	MsgID	Message ID.
	Mask	Message Mask.
	IDType	Identifier type (0- standard, 1- Extended).
	Datalength	Number of Total Data Bytes in the ID (1-8).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure CAN Rx Filter.	

Note: CAN Mask is taken care internally. As only the messages/ Message ID's configured are only received on CATS end.

2.1.42 Function for reading CAN Rx data:

readCANRxData:

Service name:	*readCANRxData*	
Syntax:	byte **readCANRxData** (byte MsgObject, ref long MsgID, ref byte IDType, ref byte Data1, ref byte Data2, ref byte Data3,ref byte Data4, ref byte Data5, ref byte Data6, ref byte Data7, ref byte Data8).	
Parameters (in):	MsgObject	CAN message object (0x01 - 0x0F).
Parameters (in-out):	none	----------
Parameters (out):	MsgID IDType Data1 to Data8	Message ID. Identifier type (0- standard, 1- Extended). Received Data Bytes.
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to read the received CAN data.	

Note: For the current release of the API, the parameters IDType, MsgId are not updated. Only Data1 – Data8 is updated according to the DLC set during configuration. User has to maintain a copy of the 'not updated' parameter for their use.

2.1.43 Function for stopping CAN Object from TX/RX:

stopCANObject:

Service name:	*stopCANObject*	
Syntax:	byte **stopCANObject** (byte MsgObject).	
Parameters (in):	MsgObject	CAN message object (0x01 - 0x0F).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to stop RX / TX from a configured CAN object.	

2.1.44 Function for Disabling CAN module:

closeCANModule:

Service name:	*closeCANModule*	
Syntax:	byte **closeCANModule**().	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to disable CAN module in CATS unit.	

Note: With the current API's only CAN Channel 1 is supported.

2.1.45 *Function For Initiating SENT Sensor Transmission:*

writeSENTdata:

Service name:	*writeSENTdata*	
Syntax:	byte **writeSENTdata** (byte status, byte nibble1, byte nibble2, byte nibble3, byte nibble4, byte nibble5, byte nibble6, byte tick, byte framelengthtype, short framelength, byte lowpulselegnth, byte crc_mode, byte polarity, byte sensortype).	
Parameters (in):	status	Status Value: Range is 0 to 3.
	nibble1 to nibble6	Data Nibbles: Range of each nibble is 0 to 15.
	tick	Basic SENT clock tick, (3 to 90 microseconds).
	framelengthtype	Frame format. 0 - Variable, 1 - Fixed.
	framelength	Frame Length: applicable if Frame format is 1. Range is – 282 to 922 microseconds.
	lowpulselength	Low Pulse Length: Range is 5 to 10.
	crc_mode	CRC Calculation method: 0 – Recommended CRC, 1 - Legacy CRC
	polarity	Active Signal polarity: 0 – Active High, 1 – Active Low.
	sensortype	SENT Sensor type: 0 – Sensor Type1 1 – Sensor Type2 2 – Sensor Type3 3 – Sensor Type4 4 – Sensor Type5
Parameters (in-out):	none	----------
Parameters (out):	none	----------

Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to initiate SENT Transmission.	

Note: The nibble arrangement has to be taken care by the user SENT data when received nibble wise looks like "status | nib1 | nib2 | nib3 | nib4 | nib5 | nib6 | crc", nibble values should be arranged according to specific requirement.

2.1.46 Function For Stopping SENT Sensor Simulation:

stopSENTTransmission:

Service name:	*stopSENTTransmission*	
Syntax:	byte **stopSENTTransmission** ().	
Parameters (in):	none	----------
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to stop SENT Transmission.	

Note: With the current set of API's only SENT Channel 1 is supported.

2.1.47 Function For Saving Pattern from CATS Unit to a File:

saveESMPattern:

Service name:	*saveESMPattern*	
Syntax:	byte **saveESMPattern** (byte Patternslot , string Filepath).	
Parameters (in):	Patternslot Filepath	ESM Pattern slot number (1 to 6) File name with a valid file path for saving ESM Pattern.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to save ESM pattern inside CATS unit to a user defined Pattern file. It will be stored in .txt format.	

2.1.48 Function For Loading ESM Pattern from a File to CATS Unit:

loadESMPattern:

Service name:	*loadESMPattern*	
Syntax:	byte **loadESMPattern** (byte Patternslot, string Filepath).	
Parameters (in):	Patternslot Filepath	ESM Pattern slot number (1 to 6) File name with a valid file path to a saved ESM pattern file.
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to load ESM pattern from a pattern file to the Respective CATS Hardware ESM pattern Slot.	

2.1.49 *Function for Digital/PWM Input Configuration:*

writeDigPWMinputPinconfiguration:

Service name:	*writeDigPWMinputPinconfiguration*	
Syntax:	byte **writeDigPWMinputPinconfiguration** (byte channelId, byte Pinconfiguration)	
Parameters (in):	channelId	Identifier of the digital/PWM input channel, according to connector panel labelling(1 to 8)
	Pinconfiguration	Respective Digital/PWM Channel Input Configuration (0 – configure as Digital input, 1 – Configure as PWM input).
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to configure multiplexed input lines as Digital or PWM channels.	

2.1.50 Function to Configure the Digital/PWM Output:

***configDigPWMoutput*:**

Service name:	*configDigPWMoutput*	
Syntax:	byte **configDigPWMoutput** (byte channelId, byte outputtype)	
Parameters (in):	channelId Outputtype	Identifier of the digital output channel, according to connector panel labelling(1 to 8) Digital/PWM channel Output type Configuration (0 – DIgital, 1 - PWM)
Parameters (in-out):	none	----------
Parameters (out):	none	----------
Return value:	byte	**0x10** if everything was OK. **0x20** in case device not detected. **0x22** in case of invalid input parameters. **0x25** any other exceptions
Description:	This function can be used to set the multiplexed output channels as digital or PWM.	

RBEI/EEG1