ECU 1 : Input Control Unit

API	Name	Description	Parameters	Return
Sensor Manager	SensorManager_t	Structure contains(2 pointers to functions)	Creating an instance from this structure to start calling functions inside sensor manager: SensorManager_t S; S.initSensor(Door_Sensor);	
	void initSensor(SensorState_t S)	Initialize any sensor by calling it's driver to initialize it with GPIO Pins	Specifiy which sensor you want to initialize you Have 3 Options for this project: Door_Sensor, Speed_Sensor, Light_Switch	Void
	SensorState_t readSensor(SensorState_t S)	Read the value of specific sensor by calling it's driver to read it with GPIO Pins	Specifiy which sensor you want to read you Have 3 Options for this project: Door_Sensor, Speed_Sensor, Light_Switch	Return a value of reading sensor SensorState_t = unsigned char
Door_Sensor	void initDoorSensor(void);	Initialize Door sensor by GPIO Driver	Void	Void
	DoorState_t getDoorState(void);	Read the value of specific sensor by calling GPIO Driver	Void	Return a value of reading sensor DoorState_t = unsigned char
Speed_Sensor	void initSpeedSensor(void);	Initialize Speed sensor by GPIO Driver	Void	Void
	SpeedState_t getSpeedState(void);	Read the value of specific sensor by calling GPIO Driver	Void	Return a value of reading sensor SpeedState_t = unsigned char
Light_Switch	void initLightSwitch(void);	Initialize Light Switch by GPIO Driver	Void	Void
	SwitchState_t getSwitchState(void);	Read the value of specific sensor by calling GPIO Driver	Void	Return a value of reading sensor SwitchState_t = unsigned char
GPIO	void GPIO_init(portX_t PortName, pinX_t pinNum, pinDir_t mode)	Initialize pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin , pinDir_t: Specify Input or Output	VOID
	pinState_t GPIO_readPin(portX_t PortName, pinX_t pinNum)	Read pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin	pinState_t: return the value of pin HIGH or LOW
	void GPIO_writePin(portX_t portName, pinX_t pinNum, pinState_t pinState)	Write pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin , pinState_t: Value of pin HIGH or LOW	void
Communication Manager	CommunicationManager_t	Structure contains(3 pointers to functions)	Creating an instance from this structure to start calling functions inside Communication Manager: CommunicationManager_t Serial; Serial.initSerial(void);	
	void initSerial(Protocol_t protocol);	Initialize any Serial Communication found in System	Protocol_t protocol: Specify which Serial communication you will use to intialize it	Void
	void Send_message(Protocol_t protocol,unsigned char* message, unsigned int size);	Sending Message to serial communication	Protocol_t protocol: Specify Which Serial unsigned char* message: your message unsigned int size: size of your message	Void
	void Recieve_message(Protocol_t protocol,unsigned char* message, unsigned int size);	Receive Message from serial communication	Protocol_t protocol: Specify Which Serial unsigned char* message: your message unsigned int size: size of your message	Void
CAN	void CAN_init(void)	Initialize CAN Protocol	Void	Void
	int_fast32_t CAN_write(CAN_Handle handle, const void *buffer, size_t size)	Write Data on CAN BUS	CAN_Handle handle: Passed Configurations const void *buffer: Your message size_t size: Size	int_fast32_t: signed int return success or invalid
	int_fast32_t CAN_read(CAN_Handle handle, void *buffer, size_t size)	Read Data From CAN BUS	CAN_Handle handle: Passed Configurations const void *buffer: Your message size_t size: Size	int_fast32_t: signed int return success or invalid

ECU 2 : Output Control Unit

API	Name	Description	Parameters	Return
ExtDvsMng	DevicesManager_t	Structure contains(3 pointers to functions)	Creating an instance from this structure to start calling functions inside External Device manager: DevicesManager_t Device; Device.initDevice(Buzzer_device);	
	void initDevice(Device_t Device)	Initialize any Device by calling it's driver to initialize it with GPIO Pins	Specifiy which Device you want to initialize you Have 2 Options for this project: Buzzer_device, Light_device	Void
	void writeOutput(Device_t Device, Device_t DeviceState)	Write High or Low on specified Device	Device_t Device: Specifiy which Device Device_t DeviceState: HIGH or LOW	Void
	Device_t readOutput(Device_t Device)	Read Value of specified device	Device_t Device: Specifiy which Device	Return a value of reading device Device_t = unsigned char
Buzzer	void initBuzzer(void)	Initialize Buzzer Device by GPIO Driver	Void	Void
	void setBuzzerState(BuzzerState_t Buzzer)	Set a value to Buzzer by Calling GPIO Driver	BuzzerState_t Buzzer: HIGH or LOW	Void
	BuzzerState_t getBuzzerState(void)	Read the value of Buzzer by calling GPIO Driver	Void	Return the Value of Buzzer state
Lights	void initLights(void)	Initialize Light Device by GPIO Driver	Void	Void
	void setLightState(LightState_t Lights)	Set a value to Lights by Calling GPIO Driver	LightState_t Lights: High or LOW	Void
	LightState_t getLightState(void)	Read the value of Lights by calling GPIO Driver	Void	Return the Value of Light state
GPIO	void GPIO_init(portX_t PortName, pinX_t pinNum, pinDir_t mode)	Initialize pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin , pinDir_t: Specify Input or Output	VOID
	pinState_t GPIO_readPin(portX_t PortName, pinX_t pinNum)	Read pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin	pinState_t: return the value of pin HIGH or LOW
	void GPIO_writePin(portX_t portName, pinX_t pinNum, pinState_t pinState)	Write pin inside GPIO Driver	portX_t: Passing PORT name , pinX_t: Passing which pin , pinState_t: Value of pin HIGH or LOW	void
Communication Manager	CommunicationManager_t	Structure contains(3 pointers to functions)	Creating an instance from this structure to start calling functions inside Communication Manager: CommunicationManager_t Serial; Serial.initSerial(void);	
	void initSerial(Protocol_t protocol);	Initialize any Serial Communication found in System	Protocol_t protocol: Specify which Serial communication you will use to intialize it	Void
	void Send_message(Protocol_t protocol,unsigned char* message, unsigned int size);	Sending Message to serial communication	Protocol_t protocol: Specify Which Serial unsigned char* message: your message unsigned int size: size of your message	Void
	void Recieve_message(Protocol_t protocol,unsigned char* message, unsigned int size);	Receive Message from serial communication	Protocol_t protocol: Specify Which Serial unsigned char* message: your message unsigned int size: size of your message	Void
CAN	void CAN_init(void)	Initialize CAN Protocol	Void	Void
	int_fast32_t CAN_write(CAN_Handle handle, const void *buffer, size_t size)	Write Data on CAN BUS	CAN_Handle handle: Passed Configurations const void *buffer: Your message size_t size: Size	int_fast32_t: signed int return success or invalid
	int_fast32_t CAN_read(CAN_Handle handle, void *buffer, size_t size)	Read Data From CAN BUS	CAN_Handle handle: Passed Configurations const void *buffer: Your message size_t size: Size	int_fast32_t: signed int return success or invalid