maxAI 430 Design Studio Software Guide

Contents

Revision History

Version	Change Description	Date	Author	Approve
2.0	Release for Beta Testing	Jul 5, 2022	Victor Rios	Francisco Lopez
2.1	Adde steps to open the SDK project	July 12, 2022	Victor Rios	
2.2	Update Light Sensor section	September 26, 2022	Emmanuel Hernandez	
2.3	Overall corrections. See attached excel file for details:	Oct 21st, 2022	Luis Figueroa	Victor Rios
2.4	Added section “Updates and fixes”	Oct 31, 2022	Victor Rios	Victor Rios
2.5	Updated STM32 Cube IDE and Touch GFX Software Versions to 1.11.0 and 4.22.0 respectively. Improved document readability.	June 21, 2023	Victor Trimmer	Victor Rios
2.6	Updated memory information per section. Added section “flashing guide” via USB and CAN. Added section “Project Migration” to update old projects with the new SDK template.	September 05, 2023	Eduardo Martinez	Victor Rios

Acronyms & Abbreviations

Acronyms	Definition
AI430	maxAI 430
API	Application Programming Interface
BLE	Bluetooth Low Energy
CAN	Control Network Area
DB	Data Base
DIO	Digital Output
DL	Data Layer
DM	Diagnostic Message
IDE	Integrated Development Kit
LCD	Liquid Crystal Display

LED	Light Emitting Diode
NTSC	National Television System Committee
PAL	Phase Alternate Line
RTC	Real Time Clock
SDK	Software Development Kit (maxAI 430 Design Studio)
USB	Universal Serial Bus
WL	Warning Lights

Maximatecc Software Overview

maxAI™ Configurator

For quick and easy setup, use the Configurator Tool to automatically populate your engine monitoring data with preset options and layouts. No need for complex coding or additional resources.

maxAI™ Design Studio

The Design Studio is a Software Development Kit (SDK) that provides a higher level of flexibility and control. You choose the advanced engine monitoring parameters to equip your display with all the information you need to know.

maxAI™ Specialized

For larger projects the maximatecc engineering team can help with the development of a custom interface that meets specific application needs. The team can support all elements of the engineering and setup process for ease and flexibility. To receive support and a quotation please reach out to your established maximatecc agent with your Software and Hardware Specifications.

Purpose of document

The purpose of this document is to enable an application developer to write TouchGFX applications for the maxAI 430 SDK using the features, modules, interfaces, and possible configurations that are available with the maxAI 430 SDK hardware platform.

Scope of document

The scope of the document is to list all the features and functionalities of the maxAI 430 SDK which are of relevance to the Touch GFX application developer using the SDK.

Updates and Fixed for maxAI 430 SDK 1.0.0

- Camera Video
  - o Added support to PAL format (before only NTSC format was supported)
  - o Increased framerate
- Bluetooth
  - o Fixed issues regarding connectivity with mobile devices
- Power monitoring
  - o Added internal microcontroller temperature reading
- System
  - o Improve boot time
  - o Moved the files that the user normally will modify into a folder called “user_modfy_files”.
- Keypad
  - o Added continuous pressed state
- Watchdog
  - o Disable the independent watchdog in stop mode to avoid a reset
- Light sensor
  - o Fixed one shot mode and corrected equation to convert the data from the sensor
- EEPROM
  - o Moved shadow EEPROM from external SDRAM to internal RAM to void conflicts with frame buffers

Updates and fixes for maxAI 430 SDK 1.0.1

- Migrate STMCubeIde from 1.8.0 to 1.11.0
  o Minor adjustments to adapt to the gcc compiler from the new STM32CUBE IDE versions (1.8 to 1.11)
- Migrate TouchGFX designer from 4.18.1 to 4.22.
  o Minor adjustments migrate to touchgfx 4.18.1 to 4.22.
- Added code to analyze stack when a hard fault occurs.
  o Added the fault stack pulling to check the program counter, link register and program counter status
- Fixed correct size of the external flash.
  o The configuration for the external flash had an incorrect size
- Removed dummy variables from the data layer.
  o The dummy variables were removed from the data layer to avoid enumeration issues with the USB/BLE debuggers
- Fixed issue queue overrun in the platform services calls.
  o Relocated the memory freeing for the allocated memory from the sdk_api to each platform service
- Fixed enumeration issue of the data layer in the PC and mobile debuggers
  o Corrected the enumeration of the data layer
- Added jump to bootloader command.
  o Currently active on the terminal conditional of the ECU reset.

Introduction

The AI430 SDK platform is an embedded software solution for custom applications based on the AI430 hardware only. This platform provides a set of software components to reduce the development effort to create a complete embedded application compliant with all the customer requirements. The SDK solution potentiates the scope of the AI430 platform. The user can explore all the possibilities to cover the requirements and needs by using the AI430 peripherals and by creating their own custom graphical applications.

The SDK platform has the following benefits:

• Short development time	• Pre-established low level driver administration.
• Portable software components	• Low technical development skills required
• Pre-configured and stable SW architecture.	• Secure custom and private algorithms implementation.

SDK Setup and installation

To get started with the AI430 SDK, you will need to setup the right environment. Please follow the procedure described in this section to install the necessary tools required to use the SDK and create a TouchGFX application.

SDK Development Environment

The AI430 SDK allows TouchGFX Applications to be custom built on the AI430 platform. Please ensure the below hardware and software setup is available.

Hardware Requirements

Host PC	WINDOWS (64-bit OS)
RAM Size	4 GB RAM required minimum
Disk Space	2 GB disk space required minimum
Board with Power Supply	MAXAI430 kit
Debugger	ST Link V2 in-Circuit debugger with USB cable

Software Requirements

Development IDE	STM32 Cube (1.11.0)
Development IDE	TOUCH GFX (4.22.0)
Software package	S/W package released with the MAXAI430 kit

If another version of TouchGFX it used it will require a migration (from version 4.18.1 to the newer one) and additional modifications in the code, for this reason its recommended to use the versions of the software previously mentioned.

IDE Installations

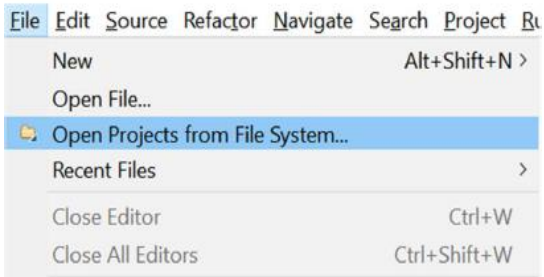
To get started with the AI430 SDK, please follow the below links to install the STM32 Cube IDE and the Touch GFX IDE.

1) Install the STM32 Cube IDE following the instructions in the document
https://www.st.com/resource/en/user_manual/um2563-stm32cubeide-installation-guidestmicroelectronics.pdf

2) Install the Touch GFX IDE following the instructions listed in the document
https://support.touchgfx.com/docs/introduction/installation

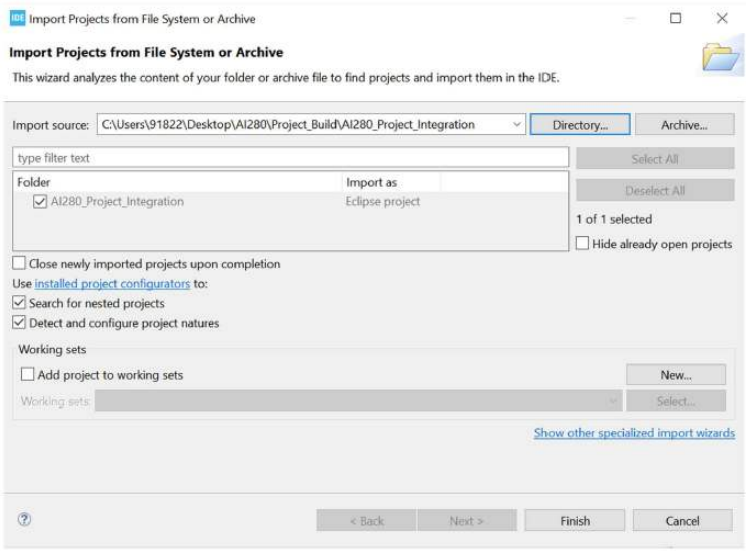
3) Once the STM32 Cube IDE is installed to open one of the SDK projects (integration test project, demo project or blank template project) open the Cube IDE and from the File menu select "open project from file system…" option.

For reliability within TouchGFX please ensure the project files are in a file path directory with no spaces. Example:



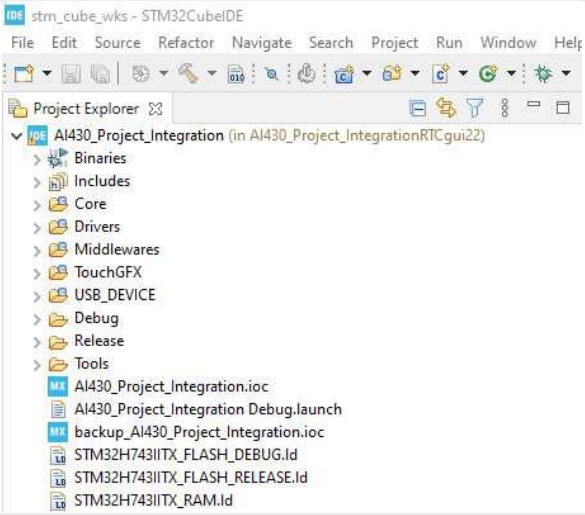
C:\User\JohnSmith\CANBUSProject\Template_Directory

To avoid conflicts only open one SDK project at the time.



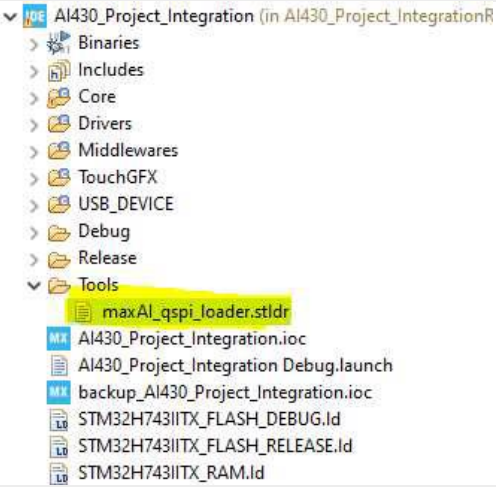
Select the option “Directory…” to look for the folder where the SDK project is located.

After some seconds the option “Finish” will be enable and need to be clicked to finish the process.



Once the process to import the project is done the files will be available in the left section of the IDE.

Once the SDK project is imported an external loader needs to be copied into the installation folder. Inside the AI430 project in Tools folder the external loader is located.



The maxAI_qspi_loader. stldr needs to be copied in the following path:

“LOCAL_DIRECTORY\STM32CubeIDE_1.11.0\STM32CubeIDE\plugins\com.st.stm32cube.ide.mcu.externaltools.cubeprogrammer.win32_2.0.100.202110141430\tools\bin\ExternalLoader”
Note: The LOCAL_DIRECTORY is the directory were the STMCubeIde was installed.

S/W Release Package Details

The MAXAI430 SDK kit comes with the below S/W release package. It has 3 released project files.

1) Demo Project File
2) Application Project File
3) Blank Project File

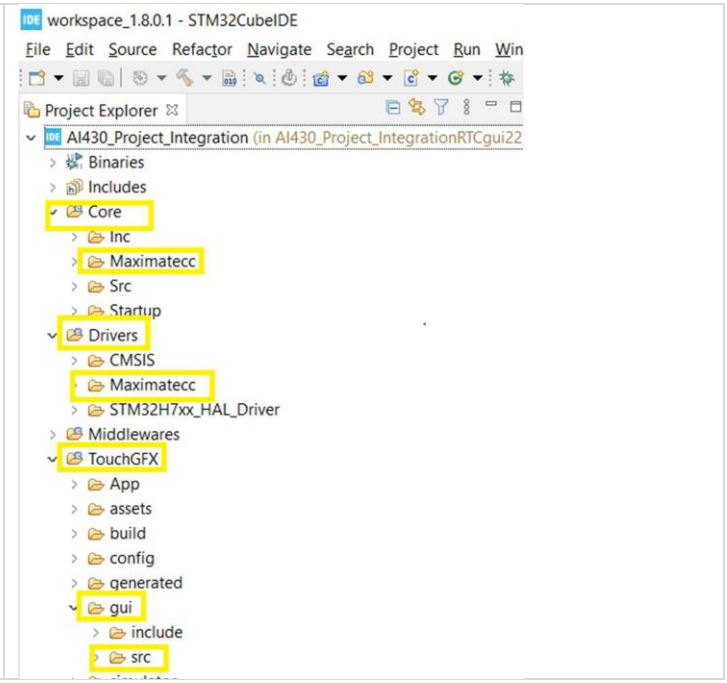
AI430 Project Structure

The maxAI 430 project files are integrated source code which include the TouchGFX application integrated with the AI430 SDK. These applications leverage the hardware capabilities of the AI430 platform via the SDK interface. In this section we will describe to you the variations in the three project files released with the maxAI 430 SDK which will enable you to write full-fledged applications using the AI430 SDK.

Demo Project File

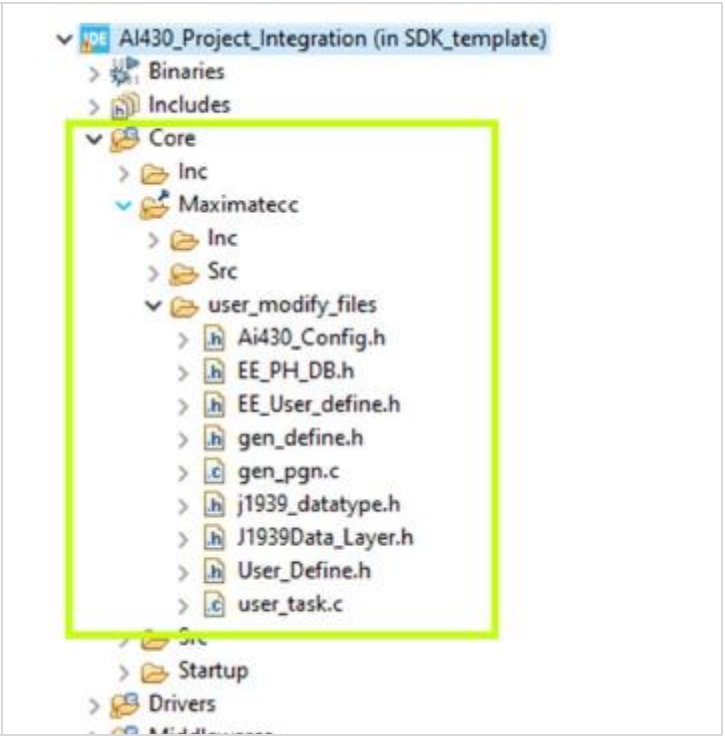
The Demo Project File is a fully graphical pre-built project file which leverages all the functionality of the AI430 SDK. This application is an integrated example which communicates with different modules in the SDK in a single UI screen. This project can be used as a reference for all users who are working on creating integrated applications for their specific needs. This project has 5 UI screen and the details of how-to setup and test are described in Digital Output Module section.
The below image shows the folder structure of the AI430 demo project file.

The Main folders of interest are:
a. Core
b. Drivers
c. Touch GFX



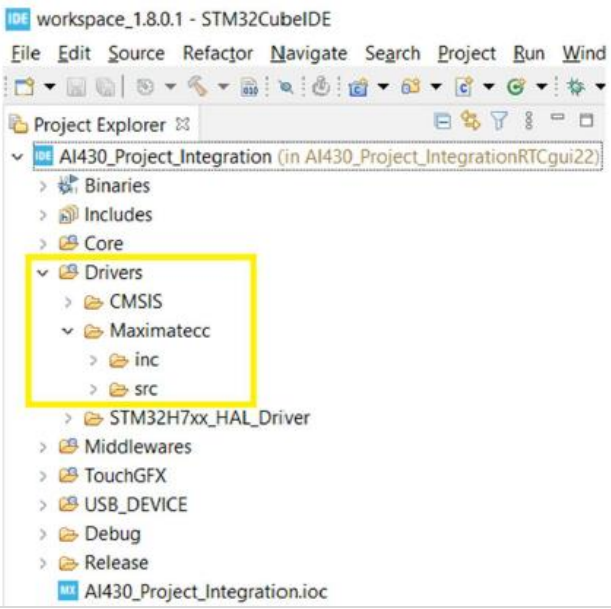
Core Directory

The core directory includes the files which form the core of the SDK architecture which include the platform service files for all the modules. (Platformservice.h and Platformservice.c) They are located under the Core\Maximatecc\Inc and Core\Maximatecc\Src directories.



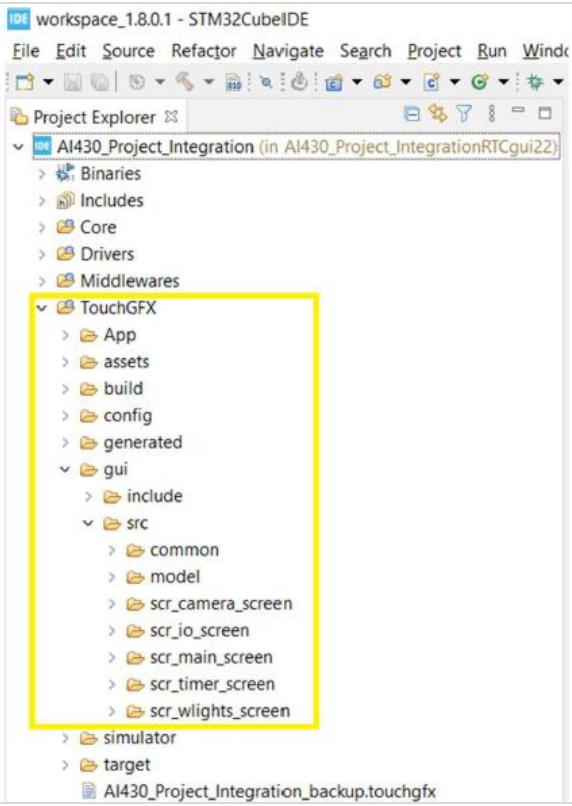
Driver Directory

The Driver files for all the modules in the MAXAI430 are located under the folder structure Drivers\Maximatecc\inc and Drivers\Maximatecc\src.



TouchGFX Directory

The GUI files for Demo purpose and understanding are available under the folder structure TouchGFX\gui\src. The screens that are available as demo are scr_io_screen,scr_main_screen and scr_timer_screen.
As a TouchGFX developer any GUI code that is developed by you would go into this directory.



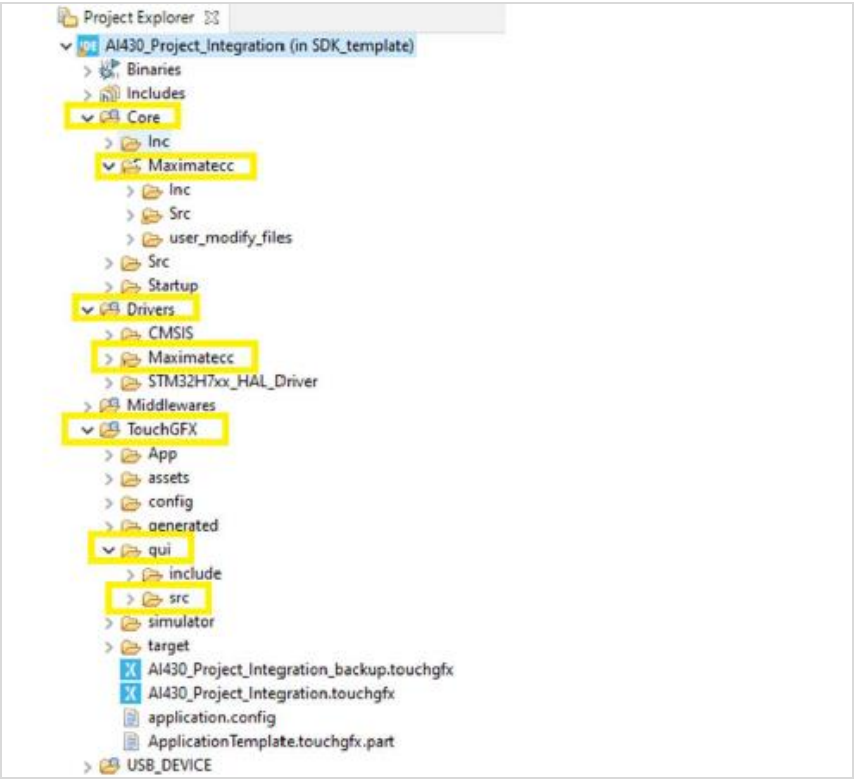
Sample Application Project File

The Application Project File is a semi graphical pre-built project file which details each module available in the AI430 hardware. References for all the functionality of the AI430 SDK can be found in the application project file. This sample application has standalone screens for each module in the SDK and elaborates in detail the possible ways you can interact with each module in the SDK. This project can be used as a reference for all users to understand in detail the individual modules of the SDK and get sample reference of how to use the various functionalities in the individual modules.

The below image shows the folder structure of the AI430 sample application project file.

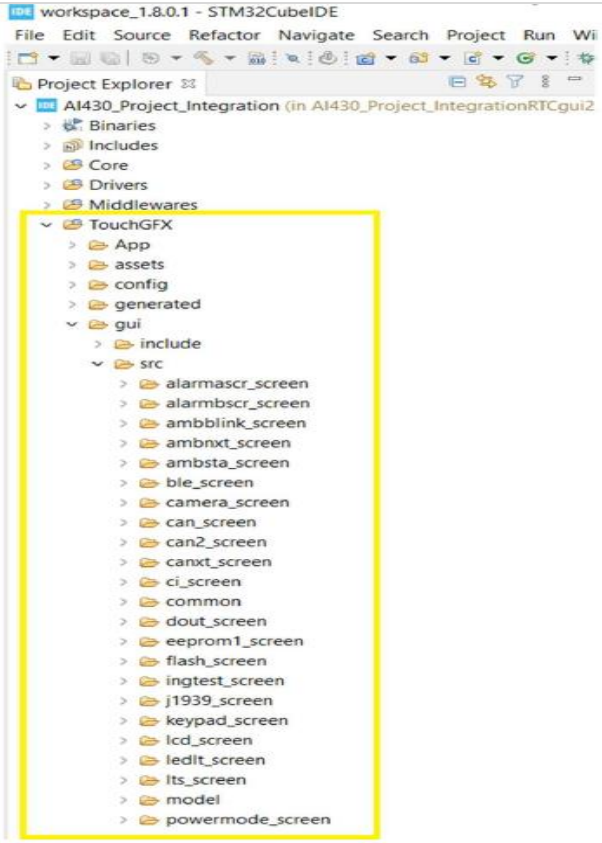
The Main folders of interest are:

a. Core: Like the Demo project
b. Drivers: Like the Demo project.
c. TouchGFX



TouchGFX Directory

The GUI files for application purpose and understanding are available under the folder structure TouchGFX\gui\src. The screens for all the possible user applications are available under mentioned folder as shown in the next diagram.

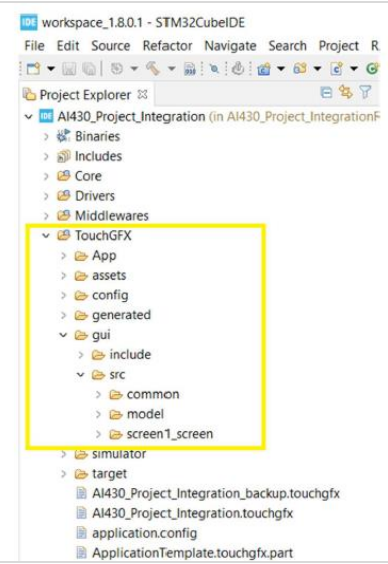


Blank Project File

The Blank Project File template is provided as a convenience for the end user to begin the firmware development. The Blank Project file can be unzipped to the desired location (folder) and renamed to a name as per users' choice.

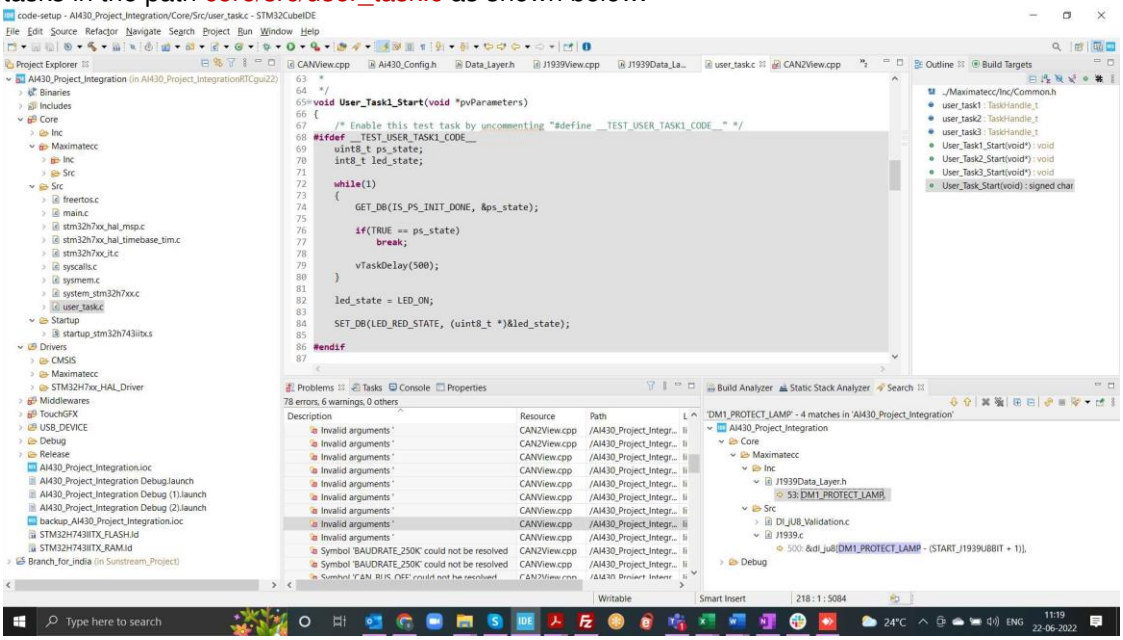
TouchGFX Directory

The GUI files for creating new applications should be added under the folder structure TouchGFX\gui\src. Section 3 describes in detail how a sample application can be written and
integrated with the SDK and tested on a MAX AI430 board.



Blank User Task Files

The SDK has included some blank user tasks that can be used by the application developers if they would like to create some tasks that run in the platform independent of the TouchGFX. As shown in the image below, you can find the user tasks in the path core/src/user_task.c as shown below.



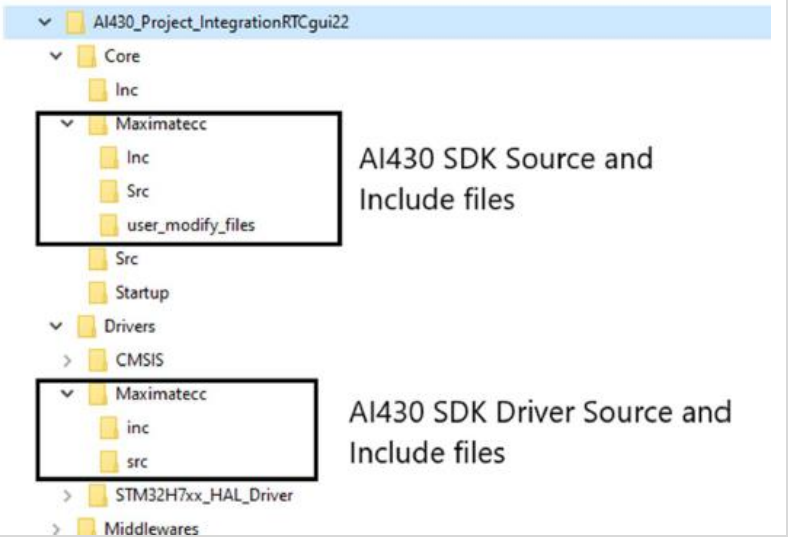
Please refer to User Task Edit Details to understand how to edit these user tasks and integrate with the SDK and tested on a MAX AI430 board.

SDK Application Development Procedure

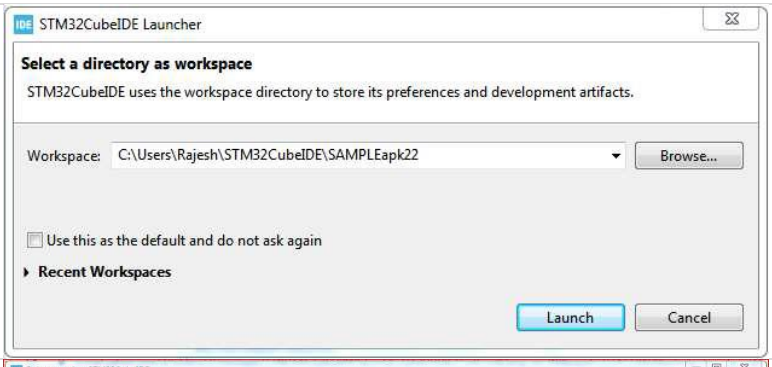
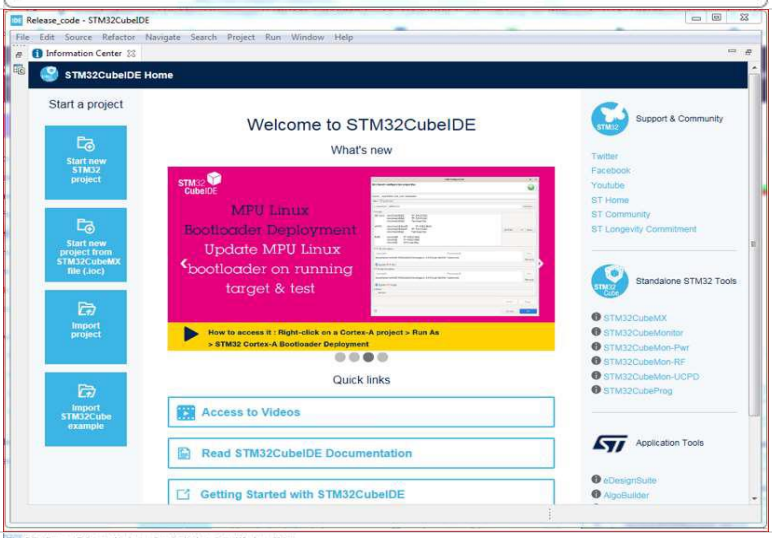
In this section we will walk you through the procedure to write a simple TouchGFX application using the blank project file provided in the S/W release package and compile the same with the STM Cube IDE and flash it on the maxAI 430 hardware and test it. We will also provide you the details on how you can debug the application using the ST link debugger.

Blank Project Installation and loading

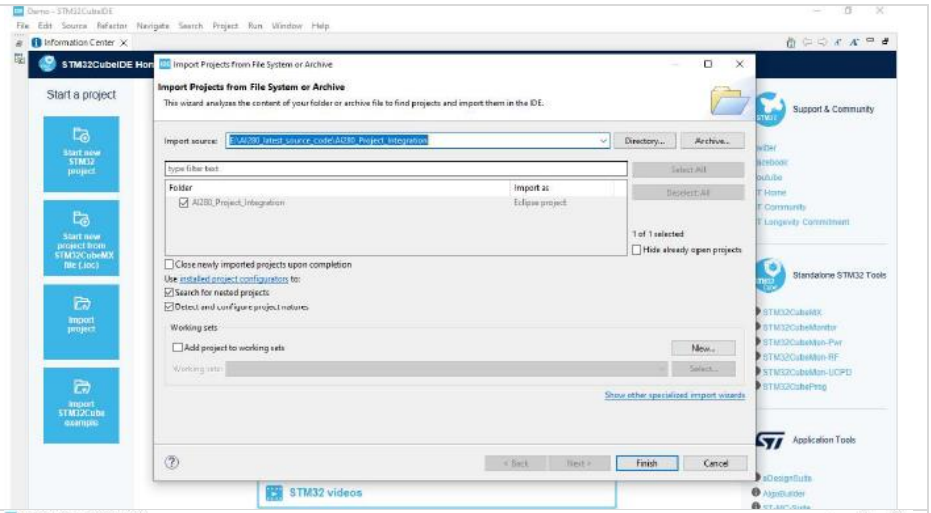
The MaxAI 430 SDK release comes with 3 project files as described in the Section 2.3. Please go to the folder with the zip file (AI430_GettingStart.zip) containing the blank project file and unzip it. You will get the directory, AI430_Project_IntegrationRTCgui22 after unzipping the file. The next image shows the contents and path of the blank project after it has been unzipped.



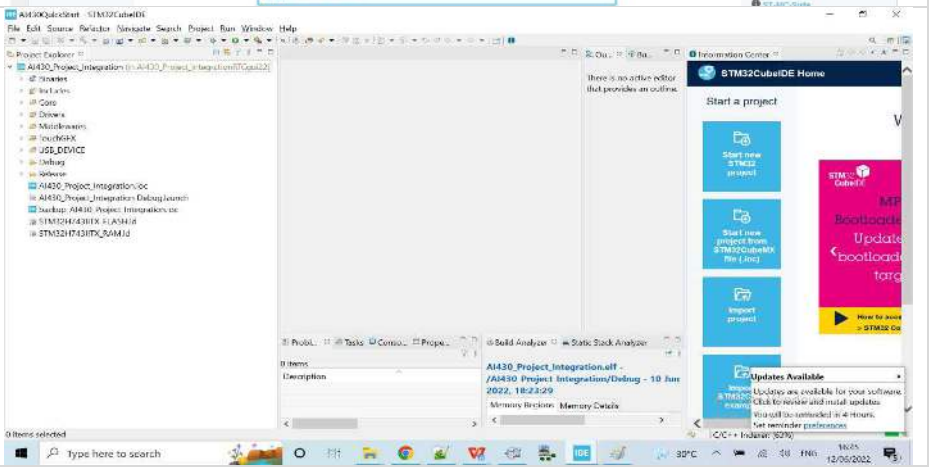
STM32 Cube IDE Setup.

Click on STM32CubeIDE Launcher and provide the Workspace name on the pop-up window given below and then click on Launch option given at the bottom right corner of the Pop-up notification to open the IDE with any desired workspace. We will then be adding our project into this workspace.	
Once the user enters the Workspace, he can see the next screen.	
The user must now import the blank project into this workspace. Click file menu and select the option “Open Projects from File System”. Refer the next image.	

Clicking on the “Open Projects from File System” will bring up the next window. Please provide the path of the unzipped blank folder in the import source option.

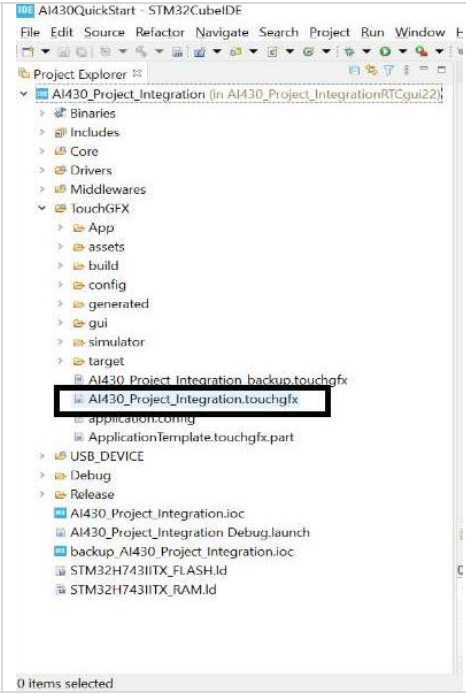


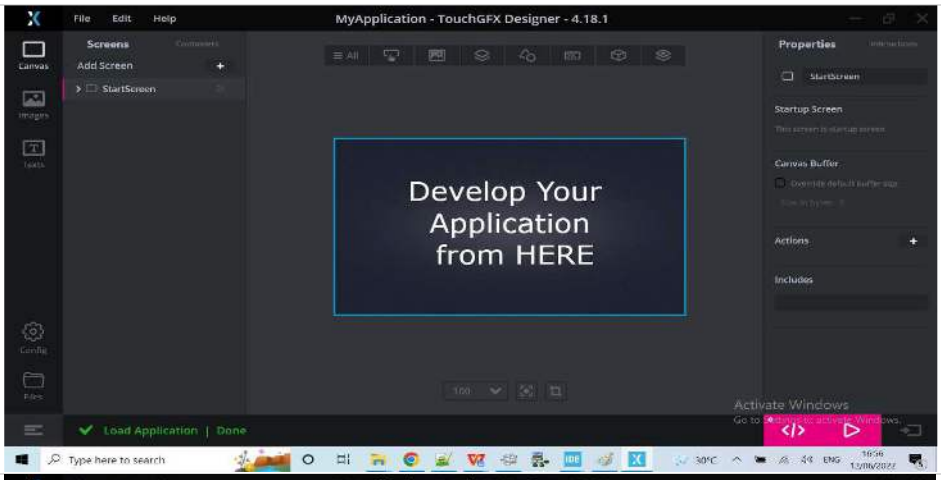
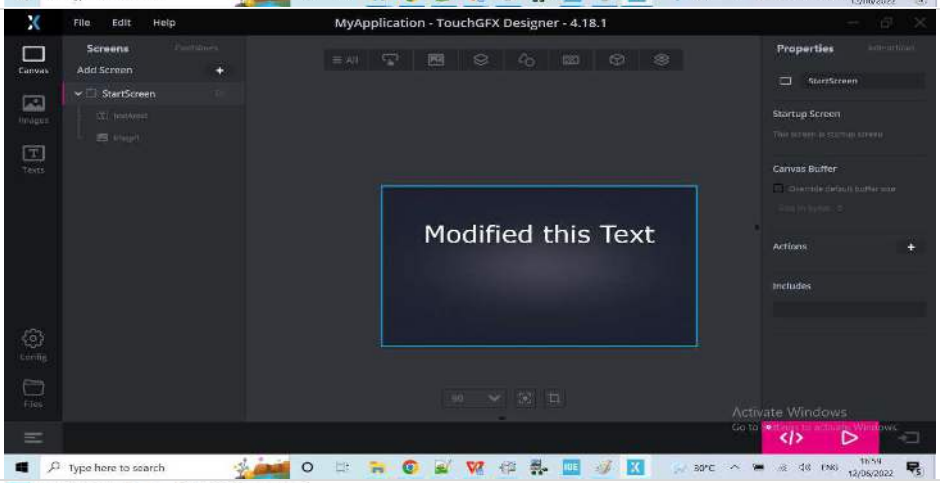
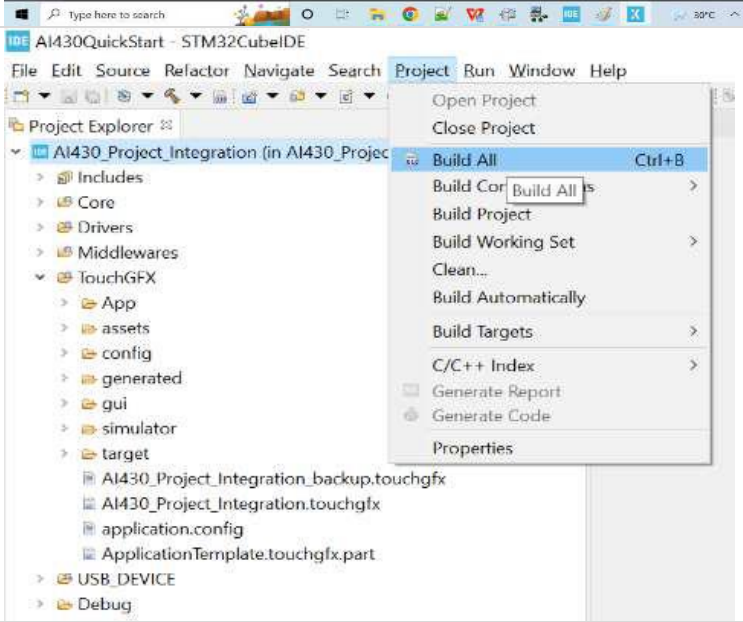
Now click on Finish once it recognizes the project file. You will get the next screen once you click on Finish.



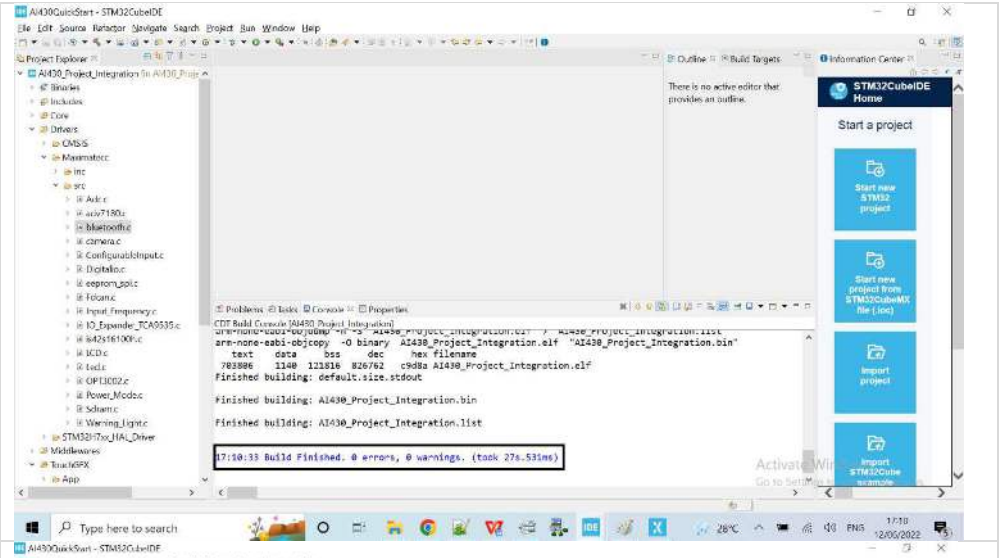
Build and Flash Procedure

Now we will create our own TouchGFX UI and compile and test it on the board. Expand the TouchGFX folder from the STM32CubeIDE and double click on the AI430_Project_Integration.touchgfx file. Refer the image given

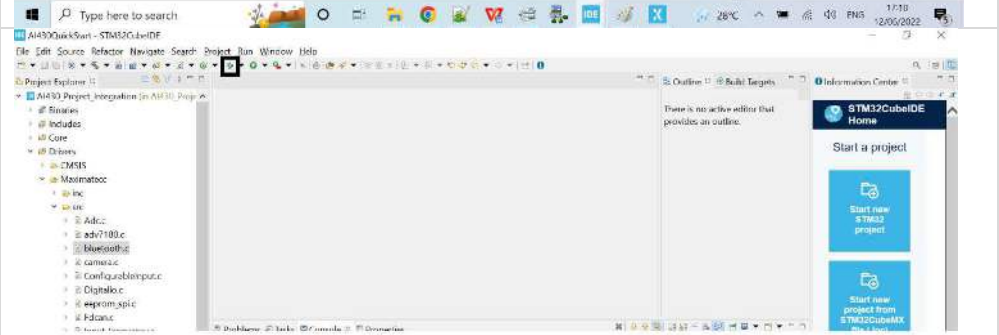


Once you double click the AI430_Project_Integration.touchgfx file, TouchGFX IDE will be opened with our designed UI as shown.	
Now we will modify some text and test it on the board. Change the text to “Modified this Text”. Click “</>” button for generating the TouchGFX code. Refer the next image	
Go to STM32CubeIDE Screen. And click Project ===> Build All. Refer the next image.	

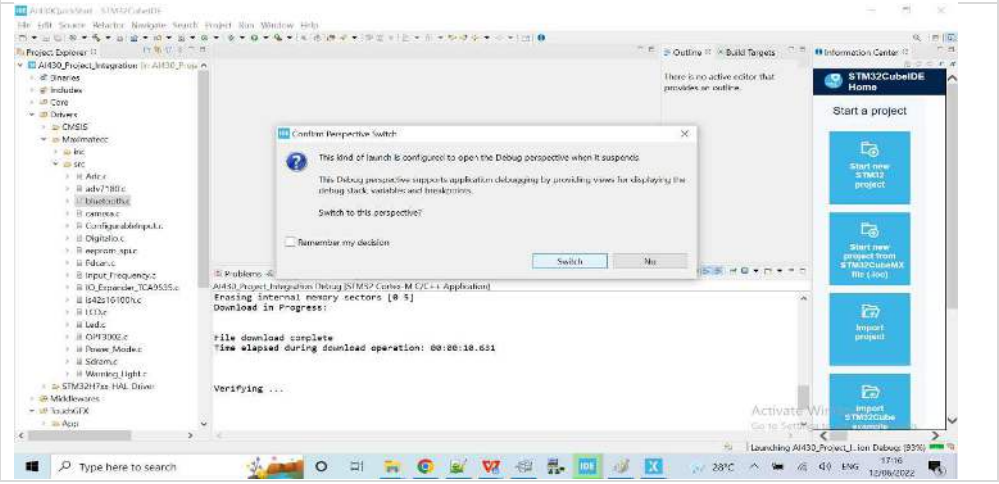
Once the build is success, you will get the console windows with “0 errors, 0 warnings”.

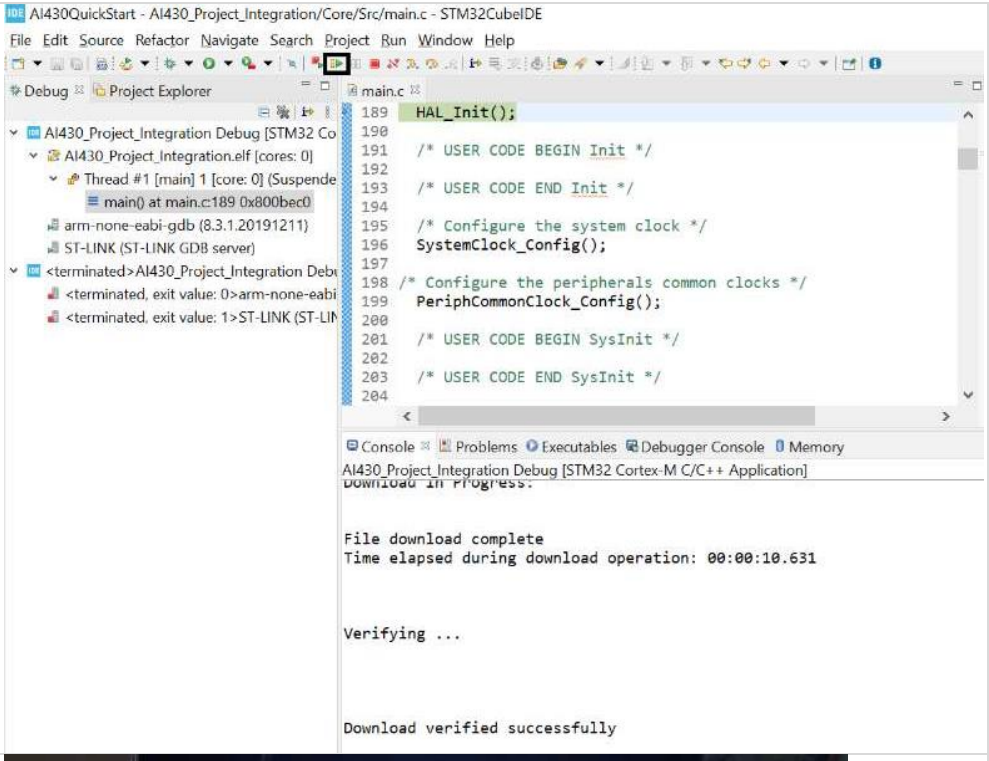
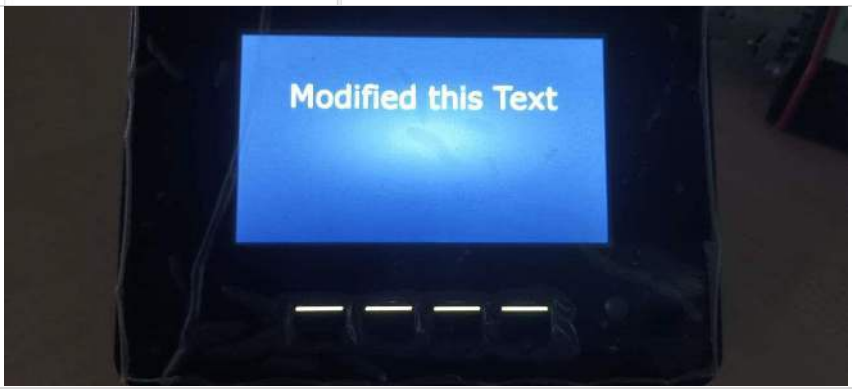


Now we will flash the binary using the ST JTAG, click debug icon to flash the code. Refer the highlighted (■) part in the next screen.



During the flashing, the code you will get the next screen. Please click “Switch” button to continue.



Once the flashing is completed you will get the next screen.	
Click Resume button (Highlighted ( ) in the above image) to run the application. You will get the next Screen on the AI430 board.	

We were successfully able to compile and flash the blank project with minimal changes to the MAX AI430 board. In the next section we will elaborate with an example on how you can leverage the features of the SDK.

Memory Sections

Debug and Release configurations

There are two configurations to compile the code, the example used the Debug configuration. This configuration allocates the code in the beginning of the internal flash (address 0x08000000) for debug process, rewriting the whole internal flash in the process, hereby overwriting the bootloader in case it was on the memory before. The second configuration Release allocates the code after the memory space dedicated for the bootloader (the address of the SDK is 0x08020000).

TouchGFX memory allocation

The images, fonts and texts added in TouchGFX are stored in the external flash. The external flash has a size of 16Mbyte and its only used to store the data of TouchGFX. This memory section is identified with the Build Analyzer as “QSPI”.

Memory allocation

The internal flash, as specified above, starts in the address 0x08000000, and ends right before the QSPI address (0x09000000). The flash can store up to 2MB. The RAM sections start in the address 0x2000000, and it is divided into multiple sections (See the figure next showing the Build example):

AI280_Project_Integration.elf - /AI280_Project_Integration/Release - Aug 29, 2023, 2:41:19 PM

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08020000	0x08220000	2 MB	1.37 MB	646.81 KB	31.58%
SHARED	0x20000000	0x20000040	64 B	0 B	64 B	100.00%
DTCMRAM	0x20000040	0x20020000	127.94 KB	127.94 KB	0 B	0.00%
ITCMRAM	0x00000000	0x00010000	64 KB	64 KB	0 B	0.00%
RAM_D1	0x24000000	0x24080000	512 KB	82.5 KB	429.5 KB	83.89%
RAM_D2	0x30000000	0x30048000	288 KB	288 KB	0 B	0.00%
RAM_D3	0x38000000	0x38010000	64 KB	64 KB	0 B	0.00%
QSPI	0x90000000	0x91000000	16 MB	15.57 MB	439.17 KB	2.68%

- **Shared**: Stores up to 64 Bytes that allow sharing information between the bootloader and the main application (0x20000000 to 0x20000040).
- **DTCMRAM**: Stores up to 128KB, address 0x20000040 to 0x20020000.
- **ITCMRAM**: Stores up to 64KB, address 0x00000000 to 0x00010000.
- **RAM_D1**: Stores up to 512 KB, address 0x2400000 to 0x24080000.
- **RAM_D2**: Stores up to 288KB, address 0x3000000 to 0x30048000.
- **RAM_D3**: Stores up to 64KB, address 0x38000000 to 0x38010000.

User Accessible memory

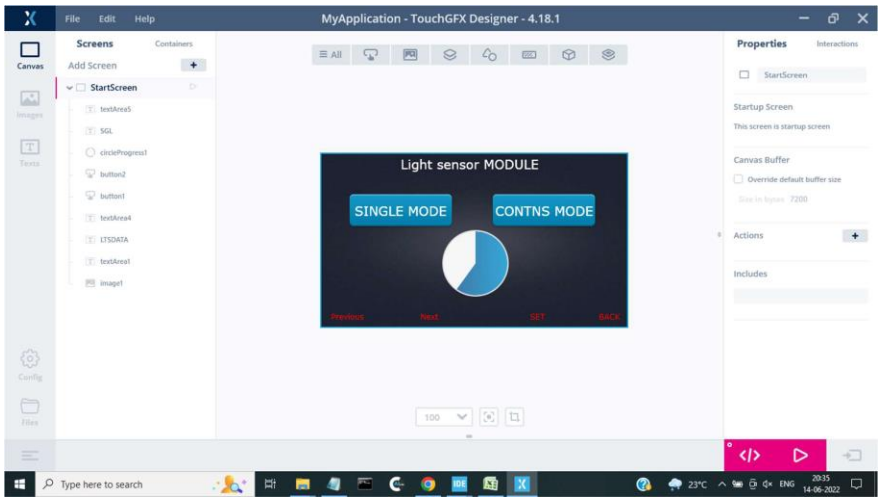
The memory that the user can modify, access or reserve depends on the build of the project. Firstly, the QSPI external memory linking directly to what is uploaded to the assets for the TouchGFX interface. Then, the remaining RAM memory that wasn't built is accessible to the user. Also, the internal flash memory out of the build from the application that wasn't utilized. The shared memory is for internal use only, as well as the memory used by the initial build of the application, which won't be accessible to the user for any type of operation.

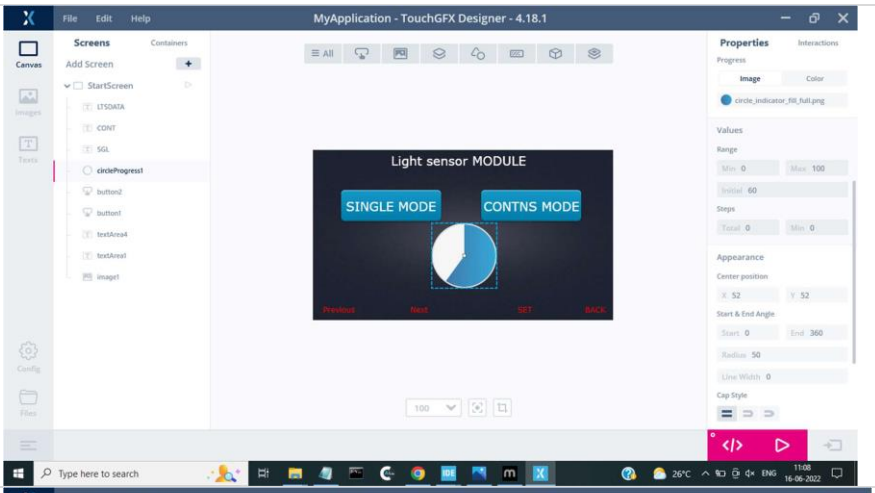
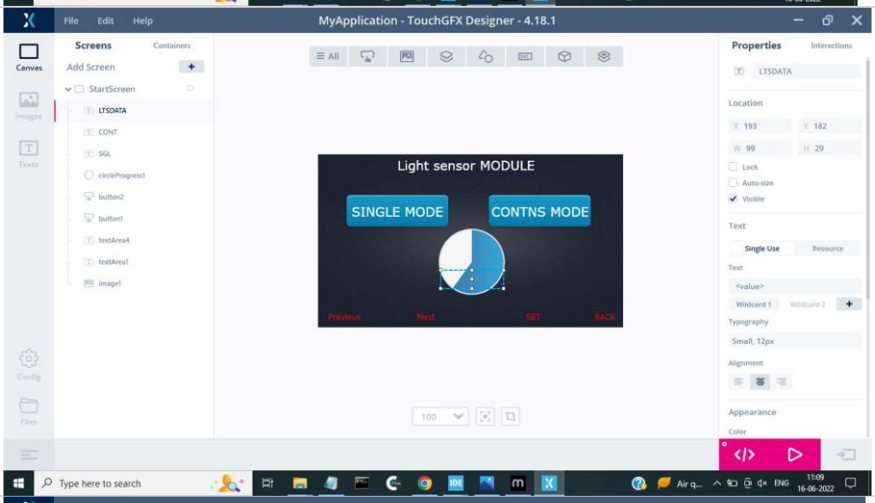
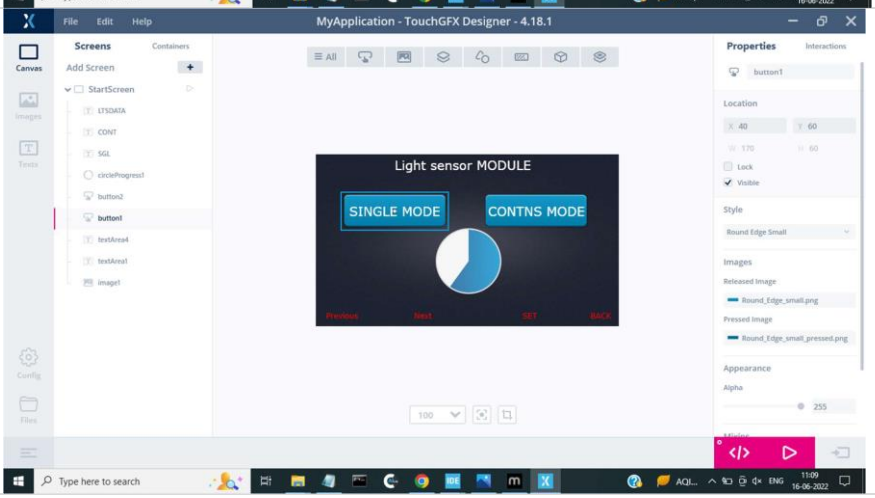
Light Sensor Module Demo

This section elaborates the procedure on how the TouchGFX application interacts with the SDK. We have used the Light Sensor module demo to explain this procedure.

Adding new GUI elements in the TouchGFX Screen

In this section, we will elaborate how we can add new elements in the touch GFX screen and then link them with the SDK. Select the "Text Area" under "All". Now you can type the text in the text box. You need to select wildcard1 option for Sensor value. Light sensor will fetch the data from sensor and display it on the screen. Refer the below screen.

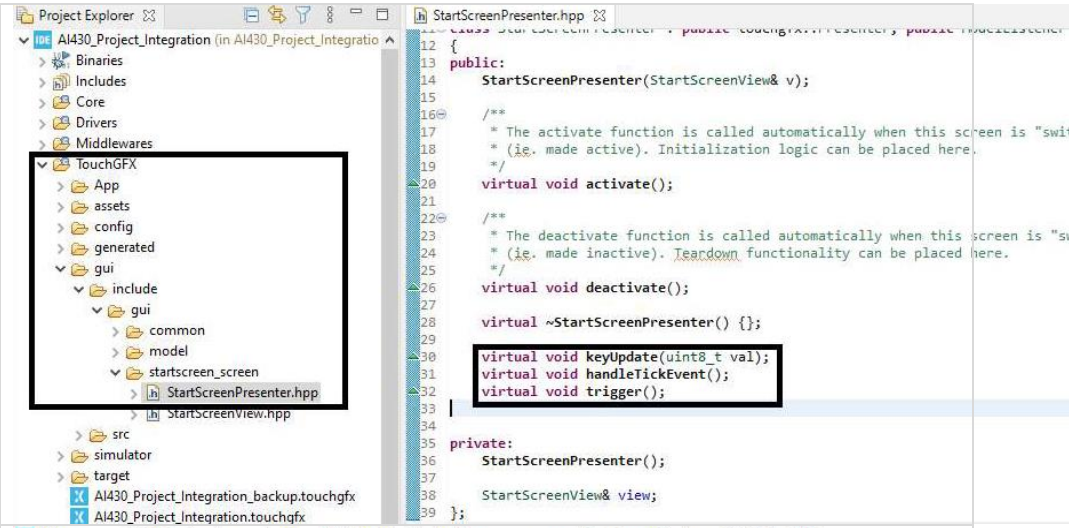


To create the circle, select Circle Progress under the section "All". The initial value can be from 0(Min) to 100(Max). You can assign the initial value under Properties section. Refer the next screen.	
Select the "Text Area" under "All". Now drag the and place the text box over the circle. You need to select wildcard1 option for Sensor value. Light sensor will fetch the data from sensor and display it on the screen. Refer the next screen.	
To create the button, select Button under the section "All". Now drag the and place the text box over the button. Now you can type the text as SINGLE MODE/CONTNS MODE.	
Click “</>” button for generating the TouchGFX code. Refer the highlighted part ( ) in the next screen.	
After the code is generated, please navigate to the STM 32 Cube IDE to edit the code and link the graphical elements to the SDK.	

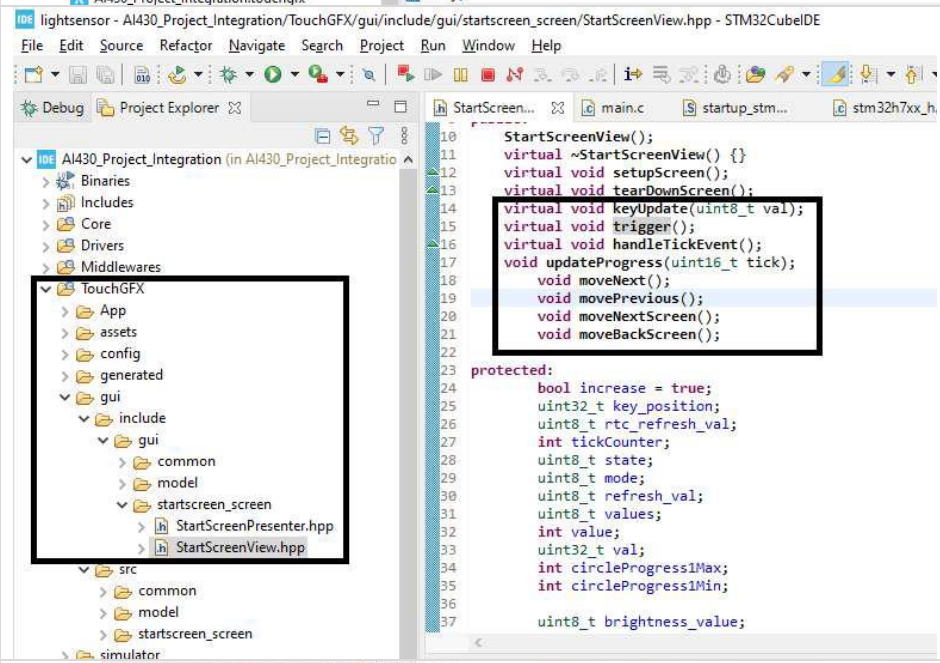
Edit the DB Variables

Now we are going to enable the graphical elements added in the last section to configure the light sensor module in single or continuous mode. To do the same we will need to link it with the SDK DB. In this section, we will describe how to link the DB Variables to the TouchGFX elements.

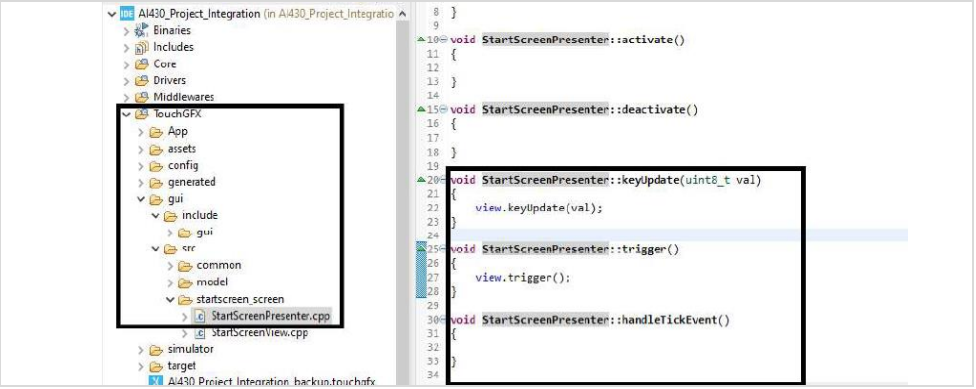
Go to STM32CubeIDE Screen. Add the following function declarations in "StartScreenPresenter.hpp" file under the "TouchGFX/gui/include/gui/startscreen_screen" folder structure.
" virtual void keyUpdate(uint8_t val);
"
" virtual void handleTickEvent();"
" virtual void trigger();"



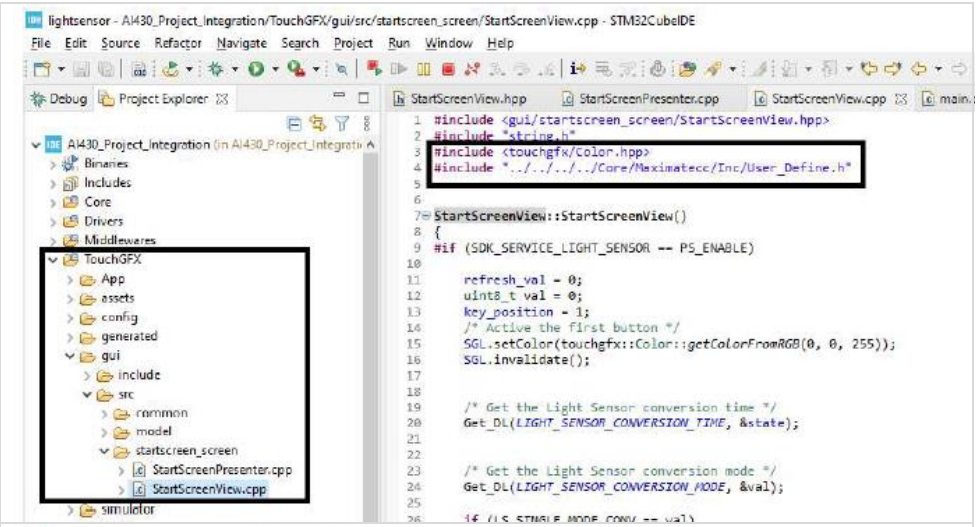
Add the following functions which will be used for the Light sensor Module in "StartScreenView.hpp" file under the "TouchGFX/gui/include/gui/startscreen_screen" folder structure.
"virtual void keyUpdate(uint8_t val);"
"void trigger();"
"void handleTickEvent();"
"void updateProgress(uint16_t tick);"
"void moveNext();"
"void movePrevious();"
Add the following functions for screen navigation purposes.
"void moveNextScreen();"
"void moveBackScreen();"



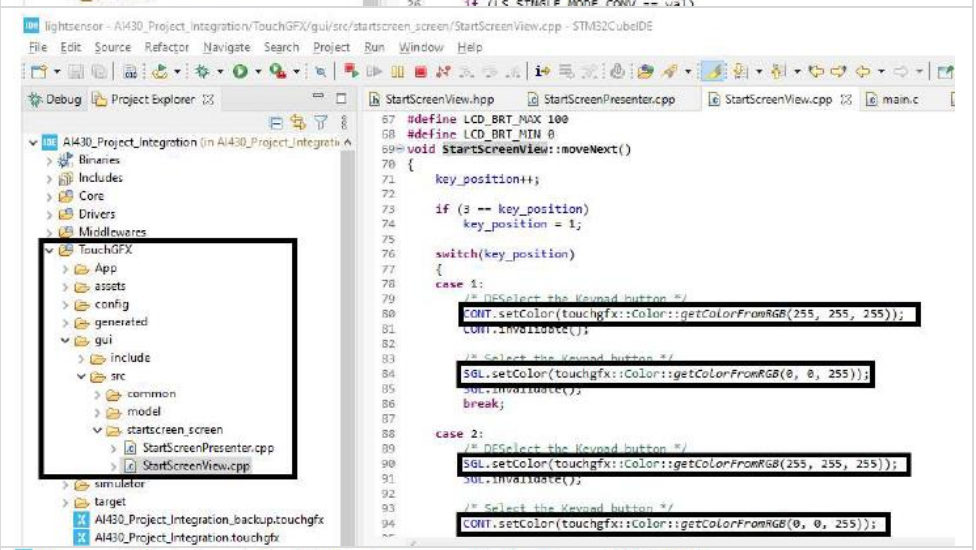
Add the following keypad function to access the functionality in "StartScreenPresenter.hpp" file under the "TouchGFX/gui/include/gui/startscreen_screen" folder structure.
void StartScreenPresenter::keyUpdate(uint8_t val);
{
view.keyUpdate(val);
}



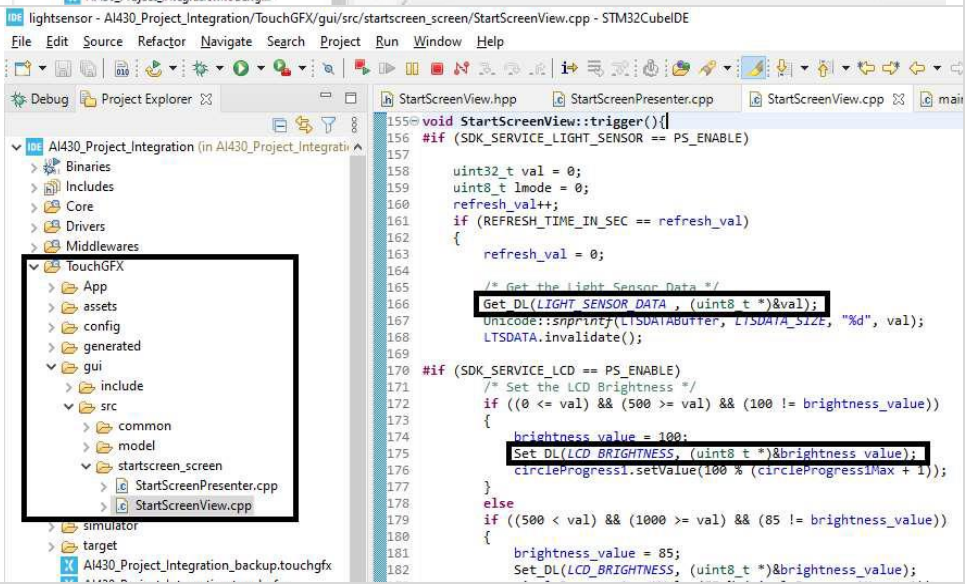
To access the DB variables, the following header needs to be added.
#include "../../../../Core/Maximatecc/Inc/User_Define.h"
To highlight the selected text, include the following line.
#include <touchgfx/Color.hpp>
Refer the next screen to see the code snippet and file path.



The 'setColor' function is used to highlight the selected text. Refer the next screen for code snippet.



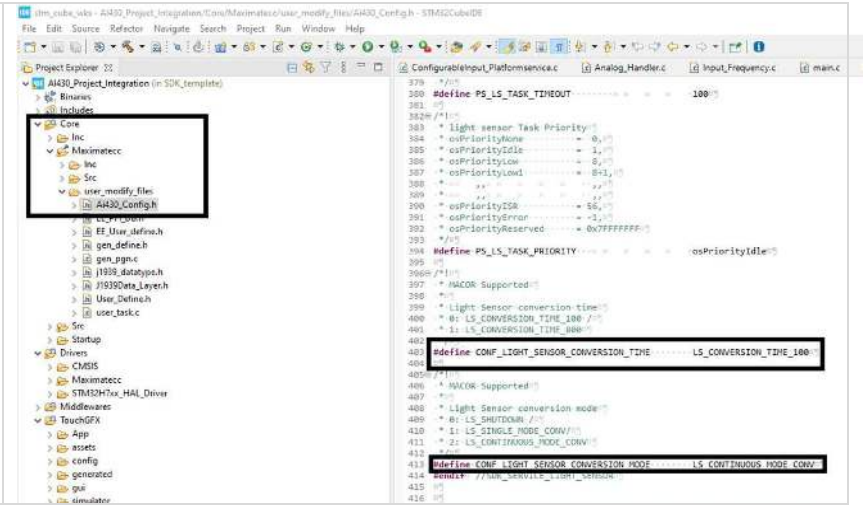
Based on the Light sensor data, we will adjust the LCD screen brightness. Get_DL function is used to read the DB variable of Light sensor data. Set_DL function is used to modify the DB variable of LCD brightness.



Configurations

This section describes the configurations required for Light sensor module. This default configuration can be done in the AI430_config.h file under the section core/Maximatecc/Inc. The configurable parameters available for light sensor module are conversion time and conversion mode.

To configure these, we are using CONF_LIGHT_SENSOR_CONVERSION_TIME and CONF_LIGHT_SENSOR_CONVERSION_MODE macros.
Kindly refer the next screen for code snippet.



Now we need to compile and test it on the board. For compilation, follow the instructions provided under the section Build and Flash Procedure.

Output

Once we flashed the updated binary on MAX AI430 board we will observe that the application we designed is launched on the GUI. Since the Conversion mode is set to Continuous mode by default, Light sensor will continuously fetch the sensor data and update in the DB and the UI will also get updated. Thus, the default configuration is working fine. Refer the next screens captured to show different sensor values being updated in the UI.

User can change the conversion mode on the board by clicking the single mode button. This will cause the mode to switch from continuous to single shot and the light sensor module will fetch only one data from the hardware and update in the DB. The next screen shows the sensor value for Single Mode.

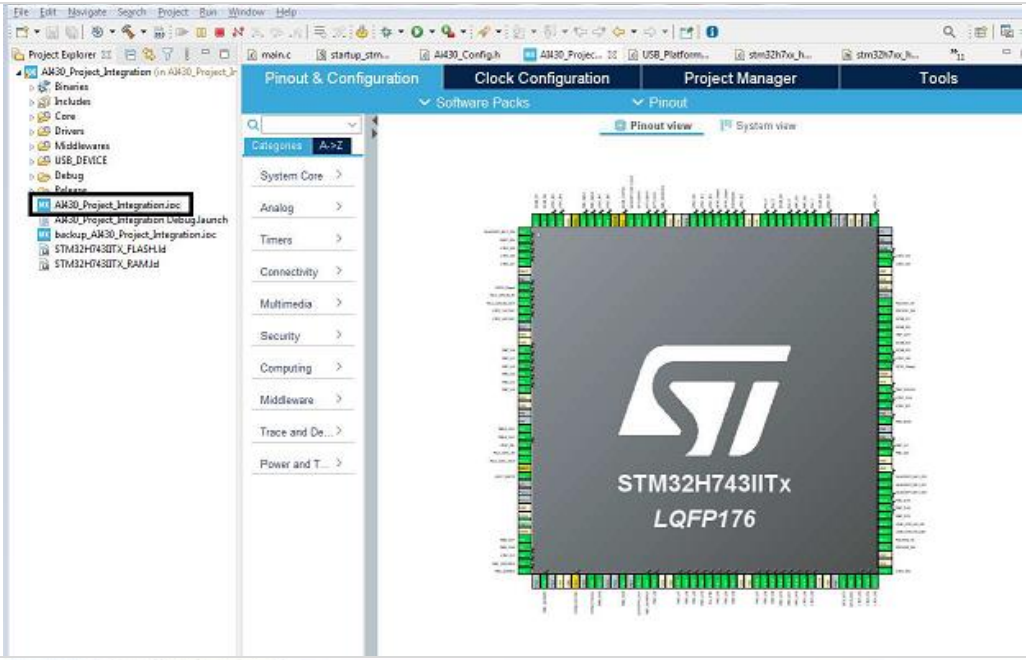
The UI will remain in the above screen unless the user changes the configuration.

Warning Light Demo

The demo application is written for user who would like to write an application without using the TouchGFX. They can use the same blank project released in the SDK, but need to disable the TouchGFX and then write code which will link with the SDK and use the functionalities but will have a blank UI.

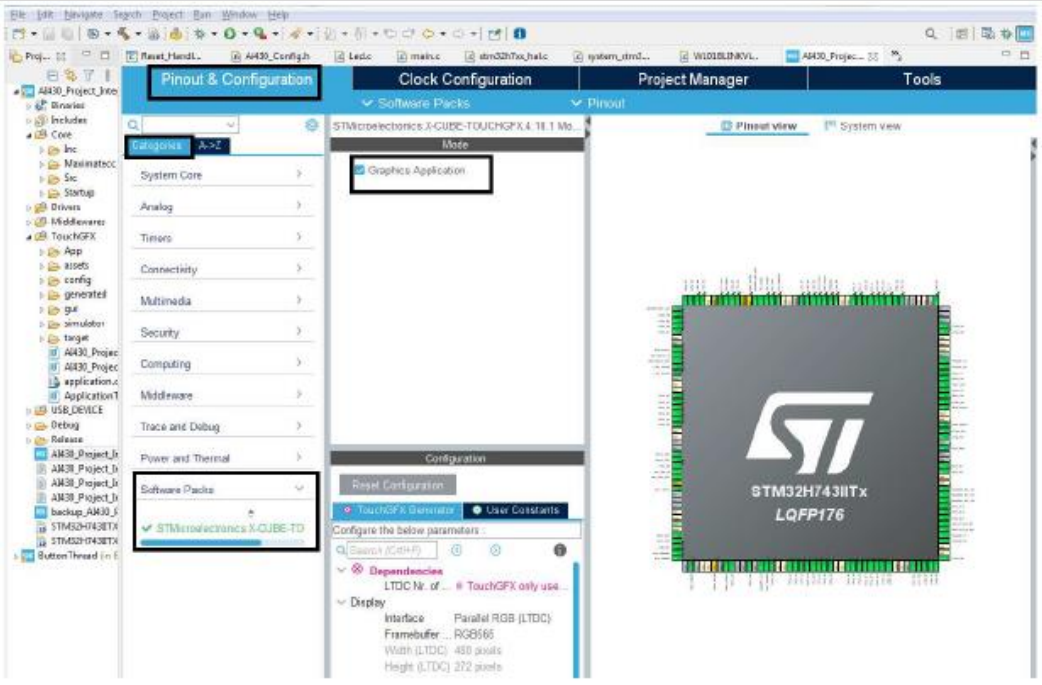
To Disable TouchGFX we need to click on AI430_Project_Integration.ioc.

Go to STM32CubeIDE Screen. Now double click on AI430_Project_Integration.ioc. The next screen will come up.

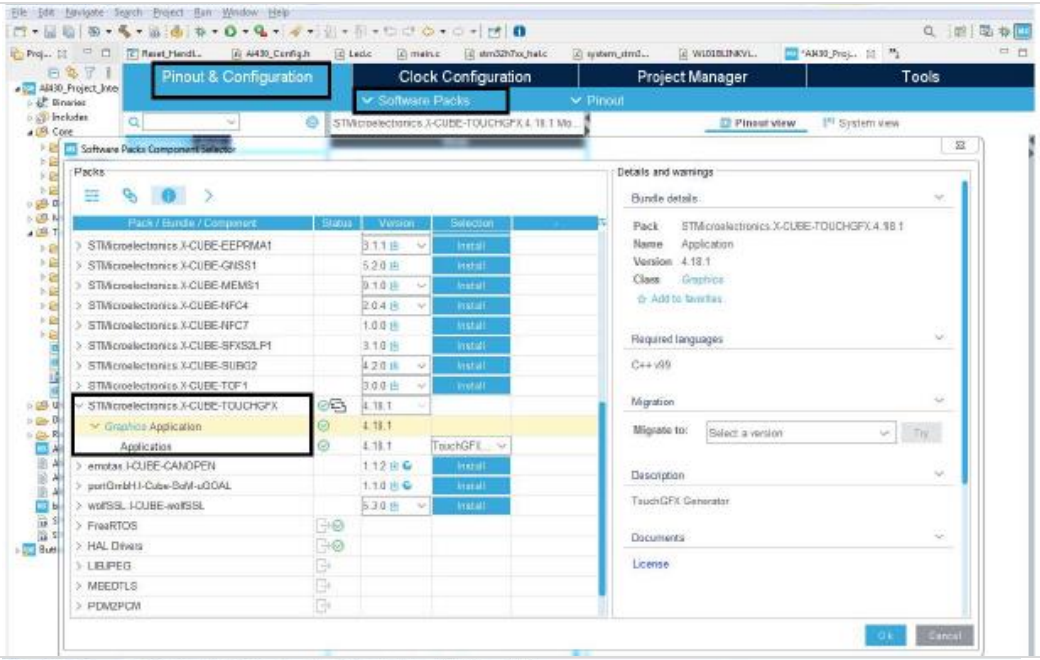


You will get the next screen once you click on AI430_Project_Integration.ioc. Now click Pinout & Configuration->Categories -> Software Packs-> STMicroelectronics.X-CUBETOUCHGFX.

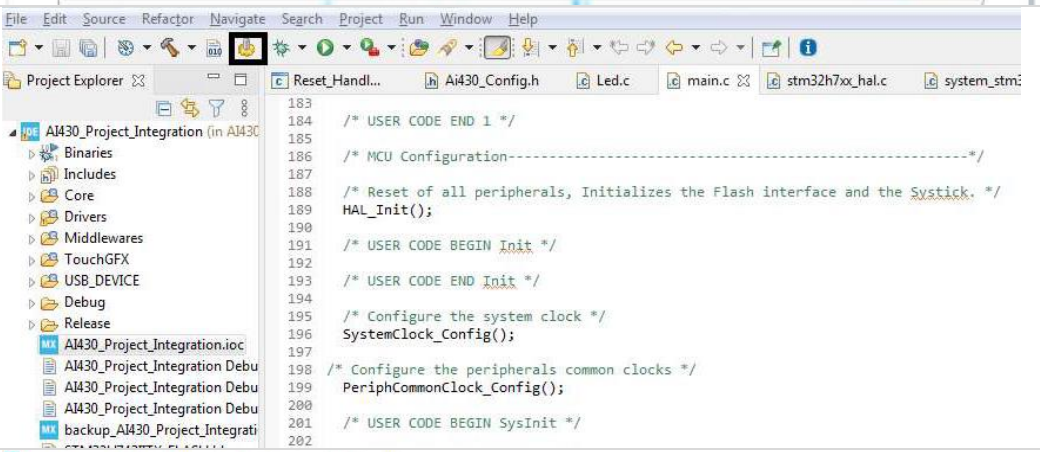
Now you can untick Graphics Application box. This will disable the TouchGFX configurations. Refer the highlighted areas in the below screen.



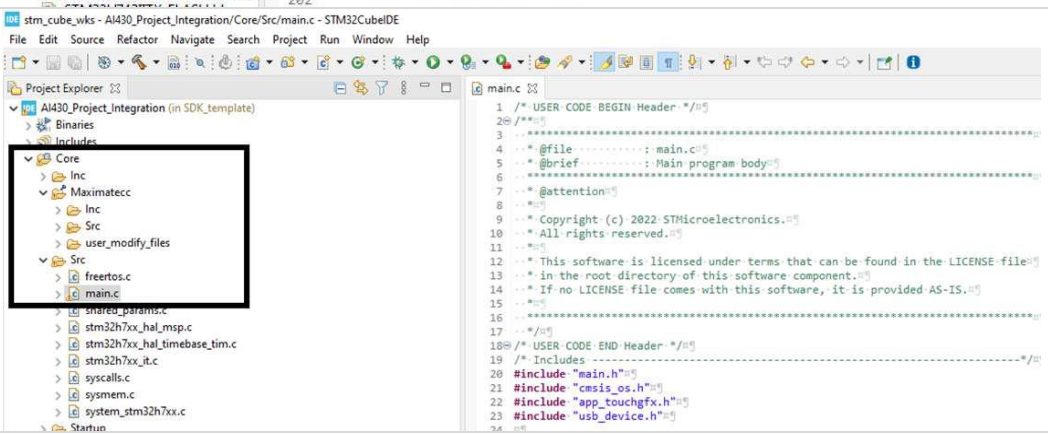
Now click Pinout & Configuration-> Software Packs->Select Components-> STMicroelectronics.X-CUBE-TOUCHGFX Graphics Application . Then select "Not selected" from the list. Then click on OK.



Now TouchGFX is disabled. The next step would be for the user to generate the code so that he can add his features.
Click Device Configuration Tool Code Generation button.

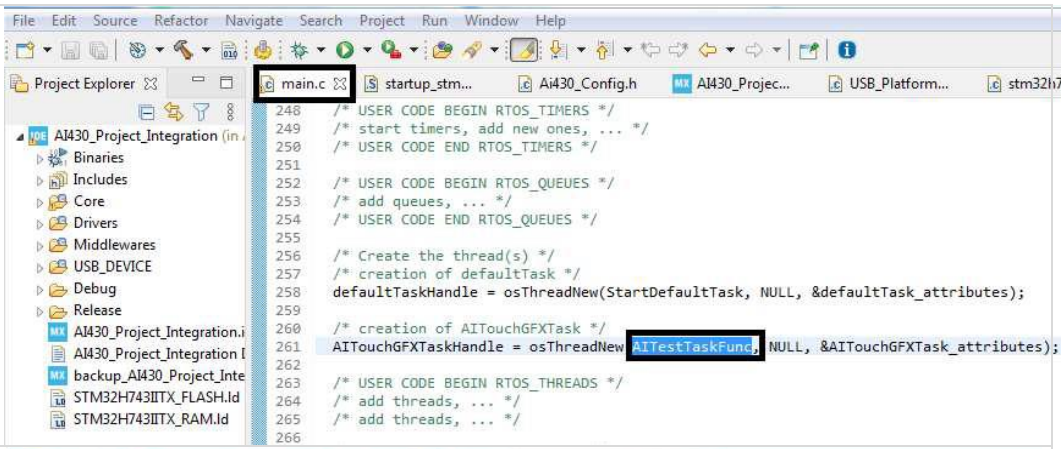


To add the Test task function, go to main.c file under the section core/src. Refer the next screen.

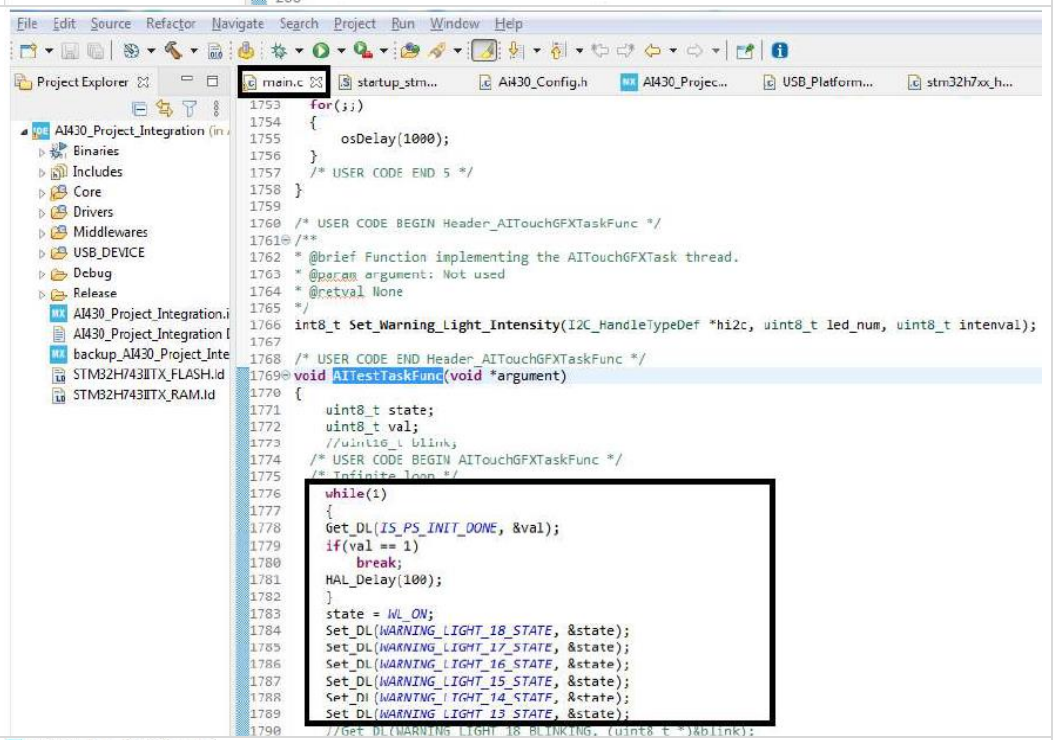


Now create the test task function. Refer the screen below for code snippet.

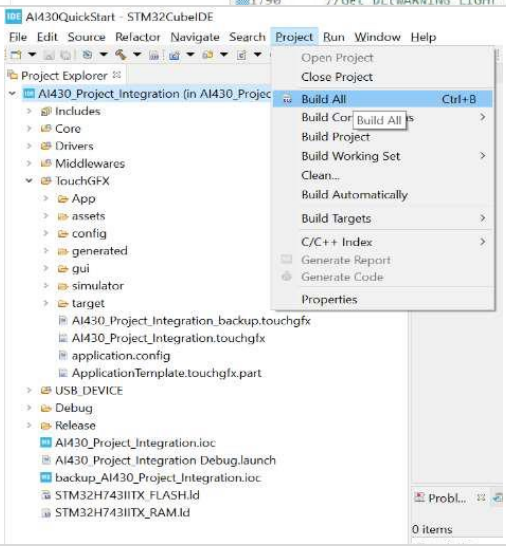
The user can add his code inside this test task.



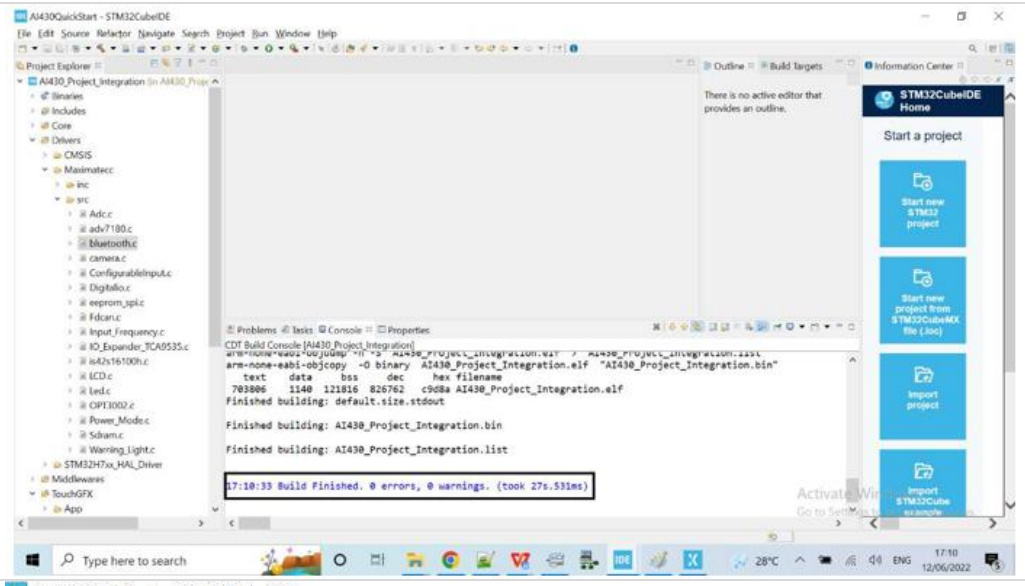
For example, we have added the below lines for warning light functionality. User can call Get_DL and Set_DL functions inside AITestTaskFunc. Refer the next screen for code snippet.



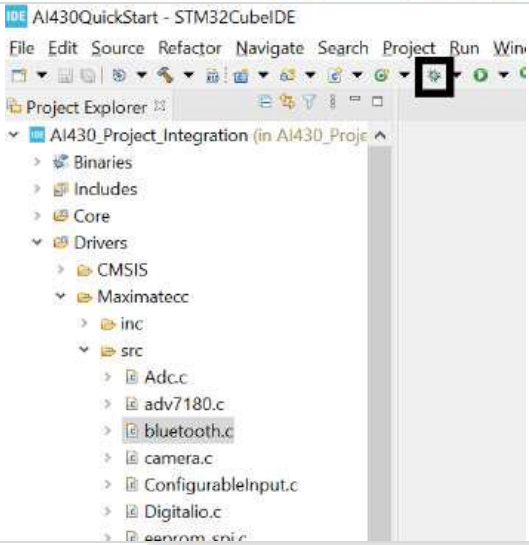
Now click Project ===> Build All. Refer the next image.



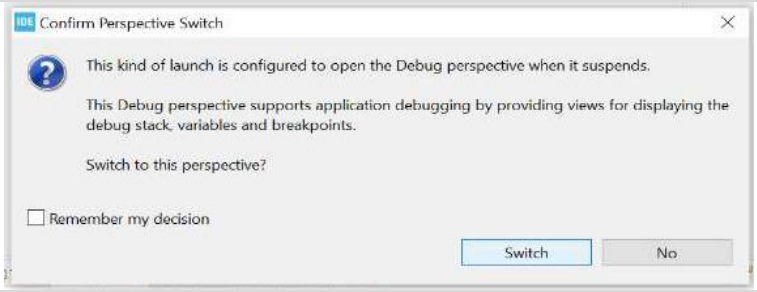
Once the build is successful, you will get the console windows with “0 errors, 0 warnings.” As shown.



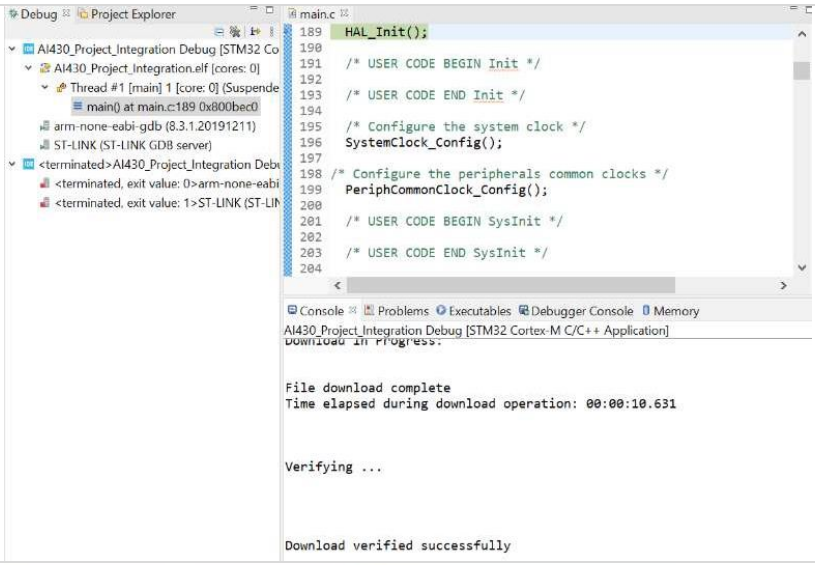
Now we will flash the binary using the ST JTAG, click debug icon to flash the code. Refer the highlighted part in the next screen.



During the flashing, the code you will get the below screen. Please click “Switch” button to continue.



Once the flashing is completed you will get the next screen.

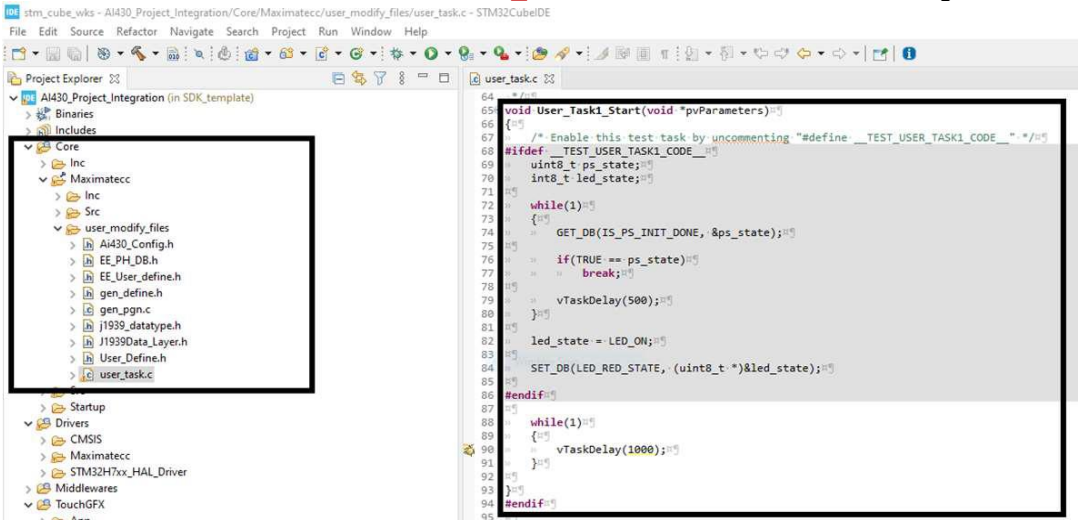


Click Resume button(Highlighted ( ) in the above image) to run the application. You will get the next Screen on the AI430 board.

In the AITestTaskFunc, we enabled WarningLight 13 to Warning Light 18. You can see the next screen shot where the warning lights are enabled.

User Task Edit Details

This section elaborates how user can initialize their user task and call the APIs, Set_DL and Get_DL without using the TouchGFX. They can also perform any non-UI related tasks here. User can create the tasks as shown below. Below images show that the user is creating three tasks. Refer to the next two images for the sample code snippet to create the tasks. These tasks are created in their user_task.c file as shown in the next image:



```
int8_t User_Task_Start(void)
{
#if((USER_TASK1 == PS_ENABLE) || (USER_TASK2 == PS_ENABLE) || (USER_TASK3 == PS_ENABLE)
        BaseType_t xReturned;
#endif

#if(USER_TASK1 == PS_ENABLE)
        /* Start the User_Task1_Start service thread */
        xReturned = xTaskCreate(User_Task1_Start, "User_Task1",
                        512, NULL, osPriorityIdle,   &user_task1);
        if (xReturned != pdPASS)
        {
                LOGERROR("Unable to initialize the User_Task1 \r\n");
                return PS_FAILURE;
        }
#endif

#if(USER_TASK2 == PS_ENABLE)
        /* Start the User_Task2_Start service thread */
        xReturned = xTaskCreate(User_Task2_Start, "User_Task2",
                        512, NULL, osPriorityIdle,   &user_task2);
        if (xReturned != pdPASS)
        {
                LOGERROR("Unable to initialize the User_Task2 \r\n");
                return PS_FAILURE;
        }
#endif
```

```
#if(USER_TASK3 == PS_ENABLE)
        /* Start the User_Task3_Start service thread */
        xReturned = xTaskCreate(User_Task3_Start, "User_Task3",
                        512, NULL, osPriorityIdle,   &user_task3);
        if (xReturned != pdPASS)
        {
                LOGERROR("Unable to initialize the User_Task3 \r\n");
                return PS_FAILURE;
        }
#endif

        return PS_SUCCESS;
}

#endif
```

User can call SET_DB and GET_DB APIs inside the created user tasks to access the required functionalities. Please see the below image for the code snippets.

```
void User_Task1_Start(void *pvParameters)
{
        /* Enable this test task by uncommenting "#define __TEST_USER_TASK1_CODE__" */
#ifdef __TEST_USER_TASK1_CODE__
        uint8_t ps_state;
        int8_t led_state;

        while(1)
        {
                GET_DB(IS_PS_INIT_DONE, &ps_state);

                if(TRUE == ps_state)
                        break;

                vTaskDelay(500);
        }
        led_state = LED_ON;

        SET_DB(LED_RED_STATE, (uint8_t *)&led_state);
#endif

        while(1)
        {
                vTaskDelay(1000);
        }

}
#endif
```

```
#if(USER_TASK2 == PS_ENABLE)
void User_Task2_Start(void *pvParameters)
{

        /* Enable this test task by uncommenting "#define __TEST_USER_TASK2_CODE__" */
#ifdef __TEST_USER_TASK2_CODE__
        uint8_t ps_state;
        int8_t state;

        while(1)
        {
                GET_DB(IS_PS_INIT_DONE, &ps_state);
                if(TRUE == ps_state)
                        break;

                vTaskDelay(500);
        }
        state = KEY_BACKLIGHT_ON;

        Set_DL(KEYPAD_BACKLIGHT , (uint8_t *)&state);
#endif

        while(1)
        {
                vTaskDelay(1000);
        }

}
#endif
```

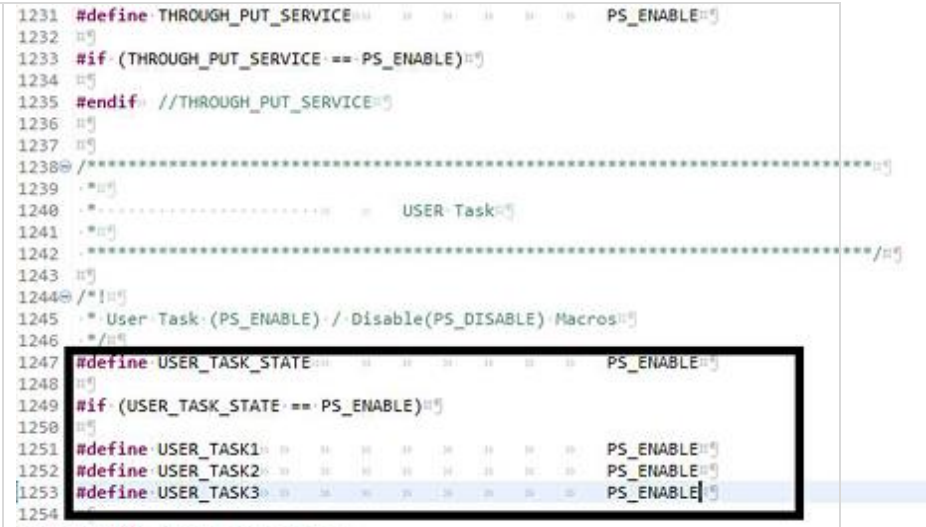
```
#if(USER_TASK3 == PS_ENABLE)
void User_Task3_Start(void *pvParameters)
{
        /* Enable this test task by uncommenting "#define __TEST_USER_TASK3_CODE__" */
#ifdef __TEST_USER_TASK3_CODE__
        uint8_t ps_state;
        int8_t led_state;

        while(1)
        {
                GET_DB(IS_PS_INIT_DONE, &ps_state);

                if(TRUE == ps_state)
                        break;

                vTaskDelay(500);

        }
        led_state = LED_ON;

        SET_DB(LED_AMB_STATE, (uint8_t *)&led_state);
#endif

        while(1)
        {
                vTaskDelay(1000);
        }

}
#endif
```

If user enables the USER_TASK_STATE in Ai430_Config.h file, User_Task_Start function gets called from Platform_Service.c. Refer the below screen for code snippet.

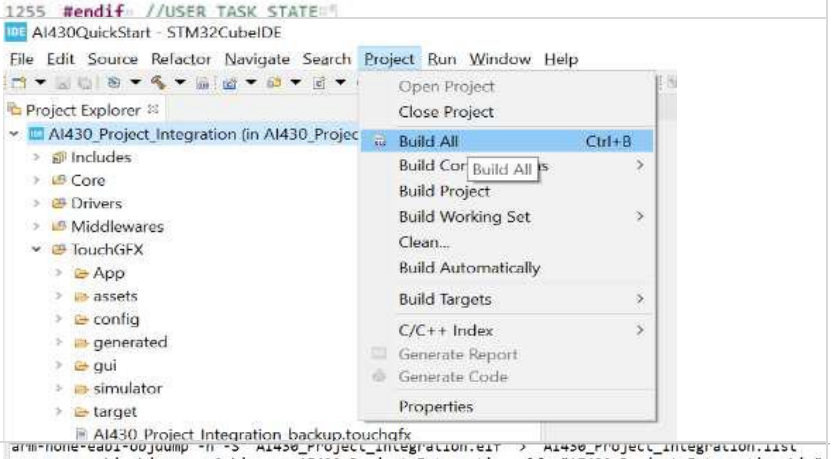
```
#if (SDK_SERVICE_FDCAN == PS_ENABLE)
    /* Start FDCAN Platform Service */
    FDCAN_PS_Start();
#endif

#if (SDK_SERVICE_J1939 == PS_ENABLE)
    /* Start J1939 Platform Service */
    J1939_PS_Start();
#endif

#if (SDK_SERVICE_CFG_INPUT == PS_ENABLE)
    /* Start ConfigurableInput Platform Service */
    CI_PS_Start();
#endif

#if (THROUGH_PUT_SERVICE == PS_ENABLE)
    Throughput_service_Start();
#endif

#if (USER_TASK_STATE == PS_ENABLE)
    User_Task_Start();
#endif
```

Before using the tasks, the user will need to initialize the tasks from the main.c. User can call there USERTASK1_PS_Start under platform service init as shown in the below image.

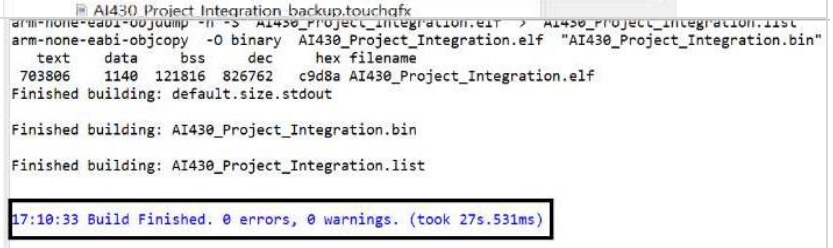
Users need to create and enable the User Tasks in the configuration file. Refer the next screen for code snippet.



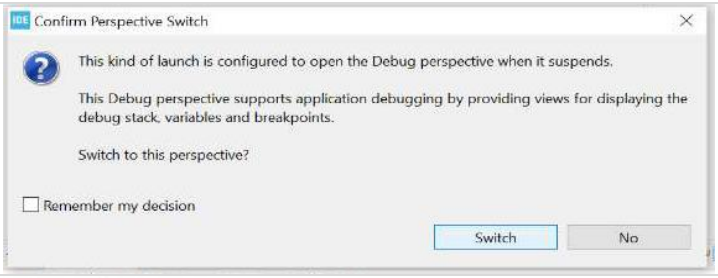
The user can then compile and flash the same following the below procedure.
Now click Project ===> Build All. Refer the next image.



Once the build is successful, you will get the console windows with “0 errors, 0 warnings.” As shown.

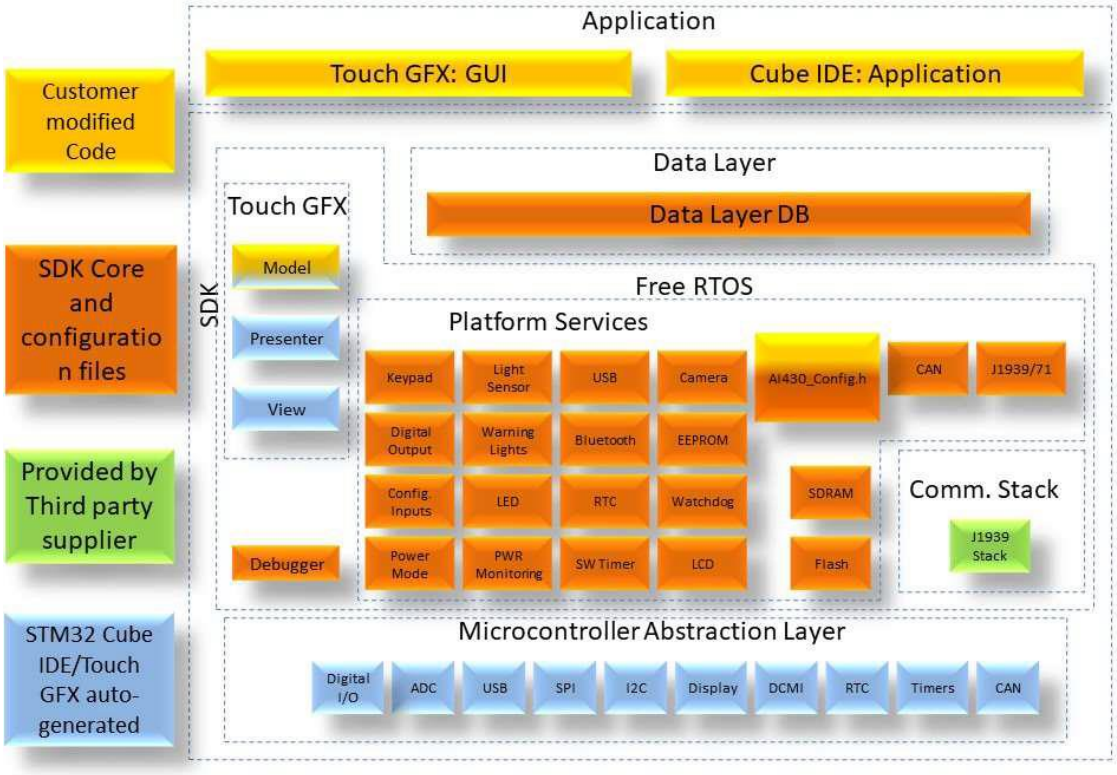


Now we will flash the binary using the ST JTAG, click debug icon to flash the code. Refer the highlighted part in the next screen.

During flashing the code, you will get the next screen. Please click “Switch” button to continue.	
Once the flashing is completed you will get the next screen	File download complete Time elapsed during download operation: 00:00:10.631 Verifying ... Download verified successfully
Click Resume button to run the application. You will get the next Screen on the AI430 board. Once the device powers up with this build user can see the output on the board. By default, the maxAI 430 LED state is OFF but, in the user, task3 that we created and flashed we have changed the LED state as ON. Hence if we power up the board with this build, the user can see that LED is ON as shown next which confirms that the user task 3 has executed successfully. Hence the user can add any non-UI based functionalities in these tasks and execute them in the background.	

SDK Architecture

The AI430 SDK is built with the below four layers which are well organized to give the application the flexibility to be written independently with minimum knowledge regarding the internal functionality of the underlying hardware.



A quick overview of the layers is described below

Application

Blank Touch GFX will be provided where USER will be able to create the GUI layout using Touch GFX or USER developed widgets, Hardware configuration of the Touch GFX project would be predefined in the SDK. The user can create the graphical elements and link them with the SDK modules to achieve the desired results. The user modified code resides here.

Data Layer Data Base (DB)

Data Layer DB is the interface between USER application and the SDK Platform services. It acts as an intermediate layer and is used for communication between user application and platform services.

Data Layer DB consist of a collection of RAM variables containing the data of the Platform services, this data shall be updated with latest data from each platform service iteration/event. The Data Layer DB will also work as a channel to input data from the application to the Platform services.

Platform Services

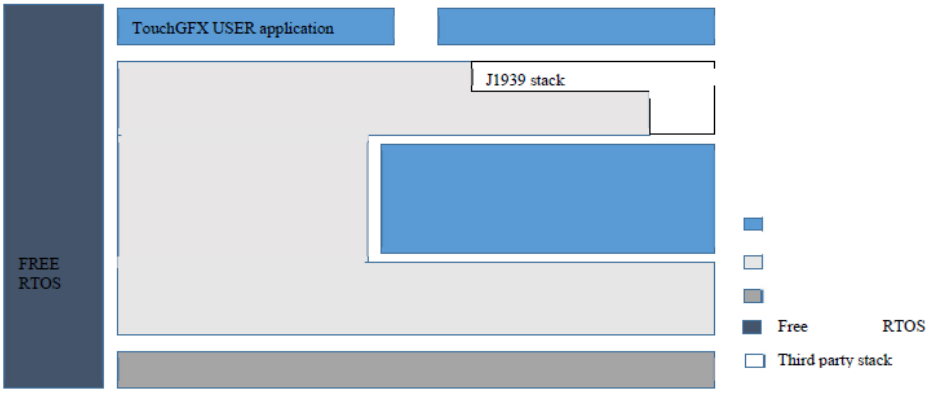
Platform Services will work as an interface between Data Layer DB and Platform driver. It will create and manage tasks for hardware peripherals based on user configuration/application requirements. These created tasks will run in RTOS. Data requested from user application will be obtained by Platform services from Platform driver. After receiving data, Platform services will push that information to Data Layer DB. Then, user application can fetch requested data from the Data Layer DB.

Platform Drivers

Platform Drivers will be used for accessing and controlling the hardware. Platform drivers will configure the hardware based on user configuration/application requirements. It will receive the relevant data needed by user application. Data received by platform driver will be sent to Platform services.

SDK Interfaces

The SDK adaptation software provides an interface between the USER tasks and platform driver layer on AI340 hardware platform. This design provides the easy to include / exclude design for the SDK modules/drivers using the configuration file (.h) in the final firmware application. And the USER can easily integrate the TouchGFX UI into the SDK and use it on the AI340 hardware platform. Using this design, the USER can easily focus on the design of the end application. The below diagram depicts the overall design architecture of the final firmware application.



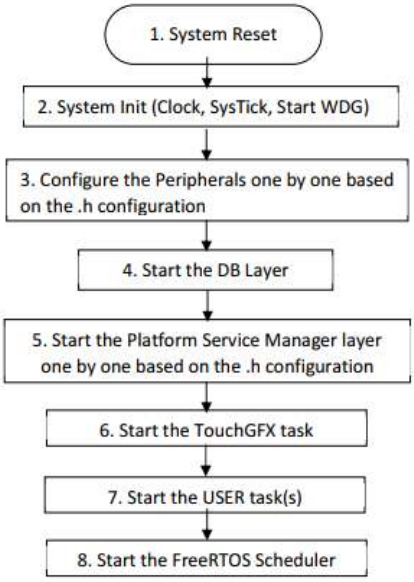
The user interacts with the SDK for the below functionalities:

1) Enable and disable the modules via the configuration file.
2) Provide default configuration for the properties of the modules as per their requirements.
3) Access Data Layer Data Base to get/set individual properties of the modules.

The user is free to enable any of the services or modules if needed to improve memory constraints.
So, the first step is to enable the desired services in the configuration file and then configure the properties for each manager. The managers are directly connected to the OS and work automatically on the background, so the user does not need to worry about the usage or the error management. In the below sections a detailed description is provided on how each module of the SDK can be accessed by the user for full filling their requirements.

SDK Boot flow

The diagram depicts the SDK boot flow for the MAXAI430 platform



Application and SDK Interaction

The user can interact with the SDK to configure the individual modules of the SDK. This can happen either during power up configuration or during the run time configuration.

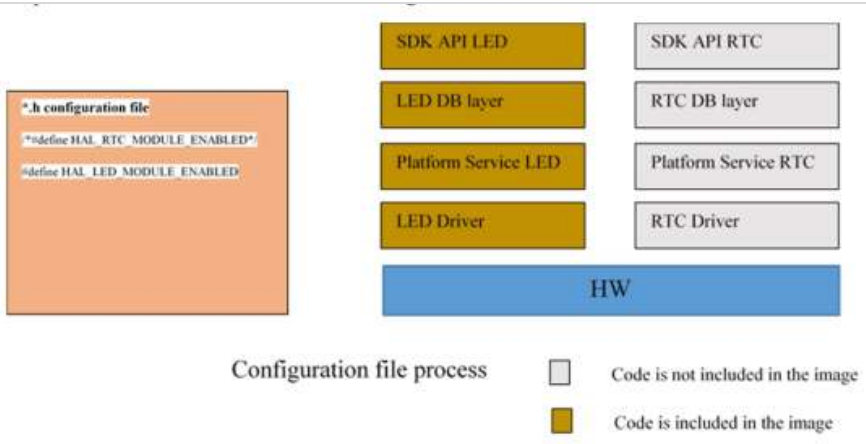
SDK Module Default Configuration

The user can configure the SDK by enabling and disabling individual modules. If a user enables or disables a particular module the entire stack is disabled for that module as shown in the below diagram

For Example, If the USER disables the RTC driver, the RTC related (SDK USER APIs, RTC DB Layer, RTC Platform service and RTC Driver) codes will not be included in the final firmware image.

That is, each device driver related SDK APIs, DB layer service, platform service and driver will be blocked with preprocessor MACROs (e.g., HAL_RTC_MODULE_ENABLED).

This implementation will reduce the final image size.



For example: - If user wants to enable the keypad module in the current build, user must configure the below mentioned variable as PS_ENABLE. **#define SDK_SERVICE_KEYPAD PS_ENABLE**

The user can find the configuration file in the source code in the below-mentioned path. Users can configure variables for any modules based on the requirements in the AI430_Config.h.

AI430_Project\Core\Maximatecc\Inc\AI430_Config.h

Each module in the configuration file is differentiated with Headers/comments and users can easily find the SDK modules they are looking for.

For Example: Keypad Module configuration

```
/*****************************************************************************
 *
 *                        Keypad Module Configuration
 *
 *****************************************************************************/
```

As shown in the above picture, the keypad module configurations are listed in the header file after the above comments. You can find similar comments for each module section in the configuration file.

The user can configure certain parameters per module which will impact the default configuration of the individual modules. This can be done by modifying the configuration file, which is used by the SDK to configure the individual modules during the power up sequence. Once configured, the modules continue using the same configuration until it is changed by the user.

For Example: - The below parameter configures the keypad backlight. It can be configured as ON / OFF and when the device powers up the SDK Reads this configuration file and updates the keypad backlight accordingly. In the MAXAI430 the default configuration for this parameter is true and hence the keypad backlight is always ON after device powers up unless the application turns it of during runtime.

```
/*!
 * MACRO Supported
 *
 * Keypad backlight configuration state
 * 1: KEY_BACKLIGHT_ON /
 * 0: KEY_BACKLIGHT_OFF
 */
#define KEYPAD_BACKLIGHT_CFG_STATE                    KEY_BACKLIGHT_ON
```

When the user modifies any configuration in the config.h file, the user will have to re – compile the source code and flash the updated binaries to the device and verify the changes.

Run Time Configuration

The user can modify certain parameters per module during the run time to interact with the module and to execute their desired functionality. This can be achieved by using the Data Layer Data Base APIs to read/write into the DB entries for each module.

Data Layer DB will collect the data from the platform service / platform layers and update the data into the proper variable.

Data Layer DB will be accessed by using GET/SET APIs from the application. If the application needs any platform related data, it uses the GET/SET API of the DB layer with the proper platform field name/id.

Keypad	Digital Input	Configurable Inputs
Power Mode	Light Sensor	Warning Lights
LED	Power Monitoring	USB
Bluetooth	RTC	SW Timer
Camera	EEPROM	WatchDog
LCD	SDRAM	Flash
CAN	J1939	

The Data Layer DB will interact with the Platform service through platform service SDK GET/SET APIs.

DB Layer USER APIs

The USER shall access the DB layer field through the below set and get functions.

Function Name: GET_DL

No	API Syntax	Parameter	Return Value
1	**GET_DL()** The user can use this API to retrieve the value of the data from the database. The Field id is defined in the database.h file which identifies the data field we are interested in.	uint16_t dl_index , int8_t *buf Value: We have to pass the Data ID for the data field we are looking to retrieve the data and then pass a buffer where the data to be written will be stored when the function call returns to the application.	0: Failure 1: Success

Below is the snippet of the description of the function:

```
/** @brief Set_DL
 *
 *  This function will Set the data to the platform service
 *
 *  @param  dl_index[IN]   : DB index value
 *          buf[IN]          : input buffer
 *
 *  @return  0 : FAILURE
 *           1 : SUCCESS
 *          -3 : NULL_POINTER
 *
 */
int8_t Set_DL(uint16_t dl_index, uint8_t *buf)
```

Function Name: SET_DL

No	API Syntax	Parameter	Return Value
1	**SET_DL()** The user can use this API to 1: Success store the value of the data from the database. The field id is defined in the database.h file which identifies the data field we are interested in.	uint16_t dl_index , int8_t *buf Value: We have to pass the index ID for the data field we are looking to retrieve the data and then pass a buffer where the data to be written will be stored when the function call returns to the application	0: Failure 1: Success

Below is the snippet of the description of the function:

```
/** @brief Get_DL
 *
 *  This function will get the data from the platform service
 *
 *  @param  dl_index[IN]   : DB index value
 *          buf[IN]      : input buffer
 *
 *  @return  0 : FAILURE
 *           1 : SUCCESS
 *          -3 : NULL_POINTER
 *
 */
int8_t Get_DL(uint16_t dl_index, uint8_t *buf)
```

For Example:

If we want to get the current status of the keypad backlight and then toggle it, we can do so by using the below code snippet,

```
/* Get the current backlight status */
If (Get_DL(KEYPAD_BACKLIGHT , &state) == SUCCESS)
{
  /* If current state is ON, set it OFF */
  If ( state == KEY_BACKLIGHT_ON )
  {
   State = KEY_BACKLIGHT_OFF;
   Set_DL(KEYPAD_BACKLIGHT , &state);
   }
   /* If current state is OFF, set it ON */
  Else
  {
   State = KEY_BACKLIGHT_ON;
   Set_DL(KEYPAD_BACKLIGHT , &state);
  }
}
```

SDK Modules

Described below are the functionalities supported by each SDK module which can be used by the application developer to full fill their requirements.

Keypad Module

The User would be able to use the below functionalities of the keypad module via the DB variables and configuration file.

Keypad Module Enable/Disable

The SDK provides the user the ability to enable/disable the Keypad functionality by modifying the default configuration file. Please see section Keypad sample Configuration

No	Variables	Options	Default State	Description
1	SDK_SERVICE_KEYPAD	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the Keypad module in the SDK. **PS_DISABLE**: Disables the Keypad module in the SDK.

Keypad Backlight ON/OFF

The AI430 SDK supports default configuration of the Keypad backlight and this can be done by modifying the below parameter in the configuration file. Please see section Keypad sample Configuration.

No	Variables	Options	Default State	Description
1	KEYPAD_BACKLIGHT_CFG_STATE	1:KEY_BACKLIGHT_ON 0:KEY_BACKLIGHT_OFF	KEY_BACKLIGHT_ON	User can configure the default state of the keypad Backlight to ON/OFF using this variable.

During runtime the user can read and modify the keypad backlight by reading and writing to the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
KEYPAD_BACKLIGHT	DBu8	READ/WRITE	1	ON/OFF	This field is used to turn ON/OFF the Keypad Backlight.

The sample code for set/get the Backlight using key#1 is as below:

```
if (KEY1_SHORT_PRESS == val)
{
state = KEY_BACKLIGHT_ON;
Set_DL(KEYPAD_BACKLIGHT , &state);
}
/* Get the backlight state */
Get_DL(KEYPAD_BACKLIGHT, &state);
```

Keypad Time Out Configuration

The AI430 SDK user can configure the timeout value of keypress to differentiate between short press and long press. Short Press can be configured in the range (> 10ms && < 255ms). If the key is pressed longer than the short press timeout it would be considered as long press. This default configuration can be done in the AI430_config.h. Please see section Keypad sample Configuration.

No	Variables	Options	Default State	Description
1	SHORT_PRESS_TIMEOUT	Short Press (> 10 && < 255) Long press (> short press time)	10	The user can configure the timeout value of keypress to differentiate

Keypad Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the keypad module in the configuration file. Please see section Keypad sample Configuration.

No	Variables	Options	Default State	Description
1	PS_KEYPAD_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

Keypad Keys Enable/Disable

The AI430 SDK supports four keys and the user can enable/disable each of the keys using the default configuration file.

No	Variables	Options	Default State	Description
1	CONF_KEYPAD_01_STATE	PS_ENABLE PS DISABLE	PS_ENABLE	**PS_ENABLE**: Enable the HW Key01 in the Keypad SDK platform service. **PS_DISABLE**: Disable the HW Key01 in the Keypad SDK platform service.
2	CONF_KEYPAD_02_STATE	PS_ENABLE PS DISABLE	PS_ENABLE	**PS_ENABLE**: Enable the HW Key02 in the Keypad SDK platform service. **PS_DISABLE**: Disable the HW Key02 in the Keypad SDK platform service.
3	CONF_KEYPAD_03_STATE	PS_ENABLE PS DISABLE	PS_ENABLE	**PS_ENABLE**: Enable the HW Key03 in the Keypad SDK platform service. **PS_DISABLE**: Disable the HW Key03 in the Keypad SDK platform service.
4	CONF_KEYPAD_04_STATE	PS_ENABLE PS DISABLE	PS_ENABLE	**PS_ENABLE**: Enable the HW Key04 in the Keypad SDK platform service. **PS_DISABLE**: Disable the HW Key04 in the Keypad SDK platform service.

Keypad Keys Read Status.

The AI430 SDK user can then read the Key status variables to know if the keys are active or inactive. This DB entry has to be read first for receiving the keypress event. If the *KEY_STATUS_01* is KP_ACTIVE then the USER has to read the *KEY_PRESS_01* DB variable to check the state of the key press.

Field ID	Data Type	Permission	Size	Description	Comments
KEY_STATUS_01	DBu8	READ	1	KP_ACTIVE/KP_INACTIVE	This field is used to Read the status of the Key#1.
KEY_STATUS_02	DBu8	READ	1	KP_ACTIVE/KP_INACTIVE	This field is used to Read the status of the Key#2.
KEY_STATUS_03	DBu8	READ	1	KP_ACTIVE/KP_INACTIVE	This field is used to Read the status of the Key#3.
KEY_STATUS_04	DBu8	READ	1	KP_ACTIVE/KP_INACTIVE	This field is used to Read the status of the Key#4.

The user can read the key state (short press/ long press / inactive) by continuously monitoring the below DB variables. Once the USER gets any one of the Keypress events (SHORT_PRESS/LONG_PRESS), the USER has to ACK the key press (KEY_PRESS_01) with the value 1.

Field ID	Data Type	Permission	Size	Description	Comments
KEY_PRESS_01	DBu8	READ/WRITE	1	INACTIVE/SHORT_PRESS/LONG_PRESS	This field detects the type of KeyPress for Key#1.
KEY_PRESS_02	DBu8	READ/WRITE	1	INACTIVE/SHORT_PRESS/LONG_PRESS	This field detects the type of KeyPress for Key#2.
KEY_PRESS_03	DBu8	READ/WRITE	1	INACTIVE/SHORT_PRESS/LONG_PRESS	This field detects the type of KeyPress for Key#3.
KEY_PRESS_04	DBu8	READ/WRITE	1	INACTIVE/SHORT_PRESS/LONG_PRESS	This field detects the type of KeyPress for Key#4.

Below is a code snippet for the keypress read and acknowledge use case.

```
#if(CONF_KEYPAD_01_STATE == PS_ENABLE)
/* Read the Key Status from the DB */
res = Get_DL(KEY_STATUS_01, &key_status);
/* Proceed only if the Key # 1 is active */
if (KEY_INACTIVE != key_status)
{
res = Get_DL(KEY_PRESS_01, &key_status);
if (SHORT_PRESSED == key_status)
{
key_status = 1;
/* ACK the Keypress */
res = Set_DL(KEY_PRESS_01, &key_status); }
else
if (LONG_PRESSED == key_status)
{
key_status = 1;
/* ACK the Keypress */
res = Set_DL(KEY_PRESS_01, &key_status);
modelListener->keyUpdate(KEY1_LONG_PRESS);
}
}
#endif //KEYPAD_01
```

Keypad Sample Configuration

```
/*****************************************************************************************************************************************************************
*
                                                    * Keypad Module Configuration
*
*****************************************************************************************************************************************************************/
/*!
* Keypad Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_KEYPAD PS_ENABLE
#if (SDK_SERVICE_KEYPAD == PS_ENABLE)
/*!
* Keypad Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_KPD_TASK_PRIORITY osPriorityIdle
/*!
* MACRO Supported
*
* Keypad backlight configuration state
* 1: KEY_BACKLIGHT_ON /
* 0: KEY_BACKLIGHT_OFF
*/
* SHORT Press timeout (Millisecond)
*/
#define SHORT_PRESS_TIMEOUT 10
#define KEYPAD_BACKLIGHT_CFG_STATE KEY_BACKLIGHT_ON
/*!


/*!
* MACOR Supported
*
* Keypad configuration state
* 1: PS_ENABLE /
* 0: PS_DISABLE
*/
#define CONF_KEYPAD_01_STATE PS_ENABLE
#define CONF_KEYPAD_02_STATE PS_ENABLE
#define CONF_KEYPAD_03_STATE PS_ENABLE
#define CONF_KEYPAD_04_STATE PS_ENABLE
#endif //SDK_SERVICE_KEYPAD
```

Digital Output Module

The User would be able to use the below functionalities of the digital output module via the DB variables and configuration file.

Digital Output Module Enable/Disable

The SDK provides the user the ability to enable/disable the Digital Output functionality by modifying the default configuration file. Please see section Digital Output Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_DIGITAL_OUTPUT	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the Digital Output module in the SDK. **PS_DISABLE**: Disables the Digital Output module in the SDK.

Digital Output Configuration

The AI430 SDK supports default configuration of the digital output status and this can be done by modifying the below parameter in the configuration file. Please see section Digital Output Sample Configuration.

No	Variables	Options	Default State	Description
1	DIGITAL_OUTPUT_CFG	CONF_OPEN_DRIVE_DRIVER/ CONF_LOW_SIDE_DRIVER/ CONF_HIGH_SIDE_DRIVER	CONF_OPEN_DRIVE_DRIVER	User can select the Digital output configuration as open drive driver when the digital output status is OFF User can select the Digital output as configuration as low side driver. User can select the Digital output as configuration as high side driver.

The user can do the same configuration during the runtime via the DB variables and configuration file as shown:

Field ID	Data Type	Permission	Size	Description	Comments
DIGITAL_OUTPUT_01_CFG	DBu8	READ/WRITE	1	OPEN_DRIVE/ HIGH_SIDE_DRIVER / LOW_SIDE_DRIVER	This field is used to configure the digital output as high side, low side or open drive. The field is also used to enable/ disable the Digital Output. The status of the field can also be read back once enabled.

Below is the sample code for accessing the Digital output configuration DB variables.

```
/* Read the Digital Output configuration */
Get_DL(DIGITAL_OUTPUT_01_CFG, &state);
if(state == CONF_LOW_SIDE_DRIVER)
{
state = CONF_HIGH_SIDE_DRIVER;
/* Set the High side Digital Output */
Set_DL(DIGITAL_OUTPUT_01_CFG, &state);
}
```

```
Get_DL(DIGITAL_OUTPUT_01_CFG, &state);
if(state == CONF_HIGH_SIDE_DRIVER)
{
state = CONF_LOW_SIDE_DRIVER;
/* Set the Low side Digital Output */
Set_DL(DIGITAL_OUTPUT_01_CFG, &state);
}
```

Digital Output ON/OFF

The AI430 SDK user can turn ON / OFF the digital output during runtime. To do so he can use the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
DIGITAL_OUTPUT_01_STATE	DBU8	READ/WRITE	1	CONF_DIGITAL_OUTPUT_ON/CONF_DIGITAL_OUTPUT_OFF	This field is used to turn ON or OFF the digital input. The status of the field can also be read back once to get the current status of the pins.

Below is the sample code for turning ON/OFF the Digital output DB variable.

```
state = CONF_DIGITAL_OUTPUT_ON;
/* Set the open drive Digital Output */
Set_DL(DIGITAL_OUTPUT_01_STATE, &state);
state = CONF_DIGITAL_OUTPUT_OFF;
/* Set the open drive Digital Output */
Set_DL(DIGITAL_OUTPUT_01_STATE, &state);
```

Digital Output Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would go and read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. configuration.

No	Variables	Options	Default State	Description
1	PS_DIO_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100ms	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the values in the Database.

Digital Output Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the digital output module in the configuration file. Please see section Digital Output Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_DIO_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

Digital Output Sample Configuration

```
/*****************************************************************************************************************************************************
*
                                                    * Digital Output Module Configuration
*
*****************************************************************************************************************************************************/
 /*!
 * DIO Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE)
 Macros
 */
 #define SDK_SERVICE_DIGITAL_OUTPUT PS_ENABLE
 #if (SDK_SERVICE_DIGITAL_OUTPUT == PS_ENABLE)
 /*!
 * DIO Task Periodicity 100ms
 */
 #define PS_DIO_TASK_TIMEOUT 100
 /*!
 * DIO Task Priority
 * osPriorityNone = 0,
 * osPriorityIdle = 1,
 * osPriorityLow = 8,
 * osPriorityLow1 = 8+1,
 * ,, ,,
 * ,, ,,
 * osPriorityISR = 56,
 * osPriorityError = -1,
 * osPriorityReserved = 0x7FFFFFFF
 */
 #define PS_DIO_TASK_PRIORITY osPriorityIdle
 /*!
 * Select the DIGITAL_OUTPUT_CFG 00 : CONF_LOW_SIDE_DRIVER
 * 01 : CONF_HIGH_SIDE_DRIVER
 * 02 : CONF_OPEN_DRIVE_DRIVER
 */
 #define DIGITAL_OUTPUT_CFG CONF_OPEN_DRIVE_DRIVER
 #endif //SDK_SERVICE_DIGITAL_OUTPUT
```

Configurable Inputs Module

The User would be able to use the below functionalities of the keypad module via the DB variables and configuration file.

Configurable Inputs Module Enable/Disable

The SDK provides the user the ability to enable/disable the configurable functionality by modifying the default configuration file. Please see section Configurable Inputs Default Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_CG_INPUT	PS_ENABLE PS_DISABLE	PS_ENABLE	PS_ENABLE: Enables the Configurable input module in the SDK. PS_DISABLE: Disables the configurable input module in the SDK.

Configurable Inputs Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the configurable inputs module in the configuration file. Please see section Configurable Inputs Default Configuration.

No	Variables	Options	Default State	Description
1	PS_CFG_INPUT_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 ,	osPriorityIdle	User can select any one of the priorities based on the application requirement.

		osPriorityISR , osPriorityError , osPriorityReserved		

Configurable Inputs Task Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would go and read the hardware for the configured inputs and update it in the DB so that when the user reads the DB, he will receive the latest updated data. This default configuration can be done in the AI430_config.h. Please see section Configurable Inputs Default Configuration.

No	Variables	Options	Default State	Description
1	PS_CFG_INPUT_TASK _TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

Configurable Inputs – Configure the Number of Samples.

The AI430 SDK user can configure the number of samples to be considered for the average calculation of the readings from the hardware before it is updated to the database. This would improve the accuracy of the data updated in the DB. This default configuration can be done in the AI430_config.h. Please see section Configurable Inputs Default Configuration.

No	Variables	Options	Default State	Description
1	CONF_INPUT_01_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 1.
2	CONF_INPUT_02_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 2.
3	CONF_INPUT_03_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 3.
4	CONF_INPUT_04_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 4.
5	CONF_INPUT_05_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 5.
6	CONF_INPUT_06_NUMB _SAMPLES	MIN VALUE: 1 MAX VALUE: 255	1	User can set the number of sample values to be considered for the average calculation for configurable input 6.

The user can dynamically set/get the number of samples to be considered for the average calculation for configurable input during the run time. To do so, the user can use the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
CFG_INPUT_01_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.
CFG_INPUT_02_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.
CFG_INPUT_03_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.
CFG_INPUT_04_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.
CFG_INPUT_05_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.
CFG_INPUT_06_NUMB_SAMPLES	DBu8	READ/WRITE	1	1-255 range	This field is used to set number of samples to get An average value. The field is also used to read the number of samples set.

The below code snippet shows how the sample configuration can be altered from the application code:

```
if(update_sample)
{
int num_sample = 10;
/* Set the sample count to 10 for configurable input 1 */
res = DL_Set(CFG_INPUT_01_NUMB_SAMPLES,&num_sample);
}
```

Configurable Inputs configuration

The AI430 SDK user can configure the 5 available configurable inputs as per his desired requirement as supported by the platform. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Configurable Inputs Default Configuration.

No	Variables	Options	Default State	Description
1	CONF_INPUT_TYPE_01	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V	CI_DIGITAL_STB	User can configure the configurable input type 1 for input voltage, input frequency, input resistance, Digital STB and Digital STG
	CONF_INPUT_TYPE_02	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V	CI_DIGITAL_STG	User can configure the configurable input type 1 for input voltage, input frequency, input resistance, Digital STB and Digital STG
	CONF_INPUT_TYPE_03	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V	CI_INPUT_RESISTANCE	User can configure the configurable input type 1 for input voltage, input frequency, input resistance, Digital STB and Digital STG
	CONF_INPUT_TYPE_04	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V	CI_INPUT_RESISTANCE	User can configure the configurable input type 1 for input voltage, input frequency, input resistance, Digital STB and Digital STG
	CONF_INPUT_TYPE_05	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V	CI_INPUT_RESISTANCE	User can configure the configurable input type 1 for input voltage, input frequency, input resistance, Digital STB and Digital STG
	CONF_INPUT_TYPE_06	CI_INPUT_VOLTAGE_LOW_6V CI_INPUT_CURRENT	CI_INPUT_CURRENT	User can configure the input type 6 for input current and input voltage.

The user also can run time configure the 6 available configurable inputs as per his desired requirement as supported by the platform. To do so the user can use the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
CFG_INPUT_01_TYPE	DBu8	READ/WRITE	1	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF,	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input. The status of the CFG_Input#01 type can be read using this field.
CFG_INPUT_02_TYPE	DBu8	READ/WRITE	1	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input. The status of the CFG_Input#02 type can be read using this field.
CFG_INPUT_03_TYPE	DBu8	READ/WRITE	1	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG,	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input.

				CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF	The status of the CFG_Input#03 type can be read using this field.
CFG_INPUT_04_TYPE	DBu8	READ/WRITE	1	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input. The status of the CFG_Input#04 type can be read using this field.
CFG_INPUT_05_TYPE	DBu8	READ/WRITE	1	CI_INPUT_FREQUENCY, CI_INPUT_RESISTANCE, CI_DIGITAL_STG, CI_DIGITAL_STB, CI_INPUT_VOLTAGE_HIGH, CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input. The status of the CFG_Input#05 type can be read using this field.
CFG_INPUT_06_TYPE	DBu8	READ/WRITE	1	CI_INPUT_CURRENT CI_INPUT_VOLTAGE_LOW_6V, CI_INPUT_OFF	This field is used Configure the input to any one of the types suggested. The field is also used to turn off the input. The status of the CFG_Input#06 type can be read using this field.

The below code snippet shows how the Configurable inputs type can be configured during the run time.

```
/* Read the current configuration for Configurable input 1 */
Get_DL(CFG_INPUT_01_TYPE , &input1_type);
/* If it is currently configured as frequency , change it to resistance */
if(CI_INPUT_FREQUENCY == input1_type)
{
input1_type = CI_INPUT_RESISTANCE;
Set_DL(CFG_INPUT_01_TYPE , &input1_type);
}
```

Once the user configures the various configurable inputs the platform service will read the data from the hardware every time the task time out occurs and update the below DB variables. The user can then access the same by using the DL_get/DL_set API's. For example, if he has configured the Configurable input 1 as *CI_INPUT_VOLTAGE_HIGH* then he will have to read the *CFG_INPUT_01_VOLTAGE_32V* DB entry to read the voltage value in milli volts.

Field ID	Data Type	Permission	Size	Description	Comments
CFG_INPUT_01_FREQUENCY	DbU32	Read	4	10Hz-20000Hz range	This field is used to read the frequency of CFG_Input#01. The frequency is read in Hertz.
CFG_INPUT_01_VOLTAGE_32V	DBu16	READ	2	0-32000 range	This field is used to read the high voltage of CFG_Input#01. The voltage is read in milli-volts.
CFG_INPUT_01_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the Resistance of CFG_Input#01. Resistance is read in Ohms.
CFG_INPUT_01_DIGITAL_STG	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#01. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_01_DIGITAL_STB	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#01. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_02_FREQUENCY	DBu32	READ	4	10Hz-20000Hz range	This field is used to read the frequency of CFG_Input#02. The frequency is read in milli-Hertz.
CFG_INPUT_02_VOLTAGE_32V	DBu16	READ	2	0-32000	This field is used to read the voltage of CFG_Input#02. The voltage is read in milli-volts.
CFG_INPUT_02_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the low voltage of CFG_Input#02. The voltage is read in Volts
CFG_INPUT_02_RESISTANCE	DBu16	READ	2	1ohm – 10KOhm range	This field is used to read the Resistance of CFG_Input#02. Resistance is read in Ohms.
CFG_INPUT_02_DIGITAL_STG	DBu8	READ	1	TRUE /FALSE	This field is used to read the Digital Input level of CFG_Input#02. If TRUE = digital active and FALSE = digital Inactive.
CFG_INPUT_02_DIGITAL_STB	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#02. If TRUE = digital active and FALSE = digital Inactive.

CFG_INPUT_03_FREQUENCY	DBu32	READ	4	0-20000 range	This field is used to read the frequency of CFG_Input#03. The frequency is read in milli-Hertz.
CFG_INPUT_03_VOLTAGE_32V	DBu16	READ	2	0-32000 range	This field is used to read the voltage of CFG_Input#03. The voltage is read in milli-Volts.
CFG_INPUT_03_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the low voltage of CFG_Input#03 The voltage is read in Volts.
CFG_INPUT_03_RESISTANCE	DBu16	READ	2	1ohm – 10KOhm range	This field is used to read the Resistance of CFG_Input#03. Resistance is read in ohms.
CFG_INPUT_03_DIGITAL_STG	DBu8	READ	1	TRUE /FALSE	This field is used to read the Digital Input level of CFG_Input#03. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_03_DIGITAL_STB	DBU8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#03. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_04_FREQUENCY	DBU32	READ	4	0-20000 range	This filed is used to read the frequency of CFG_Input#04. The frequency is read in milli-Hertz.
CFG_INPUT_04_VOLTAGE_32V	DBu16	READ	2	0-32000 range	This field is used to read the voltage of CFG_Input#04. The voltage is read in milli-volts.
CFG_INPUT_04_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the low voltage of CFG_Input#04. The voltage is read in Volts.
CFG_INPUT_04_RESISTANCE	DBu16	READ	2	1ohm – 10Kohm range	This filed is used to read the Resistance of CFG_Input#04. Resistance is read in Ohms
CFG_INPUT_04_DIGITAL_STG	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#04. If TRUE = digital active and FALSE = digital inactive
CFG_INPUT_04_DIGITAL_STB	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#04. If TRUE = digital active and FALSE = digital inactive
CFG_INPUT_05_FREQUENCY	DBu32	READ	4	0-20000 range	This filed is used to read the frequency of CFG_Input#05. The frequency is read in milli-Hertz.
CFG_INPUT_05_VOLTAGE_32V	DBu16	READ	2	0-32000 range	This field is used to read the voltage of CFG_Input#05. The voltage is read in milli-volts
CFG_INPUT_05_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the low voltage of CFG_Input#05. The voltage is read in Volts
CFG_INPUT_05_RESISTANCE	DBu16	READ	2	1ohm – 10Kohm range	This filed is used to read the Resistance of CFG_Input#05. Resistance is read in Ohms
CFG_INPUT_05_DIGITAL_STG	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#05. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_05_DIGITAL_STB	DBu8	READ	1	TRUE / FALSE	This field is used to read the Digital Input level of CFG_Input#05. If TRUE = digital active and FALSE = digital Inactive
CFG_INPUT_06_CURRENT	DBu32	READ	4	0-20000 range	This field is used to read the current of CFG_Input#06. The current is read in milli-amps
CFG_INPUT_02_VOLTAGE_LOW_6V	DBu16	READ	2	0-6V	This field is used to read the low voltage of CFG_Input#01. The voltage is read in Volts

The below code sample shows the configuration values read from CI1 during runtime:

```
if(CI_INPUT_FREQUENCY == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_FREQUENCY , (uint8_t *)&val);
}
else
if(CI_INPUT_VOLTAGE_HIGH == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_VOLTAGE_32V , (uint8_t *)&val);
}
else
if(CI_INPUT_VOLTAGE_LOW_6V == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_VOLTAGE_LOW_6V , (uint8_t *)&val);
}
else
if(CI_INPUT_RESISTANCE == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_RESISTANCE , (uint8_t *)&val);
}
else
if(CI_DIGITAL_STG == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_DIGITAL_STG , (uint8_t *)&val);
}
else
if(CI_DIGITAL_STB == input1_type)
{
val = 0;
Get_DL(CFG_INPUT_01_DIGITAL_STB , (uint8_t *)&val);
}
```

Configurable Inputs Default Configuration

```
/*****************************************************************************************************************************************************************
                                                                        *
                                                                * Configurable Input Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * Config input Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_CFG_INPUT PS_ENABLE
  #if (SDK_SERVICE_CFG_INPUT == PS_ENABLE)
  /*!
  * Config_input Task Periodicity 100ms
  */
  #define PS_CFG_INPUT_TASK_TIMEOUT 100
  /*!
  * CFG Input Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  * ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_CFG_INPUT_TASK_PRIORITY osPriorityIdle
  #define CONF_INPUT_TYPE_01 CI_DIGITAL_STB
  #define CONF_INPUT_TYPE_02 CI_DIGITAL_STG
  #define CONF_INPUT_TYPE_03 CI_INPUT_RESISTANCE
  #define CONF_INPUT_TYPE_04 CI_INPUT_RESISTANCE
  #define CONF_INPUT_TYPE_05 CI_INPUT_RESISTANCE
  #define CONF_INPUT_TYPE_06 CI_INPUT_CURRENT
  #define CONF_INPUT_01_NUMB_SAMPLES 1
  #define CONF_INPUT_02_NUMB_SAMPLES 1
  #define CONF_INPUT_03_NUMB_SAMPLES 1
  #define CONF_INPUT_04_NUMB_SAMPLES 1
  #define CONF_INPUT_05_NUMB_SAMPLES 1
  #define CONF_INPUT_06_NUMB_SAMPLES 1
  #endif //SDK_SERVICE_CFG_INPUT
  #endif //SDK_SERVICE_CFG_INPUT
```

Light Sensor Module

The User would be able to use the below functionalities of the light sensor module via the DB variables and configuration file.

Light Sensor Enable/Disable

The SDK provides the user the ability to enable/disable the Light Sensor functionality by modifying the default configuration file. Please see section Light Sensor Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_LIGHT_SENSOR	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the light sensor module in the SDK. **PS_DISABLE**: Disables the light sensor module in the SDK.

Light Sensor Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Light Sensor Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_LS_TASK_TIMEOUT	MIN VALUE :50 MAX VALUE :500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

Light Sensor Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section Light Sensor Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_LS_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

Light Sensor Conversion Time

The AI430 SDK allows the user to modify the conversion time for the light sensor. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Light Sensor Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_LIGHT_SENSOR_ CONVERSION_TIME	LS_CONVERSION_ TIME_100/ LS_CONVERSION_ TIME_800	LS_CONVERSION_TIME_100	User can set the conversion time as 100 ms. User can set the conversion time as 800 ms.

During runtime the user can read and modify the light sensor conversion time as 00ms/800ms by reading and writing to the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
LIGHT_SENSOR_ CONVERSION_TI ME	DBu8	READ/WRITE	1	100MS/800MS	This field is used to read and write the Light sensor conversion time.

Below code snippet shows how the light sensor conversion time can be read/written into the DB.

```
/* Validate the Light Sensor Conversion time */
case 1:
state = LS_CONVERSION_TIME_100;
/* Set the Light Sensor conversion time */
Set_DL(LIGHT_SENSOR_CONVERSION_TIME, &state);
break;
case 2:
state = LS_CONVERSION_TIME_800;
/* Set the Light Sensor conversion time */
Set_DL(LIGHT_SENSOR_CONVERSION_TIME, &state);
break;
/* Get the Light Sensor conversion time */
Get_DL(LIGHT_SENSOR_CONVERSION_TIME, &state);
```

Light Sensor Conversion Mode

The AI430 SDK allows the user to modify the conversion mode for the light sensor based on which the light sensor will continuously fetch the data from the sensor and update the DB or do it just one time. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Light Sensor Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_LIGHT_SENSOR_ CONVERSION_MODE	LS_CONTINUOUS_ MODE_ CONV/ LS_SINGLE_MODE _CONV/ LS_SHUTDOWN	LS_CONTINUOUS_MODE_C ONV	If the user sets the single mode the sensor value is updated in the DB following the light sensor trigger variable. If the user sets the continuous mode, the light sensor value will be updated every cycle of the configured conversion time User can configure the conversion mode as shutdown mode to turn off the sensor.

The light sensor trigger variable must work as follows:

Field ID	Data Type	Permission	Size	Description	Comments
LIGHT_SENSOR_TRIGGER	DBu8	READ/WRITE	1	TRUE / FALSE	This field is used to do the conversion when the light sensor trigger variable is set to true. The platform service will automatically clear the variable every time it is set.

During runtime the user can read and modify the light sensor conversion mode by reading and writing to the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
LIGHT_SENSOR_CONVERSION_MODE	DBu8	READ/WRITE	1	SINGLE_SHOT/CONTINUOUS	This field is used to set the Light sensor conversion mode. The field is also used to read back the conversion type set.

The below code snippet shows how to set/get the Light Sensor Mode.

```
/* Set the Light Sensor Mode Conversion */
case 3:
val = LS_SHUTDOWN;
/* Set the Light Sensor conversion mode */
Set_DL(LIGHT_SENSOR_CONVERSION_MODE, &val);
mode = val;
break;
case 4:
val = LS_SINGLE_MODE_CONV;
/* Set the Light Sensor conversion mode */
Set_DL(LIGHT_SENSOR_CONVERSION_MODE, &val);
mode = val;
break;
case 5:
val = LS_CONTINUOUS_MODE_CONV;
/* Set the Light Sensor conversion mode */
Set_DL(LIGHT_SENSOR_CONVERSION_MODE, &val);
mode = val;
break;
/* *To get the LS Mode */
Get_DL(LIGHT_SENSOR_CONVERSION_MODE, &val);
```

Light Sensor Sample Data

The AI430 SDK user can configure the light sensor in single shot or continuous mode. If the user configures the conversion mode as Single shot the sensor value is read and updated the DB following the Light Sensor Trigger variable.
If the user configures the light sensor in continuous mode, then the data from the light sensor will continuously be fetched and updated in the DB. The user can refresh the UI accordingly. To read the light sensor data during the runtime, the below DB variable can be used:

Field ID	Data Type	Permission	Size	Description	Comments
LIGHT_SENSOR_DATA	float	READ	4	optical power in nW/cm2	Light sensor data value

The below code snippet shows how the UI can read the light sensor data in continuous mode:

```
#define REFRESH_TIME_IN_SEC 33
#if (SDK_SERVICE_LIGHT_SENSOR == PS_ENABLE)
uint32_t val = 0;
uint8_t lmode = 0;
refresh_val++;
if (REFRESH_TIME_IN_SEC == refresh_val)
{
  refresh_val = 0;
  /* Get the Light Sensor Data */
  Get_DL(LIGHT_SENSOR_DATA , (uint8_t *)&val);
  /* Update the UI accordingly*/
}
#endif
```

The below code snippet shows how the UI can read the light sensor data in single shot mode:

```
val = LS_SINGLE_MODE_CONV;
/* Set the Light Sensor conversion mode */
Set_DL(LIGHT_SENSOR_CONVERSION_MODE, &val);
for(;;)
{
/* Get the LS Data when the light sensor trigger variable is set to true*/
Get_DL(LIGHT_SENSOR_DATA , (uint8_t *)&val);
if (counter == 10)
{
counter = 0;
Trigger = TRUE;
Set_DL(LIGHT_SENSOR_TRIGGER, & Trigger);
}
counter++;
}
```

Light Sensor Sample Configuration

```
/*****************************************************************************************************************************************************************
*
                                                        * Light Sensor Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * Light Sensor Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_LIGHT_SENSOR PS_ENABLE
  #if (SDK_SERVICE_LIGHT_SENSOR == PS_ENABLE)
  /*!
  * Light Sensor Task Periodicity 100ms
  */
  #define PS_LS_TASK_TIMEOUT 100
  /*!
  * light sensor Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  * ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_LS_TASK_PRIORITY osPriorityIdle
  /*!
  * MACOR Supported
  *
  * Light Sensor conversion time
  * 0: LS_CONVERSION_TIME_100 /
  * 1: LS_CONVERSION_TIME_800
  */

  #define CONF_LIGHT_SENSOR_CONVERSION_TIME
  LS_CONVERSION_TIME_100
  /*!
  * MACOR Supported
  *
  * Light Sensor conversion mode
  * 0: LS_SHUTDOWN /
  * 1: LS_SINGLE_MODE_CONV/
  * 2: LS_CONTINUOUS_MODE_CONV
  */
  #define CONF_LIGHT_SENSOR_CONVERSION_MODE
  LS_CONTINUOUS_MODE_CONV
  #endif //SDK_SERVICE_LIGHT_SENSOR
```

Warning Light Module

The User would be able to use the below functionalities of the light sensor module via the DB variables and configuration file.

Warning Light Module Enable/Disable

The SDK provides the user the ability to enable/disable the Warning Light functionality by modifying the default configuration file. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_ WARNING_LIGH T	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the warning light module in the SDK. **PS_DISABLE**: Disables the warning light module in the SDK.

Warning Light Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_WL_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the latest status of warning light in the Database.

Warning Light Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_WL_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

Max Warning Lights Configuration

The AI430 SDK allows the user to configure the maximum number of warning lights he needs. The platform supports maximum 20 warning lights. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	MAX_NUM_WARNING_LIGHT	MIN VALUE: 0 MAX VALUE:20	20	User can configure how many warning lights he would like to use.

Warning Lights Frequency Configuration

The AI430 SDK allows the user to configure the frequency of the warning lights. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	WARNING_LIGHT_FREQ	WL_kHz3_FREQ WL_kHz22_FREQ	WL_kHz3_FREQ	User can configure the frequency of warning lights as 3KHz or 22 KHz.

Warning Lights Enable/Disable

The AI430 SDK allows the user to enable/disable each of the warning lights. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_WARNING_LIGHT_01_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 1 independently using this configuration.
2	CONF_WARNING_LIGHT_02_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 2 independently using this configuration.
3	CONF_WARNING_LIGHT_03_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 3 independently using this configuration.
4	CONF_WARNING_LIGHT_04_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 4 independently using this configuration.
5	CONF_WARNING_LIGHT_05_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 5 independently using this configuration.
6	CONF_WARNING_LIGHT_06_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 6 independently using this configuration.
7	CONF_WARNING_LIGHT_07_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 7 independently using this configuration.
8	CONF_WARNING_LIGHT_08_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 8 independently using this configuration.
9	CONF_WARNING_LIGHT_09_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 9 independently using this configuration.
10	CONF_WARNING_LIGHT_10_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 10 independently using this configuration.
11	CONF_WARNING_LIGHT_11_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 11 independently using this configuration.
12	CONF_WARNING_LIGHT_12_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 12 independently using this configuration.
13	CONF_WARNING_LIGHT_13_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 13 independently using this configuration.
14	CONF_WARNING_LIGHT_14_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 14 independently using this configuration.
15	CONF_WARNING_LIGHT_15_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 15 independently using this configuration.

16	CONF_WARNING_LIGHT_16_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 16 independently using this configuration.
17	CONF_WARNING_LIGHT_17_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 17 independently using this configuration.
18	CONF_WARNING_LIGHT_18_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 18 independently using this configuration.
19	CONF_WARNING_LIGHT_19_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 19 independently using this configuration.
20	CONF_WARNING_LIGHT_20_ENABLE	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable Warning Light 20 independently using this configuration.

The user is allowed to enable/disable a warning light at run time by accessing the below DB variables:

Field ID	Data Type	Permission	Size	Description	Comments
WARNING_LIGHT_01_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_01. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_02_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_02. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_03_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_03. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_04_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_04. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_05_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_05. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_06_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_06. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_07_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_07. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_08_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_08. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_09_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_09. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_10_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_10. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_11_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_11. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_12_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_12. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_13_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_13. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_14_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_14. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_15_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_15. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_16_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_16. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_17_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_17. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_18_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_18. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_19_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_19. ON = Light up the LED, OFF = Turn off the LED.
WARNING_LIGHT_20_STATE	DBu8	READ/WRITE	1	ON /OFF	This field is used to enable/disable the WARNING_LIGHT_20. ON = Light up the LED, OFF = Turn off the LED.

The sample code to get/set the Warning Light status from DB is as below:

```
WL01BLINKView::WL01BLINKView()
{
#if (SDK_SERVICE_WARNING_LIGHT == PS_ENABLE)
/* Get the Warning Light Status from the DB */
Get_DL(WARNING_LIGHT_01_STATE, &state);
if (WL_ON == state)
{
state = WL_OFF;
/* Warning Light is ON */
Set_DL(WARNING_LIGHT_01_STATE, &state);
}
else
/* Warning Light is OFF */
#endif
}
```

Warning Lights Current Configuration

The AI430 SDK allows the user to modify the current for each warning light. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_WARNING_LIGHT_01_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 1 independently using this configuration.
2	CONF_WARNING_LIGHT_02_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 2 independently using this configuration.
3	CONF_WARNING_LIGHT_03_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 3 independently using this configuration.
4	CONF_WARNING_LIGHT_04_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 4 independently using this configuration.
5	CONF_WARNING_LIGHT_05_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 5 independently using this configuration.
6	CONF_WARNING_LIGHT_06_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 6 independently using this configuration.
7	CONF_WARNING_LIGHT_07_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 7 independently using this configuration.
8	CONF_WARNING_LIGHT_08_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 8 independently using this configuration.
9	CONF_WARNING_LIGHT_09_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 9 independently using this configuration.
10	CONF_WARNING_LIGHT_10_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 10 independently using this configuration.
11	CONF_WARNING_LIGHT_11_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 11 independently using this configuration.
12	CONF_WARNING_LIGHT_12_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 12 independently using this configuration.
13	CONF_WARNING_LIGHT_13_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 13 independently using this configuration.
14	CONF_WARNING_LIGHT_14_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 14 independently using this configuration.
15	CONF_WARNING_LIGHT_15_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 15 independently using this configuration.

16	CONF_WARNING_LIGHT_16_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 16 independently using this configuration.
17	CONF_WARNING_LIGHT_17_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 17 independently using this configuration.
18	CONF_WARNING_LIGHT_18_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 18 independently using this configuration.
19	CONF_WARNING_LIGHT_19_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 19 independently using this configuration.
20	CONF_WARNING_LIGHT_20_CURRENT	WL_OUT_CURRENT_MAX WL_OUT_CURRENT_MAX_2 WL_OUT_CURRENT_MAX_4 WL_OUT_CURRENT_MAX_8	WL_OUT_CURRENT_MAX	User can modify the current value for Warning Light 20 independently using this configuration.

Warning Lights Power ON State Configuration

The AI430 SDK allows the user to modify the power on State for each warning light. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_WARNING_LIGHT_01_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 1 independently using this configuration.
2	CONF_WARNING_LIGHT_02_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 2 independently using this configuration.
3	CONF_WARNING_LIGHT_03_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 3 independently using this configuration.
4	CONF_WARNING_LIGHT_04_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 4 independently using this configuration.
5	CONF_WARNING_LIGHT_05_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 5 independently using this configuration.
6	CONF_WARNING_LIGHT_06_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 6 independently using this configuration.
7	CONF_WARNING_LIGHT_07_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 7 independently using this configuration.
8	CONF_WARNING_LIGHT_08_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 8 independently using this configuration.
9	CONF_WARNING_LIGHT_09_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 9 independently using this configuration.
10	CONF_WARNING_LIGHT_10_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 10 independently using this configuration.
11	CONF_WARNING_LIGHT_11_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 11 independently using this configuration.
12	CONF_WARNING_LIGHT_12_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 12 independently using this configuration.
13	CONF_WARNING_LIGHT_13_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 13 independently using this configuration.
14	CONF_WARNING_LIGHT_14_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 14 independently using this configuration.
15	CONF_WARNING_LIGHT_15_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 15 independently using this configuration.
16	CONF_WARNING_LIGHT_16_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 16 independently using this configuration.
17	CONF_WARNING_LIGHT_17_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 17 independently using this configuration.
18	CONF_WARNING_LIGHT_18_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 18 independently using this configuration.
19	CONF_WARNING_LIGHT_19_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 19 independently using this configuration.
20	CONF_WARNING_LIGHT_20_POWER ON_STATE	WL_CONF_OFF WL_CONF_ON	WL_CONF_OFF	User can modify the power on state for Warning Light 20 independently using this configuration.

Warning Lights PWM DC Configuration

The AI430 SDK allows the user to modify the duty cycle for each warning light. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_WARNING_LIGHT_01_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 1 independently using this configuration.
2	CONF_WARNING_LIGHT_02_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 2 independently using this configuration.
3	CONF_WARNING_LIGHT_03_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 3 independently using this configuration.
4	CONF_WARNING_LIGHT_04_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 4 independently using this configuration.
5	CONF_WARNING_LIGHT_05_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 5 independently using this configuration.
6	CONF_WARNING_LIGHT_06_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 6 independently using this configuration.
7	CONF_WARNING_LIGHT_07_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 7 independently using this configuration.
8	CONF_WARNING_LIGHT_08_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 8 independently using this configuration.
9	CONF_WARNING_LIGHT_09_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 9 independently using this configuration.
10	CONF_WARNING_LIGHT_10_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 10 independently using this configuration.
11	CONF_WARNING_LIGHT_11_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 11 independently using this configuration.
12	CONF_WARNING_LIGHT_12_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 12 independently using this configuration.
13	CONF_WARNING_LIGHT_13_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 13 independently using this configuration.
14	CONF_WARNING_LIGHT_14_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 14 independently using this configuration.
15	CONF_WARNING_LIGHT_15_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 15 independently using this configuration.
16	CONF_WARNING_LIGHT_16_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 16 independently using this configuration.
17	CONF_WARNING_LIGHT_17_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 17 independently using this configuration.
18	CONF_WARNING_LIGHT_18_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 18 independently using this configuration.
19	CONF_WARNING_LIGHT_19_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 19 independently using this configuration.
20	CONF_WARNING_LIGHT_20_PWM_DC	MIN VALUE : 0 MAX VALUE : 100	100	User can modify the duty cycle for Warning Light 20 independently using this configuration.

The user is allowed to read or write to the PWM duty cycle period during runtime using the below DB variables.
The warning light should be ON for this configuration to be enabled in the hardware when it is set from the TouchGFX.

Field ID	Data Type	Permission	Size	Description	Comments
WARNING_LIGHT_01_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_02_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_03_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_04_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_05_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_06_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_07_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.

WARNING_LIGHT_08_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_09_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_10_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_11_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_12_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_13_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_14_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_15_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_16_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_17_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_18_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_19_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.
WARNING_LIGHT_20_PWM_DC	DBu8	READ/WRITE	1	0-100 range	This field is used to set and read back the Percentage of PWM duty cycle, 100% is maximum brightness.

The below sample code gets/sets the PWM Duty cycle in the DB:

```
if((KEY2_SHORT_PRESS == val) || (KEY2_LONG_PRESS == val))
{
  Get_DL(WARNING_LIGHT_01_PWM_DC, &pwmdc);
  if ((pwmdc < 100) && (pwmdc >= 0))
  {
     pwmdc ++;
     /* Set the Warning Light Intensity */
     Set_DL(WARNING_LIGHT_01_PWM_DC, &pwmdc);
    /* Turn on the warning light */
    uint8_t WL_state = WL_ON;
    Set_DL(WARNING_LIGHT_01_STATE, (uint8_t *)&WL_state);
   }
}
```

Warning Lights Blinking Configuration

The AI430 SDK allows the user to modify the blinking period for each warning light. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Warning Lights Sample Configuration. The value configured here is multiplied by 250ms to get the blinking period. So, if we have set a value of 2 here, between every blink there will be a (2*250ms = 500ms) time lag.

No	Variables	Options	Default State	Description
1	CONF_WARNING_LIGHT_01_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 1 independently using this configuration.
2	CONF_WARNING_LIGHT_02_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 2 independently using this configuration.
3	CONF_WARNING_LIGHT_03_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 3 independently using this configuration.
4	CONF_WARNING_LIGHT_04_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 4 independently using this configuration.
5	CONF_WARNING_LIGHT_05_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 5 independently using this configuration.
6	CONF_WARNING_LIGHT_06_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 6 independently using this configuration.
7	CONF_WARNING_LIGHT_07_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 7 independently using this configuration.
8	CONF_WARNING_LIGHT_08_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 8 independently using this configuration.

9	CONF_WARNING_LIGHT_09_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 9 independently using this configuration.
10	CONF_WARNING_LIGHT_10_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 10 independently using this configuration.
11	CONF_WARNING_LIGHT_11_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 11 independently using this configuration.
12	CONF_WARNING_LIGHT_12_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 12 independently using this configuration.
13	CONF_WARNING_LIGHT_13_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 13 independently using this configuration.
14	CONF_WARNING_LIGHT_14_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 14 independently using this configuration.
15	CONF_WARNING_LIGHT_15_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 15 independently using this configuration.
16	CONF_WARNING_LIGHT_16_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 16 independently using this configuration.
17	CONF_WARNING_LIGHT_17_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 17 independently using this configuration.
18	CONF_WARNING_LIGHT_18_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 18 independently using this configuration.
19	CONF_WARNING_LIGHT_19_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 19 independently using this configuration.
20	CONF_WARNING_LIGHT_20_BLINKING_MS	MIN VALUE: 1 MAX VALUE: 1000	2	User can modify the blinking period for Warning Light 20 independently using this configuration.

The user is allowed to read or write to the blinking period during runtime using the below DB variables. The warning light should be ON for this configuration to be enabled in the hardware when it is set from the TouchGFX.

Field ID	Data Type	Permission	Size	Description	Comments
WARNING_LIGHT_01_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_02_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_03_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_04_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_05_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_06_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_07_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_08_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_09_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_10_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_11_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_12_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_13_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_14_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_15_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_16_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_17_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_18_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.
WARNING_LIGHT_19_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.

WARNING_LIGHT_20 _BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.

The sample code for **getting** the blink period from DB is as below:

```
WL01BLINKView::WL01BLINKView()
{
  #if (SDK_SERVICE_WARNING_LIGHT == PS_ENABLE)
   blink = 0;
   /* Get the Warning Light blink period from the DB */
   Get_DL(WARNING_LIGHT_01_BLINKING, (uint8_t *)&blink);
 #endif
}
```

The sample code for **setting** the blink period from DB is as below:

```
if((KEY2_SHORT_PRESS == val) || (KEY2_LONG_PRESS == val))
{
  if ((blink < 65535) && (blink >= 0))
  {
    blink ++;
   }
   /* Set the Warning Light blink period */
   Set_DL(WARNING_LIGHT_01_BLINKING, (uint8_t *)&blink);
   /* Turn on the warning light */
   uint8_t WL_state = WL_ON;
  Set_DL(WARNING_LIGHT_01_STATE, (uint8_t *)&WL_state);
}
```

Warning Lights Sample Configuration

```
/*****************************************************************************************************************************************************************
*
                                                        * Warning Light Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * Warning light Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_WARNING_LIGHT PS_ENABLE
  #if (SDK_SERVICE_WARNING_LIGHT == PS_ENABLE)
  /*!
  * Warning Light Task Periodicity 100ms
  */
  #define PS_WL_TASK_TIMEOUT 100
  /*!
  * Warning Light Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  * ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_WL_TASK_PRIORITY osPriorityIdle
  /*!
  * Maximum Number of Warning Light required for this application
  *
  * MACOR Supported
  *
  * MAX_NUM_WARNING_LIGHT : This hardware support maximum of
  20 Warning lights
  * USER can choose between 0 to 20
  *
  #define CONF_WARNING_LIGHT_07_PWM_DC 100
  #define CONF_WARNING_LIGHT_07_BLINKING_MS 2
  #define CONF_WARNING_LIGHT_08_ENABLE PS_ENABLE
  #define CONF_WARNING_LIGHT_08_CURRENT
  WL_OUT_CURRENT_MAX_2
  #define CONF_WARNING_LIGHT_08_POWERON_STATE
  WL_CONF_OFF
  #define CONF_WARNING_LIGHT_08_PWM_DC 100
  #define CONF_WARNING_LIGHT_08_BLINKING_MS 2
  #define CONF_WARNING_LIGHT_09_ENABLE PS_ENABLE
  #define CONF_WARNING_LIGHT_09_CURRENT
  WL_OUT_CURRENT_MAX_4
  #define CONF_WARNING_LIGHT_09_POWERON_STATE
  WL_CONF_OFF
  #define CONF_WARNING_LIGHT_09_PWM_DC 100
  #define CONF_WARNING_LIGHT_09_BLINKING_MS 2
  #define CONF_WARNING_LIGHT_10_ENABLE PS_ENABLE
  #define CONF_WARNING_LIGHT_10_CURRENT
  WL_OUT_CURRENT_MAX_8
  #define CONF_WARNING_LIGHT_10_POWERON_STATE
  WL_CONF_OFF
  #define CONF_WARNING_LIGHT_10_PWM_DC 100
  #define CONF_WARNING_LIGHT_10_BLINKING_MS 2
  #define CONF_WARNING_LIGHT_11_ENABLE PS_ENABLE
  #define CONF_WARNING_LIGHT_11_CURRENT
  WL_OUT_CURRENT_MAX
  #define CONF_WARNING_LIGHT_11_POWERON_STATE
  WL_CONF_OFF
  #define CONF_WARNING_LIGHT_11_PWM_DC 100
  #define CONF_WARNING_LIGHT_11_BLINKING_MS 2
```

```
* WARNING_LIGHT_FREQ WL_kHz3_FREQ / WL_kHz22_FREQ
*
* CONF_WARNING_LIGHT_xx_ENABLE PS_ENABLE
* PS_DISABLE
*
* CONF_WARNING_LIGHT_xx_CURRENT WL_OUT_CURRENT_MAX
* WL_OUT_CURRENT_MAX_2
* WL_OUT_CURRENT_MAX_4
* WL_OUT_CURRENT_MAX_8
*
* CONF_WARNING_LIGHT_xx_POWERON_STATE WL_CONF_OFF
* WL_CONF_ON
*
* CONF_WARNING_LIGHT_xx_PWM_DC <0 - 100>
*
* CONF_WARNING_LIGHT_xx_BLINKING_MS <0-65535>
*
*
*/
/* Private defines -----------------------------------------------------------*/
/* USER CODE BEGIN Private defines */
#define MAX_NUM_WARNING_LIGHT 20
#define WARNING_LIGHT_FREQ WL_kHz3_FREQ
#define CONF_WARNING_LIGHT_01_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_01_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_01_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_01_PWM_DC 100
#define CONF_WARNING_LIGHT_01_BLINKING_MS 2
#define CONF_WARNING_LIGHT_02_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_02_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_02_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_02_PWM_DC 100
#define CONF_WARNING_LIGHT_02_BLINKING_MS 2
#define CONF_WARNING_LIGHT_03_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_03_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_03_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_03_PWM_DC 100
#define CONF_WARNING_LIGHT_03_BLINKING_MS 2
#define CONF_WARNING_LIGHT_04_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_04_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_04_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_04_PWM_DC 100
#define CONF_WARNING_LIGHT_04_BLINKING_MS 2
#define CONF_WARNING_LIGHT_05_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_05_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_05_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_05_PWM_DC 100
#define CONF_WARNING_LIGHT_05_BLINKING_MS 2
#define CONF_WARNING_LIGHT_06_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_06_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_06_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_06_PWM_DC 100
#define CONF_WARNING_LIGHT_06_BLINKING_MS 2
#define CONF_WARNING_LIGHT_07_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_07_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_07_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_12_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_12_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_12_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_12_PWM_DC 100
#define CONF_WARNING_LIGHT_12_BLINKING_MS 2
#define CONF_WARNING_LIGHT_13_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_13_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_13_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_13_PWM_DC 100
#define CONF_WARNING_LIGHT_13_BLINKING_MS 2
#define CONF_WARNING_LIGHT_14_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_14_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_14_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_14_PWM_DC 100
#define CONF_WARNING_LIGHT_14_BLINKING_MS 2
#define CONF_WARNING_LIGHT_15_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_15_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_15_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_15_PWM_DC 100
#define CONF_WARNING_LIGHT_15_BLINKING_MS 2
#define CONF_WARNING_LIGHT_16_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_16_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_16_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_16_PWM_DC 100
#define CONF_WARNING_LIGHT_16_BLINKING_MS 2
#define CONF_WARNING_LIGHT_17_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_17_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_17_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_17_PWM_DC 100
#define CONF_WARNING_LIGHT_17_BLINKING_MS 2
#define CONF_WARNING_LIGHT_18_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_18_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_18_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_18_PWM_DC 100
#define CONF_WARNING_LIGHT_18_BLINKING_MS 2
#define CONF_WARNING_LIGHT_19_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_19_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_19_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_19_PWM_DC 100
#define CONF_WARNING_LIGHT_19_BLINKING_MS 2
#define CONF_WARNING_LIGHT_20_ENABLE PS_ENABLE
#define CONF_WARNING_LIGHT_20_CURRENT
WL_OUT_CURRENT_MAX
#define CONF_WARNING_LIGHT_20_POWERON_STATE
WL_CONF_OFF
#define CONF_WARNING_LIGHT_20_PWM_DC 100
#define CONF_WARNING_LIGHT_20_BLINKING_MS 2
/* USER CODE END Private defines */
#endif //SDK_SERVICE_WARNING_LIGHT
```

LED Module

The User would be able to use the below functionalities of the digital output module via the DB variables and configuration file.

LED module Enable/Disable

The SDK provides the user the ability to enable/disable the LED functionality by modifying the default configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_LED	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the LED module in the SDK. **PS_DISABLE**: Disables the LED module in the SDK.

LED Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_LED_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

LED Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_LED_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

Maximum LED'S Configuration

The AI430 SDK supports a maximum of 2 LED's and the user has the ability to configure the MAX LED's supported by the device in the configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	MAX_LED_NUM	1 OR 2	2	User can operate maximum 2 LED

Configuring RED LED Enable/Disable

The SDK provides the user the ability to enable/disable the RED LED functionality by modifying the default configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_RED_LED_PS_STATE	PS_ENABLE/ PS_DISABLE	PS_ENABLE	User can enable/Disable the RED LED

Configuring RED LED State

The AI430 SDK supports the user to configure the default state of the RED LED and this can be done by modifying the below parameter in the configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_RED_LED_STATE	LED_CONF_ON/ LED_CONF_OFF	LED_CONF_ON	User can turn ON/OFF the RED LED

During runtime, the user **can read** and **modify** the RED LED state by reading and writing to the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
LED_RED_STATE	DbU8	READ/WRITE	1	ON/OFF	This field is used to enable/disable the LED_RED. ON = Light up the LED, OFF = Turn off the LED.

Below code snippet shows how the RED LED can be read and written into the DB.

```
#if (SDK_SERVICE_LED == PS_ENABLE)
  #if(CONF_RED_LED_PS_STATE == PS_ENABLE)
   /* Get the RED LED Status from the DB */
   Get_DL(LED_RED_STATE, &state);
   if (LED_ON == state)
   {
     /* LED is on */
    state = = LED_OFF;
    /* Set the RED LED Status from the DB */
    Set_DL(LED_RED_STATE, &state);
   }
  else
  {
    /* LED is OFF; */
   }
  #endif
#endif
```

Configuring RED LED Blinking

The AI430 SDK supports the user to configure the RED LED blinking time period in milli second and this can be done by modifying the below parameter in the configuration file. Please see section LED Sample Configuration. The value configured here is multiplied by 250ms to get the blinking period. So, if we have set a value of 2 here, between every blink there will be a (1*250ms = 250ms) time lag. If this value is set as 0 then the blinking is disabled.

No	Variables	Options	Default State	Description
1	CONF_RED_LED_BLINKING_MS	0-1000	1	User can change the Blink time period for the RED LED

During runtime, the user can read and modify the RED LED state and RED LED blinking time period by reading and writing to the below DB variables:

Field ID	Data Type	Permission	Size	Description	Comments
LED_RED_BLINKING	DBU16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.

Below code snippet shows how the RED LED can be read from the DB.

```
#if (SDK_SERVICE_LED == PS_ENABLE)
  #if(CONF_RED_LED_PS_STATE == PS_ENABLE)
   blink = 0;
   /* Get the RED LED blink period from the DB */
   Get_DL(LED_RED_BLINKING, (uint8_t *)&blink);
   state = = LED_ON;
   /* Set the RED LED Status to the DB */
   Set_DL(LED_RED_STATE, &state);
  #endif
#endif
```

Configuring AMB LED Enable/Disable

The SDK provides the user the ability to enable/disable the AMBER LED functionality by modifying the default configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_AMB_LED_PS_STATE	PS_ENABLE/ PS_DISABLE	PS_ENABLE	User can enable/Disable the AMB LED

Configuring AMB LED State

The AI430 SDK supports the user to configure the default state of the AMBER LED and this can be done by modifying the below parameter in the configuration file. Please see section LED Sample Configuration.

No	Variables	Options	Default State	Description
1	CONF_AMB_LED_STATE	LED_CONF_ON/ LED_CONF_OFF	LED_CONF_ON	User can turn ON/OFF the AMB LED

During runtime, the user can read and modify the AMB LED state by reading and writing to the below DB variables:

Field ID	Data Type	Permission	Size	Description	Comments
LED_AMB_STATE	DBu8	READ/WRITE	1	ON/OFF	This field is used to enable/disable the LED_AMB. ON = Light up the LED, OFF = Turn off the LED.

Below code snippet shows how the AMB LED can be read and written into the DB.

```
#if (SDK_SERVICE_LED == PS_ENABLE)
  #if(CONF_AMB_LED_PS_STATE == PS_ENABLE)
  /* Get the RED LED Status from the DB */
  Get_DL(LED_AMB_STATE, &state);
  if (LED_ON == state)
  {
  /* LED is on */
  state = = LED_OFF;
  /* Get the RED LED Status from the DB */
  Set_DL(LED_AMB_STATE, &state);
  }
  else
  {
  /* LED is OFF; */
  }
  #endif
#endif
```

Configuring AMB LED blinking

The AI430 SDK supports the user to configure the AMBER LED blinking time period in milli second and this can be done by modifying the below parameter in the configuration file. Please see section LED Sample Configuration. The value configured here is multiplied by 250ms to get the blinking period. So, if we have set a value of 2 here, between every blink there will be a (1*250ms = 250ms) time lag. If this value is set as 0 then the blinking is disabled.

No	Variables	Options	Default State	Description
1	CONF_AMB_LED_BLINKING_MS	0-1000	1	User can change the Blink time period for the AMB LED

During runtime, the user can read and modify the AMBER LED state and AMBER LED blinking by reading and writing to the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
LED_AMB_BLINKING	DBu16	READ/WRITE	2	0-65535 range	This field is used to set and read back the Blinking. Each unit is equivalent to 250ms.

Below code snippet shows how the AMB LED can be read and written into the DB.

```
#if (SDK_SERVICE_LED == PS_ENABLE)
   #if(CONF_AMB_LED_PS_STATE == PS_ENABLE)
    blink = 0;
    /* Get the AMB LED blink period from the DB */
    Get_DL(LED_AMB_BLINKING, (uint8_t *)&blink);
```

```
  blink = 2;
   /* Set the AMB LED blink period from the DB */
   Set_DL(LED_AMB_BLINKING, (uint8_t *)&blink);
  state = = LED_ON;
   /* Set the RED LED Status to the DB */
   Set_DL(LED_AMB_STATE, &state);
 #endif
#endif
```

LED Sample Configuration

```
 /*!
 * LED Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE)
 Macros
 */
 #define SDK_SERVICE_LED PS_ENABLE
 #if (SDK_SERVICE_LED == PS_ENABLE)
 /*!
 * LED Task Periodicity 100ms
 */
 #define PS_LED_TASK_TIMEOUT 100
 /*!
 * LED Task Priority
 * osPriorityNone = 0,
 * osPriorityIdle = 1,
 * osPriorityLow = 8,
 * osPriorityLow1 = 8+1,
 * ,, ,,
 * ,, ,,
 * osPriorityISR = 56,
 * osPriorityError = -1,
 * osPriorityReserved = 0x7FFFFFFF
 */
 #define PS_LED_TASK_PRIORITY osPriorityIdle
 /*!
 * Maximum Number of LED required for this application
 *
 * MACOR Supported
 *
 * MAX_LED_NUM : This hardware support maximum of 2 LED's
 *
 * CONF_xx_LED_PS_STATE PS_ENABLE
 * PS_DISABLE
 *
 * CONF_xx_LED_STATE LED_CONF_OFF
 *
 * LED_CONF_ON
 *
 * CONF_xx_LED_BLINKING_MS <0-65535>
 *
 */
 #define MAX_LED_NUM 2
 #define CONF_RED_LED_PS_STATE PS_ENABLE
 #define CONF_RED_LED_STATE LED_CONF_ON
 #define CONF_RED_LED_BLINKING_MS 1
 #define CONF_AMB_LED_PS_STATE PS_ENABLE
 #define CONF_AMB_LED_STATE LED_CONF_ON
 #define CONF_AMB_LED_BLINKING_MS 1
 #endif
```

Power Monitor Module

The User would be able to use the below functionalities of the power monitor module via the DB variables and configuration file.

Power Monitor Module Enable/Disable

The SDK provides the user the ability to enable/disable the power monitor functionality by modifying the default configuration file. Please see section Power Monitor sample configuration. If the Configurable inputs is disabled then the power monitor module will also be disabled in the configuration file.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_POWER_MONITOR	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the power monitor module in the SDK. **PS_DISABLE**: Disables the power monitor module in the SDK

Power Monitor Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Power Monitor sample configuration.

No	Variables	Options	Default State	Description
1	PS_POWER_MONITOR_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

Power Monitor Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section Power Monitor Sample Configuration

No	Variables	Options	Default State	Description
1	PS_POWER_MONITOR_ TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

Power Monitor Functionality Support

The AI430 SDK supports the next values which are monitored by the power monitor module. They are:

Battery_Level | Ignition_Status | Thermostat_Level

During runtime, the user can read the Battery level, Ignition status and Thermostat level by reading the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
BATTERY_LEVEL	DBu16	READ	2	Voltage in mili volts	This field is used to read the BATTERY_LEVEL in milli-volts.
IGNITION_STATUS	DBu8	READ	1	ON/OFF	This field is used to read the status of IGNITION_STATUS.
THERMOSTAT_LEVEL	float	READ	4	Temperature in Celsius	This field is used to read the THERMOSTAT_LEVEL in Celsius.

The below code snippet shows how to read the Battery Level, Ignition Status, and the Thermostat level.

```
void PWRMNTRView::trigger()
{
  #if (SDK_SERVICE_POWER_MONITOR == PS_ENABLE)
  uint16_t val = 0;
  float val_thermostat = 0;
  switch(key_position)
  {
    case 1:
    /*  Get ignition Status */
    Get_DL(IGNITION_STATUS, (uint8_t*)&val);
    break;
    case 2:
    /* Get Temperature level */
     Get_DL(THERMOSTAT_LEVEL, (uint8_t*)&val_thermostat);
    break;
    case 3:
     /* Get Battery level */
    Get_DL(BATTERY_LEVEL, (uint8_t*)&val);
    break;
  }
#endif
}
```

Power Monitor Sample configuration

```
/*****************************************************************************************************************
*
                                        * Power Monitor Module Configuration
*
*****************************************************************************************************************/
#define SDK_SERVICE_POWER_MONITOR PS_ENABLE
#if (SDK_SERVICE_POWER_MONITOR == PS_ENABLE)
#if ((SDK_SERVICE_POWER_MONITOR == PS_ENABLE) &&
(SDK_SERVICE_CFG_INPUT ==
PS_DISABLE))
#undef SDK_SERVICE_POWER_MONITOR
#define SDK_SERVICE_POWER_MONITOR PS_DISABLE
#endif
/*!
* Power Monitor Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_POWER_MONITOR_TASK_PRIORITY osPriorityIdle
/*!
* Power Monitor Task Periodicity 100ms
*/
#define PS_POWER_MONITOR_TASK_TIMEOUT 100
#endif //SDK_SERVICE_POWER_MONITOR
```

```
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_POWER_MONITOR_TASK_PRIORITY osPriorityIdle
/*!
* Power Monitor Task Periodicity 100ms
*/
#define PS_POWER_MONITOR_TASK_TIMEOUT 100
#endif //SDK_SERVICE_POWER_MONITOR
```

USB Module

The User would be able to use the below functionalities of the USB module via the DB variables and configuration file.

USB Module Enable/Disable

The SDK provides the user the ability to enable/disable the USB functionality by modifying the default configuration file. Please see section USB Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_USB	PS_ENABLE/ PS_DISABLE	PS_ENABLE	PS_ENABLE: Enables the USB module in the SDK. PS_DISABLE: Disables the USB module in the SDK.

USB Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section USB Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_USB_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

USB Module Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section USB Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_USB_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

USB ECU Identification Commands

The AI430 SDK user can read the below ECU Commands and modify the below ECU commands from the USB PC terminal. These commands are sent from the USB PC tool and are used to configure the MAX AI430. These commands are supported by the SDKs USB module and their default configuration can be done in the AI430_config.h. Please see section USB Sample Configuration.

No	Variables	Options	Default State	Description
1	VEHICLE_MANU_ECU_SW_NUM	As defined in the PC Tool	0x88	User can change the vehicle manufacturing ECU software number.
2	VEHICLE_MANU_ECU_SW_VER	As defined in the PC Tool	0x89	User can change the vehicle manufacturing ECU software version
3	ECU_MANU_DATE	As defined in the PC Tool	0x8B	User can change the vehicle ECU manufacturing date
4	ECU_SERIAL_NUM	As defined in the PC Tool	0x8C	User can change the ECU Serial number
5	PROGRAMMING_DATE	As defined in the PC Tool	0x99	User can change the programming date
6	PS_USB_USER_PACKET	As defined in the PC Tool	X050	User can get the USB packet

USB Module TX

The AI430 SDK allows the users to use the USB channel to **send or receive data**. To do so please use the below variables. To send data over the USB channel, the user will fill the TX buffer and then update the status as true. The platform will then send the data over USB and then clear the status when the data is sent.

Field ID	Data Type	Permission	Size	Description	Comments
USB_TX_STATUS	DBu8	READ/WRITE	1	TRUE /FALSE	This field is used read and write the USB_TX_STATUS. One needs to write TRUE to send data. The same is cleared when data is sent
USB_TX_BUFFER_ STATUS	DBu8	READ	1	FULL/NO_FULL	This field is used to read the Status of the TX buffer.
USB_TX_DATA	DBu8	WRITE	64	Data to be send via USB	This field contains the USB TX buffer data.

USB Module RX

To **read incoming** data over the USB channel, the user will need to monitor the RX BUFFER STATUS variable and see if there is any pending data available and if yes read the data and then update the RX STATUS.

Field ID	Data Type	Permission	Size	Description	Comments
USB_RX_STATUS	DBu8	READ/WRITE	1	TRUE/FALSE	This field is used read and write the USB_RX_STATUS. One needs to write TRUE to send data. The same is cleared when data is sent
USB_RX_BUFFER _STATUS	DBu8	READ	1	MSG_PENDIN G/EMPTY	This field is used to read the Status of the RX buffer.
USB_COMMUNICA TION_STATUS	DBu8	READ	1	COM_OK/CO M_ERROR	This field is used to indicate the USB Communication Status.
USB_RX_DATA	DBu8	READ	64	DATA received	This field contains the USB RX buffer data.

During runtime, the user can **read/ write** the below DB variables in the maxAI 430 debug terminal module for USB TX Status, USB RX Status and USB Communication status.

The sample code below suggests the process to read the RX Data received.

```
void USBView::trigger()
{
#if (SDK_SERVICE_USB == PS_ENABLE)
uint8_t status;
/* Get the RX status */
Get_DL(USB_RX_BUFFER_STATUS, &status);
if(RX_MSG_PENDING == status)
{
/* Read the Rx data from the DB */
Get_DL(USB_RX_DATA, (uint8_t *)&rxbuffer[0]);
memset(&trxbuffer[0], 0x00, sizeof(trxbuffer));
Unicode::strncpy(&trxbuffer[0], (const char*)&rxbuffer[0] ,
strlen((const char*)rxbuffer));
memset(RCVTEXTBuffer, 0x00, sizeof(RCVTEXTBuffer));
status = TRUE;
/* Clear the RX buffer */
Set_DL(USB_RX_STATUS, &status);
}
#endif
}
```

USB Sample Configuration

```
/*!
* USB Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE)
Macros
*/
#define SDK_SERVICE_USB PS_ENABLE
#if (SDK_SERVICE_USB == PS_ENABLE)
/*!
* USB Task Periodicity 100ms
*/
#define PS_USB_TASK_TIMEOUT 100
/*!
* USB Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_USB_TASK_PRIORITY osPriorityIdle
/*!
* EcuIdentification commands
*/
#define VEHICLE_MANU_ECU_SW_NUM 0x88
#define VEHICLE_MANU_ECU_SW_VER 0x89
#define ECU_MANU_DATE 0x8B
#define ECU_SERIAL_NUM 0x8C
#define PROGRAMMING_DATE 0x99
#define PS_USB_USER_PACKET 0x50
#endif //SDK_SERVICE_USB
```

Bluetooth Low Energy (BLE) Module

The AI430 SDK User would be able to use the below functionalities of the BLE module via the DB variables and configuration file.

BLE Module Enable/Disable

The SDK provides the user the ability to enable/disable the BLE functionality by modifying the default configuration file. Please see section BLE Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_BLE	PS_ENABLE/ PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the BLE module in the SDK. **PS_DISABLE**: Disables the BLE module in the SDK.

BLE Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section BLE Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_BLE_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

BLE Monitor Task Priority

The AI430 SDK supports the below task priorities, and the user can modify the task priority for the light sensor module in the configuration file. Please see section BLE Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_BLE_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

BLE Module Device Name Configuration

The AI430 SDK user can configure the device name of BLE. This default configuration can be done in the AI430_config.h

No	Variables	Options	Default State	Description
1	BLE_DEVICE_NAME	Any name as per the user requirement	"maxAI12345678"	This field is used to set and read the BLE device name. The maximum length is 20 characters.

The user would be able to use the read the BLE module name during run time via the DB variables shown below.

Field ID	Data Type	Permission	Size	Description	Comments
BLE_DEVICE_NAME	DBu8	READ/WRITE	20	Devicename	This field is used to set and read the BLE device name. The maximum length is 20 characters.

The below code snippet shows how to **read** the BLE name:

```
#if (SDK_SERVICE_BLE == PS_ENABLE)
  uint8_t name;
  /* Get the BLE device name */
  Get_DL(BLE_DEVICE_NAME, &name);
}
```

BLE Module RX/TX

The AI430 SDK allows the users to use the BLE channel to send or receive data. To do so please use the below variables. To read incoming data over the BLE channel, the user will need to monitor the RX BUFFER STATUS variable and see if there is any pending data available and if yes read the data and then update the RX STATUS. To send data over the BLE channel, the user will fill the TX buffer and then update the status as true. The platform will then send the data over BLE and then clear the status when the data is sent.

Field ID	Data Type	Permission	Size	Description	Comments
BLE_TX_STATUS	DBu8	READ	1	TRUE/FALSE	This field is used read and write the BLE_TX_STATUS. One needs to write TRUE to send data. The same is cleared when data is sent
BLE_TX_BUFFER_STATUS	DBu8	READ	1	FULL/NO_FULL	This field is used to read the Status of the TX buffer
BLE_TX_DATA	DBu8	WRITE	64	Data to be send to BLE	This field contains the BLE TX buffer data.
BLE_RX_STATUS	DBu8	READ/WRITE	1	TRUE/FALSE	This field is used read and write the BLE_RX_STATUS. One needs to read TRUE to receive data. The same is cleared when data is sent.
BLE_RX_BUFFER_STATUS	DBu8	READ	1	MSG PENDING/EMPTY	This field is used to read the Status of the RX buffer.
BLE_RX_DATA	DBu8	READ	64	DATA Received	This field contains the BLE RX buffer data.
BLE_RX_DATA_SIZE	DBu8	READ	1	(Only applicable for USER_DATA_MODE)	This field is used to read the size of the BLE RX Data.

The sample code below suggests the process to **read** the RX Data received

```
{
#if (SDK_SERVICE_BLE == PS_ENABLE)
uint8_t status ;
/* Get the RX status */
Get_DL(BLE_RX_BUFFER_STATUS, &status);
if(BLE_RX_MSG_PENDING == status)
/* Clear the memory */
memset(&rxbuffer1[0], 0x00, sizeof(rxbuffer1));
/* Read the Rx data from the DB */
Get_DL(BLE_RX_DATA, (uint8_t *)&rxbuffer1[0]);
memset(&trxbuffer3[0], 0x00, sizeof(trxbuffer3));
status = TRUE;
/* Clear the RX buffer */
Set_DL(BLE_RX_STATUS, &status);
}
#endif
}
```

The sample code below suggests the process to **send** the Data over Bluetooth

```
/*set the BLE tx data */
Set_DL(BLE_TX_DATA , (uint8_t *)&buffer1[0]);
status = TRUE;
/* Clear the RX buffer */
Set_DL(BLE_TX_STATUS, &status);
```

BLE Sample Configuration

```
/*!
* BLE Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_BLE PS_ENABLE
#if (SDK_SERVICE_BLE == PS_ENABLE)
/*!
* BLE Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,

* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_BLE_TASK_PRIORITY osPriorityIdle
/*!
* BLE Task Periodicity 100ms
*/
#define PS_BLE_TASK_TIMEOUT 100
/*!
* BLE Device Name
*/
#define BLE_DEVICE_NAME "maxAI12345678"
#endif //SDK_SERVICE_BLE
```

Timer Module

The AI430 SDK User would be able to use the below functionalities of the Timer module via the DB variables and configuration file.

Timer Module Enable/Disable

The SDK provides the user the ability to enable/disable the Timer functionality by modifying the default configuration file. Please see section Timer Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_SWTIMER	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the timer module in the SDK. **PS_DISABLE**: Disables the timer module in the SDK.

Timer Module Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Timer Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_SWT_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

Timer Module Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the timer module in the configuration file. Please see section Timer Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_SWT_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

Timer Start or Stop

The AI430 SDK supports six software timers. The user can start or stop the timers during run time and get the current status of the timer. To do so he can read or write the timer state using the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
TIMER_STATUS_01	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).
TIMER_STATUS_02	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).
TIMER_STATUS_03	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).
TIMER_STATUS_04	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).
TIMER_STATUS_05	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).
TIMER_STATUS_06	DBu8	READ/WRITE	1	START/STOP	field used to set and read Timer state (START/STOP).

Next code snippet shows how the timer can be set:

```
Get_DLTIMER_STATUS_01, &state);
if(state == 1)
{
state = 2;
Set_DL(TIMER_STATUS_01, &state);
}
else if(state == 2)
{
state = 1;
set_DL(TIMER_STATUS_01, &state);
}
```

Once the timer expires the SDK updates the timer callback parameter in the DB with the status as CALLBACK_OCCURED and the user can monitor the same to know if the timer has expired. He can use the below DB variables to do the same.

Field ID	Data Type	Permission	Size	Description	Comments
TIMER_CALLBACK_01	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.
TIMER_CALLBACK_02	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.
TIMER_CALLBACK_03	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.
TIMER_CALLBACK_04	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.
TIMER_CALLBACK_05	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.
TIMER_CALLBACK_06	DBu8	READ/WRITE	1	CALLBACK_CLEAR/ CALLBACK_OCCURED	This field is used to set and clear the Timer state.

The below code snippet shows you how you can read the S/W timer status:

```
#if (SDK_SERVICE_SWTIMER == PS_ENABLE)
uint8_t timer1_state = 0;
uint8_t rtc1_val = 0;
uint8_t timeout_val = 0;
Get_DL(TIMER_CALLBACK_01, &timer1_state);
if(CALLBACK_OCCURED == timer1_state)
{
  /* Timer expired */
}
#endif
```

Timer Mode Configuration

The S/W timers can be configured as single shot and periodic. During runtime, the user **can read or write** timer mode variable in the DB to update/get the configuration of the S/W timers.

Field ID	Data Type	Permission	Size	Description	Comments
TIMER_MODE_01	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.
TIMER_MODE_02	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.
TIMER_MODE_03	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.
TIMER_MODE_04	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.
TIMER_MODE_05	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.
TIMER_MODE_06	DBu8	READ/WRITE	1	ONESHOT/ PERIODIC	This field is used to set and read Timer Mode.

Next code snippet shows how the timer state can be set and read:

```
GET_DB(TIMER_MODE_01, (uint8_t *)&shot);
if(shot == 0)
{
shot = 1;
}
else if(shot == 1)
{
shot = 0;
}
/* Set the Timer Mode_1 */
SET_DB(TIMER_MODE_01, (uint8_t *)&shot);
```

Timer Timeout Configuration

During runtime, the user can set or get the timeout period for the S/W timers using the below DB variables. Please note that the timer timeout can be increased in steps of 50ms. And the max timeout value should be less than 65535.

Field ID	Data Type	Permission	Size	Description	Comments
TIMER_TIMEOUT_01	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds
TIMER_TIMEOUT_02	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds
TIMER_TIMEOUT_03	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds
TIMER_TIMEOUT_04	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds
TIMER_TIMEOUT_05	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds
TIMER_TIMEOUT_06	DBU16	READ/WRITE	2	Time in milli seconds	This field is used to get/set the timeout in milliseconds

Next code snippet shows how the timer timeout can be set.

```
if ((timeout > 0) && (timeout <= 1300))
{
timeout --;
sw_timeout = (timeout * 50);
}
else
{
}
/* Set the Timer Timeout_1 */
SET_DB(TIMER_TIMEOUT_01, (uint8_t *)&sw_timeout);
```

Please note that the timer timeout can be increased in steps of 50ms. And the max timeout value should be less than 65535 hence the max counter in the loop is restricted to 1300.

Timer Sample Configuration

```
/*!
* SWTIMER Platform service Enable(PS_ENABLE) /
Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_SWTIMER PS_ENABLE
#if (SDK_SERVICE_SWTIMER == PS_ENABLE)
/*!
* SWTIMER Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_SWT_TASK_PRIORITY osPriorityIdle
/*!
* SWTimer Task Periodicity 100ms
*/
#define PS_SWT_TASK_TIMEOUT 100
#endif //SDK_SERVICE_SWTIMER
```

RTC Module

The AI430 SDK User would be able to use the below functionalities of the RTC module via the DB variables and configuration file.

RTC Module Enable/Disable

The SDK provides the user the ability to enable/disable the RTC functionality by modifying the default file. Please see section RTC Alarm Sample Configuration

No	Variables	Options	Default State	Description
1	SDK_SERVICE_RTC	PS_ENABLE/PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the RTC module in the SDK. **PS_DISABLE**: Disables the RTC module in the SDK.

RTC Timeout Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section RTC Alarm Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_RTC_TASK_TIME OUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

RTC Task Priority

The AI430 SDK supports the below task priority and the user can modify the task priority for the RTC module in the configuration file. Please see section RTC Alarm Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_RTC_TASK_PRIO RITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

The user would be able to use the below functionalities of the RTC module via the DB variables and configuration file.

RTC Date and Time Configuration

The user can get or set the real time clock using the following DB variables. To set the RTC , the user must set individually each of the RTC parameters and then call the SET_RTC DB variable to set the RTC TIME.

Field ID	Data Type	Permission	Size	Description	Comments
RTC_SECOND	DBu8	READ/WRITE	1	0-59	Valid values to set the real time second are from 0 to 59
RTC_MINUTE	DBu8	READ/WRITE	1	0-59	Valid values to set the real time minute are from 0 to 59
RTC_HOUR	DBu8	READ/WRITE	1	0-24	Valid values to set the real time hour are from 0 to 24
RTC_DATE	DBu8	READ/WRITE	1	1-31	Valid values to set the real time day are from 1 to 31
RTC_WEEK_DAY	DBu8	READ/WRITE	1	1-7	Valid values to set the real time from 1 = Monday to 7= Sunday
RTC_MONTH	DBu8	READ/WRITE	1	1-12	Valid values to set the real time month are from 1 to 12
RTC_YEAR	DBu8	READ/WRITE	1	00-99	Valid values to set the real time year are from 0 to 99
SET_RTC	DBu8	READ/WRITE	1	SET_RTC	(Users need to set the above RTC parameters and then enable the SET RTC to set the time)

The below snapshot is a sample for updating the RTC Time

```
Set_DL(GET_RTC_SECOND, &Seconds);
Set_DL(GET_RTC_MINUTE, &Minutes);
Set_DL(GET_RTC_HOUR, &Hours);
Set_DL(GET_RTC_DATE, &Date);
Set_DL(GET_RTC_WEEK_DAY, &WeekDay);
Set_DL(GET_RTC_MONTH, &Month);
Set_DL(GET_RTC_YEAR, &Year);
res = 1;
Set_DL(SET_RTC, &res);
```

The sample code below is an example of reading the RTC values.

```
void RTCNXTView::trigger()
{
#if (SDK_SERVICE_RTC == PS_ENABLE)
uint8_t Seconds;
uint8_t Minutes;
uint8_t Hours;
tickCounter++;
if( 10 <= tickCounter)
{
tickCounter = 0;
/* Get the RTC DB */
Get_DL(GET_RTC_SECOND, &Seconds);
Get_DL(GET_RTC_MINUTE, &Minutes);
Get_DL(GET_RTC_HOUR, &Hours);
//screenViewBase::setupScreen();
digitalHours = Hours;
digitalMinutes = Minutes;
digitalSeconds = Seconds;
digitalClock1.setTime24Hour(digitalHours, digitalMinutes,
digitalSeconds);
digitalClock1.invalidate();
}
#endif
}
```

RTC Time Format

The SDK supports the 12- and 24-hour time format. The user can read/update the RTC Time format during run time using the below DB variables.
Below are their definitions:

```
#define FORMAT_12_HOUR 1
#define FORMAT_24_HOUR 0
```

Field ID	Data Type	Permission	Size	Description	Comments
RTC_TIME_FORMAT	DBu8	READ/WRITE	1	AM/PM	RTC Time Format (AM/PM)

```
/* Get the RTC time format */
Get_DL(RTC_TIME_FORMAT, &format);
format = FORMAT_24_HOUR;
/* Set the RTC time format */
Set_DL(RTC_TIME_FORMAT, &format);
```

RTC Alarm Date and Time

The SDK platform supports 2 alarms and they can be configured by the user during run time. To set an alarm the user will need to configure the below parameters of the alarm and then enable the *SET_ALARM*.

Field ID	Data Type	Permission	Size	Description	Comments
RTC_ALARM_A_SECOND	DBu8	READ/WRITE	1	0-59	Valid values to set the alarm are from 0 to 59. From 60 to 255 the values are invalid.
RTC_ALARM_A_MINUTE	DBu8	READ/WRITE	1	0-59	Valid values to set the alarm are from 0 to 59. From 60 to 255 the values are invalid.
RTC_ALARM_A_HOUR	DBu8	READ/WRITE	1	0-24	Valid values to set the alarm are from 0 to 24. From 25 to 255 the values are invalid.
RTC_ALARM_A_DAY	DBu8	READ/WRITE	1	1-31	Valid values to set the alarm are from 1 to 31. From 32 to 255 the values are invalid.
RTC_ALARM_A_WEEK_DAY	DBu8	READ/WRITE	1	1-7	Valid values to set the alarm are from 1 to 7. From 8 to 255 the values are invalid.
RTC_ALARM_A_MONTH	DBu8	READ/WRITE	1	1-12	Valid values to set the alarm are from 1 to 12. From 13 to 255 the values are invalid.
RTC_ALARM_A_YEAR	DBu8	READ/WRITE	1	00-99	Valid values to set the alarm are from 0 to 99. From 100 to 255 the values are invalid.
SET_ ALARM_A	DBu8	READ/WRITE	1	ON/OFF	(Users need to set the above ALARM parameters and then enable the SET_ALARM1 to set the alarm time)

Field ID	Data Type	Permission	Size	Description	Comments
RTC_ALARM_B_SECOND	DBu8	READ/WRITE	1	0-59	Valid values to set the alarm are from 0 to 59. From 60 to 255 the values are invalid.
RTC_ALARM_B_MINUTE	DBu8	READ/WRITE	1	0-59	Valid values to set the alarm are from 0 to 59. From 60 to 255 the values are invalid.
RTC_ALARM_B_HOUR	DBu8	READ/WRITE	1	0-24	Valid values to set the alarm are from 0 to 24. From 25 to 255 the values are invalid.

RTC_ALARM_B_DAY	DBu8	READ/WRITE	1	1-31	Valid values to set the alarm are from 1 to 31. From 32 to 255 the values are invalid.
RTC_ALARM_B_WEEK_DAY	DBu8	READ/WRITE	1	1-7	Valid values to set the alarm are from 1 to 7. From 8 to 255 the values are invalid.
RTC_ALARM_B_MONTH	DBu8	READ/WRITE	1	1-12	Valid values to set the alarm are from 1 to 12. From 13 to 255 the values are invalid.
RTC_ALARM_B_YEAR	DBu8	READ/WRITE	1	00-99	Valid values to set the alarm are from 0 to 99. From 100 to 255 the values are invalid.
SET_ ALARM_B	DBu8	READ/WRITE	1	ON/OFF	(Users need to set the above ALARM parameters and then enable the SET_ALARM1 to set the alarm time)

The below code snippet show how we can set the alarm.

```
Set_DL(RTC_ALARM_A_HOUR, &ahours);
Set_DL(RTC_ALARM_A_MINUTE, &aminutes);
Set_DL(RTC_ALARM_A_SECOND, &aseconds);
Set_DL(RTC_ALARM_A_WEEK_DAY, (uint8_t *)&awkdays);
Set_DL(SET_ALARM_A, &ares);
```

Once the alarm is set the user can read the *ALARM_STATUS* DB variable to know the status of the alarm as seen in the below table. Once the alarm occurs the status variable will be updated to OCCURRED. After the user reads the status, he will need to reset the same in the DB.

Field ID	Data Type	Permission	Size	Description	Comments
ALARM_A_STATUS	DBu8	READ/WRITE	1	(1:OCCURRED/0:NOT OCCURRED)	Alarm1status (OCCURRED/ NOTOCCURRED)
ALARM_B_STATUS	DBu8	READ/WRITE	1	(1:OCCURRED/0:NOT OCCURRED)	Alarm1status (OCCURRED/ NOTOCCURRED)

The below code snippet shows the alarm status:

```
/* Read the Alarm A Status from the DB */
res = Get_DL(ALARM_A_STATUS, &alarm_status);
if (ALARM_OCCURED == alarm_status)
{
    alarm_status = 0;
    /* Set the ALARM A status*/
    res = Set_DL(ALARM_A_STATUS, &alarm_status);
}
```

RTC Alarm Time Format

The SDK supports the 12- and 24-hour time format. The user can read/update the RTC Alarm format during run time using the below DB variables. Below are their definitions:

```
#define FORMAT_12_HOUR 1
#define FORMAT_24_HOUR 0
```

Field ID	Data Type	Permission	Size	Description	Comments
RTC_ALARM_A_TIME_FORMAT	DBu8	READ/WRITE	1	AM/PM	RTC ALARM A Time format (AM/PM)
RTC_ALARM_B_TIME_FORMAT	DBu8	READ/WRITE	1	AM/PM	RTC ALARM A Time format (AM/PM)

RTC Alarm Sample Configuration

```
/*!
* RTC Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_RTC PS_ENABLE
#if (SDK_SERVICE_RTC == PS_ENABLE)
/*!
* RTC Task Periodicity 100ms
*/
#define PS_RTC_TASK_TIMEOUT 100
/*!
* RTC Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_RTC_TASK_PRIORITY osPriorityIdle
#endif //SDK_SERVICE_RTC
```

Camera Module

The AI430 SDK User would be able to use the below functionalities of the Camera module via the DB variables and configuration file.

Camera Module Enable/Disable

The SDK provides the user the ability to enable/disable the Camera functionality by modifying the default file. Please see section Camera Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_CAMERA	PS_ENABLE/PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the camera module in the SDK. **PS_DISABLE**: Disables the camera module in the SDK.

Camera Timeout Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Camera Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_CAMERA_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100ms	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

Camera Task Priority

The AI430 SDK supports the below task priority and the user can modify the task priority for the Camera module in the configuration file. Please see section Camera Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_CAMERA_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

Camera Mode Configuration

The AI430 SDK supports three camera modes namely:

Variable	Description
FULL_SCREEN_ON	In this camera mode, the user can view full screen on the Touch GFX
RESIZE_TO_FULL_SCREEN_ON	In this camera mode, the selected option of the image will be resized and displayed on the full screen
DISPLAY_AS_IT_IS_ON	In the camera mode, the image will be displayed when the camera is streamed on.

The user can select one of the above modes using the below configuration parameter. Please see section Camera Sample Configuration

No	Variables	Options	Default State	Description
1	PS_CAMERA_MODE	FULL_SCREEN_ON/ RESIZE_TO_FULL_SCREEN_ON/ DISPLAY_AS_IT_IS_ON	DISPLAY_AS_IT_IS_ON	User can select any one of the camera modes based on the application requirement.

The maxAI 430 SDK user can get/set the below camera video modes during runtime.

Field ID	Data Type	Permission	Size	Description	Comments
CAMERA_VIDEO_MODE	DBu8	READ/WRITE	1	1) Full Screen On 2) Resize to full screen on 3) Display as it is on	This field is used to select the mode of the camera

The next code snippet shows how to set the camera video mode:

```
/* Call the DB Variable to set the Display as it is Mode */
mode = DISPLAY_AS_IT_IS_ON;
Set_DL(CAMERA_VIDEO_MODE, &mode);
mode = CAMERA_STREAM_ON;
Set_DL(CAMERA_VIDEO_STREAM, &mode);
```

Camera Configuration Parameters

The AI430 SDK allows the user to configure the below camera parameters during run time, This default configuration can be done in the AI430_config.h. Please see section Camera Sample Configuration.

Camera X0 Display Origin | Camera Video X0 Origin | Camera Video X0 Width

Camera Y0 Display Origin | Camera Video Y0 Origin | Camera Video Y0 Height

The AI430 SDK User can get/set the below camera configuration parameters during runtime:

No	Variables	Options	Default State	Description
1	PS_CAMERA_X0 _DISP_ORIGIN	0 o 480	0	The user can configure the Camera X0 display origin
2	PS_CAMERA_Y0 _DISP_ORIGIN	0 to 272	0	The user can configure the Camera Y0 display origin
3	PS_CAMERA_X0 _ORIGIN	0 to 480	0	The user can configure the Camera X0 origin
4	PS_CAMERA_Y0 _ORIGIN	0 to 272	0	The user can configure the Camera Y0 origin
5	PS_CAMERA_X0 _WIDTH	0 to 480	480	The user can configure the width of the Camera capture
6	PS_CAMERA_Y0 _HEIGHT	0 to 272	272	The user can configure the height of the Camera capture

The AI430 SDK User can get/set the below camera configuration parameters during runtime:

Field ID	Data Type	Permission	Size	Description	Comments
CAMERA_VIDEO_X0 _WIDTH	DBu8	READ/WRITE	1	0 to 480	This field is used to set and read the Camera capture width.
CAMERA_VIDEO_Y0 _HEIGHT	DBu8	READ/WRITE	1	0 to 272	This field is used to set and read the Camera capture Height.
CAMERA_VIDEO_X0 _ORIGIN	DBu8	READ/WRITE	1	0 to 480	This field is used to set and read the Camera origin X0.
CAMERA_VIDEO_Y0 _ORIGIN	DBu8	READ/WRITE	1	0 to 272	This field is used to set and read the Camera origin Y0.
CAMERA_VIDEO_X0 _DISP_ORIGIN	DBu8	READ/WRITE	1	0 to 480	This field is used to set and read the Camera display origin X0.
CAMERA_VIDEO_Y0 _DISP_ORIGIN	DBu8	READ/WRITE	1	0 to 272	This field is used to set and read the Camera display origin Y0.

The below code snippet shows how the user can set/get the camera configuration parameters:

```
Set_DL(CAMERA_X0_WIDTH, (uint8_t *) &w0);
Set_DL(CAMERA_Y0_HEIGHT, (uint8_t *) &h0);
Set_DL(CAMERA_X0_ORIGIN, (uint8_t *) &x0);
Set_DL(CAMERA_Y0_ORIGIN, (uint8_t *) &y0);
Set_DL(CAMERA_VIDEO_FLIP_VERTICAL, &vf);
Set_DL(CAMERA_VIDEO_FLIP_HORIZONTAL, &hf);
/* Call the DB Variable to set the Full screen Mode */
Set_DL(CAMERA_VIDEO_MODE, &mode);
mode = CAMERA_STREAM_ON;
Set_DL(CAMERA_VIDEO_STREAM, &mode);
```

Camera Module Optimization Configuration

The AI430 SDK is designed currently in a way that the following modules are disabled when the camera streaming is on to improve the performance of the camera.

Keypad	CAN	Warning Light	LED	LS	USB	DIO	BLE

But the user has the option to enable the above modules when the camera streaming is on. This default configuration can be done in the AI430_config.h. Please see section Camera Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_CAM_WARNING_LIGHT_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the Warning Light module while camera is streaming. PS_DISABLE: Disables the Warning Light module while camera is streaming.
2	SDK_CAM_LED_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the LED module while camera is streaming. PS_DISABLE: Disables the LED module while camera is streaming.
3	SDK_CAM_LS_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the Light Sensor module while camera is streaming. PS_DISABLE: Disables the Light Sensor module while camera is streaming.
4	SDK_CAM_USB_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the USB module while camera is streaming. PS_DISABLE: Disables the USB module while camera is streaming
5	SDK_CAM_DIO_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the DIO module while camera is streaming. PS_DISABLE: Disables the DIO module while camera is streaming
6	SDK_CAM_BLE_STOP	PS_ENABLE/ PS_DISABLE	PS_DISABLE	PS_ENABLE: Enables the BLE module while camera is streaming. PS_DISABLE: Disables the BLE module while camera is streaming

Camera Streaming Enable/Disable

The AI430 SDK User can use the below DB variables for switching ON/OFF the camera streaming functionality during runtime.

Field ID	Data Type	Permission	Size	Description	Comments
CAMERA_VIDEO_STREAM	DBu8	READ/WRITE	1	ON/OFF	This field is used to Enable and Disable the camera streaming (ON/OFF).

The next code snippet shows how the camera streaming functionality is controlled during the runtime:

```
if (FULL_SCREEN_ON == mode)
{
  /* Call the DB Variable to set the Full screen Mode */
  Set_DL(CAMERA_VIDEO_MODE, &mode);
  mode = CAMERA_STREAM_ON;
 Set_DL(CAMERA_VIDEO_STREAM, &mode);
}
```

Camera Flip Option

The AI430 SDK User can use the below DB variables for flipping the image vertically and horizontally. The video should be streamed off and streamed on for enable the flipping functionality during runtime.

Field ID	Data Type	Permission	Size	Description	Comments
CAMERA_VIDEO_FLIP_VERTICAL	DBu8	READ/WRITE	1	TRUE/FALSE	This field is used to set and read the State of the vertical flip
CAMERA_VIDEO_FLIP_HORIZONTAL	DBu8	READ/WRITE	1	TRUE/FALSE	This field is used to set and read the State of the horizontal flip

The sample code below gives an example of flipping the video either horizontally or vertically.

```
#if (SDK_SERVICE_CAMERA == PS_ENABLE)
mode = CAMERA_STREAM_OFF;
/* Call the DB Variable to set the Full screen Mode */
Set_DL(CAMERA_VIDEO_STREAM, &mode);
mode = DISPLAY_AS_IT_IS_ON;
vf = 1;
hf = 0;
#if (SDK_SERVICE_LCD == PS_ENABLE)
Set_DL(DISPLAY_X0_ORIGIN, (uint8_t *) &dx0);
Set_DL(DISPLAY_Y0_ORIGIN, (uint8_t *) &dy0);
#endif
Set_DL(CAMERA_VIDEO_FLIP_VERTICAL, &vf);
Set_DL(CAMERA_VIDEO_FLIP_HORIZONTAL, &hf);
/* Call the DB Variable to set the Full screen Mode */
Set_DL(CAMERA_VIDEO_STREAM, &mode);
#endif
```

Camera Auto ON/OFF Functionality

The user can make the camera stream ON/OFF by enabling either keypad, CAN or STG/STB as the input source. If the user enables two of them, the camera will malfunction. For example, if he enables, SDK_CAMERA_STOP_KEY1_KEYPAD then the long press of this key can start/stop the camera from any screen. Similarly, he can configure a certain input from CAN or the Configurable inputs as the source to launch or exit the camera from the application. If the STG/STB is enabled, the user can make the camera stream on but the user cannot call the stream on function from the application. This default configuration can be done in the AI430_config.h. Please see section Camera Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_CAMERA_STOP_KEY1_KEYPAD	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the keypad 1 as the input source to disable/enable camera. **PS_DISABLE**: Disables the keypad 1 as the input source to disable/enable camera.
2	SDK_CAMERA_STOP_KEY2_KEYPAD	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the keypad 2 as the input source to disable/enable camera. **PS_DISABLE**: Disables the keypad 2 as the input source to disable/enable camera.
3	SDK_CAMERA_STOP_KEY3_KEYPAD	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the keypad 3 as the input source to disable/enable camera. **PS_DISABLE**: Disables the keypad 3 as the input source to disable/enable camera.
4	SDK_CAMERA_STOP_KEY4_KEYPAD	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the keypad 4 as the input source to disable/enable camera. **PS_DISABLE**: Disables the keypad 4 as the input source to disable/enable camera.
5	SDK_CAMERA_STOP_CI_STB	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the STB as the input source to disable/enable camera. **PS_DISABLE**: Disables the STB as the input source to disable/enable camera.
6	SDK_CAMERA_STOP_CI_STG	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the STG as the input source to disable/enable camera. **PS_DISABLE**: Disables the STG as the input source to disable/enable camera.
7	SDK_CAMERA_STOP_CAN	PS_ENABLE/PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the CAN as the input source to disable/enable camera. **PS_DISABLE**: Disables the CAN as the input source to disable/enable camera.

Camera Sample Configuration

```
/*!
* Camera Platform service Enable(PS_ENABLE) /
Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_CAMERA PS_ENABLE
#if (SDK_SERVICE_CAMERA == PS_ENABLE)
/*!
* Camera Task Periodicity 100ms
*/
#define PS_CAMERA_TASK_TIMEOUT 100
/*!
* Camera Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_CAMERA_TASK_PRIORITY osPriorityIdle
/*!
* PS_CAMERA_MODE
* =============
* FULL_SCREEN_ON1
* RESIZE_TO_FULL_SCREEN_ON
/* PS_ENABLE : During the Camera streaming, if the key
* is enabled and pressed, the camera streaming will be stopped */
#define SDK_CAMERA_STOP_KEY1_KEYPAD PS_DISABLE
#define SDK_CAMERA_STOP_KEY2_KEYPAD PS_DISABLE
#define SDK_CAMERA_STOP_KEY3_KEYPAD PS_DISABLE
#define SDK_CAMERA_STOP_KEY4_KEYPAD PS_DISABLE
#if ((SDK_CAMERA_STOP_KEY1_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY2_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY3_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY4_KEYPAD == PS_DISABLE))
/*
* Configuration Input AI1 ===> CONF_AI1
* Configuration Input AI2 ===> CONF_AI2
* Configuration Input AI3 ===> CONF_AI3
* Configuration Input AI4 ===> CONF_AI4
* Configuration Input AI5 ===> CONF_AI5
* Configuration Input AI6 ===> CONF_AI6
* Configuration Input None ===> CONF_NONE
*/
#define SDK_CAMERA_STOP_CI_STB CONF_NONE
#define SDK_CAMERA_STOP_CI_STG CONF_NONE
#endif
#if ((SDK_CAMERA_STOP_KEY1_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY2_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY3_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_KEY4_KEYPAD == PS_DISABLE) && \
(SDK_CAMERA_STOP_CI_STB == CONF_NONE) && \
(SDK_CAMERA_STOP_CI_STG == CONF_NONE))
```

```
* DISPLAY_AS_IT_IS_ON
*/
#define PS_CAMERA_MODE FULL_SCREEN_ON
/*!
*
*/
#define PS_CAMERA_X0_DISP_ORIGIN 0
#define PS_CAMERA_Y0_DISP_ORIGIN 0
#define PS_CAMERA_X0_ORIGIN 0
#define PS_CAMERA_Y0_ORIGIN 0
#define PS_CAMERA_X0_WIDTH 480
#define PS_CAMERA_Y0_HEIGHT 272
```

```
/*
* Camera ON/OFF based on the CAN packet
* PS_ENABLE : Enable the CAMERA Stream ON/OFF through CAN Message
*
*/
#define SDK_CAMERA_STOP_CAN PS_ENABLE
#endif
/* SDL Module to be stopped */
/* PS_ENABLE : Stopped the SDK Service during the Camera Streaming
*/
#define SDK_CAM_WARNING_LIGHT_STOP PS_DISABLE
#define SDK_CAM_LED_STOP PS_DISABLE
#define SDK_CAM_LS_STOP PS_DISABLE
#define SDK_CAM_USB_STOP PS_DISABLE
#define SDK_CAM_DIO_STOP PS_DISABLE
#define SDK_CAM_BLE_STOP PS_DISABLE
#endif //SDK_SERVICE_CAMERA
```

EEPROM Module

The AI430 SDK User would be able to use the below functionalities of the EEPROM module via the DB variables and configuration file.

EEPROM Module Enable/Disable

The SDK provides the user the ability to enable/disable the EEPROM functionality by modifying the default file. Please see section EEPROM Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_EEPROM	PS_ENABLE/ PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the EEPROM module in the SDK. **PS_DISABLE**: Disables the EEPROM module in the SDK.

EEPROM Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would go the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section EEPROM Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_EE_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100ms	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the configured inputs in the Database.

EEPROM Module Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the timer module in the configuration file. Please see section EEPROM Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_EE_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

EEPROM Placeholder

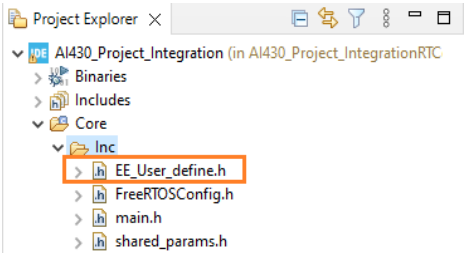
The AI430 SDK supports the below size of the placeholder and the user can modify them in the configuration file. Please see section EEPROM Sample Configuration

No	Variables	Options	Default State	Description
1	SIZE_OF_PLACEHOLDER	1 to 65535	100	User can select the size of the placeholder based on the application requirement

The SDK has currently defined 300 placeholders but the user can use 65535 placeholders. This can be used as a reference for all the additional elements that the user can use. The user would be able to read and write into the EEPROM places holder using the below DB variables. As the user would have still not vacated the house. These place holders are defined in the EE_PH_DB.h file. The user can add additional variables here.

Field ID	Data Type	Permission	Size	Description	Comments
EE_CAL01	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Place holder 1
EE_CAL02	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Place holder 2
EE_CAL03	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Place holder 3
EE_CAL04	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Place holder 4
EE_CAL05	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Place holder 5
EE_CAL0300	EEPROM_t	READ/WRITE	Variable	Place holder for EEPROM variable in EEPROM map	Considered placeholder for worst case scenario of each variable of 1 byte size

One the placeholder is defined is necessary to set the parameters of the EEPROM variable in the EE_User_define.h.




```
int8_t EE_CAL01_default = 10;

EE_Element_info     EE_user_elements[] =
{
   /* ID,    Size,       CRC_enable,                  Redundancy,                  Default_data_enable,          Default_data */
   {
     EE_CAL01 , sizeof(EE_CAL01_default), TRUE, 1, TRUE, &EE_CAL01_default},
    {EE_CAL02 , sizeof(EE_CAL01_default), TRUE, 2, TRUE, &EE_CAL01_default},
    {EE_CAL03 , sizeof(EE_CAL01_default), TRUE, 3, TRUE, &EE_CAL01_default},
    {EE_CAL04 , sizeof(EE_CAL01_default), TRUE, 4, TRUE, &EE_CAL01_default},
```

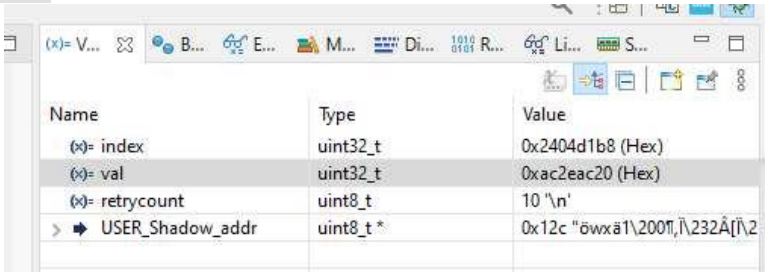
The parameters to be set is size, CRC enable, redundancy (multiple copies of the variable), enable default data and a pointer to the default data (if the reading of the variable fails it going to report the default data).
The functionality of the EEPROM platform service if all the parameters are enabled is the following: the data is going to be stored in the variable and the redundancy copies, if the principal variable fail to write the redundancy variable will be used until it fails and then a default value will be reported.
To make the EEPROM platform service update the values in the external EEPROM it is necessary to set a break point in the core/Maximatecc/src/EEPROM_Paltformservice.c in the following section.

```
void EEPROM_Shadow_Init()
{
        uint32_t index;
   uint32_t val;
   uint8_t  retrycount = 10;
   uint8_t  *USER_Shadow_addr;

   /* Read the EEPROM First page to check the Magic number */
   /* Check Magic number is present in the EEPROM          */
        /* If it is present, the EEPROM is already initialized  */
        /* with the USER data                                   */

   while(retrycount > 0)
   {
        TakeSPIBusLock();
```

```
GiveSPIBusLock();
        if (EEPROM_MAGIC_NUMBER == val)
                break;

        retrycount--;
   }

        if (EEPROM_MAGIC_NUMBER != val)
   {
                /* This is the first time writing, So initialize the
shadow memory */
                Initialize_EEPlaceHolder();
                Init_Shadow_memory();
        }
```

```
EEPROM_SPI_ReadBuffer((uint8_t *)&val, 0x00, MAGIC_NUMBER_SIZE);
```

After the breakpoint is reached in the window of local variables (upper right of the screen) the val value should be modified to make the EEPROM platform service format the external EEPROM, this is only necessary at debug stage and only when a new variable is defined.



To access the variables the user must use the START_EEPROM value + offset. For example, to access the variable To access the variable EE_CAL01 , the user will use the OFFSET as START_EEPROM + EE_CAL01. The sample code below gives an example to access the Placeholders for EEPROM.

```
if(KEY4_SHORT_PRESS == val)
{
   Get_DL((START_EEPROM+ EE_CAL01),(uint8_t *)&value);
}
```

EEPROM Sample Configuration

```
/*****************************************************************************************************************************************************************
*
                                                    * EEPROM Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * EEPROM Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_EEPROM PS_ENABLE
  #if (SDK_SERVICE_EEPROM == PS_ENABLE)
  /*!
  * EEPROM Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  *  ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_EE_TASK_PRIORITY osPriorityIdle
  /*!
  * EEPROM Task Periodicity 100ms
  */
  #define PS_EE_TASK_TIMEOUT 100
  /*!
  * EEPROM Place holder size
  */
  #define SIZE_OF_PLACEHOLDER 100
  #endif //SDK_SERVICE_EEPROM
```

WatchDog Module

The User would be able to use the below functionalities of the watch dog module via the DB variables and configuration file.

WatchDog Module Enable/Disable

The SDK provides the user the ability to enable/disable the watch dog module functionality by modifying the default configuration file. Please see section WatchDog Default Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_ WATCHDOG	PS_ENABLE/ PS_DISABLE	PS_ENABLE	PS_ENABLE: Enables the watch dog module in the SDK. PS_DISABLE: Disables the watch dog module in the SDK.

WatchDog Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any else perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section WatchDog Default Configuration.

No	Variables	Options	Default State	Description
1	PS_WD_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100ms	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the database.

WatchDog Task Priority

The AI430 SDK supports the below task priorities, and the user can modify the task priority for the light sensor module in the configuration file. Please see section WatchDog Default Configuration.

No	Variables	Options	Default State	Description
1	PS_WD_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

WatchDog User Task Enable/Disable

The SDK provides the user the ability to enable/disable the watch dog functionality by modifying the default configuration file. Please see section WatchDog Default Configuration.

No	Variables	Options	Default State	Description
1	USER_TASK_WD0	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 0 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 0 watchdog module in the SDK.
2	USER_TASK_WD1	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 1 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 1 watchdog module in the SDK.
3	USER_TASK_WD2	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 2 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 2 watchdog module in the SDK.
4	USER_TASK_WD3	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 3 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 3 watchdog module in the SDK.
5	USER_TASK_WD4	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 4 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 4 watchdog module in the SDK.
6	USER_TASK_WD5	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 5 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 5 watchdog module in the SDK.
7	USER_TASK_WD6	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 6 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 6 watchdog module in the SDK.
8	USER_TASK_WD7	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 7 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 7 watchdog module in the SDK.
9	USER_TASK_WD8	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 8 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 8 watchdog module in the SDK.
10	USER_TASK_WD10	PS_ENABLE/ PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the user task 9 watchdog module in the SDK. **PS_DISABLE**: Disables the user task 9 watchdog module in the SDK.

WatchDog Feed Timer Configuration

Watchdog is used for automatic correction of temporary hardware/software faults by resetting the MCU. The AI430 SDK allows the user to configure the watchdog timer. Once this timer expires the watchdog service would check all the registered tasks, if any of the tasks has not ping the watchdog service then it would reset the MCU. This timer value can be configured using the below parameter. Please see section WatchDog Default Configuration.

The user can configure the watchdog timer with different pre-scaler values as supported by the platform and they correspond to equivalent time. For example, when configured as IWDG_PRESCALER_256 the watchdog module expects to be refreshed every 40-50 secs else it would reset the MCU.

No	Variables	Options	Default State	Description
1	WATCHDOG_FEED_TIME	IWDG_PRESCALER_4 IWDG_PRESCALER_8 IWDG_PRESCALER_16 IWDG_PRESCALER_32 IWDG_PRESCALER_64 IWDG_PRESCALER_128 IWDG_PRESCALER_256	IWDG_PRESCALER_256	Watchdog feed time triggers a reset sequence when it is not refreshed within the expected time window

WatchDog Ping Functionality

The SDK watchdog service will reset the MCU if it finds that any of the threads are not functional. Hence as a user task it would be the users responsibility to keep pinging the watchdog service and updating the alive status. During runtime, the user can write to the below DB variable to report the alive status to the watchdog service.

Each user task has a corresponding watchdog ping variable that it needs to update. For example, user task 1 would use the WDO_PING variable as it has enabled the *USER_TASK_WD0* variable in the configuration file.

Field ID	Data Type	Permission	Size	Description	Comments
WD0_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WDO_PING variable informs the platform service task 0 is alive.
WD1_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD1_PING variable informs the platform service task 1 is alive.
WD2_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD2_PING variable informs the platform service task 2 is alive.
WD3_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD3_PING variable informs the platform service task 3 is alive.
WD4_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD4_PING variable informs the platform service task 4 is alive.
WD5_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD5_PING variable informs the platform service task 5 is alive.
WD6_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD6_PING variable informs the platform service task 6 is alive.
WD7_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD7_PING variable informs the platform service task 7 is alive.
WD8_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD8_PING variable informs the platform service task 8 is alive.
WD9_PING	DBu8	READ/WRITE	1	TASK_ID (1-10)	Setting WD9_PING variable informs the platform service task 9 is alive.

The sample code gives an example to ping for user task 5:

```
if(user_task_wd5 == 1)
{
  #if(USER_TASK_WD5 == PS_ENABLE)
    state = 6; // where 6 is the task ID
    Set_DL(WD5_PING , &state);
  #endif
}
```

WatchDog Default Configurations

```
/*****************************************************************************************************************************************************************
*
                                                        * Watchdog Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * Watchdog Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_WATCHDOG PS_DISABLE
  #if (SDK_SERVICE_WATCHDOG == PS_ENABLE)
  /*!
  * Watchdog Task Periodicity 100ms
  */
  #define PS_WD_TASK_TIMEOUT 100
  /*!
  * Watchdog Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  * ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_WD_TASK_PRIORITY osPriorityIdle
  /*!
  * Watchdog Reset timer
  */
  #define PS_WD_RESET_TIMER 500
  /*
  * IWDG_PRESCALER_4
  * IWDG_PRESCALER_16
  * IWDG_PRESCALER_32
  * IWDG_PRESCALER_64
  * IWDG_PRESCALER_128
  * IWDG_PRESCALER_256
  */

  #define WATCHDOG_FEED_TIME IWDG_PRESCALER_256
  /*!
  * Watchdog external task ping_id
  *
  * MAX Supported USER Watchdog is 10
  */
  #define USER_TASK_WD0 PS_DISABLE
  #define USER_TASK_WD1 PS_DISABLE
  #define USER_TASK_WD2 PS_DISABLE
  #define USER_TASK_WD3 PS_DISABLE
  #define USER_TASK_WD4 PS_DISABLE
  #define USER_TASK_WD5 PS_DISABLE
  #define USER_TASK_WD6 PS_DISABLE
  #define USER_TASK_WD7 PS_DISABLE
  #define USER_TASK_WD8 PS_DISABLE
  #define USER_TASK_WD9 PS_DISABLE
  #endif //SDK_SERVICE_WATCHDOG
```

* IWDG_PRESCALER_8

Power Mode Module

The User would be able to use the below functionalities of the power mode module via the DB variables and configuration file.

Power Mode Module Enable/Disable

The SDK provides the user the ability to enable/disable the power mode functionality by modifying the default configuration file. Please see section Power Mode Default Configurations.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_PM	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the power mode module in the SDK. **PS_DISABLE**: Disables the power mode module in the SDK.

Power Mode Time Out Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any and perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section Power Mode Default Configurations.

No	Variables	Options	Default State	Description
1	PS_PM_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the database.

Power Mode Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section Power Mode Default Configurations.

No	Variables	Options	Default State	Description
1	PS_PM_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement

Power Mode Wake Up Source Configuration

The AI430 SDK allows the user to configure the wake-up source, so that the device can exit from the low power mode. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Power Mode Default Configurations. The platform support wake- up from the below sources and they can be configured using the below parameters.

No	Variables	Options	Default State	Description
1	KEYPAD02_WAKE UP_SOURCE	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the keypad 2 as a wake-up source to exit low power mode. **PS_DISABLE**: Disables the keypad 2 as a wake-up source to exit low power mode.
2	KEYPAD04_WAKE UP_SOURCE	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the keypad 2 as a wake-up source to exit low power mode. **PS_DISABLE**: Disables the keypad 2 as a wake-up source to exit low power mode.
3	RTC_WAKEUP_S OURCE_STATE	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the RTC as a wake-up source to exit low power mode. **PS_DISABLE**: Disables the RTC as a wake-up source to exit low power mode.
4	IGN_WAKEUP_SO URCE	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the ignition as a wake-up source to exit low power mode. **PS_DISABLE**: Disables the ignition as a wake-up source to exit low power mode.

5	CAN_WAKEUP_SOURCE	PS_ENABLE PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the CAN as a wake-up source to exit low power mode. **PS_DISABLE**: Disables the CAN as a wake-up source to exit low power mode.

Power Mode RTC Timeout

The AI430 SDK allows the user to configure the RTC as a wake-up source, so that the device can exit from the low power mode. He also can set the timeout for the RTC to wake up the system. To do so, he can configure the below parameters in the configuration file. This default configuration can be done in the AI430_config.h. Please see section Power Mode Default Configurations.

No	Variables	Options	Default State	Description
1	RTC_WAKEUP_SOURCE_TIMEOUT	MIN VALUE : 0 MAX VALUE : 65535	1000ms	User can set the RTC wake up timeout using the configuration.

Power Mode Enable

The user would be able to enter the power mode during runtime using the below the DB variable (POWER_MODE). The platform supports the only one power mode configuration.

STOP MODE

Field ID	Data Type	Permission	Size	Description	Comments
POWER_MODE	DBu8	READ/WRITE	1	STOP	This field is used to set and read the Power mode configuration

The next code snippet shows how the user can enable the different power mode configuration:

```
#if (SDK_SERVICE_PM == PS_ENABLE)
  /* Stop Mode */
  pm_state = PM_STOP_MODE;
  Set_DL(POWER_MODE, &pm_state);
}
#endif
```

Power Mode Default Configurations

```
/*****************************************************************************************************************************************************************
*
                                                    * Power Management Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * Power Management Platform service Enable(PS_ENABLE) /
  Disable(PS_DISABLE) Macros
  */
  #define SDK_SERVICE_PM PS_ENABLE
  #if (SDK_SERVICE_PM == PS_ENABLE)
  /*!
  * Power Management Task Periodicity 100ms
  */
  #define PS_PM_TASK_TIMEOUT 100
  /*!
  * GPIO Wake up Source
  */
  #define KEYPAD01_WAKEUP_SOURCE PS_ENABLE
  #define KEYPAD02_WAKEUP_SOURCE PS_ENABLE
  #define KEYPAD03_WAKEUP_SOURCE PS_ENABLE
  #define KEYPAD04_WAKEUP_SOURCE PS_ENABLE
  /*!
  * RTC Wake up Source
  */
  #define RTC_WAKEUP_SOURCE_STATE PS_DISABLE
  #define RTC_WAKEUP_SOURCE_TIMEOUT 10000
  /*!
  * IGN Wake up Source
  */
  #define IGN_WAKEUP_SOURCE PS_ENABLE
  #endif //SDK_SERVICE_PM
```

LCD Module

The User would be able to use the below functionalities of the LCD module via the DB variables and configuration file.

LCD Mode Module Enable/Disable

The SDK provides the user the ability to enable/disable the LCD functionality by modifying the default configuration file. Please see section LCD Default Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_LCD	PS_ENABLE / PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the lcd module in the SDK. **PS_DISABLE**: Disables the lcd module in the SDK.

LCD Module Timeout Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any and perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section LCD Default Configuration.

No	Variables	Options	Default State	Description
1	PS_LCD_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100ms	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the database.

LCD Task Priority

The AI430 SDK supports the below task priorities and the user can modify the task priority for the light sensor module in the configuration file. Please see section LCD Default Configuration.

No	Variables	Options	Default State	Description
1	PS_LCD_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

LCD State

The AI430 SDK supports the user to configure the default state of the LCD (either OFF/ON) and this can be done by modifying the next parameter in the configuration file. Please see section LCD Default Configuration.

No	Variables	Options	Default State	Description
1	CONF_LCD_STATE	LCD_CONF_ON LCD_CONF_OFF	LCD_CONF_ON	User can configure the LCD state as ON/OFF.

The user can also get/set the default state of the LCD (either OFF/ON) during runtime using the next DB variable:

Field ID	Data Type	Permission	Size	Description	Comments
LCD_STATE	DBu8	READ/WRITE	1	ON/OFF	This field sets and reads back the Turn ON or OFF the LCD

The next code snippet shows how you can set or get the LCD State

```
/* Get the LCD State value from the DB */
Get_DL(LCD_STATE , &state);
if (LCD_CONF_OFF == state)
{
    state = LCD_CONF_ON;
    /* Set the LCD state to ON */
    Set_DL(LCD_STATE, &state);
}
```

LCD Brightness

The AI430 SDK supports the user to configure the LCD brightness, and this can be done by modifying the below parameter in the configuration file. Please see section LCD Default Configuration. The next configuration means the screen is at 30% brightness level. If the user needs full brightness, then it will need to be set at 100.

No	Variables	Options	Default State	Description
1	CONF_LCD_BRIGHTNESS	0-100	30%	User can configure the LCD brightness.

The user would be able to read and modify the below functionalities of the LCD module via the DB variables and configuration file:

Field ID	Data Type	Permission	Size	Description	Comments
LCD_BRIGHTNESS	DBu8	READ/WRITE	1	0-100	This field sets the percentage of brightness from 0 to 100 (full brightness for the LCD)

The sample code gives an example to set the brightness of the LCD

```
void LCDView::brightnessinc()
{
#if (SDK_SERVICE_LCD == PS_ENABLE)
brightness_value++;
if (!(LCD_BRT_MAX >= brightness_value))
brightness_value = LCD_BRT_MAX;
Set_DL(LCD_BRIGHTNESS, (uint8_t *)&brightness_value);
#endif
}
```

The sample code gives an example to get the brightness of the LCD

```
LCDView::LCDView()
{
#if (SDK_SERVICE_LCD == PS_ENABLE)
brightness_value = 0;
/* Get the LCD Brightness value from the DB */
Get_DL(LCD_STATE , &state);
Get_DL(LCD_BRIGHTNESS, (uint8_t *)&brightness_value);
#endif
}
```

LCD Default Configuration

```
/*****************************************************************************************************************************************************************
*
                                                        * LCD Module Configuration
*
*****************************************************************************************************************************************************************/
  /*!
  * LCD Platform service Enable(PS_ENABLE) / Disable(PS_DISABLE)
  Macros
  */
  #define SDK_SERVICE_LCD PS_ENABLE
  #if (SDK_SERVICE_LCD == PS_ENABLE)
  /*!
  * LCD Task Periodicity 100ms
  */
  #define PS_LCD_TASK_TIMEOUT 100
  /*!
  * LCD Task Priority
  * osPriorityNone = 0,
  * osPriorityIdle = 1,
  * osPriorityLow = 8,
  * osPriorityLow1 = 8+1,
  * ,, ,,
  * ,, ,,
  * osPriorityISR = 56,
  * osPriorityError = -1,
  * osPriorityReserved = 0x7FFFFFFF
  */
  #define PS_LCD_TASK_PRIORITY osPriorityIdle

  /*!
  * MACOR Supported
  *
  * CONF_LCD_STATE LCD_CONF_ON/
  * LCD_CONF_OFF
  *
  * CONF_LCD_BRIGHTNESS <0 - 100>
  *
  */
  #define CONF_LCD_STATE LCD_CONF_ON
  #define CONF_LCD_BRIGHTNESS 30
  #endif //SDK_SERVICE_LCD
```

CAN Module

The SDK supports two can channels CAN0 and CAN1. These channels can be used together or independently. Please note that standalone CAN module will be disabled when J1939 is enabled in the configuration file. To use CAN in a standalone mode J1939 must be disabled in the configuration file. Below code snippet from the AI430_config.h that shows the same

```
#if ((SDK_SERVICE_J1939 == PS_ENABLE) && (SDK_SERVICE_FDCAN == PS_ENABLE))
#undef SDK_SERVICE_FDCAN
#define SDK_SERVICE_FDCAN PS_DISABLE
#endif
```

CAN Module Configuration Support

The SDK provides the user the ability to Enable/disable the CAN functionality by modifying the default configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_FDCAN	PS_ENABLE / PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the FDCAN module in the SDK. **PS_DISABLE**: Disables the FDCAN module in the SDK.

CAN Enable/Disable

The SDK provides the user the ability to enable/disable the CAN0/CAN1 functionality by modifying the default configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FDCAN0_ENABLE	PS_ENABLE / PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the FDCAN0 module in the SDK. **PS_DISABLE**: Disables the FDCAN0 module in the SDK.
2	FDCAN1_ENABLE	PS_ENABLE / PS_DISABLE	PS_ENABLE	**PS_ENABLE**: Enables the FDCAN1 module in the SDK. **PS_DISABLE**: Disables the FDCAN1 module in the SDK.

CAN Module Timeout Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_FDCAN_TASK_TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the database.

CAN Task Priority

The AI430 SDK supports the next task priorities and the user can modify the task priority for the CAN module in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_FDCAN_TASK_PRIORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can select any one of the priorities based on the application requirement.

CAN Baud Rate

The AI430 SDK supports the next Baud rate and the user can modify the Baud rates for the CAN module in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FDCAN0_BAUDRATE	BAUDRATE_50K BAUDRATE_100 BAUDRATE_125 BAUDRATE_250 BAUDRATE_500 BAUDRATE_1000	BAUDRATE_5	User can set the Baud rate for CAN0

The AI430 SDK user can change the baud rate during runtime using the below DB variables:

Field ID	Data Type	Permission	Size	Description	Comments
CAN_CH0_BAUDRATE	DBu32	READ/WRITE	1	Default 250Kbaud Supported baud rates: AUTO, 50K, 100K, 125K, 250K, 500K, 1M	CAN Channel0 Baudrate
CAN_CH1_BAUDRATE	DBu32	READ/WRITE	1	Default 250Kbaud Supported baud rates: AUTO, 50K, 100K, 125K, 250K, 500K, 1M	CAN Channel1 Baudrate

The next code snippet shows how the baud rate can be changed during the runtime:

```
/*
* CAN1 Supporting baud rate                    can1_buf[0] = BAUDRATE_250K;
```

```
  * BAUDRATE_50K
  * BAUDRATE_100K
  * BAUDRATE_125K
  * BAUDRATE_250K
  * BAUDRATE_500K
* BAUDRATE_1000K
*/
```

```
Set_DL(CAN_CH1_BAUDRATE, &can1_buf[0]);
break;
```

CAN Identifier Configurations

The AI430 SDK supports the next configuration parameters for the CAN and the user can modify the same in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FDCAN0_IDENTIFIER	User Configurable ID	0x19FEFCFE	User can configure the CAN0 Identifier
2	FDCAN0_IDTYPE	FDCAN_EXTENDED_ID/ FDCAN_STANDRD_ID	FDCAN_EXTENDED_ID	User can configure the CAN0 IDTYPE as Extended/Standard
3	FDCAN0_ID	FDCAN_STANDARD_ID/ FDCAN_EXTENDED_ID	FDCAN_STANDARD_ID	User can configure the CAN0 ID
4	FDCAN1_IDENTIFIER	User Configurable ID	0x19FEFCFE	User can configure the CAN1 Identifier
5	FDCAN1_IDTYPE	FDCAN_EXTENDED_ID/ FDCAN_STANDRD_ID	FDCAN_EXTENDED_ID	User can configure the CAN1 IDTYPE as Extended/Standard
6	FDCAN1_ID	FDCAN_STANDARD_ID/ FDCAN_EXTENDED_ID	FDCAN_STANDARD_ID	User can configure the CAN1 ID

CAN Channel Configurations

The AI430 SDK supports the below filter properties for the CAN module in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FFDCAN0_RXBUFFERIND	0-65535	1	User can configure the Rx buffer
2	FDCAN0_TXFRAMETYPE	FDCAN_DATA_FRAME/FDCAN_REMOTE_FRAME	FDCAN_DATA_FRAME	User can configure the TX frame type as Data Frame or Remote Frame
3	FDCAN0_DATALENGTH	FDCAN_DLC_BYTES_0 FDCAN_DLC_BYTES_1 FDCAN_DLC_BYTES_2 FDCAN_DLC_BYTES_3 FDCAN_DLC_BYTES_4 FDCAN_DLC_BYTES_5 FDCAN_DLC_BYTES_6 FDCAN_DLC_BYTES_7 FDCAN_DLC_BYTES_8 FDCAN_DLC_BYTES_12 FDCAN_DLC_BYTES_16 FDCAN_DLC_BYTES_20 FDCAN_DLC_BYTES_24 FDCAN_DLC_BYTES_32 FDCAN_DLC_BYTES_48 FDCAN_DLC_BYTES_64	FDCAN_DLC_BYTES_8	User can configure the length of the data.
4	FDCAN0_ERRORSTATEIND	FDCAN_ESI_ACTIVE/ FDCAN_ESI_PASSIVE	FDCAN_ESI_ACTIVE	User can configure The errors state as Active or Passive.
5	FDCAN0_BITRATESWITCH	FDCAN_BRS_ON/FDCAN_BRS_OFF	FDCAN_BRS_ON	User can configure the Bit rate switch on/off.
6	FDCAN0_FDFORMATE	FDCAN_CLASSIC_CAN FDCAN_FD_CAN	FDCAN_FD_CAN	User can configure the FDCAN format.
7	FDCAN0_TXEVENTFIFOCONTROL	FDCAN_STORE_TX_EVENTS/FDCAN_NO_TX_EVENTS	FDCAN_STORE_TX_EVENTS	User can configure the event FIFO control.
8	FDCAN0_MESSAGEMARKER	0-65535	0	User can configure message marker.
9	FDCAN0_RECEIVE_TASK_DELAY	0-500ms	100ms	User can configure the delay for the receive task.
10	FFDCAN1_RXBUFFERIND	0-65535	1	User can configure the TX frame type as Data Frame or Remote Frame.
11	FDCAN1_TXFRAMETYPE	FDCAN_DATA_FRAME/FDCAN_REMOTE_FRAME	FDCAN_DATA_FRAME	User can configure the TX frame type as Data Frame or Remote Frame
12	FDCAN1_DATALENGTH	FDCAN_DLC_BYTES_0 FDCAN_DLC_BYTES_1 FDCAN_DLC_BYTES_2 FDCAN_DLC_BYTES_3 FDCAN_DLC_BYTES_4 FDCAN_DLC_BYTES_5 FDCAN_DLC_BYTES_6 FDCAN_DLC_BYTES_7 FDCAN_DLC_BYTES_8 FDCAN_DLC_BYTES_12	FDCAN_DLC_BYTES_8	User can configure the length of the data.

		FDCAN_DLC_BYTES_16 FDCAN_DLC_BYTES_20 FDCAN_DLC_BYTES_24 FDCAN_DLC_BYTES_32 FDCAN_DLC_BYTES_48 FDCAN_DLC_BYTES_64		
13	FDCAN1_ERRORSTATEIND	FDCAN_ESI_ACTIVE/ FDCAN_ESI_PASSIVE	FDCAN_ESI_ACTIVE	User can configure The errors state as Active or Passive.
14	FDCAN1_BITRATESWITCH	FDCAN_BRS_ON/FDCAN_BRS_OFF	FDCAN_BRS_ON	User can configure the Bit rate switch on/off.
15	FDCAN1_FDFORMATE	FDCAN_CLASSIC_CAN FDCAN_FD_CAN	FDCAN_FD_CAN	User can configure the FDCAN format.
16	FDCAN1_TXEVENTFIFOCONTROL	FDCAN_STORE_TX_EVENTS/FDCAN_NO_TX_EVENTS	FDCAN_STORE_TX_EVENTS	User can configure the event FIFO control.
17	FDCAN1_MESSAGEMARKER	0-65535	0	User can configure message marker.
18	FDCAN0_RECEIVE_TASK_DELAY	0-500ms	100ms	User can configure the delay for the receive task.

CAN Filter Configurations

The AI430 SDK supports the CAN Filter configurations, and the user can modify the same in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FDCAN0_FILTERINDEX	0-65535	0	User can configure the filter index.
2	FDCAN0_FILTER TYPE	FILTER_DUAL or FILTER_RANGE or FILTER_MASK or FILTER_RANGE_NO_EIDM	FDCAN_FILTER_DUAL	User can configure the filter Type.
3	FDCAN0_FILTER CONFIG	FDCAN_FILTER_DISABLE FDCAN_FILTER_TO_RXFIFO0 FDCAN_FILTER_TO_RXFIFO1 FDCAN_FILTER_REJECT FDCAN_FILTER_HP FDCAN_FILTER_TO_RXFIFO1_HP FDCAN_FILTER_TO_RXBUFFER	FDCAN_FILTER_TO_RXBUFFER	User can configure the Filter.
4	FDCAN0_FILTERID1	0-65535	FDCAN_DEFAULT_FILTERID1	User can configure the default filter id 1.
5	FDCAN0_FILTERID2	0-65535	FDCAN_DEFAULT_FILTERID2	User can configure the default filter id 2.
6	FDCAN1_FILTERINDEX	0-65535	0	User can configure the filter index.
7	FDCAN1_FILTER TYPE	FILTER_DUAL or FILTER_RANGE or FILTER_MASK or FILTER_RANGE_NO_EIDM	FDCAN_FILTER_DUAL	User can configure the filter Type.
8	FDCAN1_FILTER CONFIG	FDCAN_FILTER_DISABLE FDCAN_FILTER_TO_RXFIFO0 FDCAN_FILTER_TO_RXFIFO1 FDCAN_FILTER_REJECT FDCAN_FILTER_HP FDCAN_FILTER_TO_RXFIFO1_HP FDCAN_FILTER_TO_RXBUFFER	FDCAN_FILTER_TO_RXBUFFER	User can configure the Filter.
9	FDCAN1_FILTERID1	0-65535	FDCAN_DEFAULT_FILTERID1	User can configure the default filter id 1.
10	FDCAN1_FILTERID2	0-65535	FDCAN_DEFAULT_FILTERID2	User can configure the default filter id 2.

The user can also read and write to the next CAN filter properties during the runtime using the CAN DB variables:

Field ID	Data Type	Permission	Size	Description	Comments
CAN_CH0_FILTER_INDEX_ENABLE	DBu8	READ/WRITE	1	Enable / Disable receive data with filter. **Index range 0-32** (CAN_MODE_EXTENDED_ID | CAN_MODE_BUS_MONITORING_EXTENDED_ID) **Index range 0-64** (CAN_MODE_STANDARD_ID | CAN_MODE_BUS_MONITORING_STANDARD_ID)	CAN Channel0 filter index state.
CAN_CH0_FILTER_INDEX_ID	DBu32	READ/WRITE	4	FIFO ID use to filter	CAN Channel0 filter index ID.

CAN_CH0_FILTER_INDEX_IDMASK	DBu32	READ/WRITE	4	ID MASK use to filter	CAN Channel0 index ID Mask.
CAN_CH1_FILTER_INDEX_ENABLE	DBu8	READ/WRITE	1	Enable / Disable receive data with filter. **Index range 0-32** (CAN_MODE_EXTENDED_ID | CAN_MODE_BUS_MONITORING_EXTENDED_ID) **Index range 0-64** (CAN_MODE_STANDARD_ID | CAN_MODE_BUS_MONITORING_STANDARD_ID)	CAN Channel1 filter index state.
CAN_CH1_FILTER_INDEX_ID	DBu32	READ/WRITE	4	FIFO ID use to filter	CAN Channel1 filter index ID.
CAN_CH1_FILTER_INDEX_IDMASK	DBu32	READ/WRITE	4	ID MASK use to filter	CAN Channel1 filter index ID Mask.

The below code sample show how the filter index is updated from the application:

```
can0_buf[0] = ENABLE;
Set_DL(CAN_CH0_FILTER_INDEX_ENABLE, &can0_buf[0]);
can0_buf[0] = 10;
Set_DL(CAN_CH0_FILTER_INDEX_ID, &can0_buf[0]);
```

CAN Receive Task Delay

The AI430 SDK supports the configuration of the CAN Task Delay and the user can modify the CAN1/2 Receive Task Delay for the CAN module in the configuration file. Please see section CAN Sample Configuration.

No	Variables	Options	Default State	Description
1	FDCAN0_RECEIVE_TASK_DELAY	0-500ms	100ms	User can configure the delay for the receive task.
2	FDCAN0_RECEIVE_TASK_DELAY	0-500ms	100ms	User can configure the delay for the receive task.

CAN Channel Modes and States

The user would be able to query the database to get the mode and current state of the CAN channels using the below CAN module DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
CAN_CH0_MODE	DBu8	READ/WRITE	1	**Default:** CAN_MODE_EXTENDED_ID CAN_MODE_STANDARD_ID, CAN_MODE_EXTENDED_ID, CAN_MODE_BUS_MONITORING_STANDARD_ID, CAN_MODE_BUS_MONITORING_EXTENDED_ID,	This field sets and reads the CAN Channel0 Modes.
CAN_CH1_MODE	DBu8	READ/WRITE	1	**Default:** CAN_MODE_EXTENDED_ID CAN_MODE_STANDARD_ID, CAN_MODE_EXTENDED_ID, CAN_MODE_BUS_MONITORING_STANDARD_ID, CAN_MODE_BUS_MONITORING_EXTENDED_ID,	This field sets and reads the CAN Channel1 Modes.
CAN_CH0_STATES	DBu8	READ	1	CAN_BUS_OFF, CAN_BUS_ON, CAN_STATE_PASSIVE, CAN_STATE_UNCHANGED	This field is used to read the CAN Channel 0 state.
CAN_CH1_STATES	DBu8	READ	1	CAN_BUS_OFF, CAN_BUS_ON, CAN_STATE_PASSIVE, CAN_STATE_UNCHANGED	This field is used to read the CAN Channel 0 state.
CAN_CH0_COMM_STATE_EVENTS	DBu8	READ	1	STATE_EVENT_NONE, STATE_EVENT_BUS_OFF, STATE_EVENT_BUS_OFF_RECOVERY, STATE_EVENT_BUS_ON, STATE_EVENT_PASSIVE, STATE_EVENT_ACTIVE, STATE_EVENT_OVERRUN, STATE_EVENT_QUEUE_FULL, STATE_EVENT_QUEUE_OVERFLOW, STATE_EVENT_QUEUE_EMPTY, STATE_EVENT_DRIVER_ERROR	This field is used to read the CAN Channel0 communication state event.
CAN_CH1_COMM_STATE_EVENTS	DBu8	READ	1	STATE_EVENT_NONE, STATE_EVENT_BUS_OFF, STATE_EVENT_BUS_OFF_RECOVERY,	This field is used to read the CAN Channel1 communication state event.

				STATE_EVENT_BUS_ON, STATE_EVENT_PASSIVE, STATE_EVENT_ACTIVE, STATE_EVENT_OVERRUN, STATE_EVENT_QUEUE_FULL, STATE_EVENT_QUEUE_OVERFLOW, STATE_EVENT_QUEUE_EMPTY, STATE_EVENT_DRIVER_ERROR	

The next code snippet shows how we can access the CAN mode and states:

```
/*
* Get the CAN1 state and update the mode
*/
Get_DL(CAN_CH1_STATES, &can1_buf[0]);
can1_buf[0] = CAN_MODE_STANDARD_ID;
Set_DL(CAN_CH1_MODE, &can1_buf[0]);
```

```
/*
* Get the CAN1 communication state event
*/
Get_DL(CAN_CH1_COMM_STATE_EVENTS, &can1_buf[0]);
```

CAN Channel Reset

The user would be able to reset the CAN channel and CAN driver using the below CAN module DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
CAN_CH0_RESET	DBu8	READ/WRITE	1	TRUE = Resets the CAN controller and Bus Off mode.	This variable supports the reset of the CAN Channel 0
CAN_CH1_RESET	DBu8	READ/WRITE	1	TRUE = Resets the CAN controller and Bus Off mode.	This variable supports the reset of the CAN Channel 1
CAN_CH0_DRIVER_RESET	Dbu8	READ/WRITE	1	TRUE = Reinitialize the CAN driver if Driver Error is Set.	This variable supports the reset of the CAN Channel 0 Driver
CAN_CH1_DRIVER_RESET	Dbu8	READ/WRITE	1	TRUE = Reinitialize the CAN driver if Driver Error is Set.	This variable supports the reset of the CAN Channel 1 Driver

The next code snippet shows how the user can reset the CAN channel and driver

```
/*
* Reset the CAN CHANNEL 0
*/
case 4:
can0_buf[0] = TRUE;
Set_DL(CAN_CH0_RESET, &can0_buf[0]);
break;
```

```
/*
* Reset the CAN Channel Driver 0
*/
case 5:
can0_buf[0] = TRUE;
Set_DL(CAN_CH0_DRIVER_RESET, &can0_buf[0]);
break;
```

CAN Module RX/TX

The AI430 SDK allows the users to use the CAN channel to send or receive data. To do so please use the below variables. To read incoming data over the CAN channel, the user will need to monitor the RX DATA SIZE variable and see if there is any pending data available and if so, read the data. To send data over the CAN channel, the user will fill the TX buffer and send the data over CAN. The SDK will take care of handling the pending data. The CAN default data packet size is defined as 64 bytes hence the user is expected to create a buffer of this size while reading the data.

Field ID	Data Type	Permission	Size	Description	Comments
CAN0_RX_BYTE_COUNT	DBu8	READ	1	CAN1 RX Byte count value	CAN Channel0 RX Byte count.
CAN0_TX_BYTE_COUNT	DBu8	READ	1	CAN1 TX Byte count value	CAN Channel0 TX Byte count.
CAN0_RX_DATA_IS_AVAILABLE	DBu8	READ	1	CAN1 data available flag	CAN channel0 RX data available flag.
CAN0_RX_DATA_SIZE	DBu8	READ	1	CAN1 RX data size	CAN Channel0 RX data size.
CAN1_RX_BYTE_COUNT	DBu8	READ	1	CAN2 RX Byte count value	CAN Channel1 RX Byte count.
CAN1_TX_BYTE_COUNT	DBu8	READ	1	CAN2 TX Byte count value	CAN Channel0 TX Byte count.
CAN1_RX_DATA_IS_AVAILABLE	DBu8	READ	1	CAN2 data available flag	CAN channel1 RX data available flag.
CAN1_RX_DATA_SIZE	DBu8	READ	1	CAN2 RX data size	CAN Channel1 RX data size.

The sample code for sending and receiving the data from CAN is shown below:

```
if(CAN_BUS_OFF != can1_buf[0])
{
Get_DL(CAN1_RX_DATA_IS_AVAILABLE, &can1_buf[0]);
```

```
if(0 != status)
{
memset(&can_rxbuffer1[0], 0x00, sizeof(can_rxbuffer1));
```

```
}
#if (SDK_SERVICE_FDCAN == PS_ENABLE)
uint8_t status;
/* Get the CAN RX status */
Get_DL(CAN1_RX_DATA_SIZE, &status);
/* Read the Rx data from the DB */
Get_DL(CAN1_RX_DATA, (uint8_t *)&can_rxbuffer1[0]);
}
/*To send the data over the CAN bus, set the CAN tx data */
Set_DL(CAN0_TX_DATA , (uint8_t *)&can0buf[0]);
```

CAN Sample Configuration

```
/*****************************************************************************************************************************************************
*
                                                        * FDCAN Module
*
*****************************************************************************************************************************************************/

/*!
* FDCAN Platform service (PS_ENABLE) / Disable(PS_DISABLE)
Macros
*/
#define SDK_SERVICE_FDCAN PS_ENABLE
#if (SDK_SERVICE_FDCAN == PS_ENABLE)
#define FDCAN1_ENABLE PS_ENABLE
#define FDCAN2_ENABLE PS_ENABLE
/*!
* FDCAN Task Periodicity 100ms
*/
#define PS_FDCAN_TASK_TIMEOUT 100
/*!
* FDCAN Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_FDCAN_TASK_PRIORITY osPriorityIdle
#if (FDCAN1_ENABLE == PS_ENABLE)
/*
* BAUDRATE_50K
* BAUDRATE_100K
* BAUDRATE_125K
* BAUDRATE_250K
* BAUDRATE_500K
* BAUDRATE_1000K
*/
#define FDCAN1_BAUDRATE BAUDRATE_50K
#define FDCAN1_IDENTIFIER 0x19FEFCFE //0x111
#define FDCAN1_IDTYPE FDCAN_EXTENDED_ID
#define FDCAN1_TXFRAMETYPE FDCAN_DATA_FRAME
#define FDCAN1_DATALENGTH FDCAN_DLC_BYTES_8
#define FDCAN1_ERRORSTATEIND FDCAN_ESI_ACTIVE
#define FDCAN1_BITRATESWITCH FDCAN_BRS_ON
#define FDCAN1_FDFORMATE FDCAN_FD_CAN
#define FDCAN1_TXEVENTFIFOCONTROL
FDCAN_STORE_TX_EVENTS
#define FDCAN1_MESSAGEMARKER 0
#define FDCAN1_ID FDCAN_STANDARD_ID
#define FDCAN1_FILTERINDEX 0
#define FDCAN1_FILTERTYPE FDCAN_FILTER_DUAL
#define FDCAN1_FILTERCONFIG
FDCAN_FILTER_TO_RXBUFFER
#define FDCAN1_FILTERID1 FDCAN_DEAFULT_FILTERID1
#define FDCAN1_FILTERID2 FDCAN_DEAFULT_FILTERID2
#define FDCAN1_RXBUFFERIND 1
/* CAN1 Receive Task Delay */
#define FDCAN1_RECEIVE_TASK_DELAY 100
#endif /* FDCAN1_ENABLE */
#if (FDCAN2_ENABLE == PS_ENABLE)
/*
* BAUDRATE_50K
* BAUDRATE_100K
* BAUDRATE_125K
* BAUDRATE_250K
* BAUDRATE_500K
* BAUDRATE_1000K
*/
#define FDCAN2_BAUDRATE BAUDRATE_50K
#define FDCAN2_IDENTIFIER 0x18FEFCFE //0x222
#define FDCAN2_IDTYPE FDCAN_EXTENDED_ID
#define FDCAN2_TXFRAMETYPE FDCAN_DATA_FRAME
#define FDCAN2_DATALENGTH FDCAN_DLC_BYTES_8
#define FDCAN2_ERRORSTATEIND FDCAN_ESI_ACTIVE
#define FDCAN2_BITRATESWITCH FDCAN_BRS_ON
#define FDCAN2_FDFORMATE FDCAN_FD_CAN
#define FDCAN2_TXEVENTFIFOCONTROL
FDCAN_STORE_TX_EVENTS
#define FDCAN2_MESSAGEMARKER 0
#define FDCAN2_ID FDCAN_STANDARD_ID
#define FDCAN2_FILTERINDEX 0
#define FDCAN2_FILTERTYPE FDCAN_FILTER_DUAL
#define FDCAN2_FILTERCONFIG
FDCAN_FILTER_TO_RXBUFFER
#define FDCAN2_FILTERID1 FDCAN_DEAFULT_FILTERID1
#define FDCAN2_FILTERID2 FDCAN_DEAFULT_FILTERID2
#define FDCAN2_RXBUFFERIND 1
/* CAN2 Receive Task Delay */
#define FDCAN2_RECEIVE_TASK_DELAY 100
#endif /* FDCAN2_ENABLE */
#endif //SDK_SERVICE_FDCAN
```

J1939

J1939 Module Configuration Support

The SDK provides the user the ability to enable/disable the J1939 functionality by modifying the default configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	SDK_SERVICE_J1939	PS_ENABLE PS_DISABLE	PS_DISABLE	**PS_ENABLE**: Enables the J1939 module in the SDK. **PS_DISABLE**: Disables the J1939 module in the SDK.

Module Timeout Configuration

The AI430 SDK user can configure the timeout value of the task such that, every time the timeout occurs the task would read the hardware and update it in the DB so that when the user reads the DB, he will receive the latest updated data if there is any or perform any other routine tasks. This default configuration can be done in the AI430_config.h. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_J1939_TASK_ TIMEOUT	MIN VALUE : 50 MAX VALUE : 500	100	The user can configure the timeout value of task so that the platform service would go and read the hardware and update the database.

J1939 Task Priority

The AI430 SDK supports the next task priorities and the user can modify the task priority for the J1939 module in the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	PS_J1939_TASK_PRI ORITY	osPriorityNone , osPriorityIdle , osPriorityLow , osPriorityLow1 , osPriorityISR , osPriorityError , osPriorityReserved	osPriorityIdle	User can configure this macro to default priority.

J1939 Claim Address Enable/Disable

The AI430 SDK supports the address claim functionality can be enabled or disable from the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	J1939_CLAIM_ADDRESS	PS_ENABLE PS_DISABLE	PS_ENABLE	User can enable/disable this macro to J1939 claim address

J1939 CAN Enable/Disable

The AI430 SDK supports the enabling CAN0 and CAN1 for J1939 and they can be enabled or disable from the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	J1939_CAN0_ENABLE	PS_ENABLE/PS_DISABLE	PS_ENABLE	User can enable/disable J1939 CAN0
2	J1939_CAN1_ENABLE	PS_ENABLE/PS_DISABLE	PS_ENABLE	User can enable/disable J1939 CAN1

J1939 Claim Address

The AI430 SDK supports the address claim for channel CAN0 and CAN1 for J1939 and they can be configured as any value between 0-255 based on your requirement from the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	J1939_CAN0_CLAIM_ADDRESS	0-255	23	User can configure this macro to the required claim address.
2	J1939_CAN1_CLAIM_ADDRESS	0-255	85	User can configure this macro to the required claim address.

J1939 CAN Bit Rate

The AI430 SDK supports the bit rate configuration for the CAN0 and CAN1 for J1939 and they can be configured based on your requirement from the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	J1939_CAN0_BITRATE	0/250/500/1000	250	User can configure this macro to default J1939 CAN0 BITRATE.
2	J1939_CAN1_BITRATE	0/250/500/1000	250	User can configure this macro to default J1939 CAN1 BITRATE.

J1939 Diagnostics Support

The AI430 SDK supports the enabling and disabling the diagnostics support for J1939 and they can be configured based on your requirement from the configuration file. Please see section J1939 Sample Configuration.
The SDK Currently supports the below messages: DM1 DM2

No	Variables	Options	Default State	Description
1	EMTOS_J1939_DM1	PS_ENABLE / PS_DISABLE	PS_ENABLE	User can enable/disable the Emotas J1939 DM1.
2	EMTOS_J1939_DM2	PS_ENABLE / PS_DISABLE	PS_ENABLE	User can enable/disable the Emotas J1939 DM2.

Emotas: user can reach out to an established maximatecc agent to add a customized J1939 signals and receive a quotation.

J1939 Dynamic Address Claim

The AI430 SDK supports the dynamic address claim for CAN0 and CAN1 for J1939 and they can be configured based on your requirement from the configuration file. Please see section J1939 Sample Configuration.

No	Variables	Options	Default State	Description
1	J1939_CAN0_ADDRESS_CLAIM_DYNAMIC	PS_ENABLE/PS_DISABLE	PS_ENABLE	User can enable/disable CAN0 dynamic address claim.
2	J1939_CAN1_ADDRESS_CLAIM_DYMAMIC	PS_ENABLE/PS_DISABLE	PS_ENABLE	User can enable/disable CAN1 dynamic address claim.

J1939 Dynamic Address Claim Next Address Configuration

The AI430 SDK supports the dynamic address claim next address for CAN0 and CAN1 for J1939 and they can be configured based on your requirement from the configuration file. Please see section J1939 Sample Configuration. When dynamic address claim is set to 1, search will start from this value and up to claim the address.

No	Variables	Options	Default State	Description
1	J1939_CAN0_ADDRESS_CLAIM_NEXT_ADDRESS	0-255	80	User can configure the start address for address claim in dynamic mode for CAN0.
2	J1939_CAN0_ADDRESS_CLAIM_NEXT_ADDRESS	0-255	90	User can configure the start address for address claim in dynamic mode for CAN1.

J1939 Configure Number of PGN's Supported.

The AI430 SDK supports the configuration of the number of RX and TX PGNS on CAN0 and CAN1 for J1939 and they can be configured based on your requirement from the configuration file. Please see section J1939 Sample Configuration.
Please note that the SDK can support a maximum of 300 PGNs including RX and TX over CAN0 and CAN1 put together.

No	Variables	Options	Default State	Description
1	CAN0_NUMBER_PGNS_RX	MIN VALUE = 0 MAX VALUE = 300	17	User can configure the number of RX PGNs supported on CAN0.
2	CAN1_NUMBER_PGNS_RX	MIN VALUE = 0 MAX VALUE = 300	3	User can configure the number of RX PGNs supported on CAN1.
3	CAN0_NUMBER_PGNS_TX	MIN VALUE = 0 MAX VALUE = 300	1	User can configure the number of TX PGNs supported on CAN0.
4	CAN1_NUMBER_PGNS_TX	MIN VALUE = 0 MAX VALUE = 300	1	User can configure the number of TX PGNs supported on CAN1.

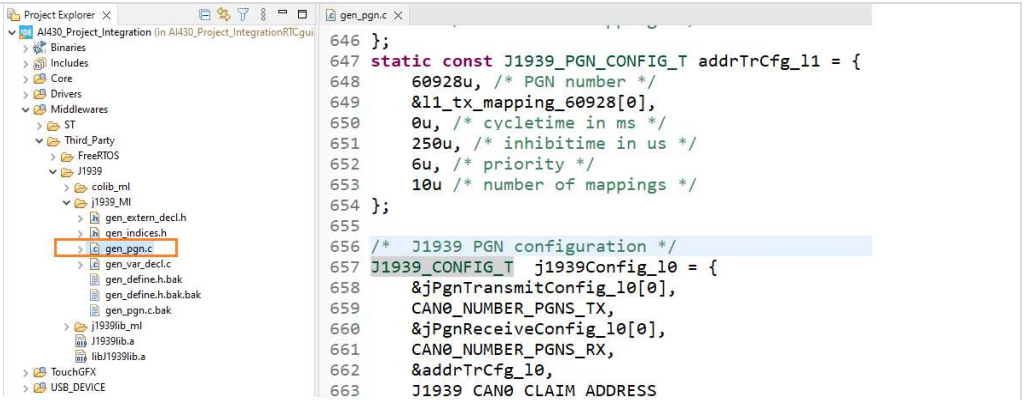
For example, if the user wants to receive 6 PGNs on the channel 2, the user must do as follows: CAN1_NUMBER_PGNS_RX 6

J1939 PGN and SPN Configuration

The AI430 SDK supports up to 300 PGN values and the same can be configured in the J1939 stack and their values will be read and updated in the Data layer for the application's access. The user can replace existing PGNs' or configure new PGN and SPN values for CAN0 and CAN1 for J1939 during compile time and same can be accessed from the DB during runtime.

J1939 Source Code

The J1939 source and its configuration files are available in the below path as highlighted in the following diagram. Middlewares/Third_Party/J1939 /j1939_ML



J1939 Supported PGN List

The AI430 SDK supports the below PGN values currently.

No	PGN	Signal Name
1	61444	Electronics Engine Controller 1
2	65110	Diesel Exhaust Fluid Tank 1 Information
3	65276	Dash Display
4	65272	Transmission Fluids
5	64774	Direct Lamp Control Command 2
7	65213	Fan Drive
8	65237	Alternator Information
9	65252	Shutdown
10	64892	Diesel Particulate Filter Control 1

No	PGN	Signal Name
11	65128	Vehicle Fluids VF
12	65237	Alternator Information
13	65252	Shutdown
14	64998	Hydraulic Braking System
15	65089	Lighting Command
16	65274	Brakes 1
17	64586	SCR System Cleaning
18	64523	Electronics Engine Controller 20
19	64525	Fire Pump Statistics 1
20	64529	Total Gaseous Fuel Information

J1939 Add PGN Configuration

The first step would be for the user to add the new PGN's in the configuration file. The PGN configuration is available in the next structure which can be found in the file Middlewares/Third_Party/J1939/j1939 _ML/gen_pgn.c This defines generic PGB structure for CAN0 and CAN1.

```
J1939_CONFIG_T j1939Config_l0 = {
&jPgnTransmitConfig_l0[0],
CAN0_NUMBER_PGNS_TX,
&jPgnReceiveConfig_l0[0],
CAN0_NUMBER_PGNS_RX,
&addrTrCfg_l0,
J1939_CAN0_CLAIM_ADDRESS
};
```

```
J1939_CONFIG_T j1939Config_l1 = {
&jPgnTransmitConfig_l1[0],
CAN1_NUMBER_PGNS_TX,
&jPgnReceiveConfig_l1[0],
CAN1_NUMBER_PGNS_RX,
&addrTrCfg_l1,
J1939_CAN1_CLAIM_ADDRESS
};
```

The same file has the TX and RX structures where the user can include their own PGN's into the specific channel structure based on their requirement. Please note that for each PGN added, the minimum configuration that needs to be provided is

1. PGN Number
2. SPN Mapping Structure (with entire list of PGNs and SPNs)
3. Cycle time in milli seconds
4. Inhibit time in milli seconds
5. Priority
6. Number of mappings (The number of SPNs that PGN contains)

The next code snippet shows sample PGN definitions for the CAN channel 1.

```
/* TX PGN definitions */
static const J1939_PGN_CONFIG_T jPgnTransmitConfig_I0[CAN0_NUMBER_PGNS_TX] = {
{
65262u, /* PGN number */
&I0_tx_mapping_65262[0],
10000u, /* cycletime in ms */
0u, /* inhibit time in ms */
6u, /* priority */
6u /* number of mappings */
}
};
```

```
/* RX PGN definitions */
static const J1939_PGN_CONFIG_T jPgnReceiveConfig_I0[CAN0_NUMBER_PGNS_RX]= {
{
61444u, /* PGN number */
&rx_mapping_61444[0],
10u, /* cycletime in ms */
0u, /* inhibit time in ms */
3u, /* priority */
9u /* number of mappings */
},
```

J1939 Add SPN Configuration

For each PGN included by the user, the supported SPN structure will need to be included in the same file. The SPN definition structure will include the below information:

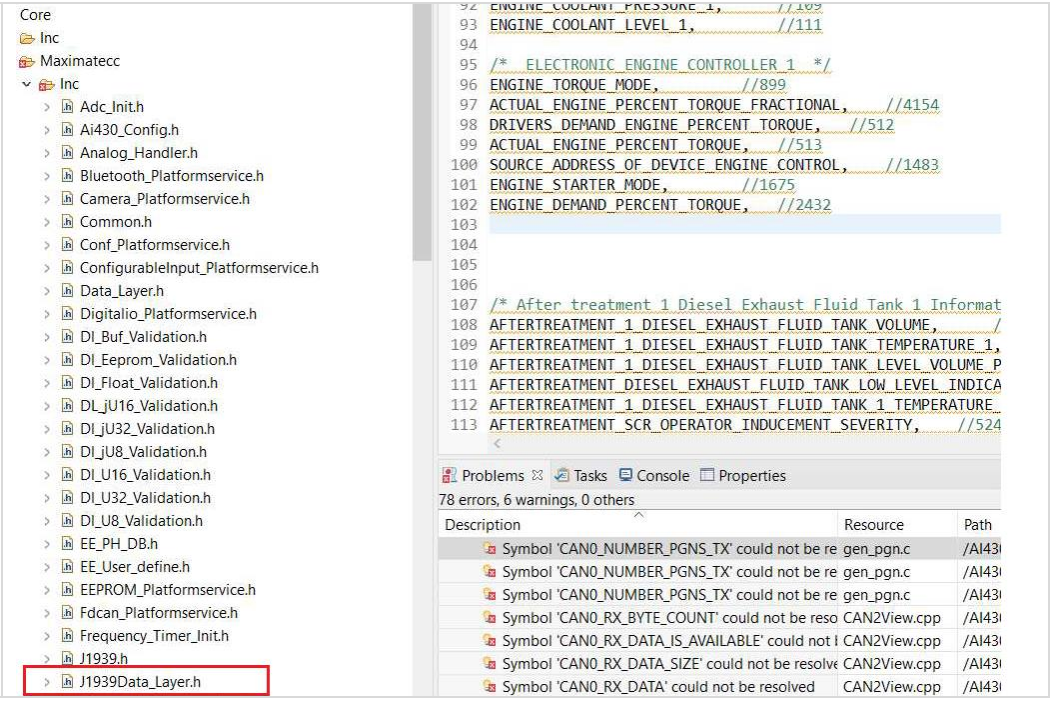
1. DB Buffer with index location
2. SPN Data Size
3. SPN Number
4. Data Type
5. If it's a dynamic PGN include the dynamic variable.

For example, we have the PGN 61444 included in the RX structure and hence the SPN's supported by 61444 are defined in the below code snippet:

```
/* +++PGN 61444/0xf004 EEC1 Electronic Engine Controller 1 */
static const J1939_MAPPING_T rx_mapping_61444[] = {
{ &dl_ju8[ENGINE_TORQUE_MODE - (START_J1939U8BIT + 1) ], 4u,
899u, J1939_DTYPE_U4 } /*
Engine Torque Mode */,
{ &dl_ju8[ACTUAL_ENGINE_PERCENT_TORQUE_FRACTIONAL -
(START_J1939U8BIT + 1)], 4u, 4154u,
J1939_DTYPE_U4 } /* Actual Engine - Percent Torque (Fractional) */,
{ &dl_ju8[DRIVERS_DEMAND_ENGINE_PERCENT_TORQUE -
(START_J1939U8BIT + 1)], 8u, 512u, J1939_DTYPE_U8
} /* Drivers Demand Engine - Percent Torque */,
{ &dl_ju8[ACTUAL_ENGINE_PERCENT_TORQUE -
(START_J1939U8BIT + 1)], 8u, 513u, J1939_DTYPE_U8 }
/* Actual Engine - Percent Torque */,
{ &dl_ju16[ENGINE_SPEED - (START_J1939U16BIT + 1)], 16u, 190u,
J1939_DTYPE_U16 } /* Engine Speed
*/,
{ &dl_ju8[SOURCE_ADDRESS_OF_DEVICE_ENGINE_CONTROL -
(START_J1939U8BIT + 1)], 8u, 1483u,
J1939_DTYPE_U8 } /* Source Address of Controlling Device for Engine
Control */,
{ &dl_ju8[ENGINE_STARTER_MODE - (START_J1939U8BIT + 1)], 4u,
1675u, J1939_DTYPE_U4 } /*
Engine Starter Mode */,
{ &mv_u8[0], 4u, 10001u, J1939_DTYPE_U4 },
{ &dl_ju8[ENGINE_DEMAND_PERCENT_TORQUE -
(START_J1939U8BIT + 1)], 8u, 2432u, J1939_DTYPE_U8 }
/* Engine Demand Percent Torque */
};
```

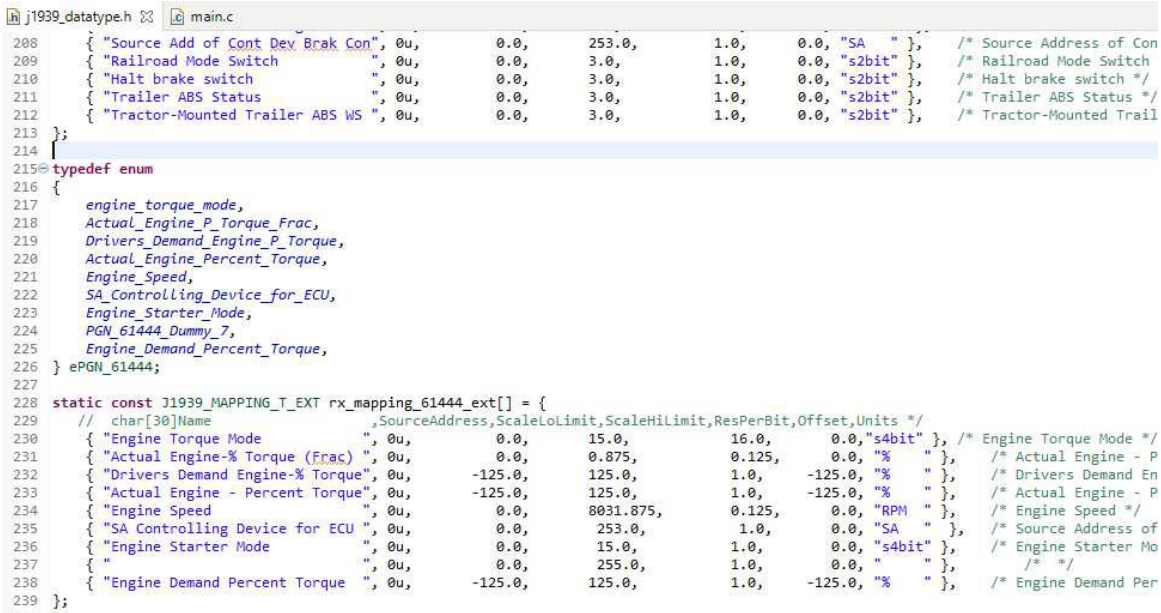
NOTE: For every PGN added, the user must put all the SPNs that the PGN contains on the structure in the same order that the J1939 SAE indicates and fill the empty bits for the frame as we made on the bottom example for the SPN 1001 (the SPN 1001 is only for reference purpose), because if we don't do it like that, the stack will report erroneous data, also the size of every SPN must be as the J1939 SAE indicates.

The #define for the SPN values can be added in the J1939Data_Layer.h. on the corresponding size variables, The path for this file and the defines are highlighted in the next image:



Translate the SPN's Raw Data to Real Value

Once the user has successfully defined the PGN's and added them in the required structure the SDK will take care of capturing them. Once the SPNs' are updated in the DB the user will need to use an API function to convert the raw value to real value and fill a structure where the user must put the necessary parameters to translate the specific SPN, for example, if we want to get the real value for every SPN of the PGN 61444, we need to do as follows:



The below code snippet shows a sample translation by SDK to support SPN conversion to real data, Engine speed SPN 190 is contained on PGN 61444

```
Get_DL(ENGINE_SPEED , (uint8_t *)&engine_speed);
float fengine_sp= ConvertReadableData(engine_speed,rx_mapping_61444_ext,Engine_Speed);
```

The above ENGINE SPEED PGN is the actual engine speed which is calculated over a minimum crankshaft and is at a resolution of 0.125 rpm per bit. It occupies 2 bytes of data and hence is stored in the dl_ju16 array. For any new PGN added by the user, a similar translation may be required based on the J1939 stack specification and PGN definition support.

Access the new SPN's from DB

The application user can directly access the DB variables of each SPN value that he has configured in the stack. The below code snippet shows the example for the same.

```
/* Get ENGINE_SPEED */
Get_DL(ENGINE_SPEED , (uint16_t*)&val);
/* Get ENGINEOILLEVEL */
Get_DL(ENGINEOILLEVEL, (uint8_t*)&val);
```

```
/* Get ENGINEOILPRESSURE */
Get_DL(ENGINEOILPRESSURE, (uint8_t*)&val);
/* Get COOLANTPRESSURE */
Get_DL(COOLANTPRESSURE, (uint8_t*)&val);
```

J1939 Diagnostic Message Configuration

The AI430 SDK supports the J1939 Diagnostic Messages (DM) which help the user understand the current state of the device.

J1939 DM1 and DM2 Support in SDK

The AI430 SDK supports the configuration of the DM1, DM2 messages. These messages are Already configured in the stack and the SDK DB has entries for each of these variables. Hence the user can directly access these SPN's from the application by accessing the below DB variables.

In the next image will list a table with the DM2. If user needs to use the messages for DM1 replace the Name DM1 instead of DM2

Field ID	Data Type	Permission	Size	Description	Comments
DM1_PROTECT_LAMP	DBu8	READ	1	Raw Data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_AMBER_WARN_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_RED_STOP_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_MAL_FUNC_IND_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_FLASH_PROT_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1__FLASH_AMBER_WARN_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_FLASH_RED_STOP_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_FLASH_MAL_FUNC_IND_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_FMI	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_SPN_CONV_METHOD	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM1_OC	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.

Field ID	Data Type	Permission	Size	Description	Comments
DM2_PROTECT_LAMP	DBu8	READ	1	Raw Data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_AMBER_WARN_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_RED_STOP_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_MAL_FUNC_IND_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_FLASH_PROT_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.

DM2__FLASH_AMBER_WARN_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_FLASH_RED_STOP_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_FLASH_MAL_FUNC_IND_LAMP	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_FMI	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_SPN_CONV_METHOD	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.
DM2_OC	DBu8	READ	1	Raw data from J1939	Raw data value from J1939 is updated into this DB variable.

The user will need to set the two variables from the Application to request the DM messages, shown in next code snippet:

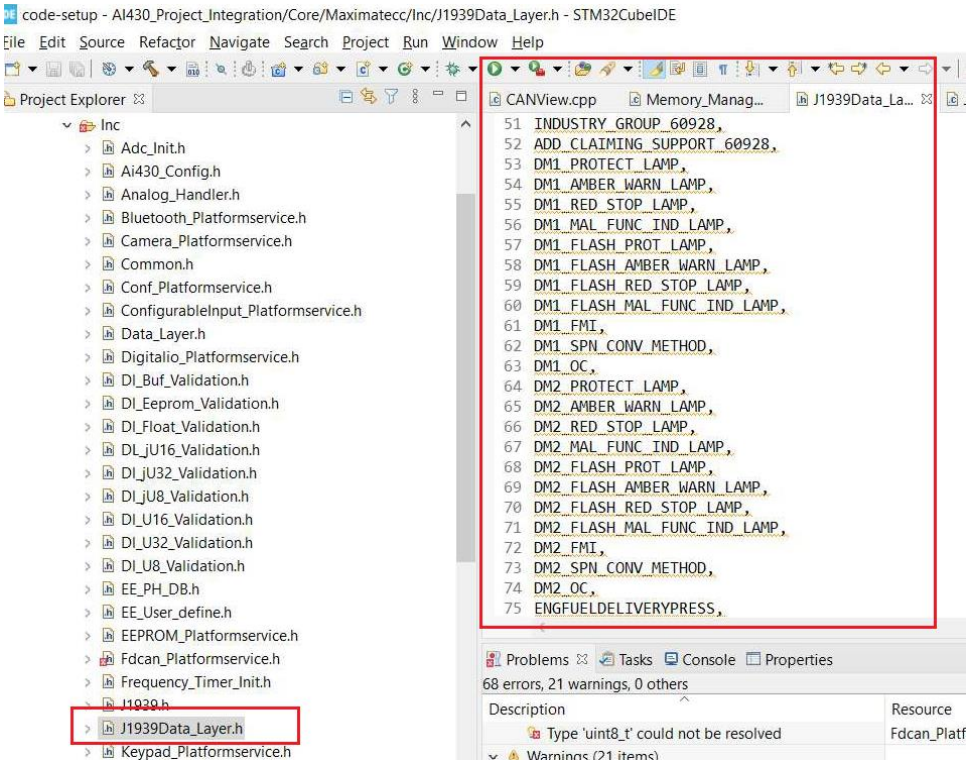
```
/* Set the CAN channel address and enable the DM1 messages */
val = 0x84;
Set_DL(CAN0_DEST_ADD, (uint8_t *)&val);
val = 1;
Set_DL(SEND_CAN0_DM1, (uint8_t *)&val);
/* Set the CAN channel address and enable the DM2 messages */
val = 0x84;
Set_DL(CAN0_DEST_ADD, (uint8_t *)&val);
val = 1;
Set_DL(SEND_CAN0_DM2, (uint8_t *)&val);
```

One the SDK receives the above messages, the DM packets will be process from the J1939 stack and then the values updated in the DB. The below code snapshot shows how the DM variables can be accessed from the TouchGFX application.

```
/* Get DM1 protect Lamp diagnostic data */
Get_DL(DM1_PROTECT_LAMP , (uint8_t*)&val);
```

J1939 Additional DM Support

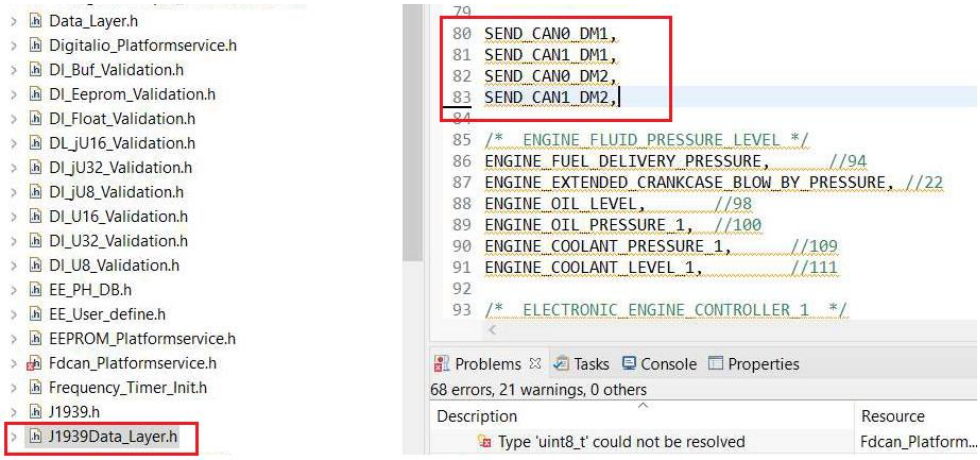
The SDK also supports the user to add support for additional DM messages. To do the same the user will have to configure the stack. The user can add the SPN definitions in the J1939Data_layer.h file as shown in the next snapshot.



For example, to add the support for DM2, the following #defines were added in this file

```
DM2_PROTECT_LAMP,
DM2_AMBER_WARN_LAMP,
DM2_RED_STOP_LAMP,
DM2_MAL_FUNC_IND_LAMP,
DM2_FLASH_PROT_LAMP,
DM2_FLASH_AMBER_WARN_LAMP,
DM2_FLASH_RED_STOP_LAMP,
DM2_FLASH_MAL_FUNC_IND_LAMP,
DM2_FMI,
DM2_SPN_CONV_METHOD,
DM2_OC,
```

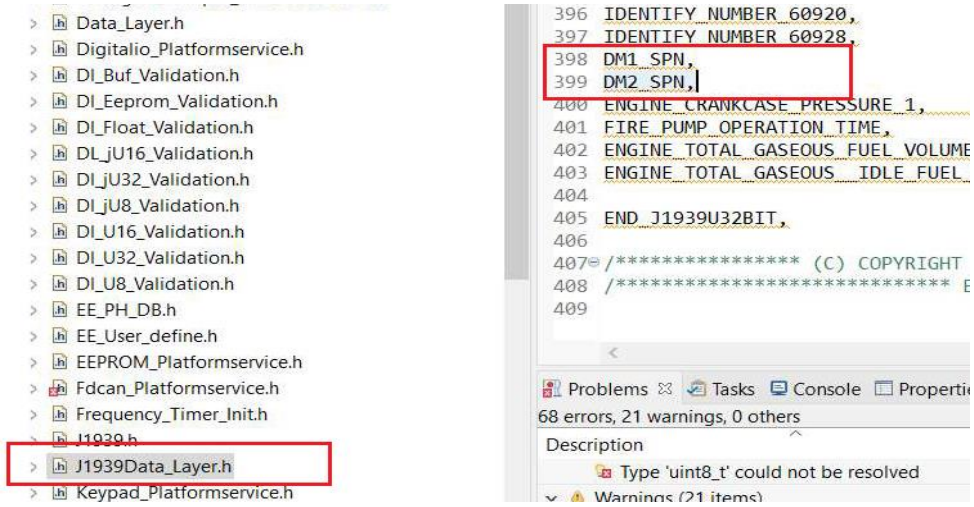
The user will need to add the below variables so that the user can use these variables to enable the DM from the UI.



For example, to add the support for DM2, the following #defines were added in this file:

```
SEND_CAN0_DM2,
SEND_CAN1_DM2,
```

The next step would be to add a 32-bit entry as shown in the below image so that the access to the DM variables can be enabled



The next step would be to enable it in the code. Please navigate to j1939.c as shown in the next image and add the details of the new DM message in the diagnosticReceive API as shown below. The code snippet shows the entries for DM1 and DM2 messages.

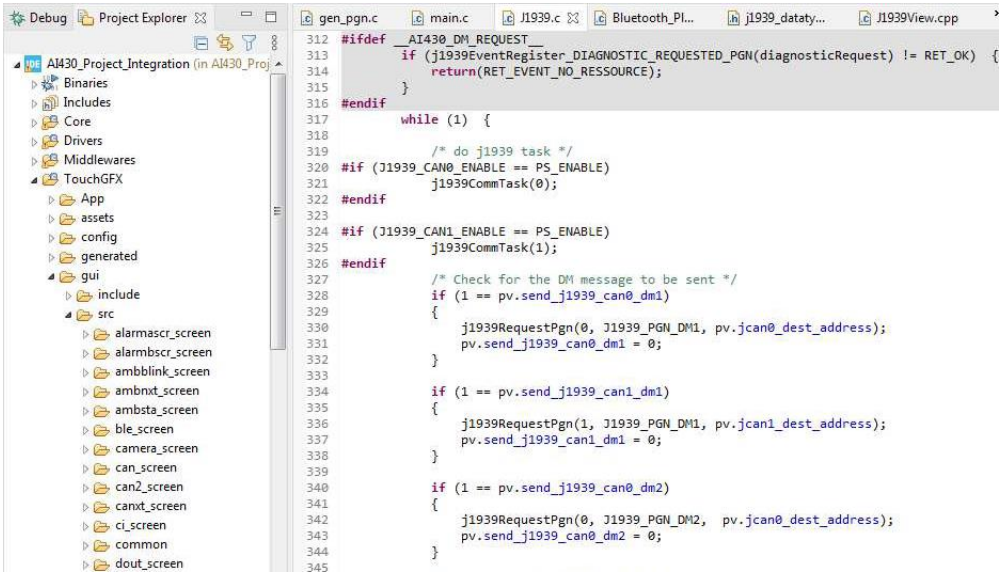
```
/*****************************************************************/
/*
* registered function for diagnostic requests
*/
static void diagnosticReceive(
UNSIGNED8 canLine,
UNSIGNED32 pgn, /* PGN requested */
UNSIGNED8 srcNode /* requested node */
)
{
&dl_ju32[DM1_SPN - (START_J1939U32BIT + 1)],
&dl_ju8[DM1_FMI - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_SPN_CONV_METHOD - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_OC - (START_J1939U8BIT + 1)]);
} while (retVal == RET_SERVICE_BUSY);
#if (EMTOS_J1939_DM2 == PS_ENABLE)
//j1939RequestPgn(canLine, J1939_PGN_DM2, 0x84);
#endif
}
#endif
```

```
RET_T retVal;
//UNSIGNED32 spn;
#if (EMTOS_J1939_DM1 == PS_ENABLE)
if (pgn == J1939_PGN_DM1) {
printf("DM1 received\n");
do {
retVal = j1939DiagnosticGet_DM1(canLine,
&dl_ju8[DM1_PROTECT_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_AMBER_WARN_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_RED_STOP_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_MAL_FUNC_IND_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_FLASH_PROT_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM1_FLASH_AMBER_WARN_LAMP - (START_J1939U8BIT
+
1)],
&dl_ju8[DM1_FLASH_RED_STOP_LAMP - (START_J1939U8BIT +
1)],
&dl_ju8[DM1_FLASH_MAL_FUNC_IND_LAMP -
(START_J1939U8BIT +
1)],
#if (EMTOS_J1939_DM2 == PS_ENABLE)
if (pgn == J1939_PGN_DM2) {
printf("DM2 received\n");
do {
retVal = j1939DiagnosticGet_DM2(canLine,
&dl_ju8[DM2_PROTECT_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_AMBER_WARN_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_RED_STOP_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_MAL_FUNC_IND_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_FLASH_PROT_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_FLASH_AMBER_WARN_LAMP - (START_J1939U8BIT +
1)],
&dl_ju8[DM2_FLASH_RED_STOP_LAMP - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_FLASH_MAL_FUNC_IND_LAMP - (START_J1939U8BIT +
1)],
&dl_ju32[DM2_SPN - (START_J1939U32BIT + 1)],
&dl_ju8[DM2_FMI - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_SPN_CONV_METHOD - (START_J1939U8BIT + 1)],
&dl_ju8[DM2_OC - (START_J1939U8BIT + 1)]);
} while (retVal == RET_SERVICE_BUSY);
}
#endif
}
```

Once this is enabled, the SDK will capture the new DM packets and store them in the DB. The TouchGFX user can access them from the DB as described in the previous section.

J1939 DM1 API Configuration

The below API support has been enabled in the J1939 file to be get the DM messages.



J1939 Sample Configuration

```
/*****************************************************************************************************************************************************************
                                                                * J1939 Module
*****************************************************************************************************************************************************************/
/*!
* J1939 Platform service (PS_ENABLE) / Disable(PS_DISABLE) Macros
*/
#define SDK_SERVICE_J1939 PS_ENABLE //PS_DISABLE
#if ((SDK_SERVICE_J1939 == PS_ENABLE) &&
(SDK_SERVICE_FDCAN == PS_ENABLE))
#undef SDK_SERVICE_FDCAN
#define SDK_SERVICE_FDCAN PS_DISABLE
#endif
#define J1939_CAN0_CLAIM_ADDRESS 23
#define J1939_CAN1_CLAIM_ADDRESS 85
/*!
* J1939_ADDRESS_CLAIM_DYNAMIC will search address when set to
1, Fixed = 0
*/
#define J1939_CAN0_ADDRESS_CLAIM_DYNAMIC 1
#define J1939_CAN1_ADDRESS_CLAIM_DYNAMIC 1
/*!
```

```
#if (SDK_SERVICE_J1939 == PS_ENABLE)
/*!
* J1939 Task Periodicity 100ms
*/
#define PS_J1939_TASK_TIMEOUT 100
/*!
* J1939 Task Priority
* osPriorityNone = 0,
* osPriorityIdle = 1,
* osPriorityLow = 8,
* osPriorityLow1 = 8+1,
* ,, ,,
* ,, ,,
* osPriorityISR = 56,
* osPriorityError = -1,
* osPriorityReserved = 0x7FFFFFFF
*/
#define PS_J1939_TASK_PRIORITY osPriorityIdle
/*!
* J1939_CLAIM_ADDRESS
*/
#define J1939_CLAIM_ADDRESS PS_ENABLE
/*
* State BLE J1939 Debug data
*/
#define BLE_J1939_DEBUG_DATA PS_ENABLE
#define J1939_CAN0_ENABLE PS_ENABLE
#define J1939_CAN1_ENABLE PS_ENABLE
/*!
* J1939_CLAIM_ADDRESS will be between 0 to 255
*/
* J1939_ADDRESS_CLAIM_NEXT_ADDRESS When dynamic set to 1, search will starting from
this value and up
*/
#define J1939_CAN0_ADDRESS_CLAIM_NEXT_ADDRESS 80
#define J1939_CAN1_ADDRESS_CLAIM_NEXT_ADDRESS 90
/*!
* J1939_CAN0_BITRATE will be between 0/250/500/1000
*/
#define J1939_CAN0_BITRATE 250
#define J1939_CAN1_BITRATE 250
/*!
* J1939 ENABLE/Disable DMx
*/
#define EMTOS_J1939_DM1 PS_ENABLE
#define EMTOS_J1939_DM2 PS_ENABLE
#define EMTOS_J1939_DM3 PS_ENABLE
//#define EMTOS_J1939_DM11 PS_ENABLE
/*!
* J1939 Set number of PGN'S to receive and transmit
*/
#define CAN0_NUMBER_PGNS_RX 17u
#define CAN1_NUMBER_PGNS_RX 3u
#define CAN0_NUMBER_PGNS_TX 1u
#define CAN1_NUMBER_PGNS_TX 1u
/*!
* J1939 Set COB handlers
*/
#define CAN0_CO_COB_COUNTS 35u
#define CAN1_CO_COB_COUNTS 21u
#endif //SDK_SERVICE_J1939
```

Through put module

The User would be able to use the below functionalities of the through put module via the DB variables and configuration file.

Through put module Enable/Disable

The SDK provides the user the ability to enable/disable the Through put functionality by modifying the default configuration file. Please see section Through put Sample Configuration.

No	Variables	Options	Default State	Description
1	THROUGH_PUT_SERVICE	PS_ENABLE PS_DISABLE	PS_ENABLE	PS_ENABLE: Enables the Through put module in the SDK. PS_DISABLE: Disables the Through put module in the SDK.

Through put maxAI 430 SDK Statistics

The user can also get the absolute time that the task has been executing (the total time that the task has been in the Running state), and the percentage time that shows essentially the same information but as a percentage of the total processing time rather than as an absolute time, for every Tasks during runtime using the below DB variables.

Field ID	Data Type	Permission	Size	Description	Comments
KEYPAD_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for keypad task
DIO_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for digital output task
CI_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for configurable inputs task
POWERMODE_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for power mode task
LIGHT_SENSOR_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for light sensor task
WARNINGLIGHT_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT time for warning lights task
LED_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for led task
POWER_MONITOR_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for power monitor task
USB_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for USB task
DEFAULT_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for default task
BLUETOOTH_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for Bluetooth task
RTC_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for RTC task

SW_TIMER_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for SW timer task
CAMERA_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for camera task
EEPROM_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for EEPROM task
WATCHDOG_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for watchdog task
LCD_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for LCD task
CAN_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for CAN task
J1939_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for J1939 task
TOUCHGFX_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for Touchgfx task
J1939CTRL_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for J1939 control task
IDLE_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for idle task
USERTASK1_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for user task 1
USERTASK2_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for user task 2
USERTASK3_ABSTIME	DBu32	READ	4	Absolute time (AT)	Variable used to get the AT for user task 3

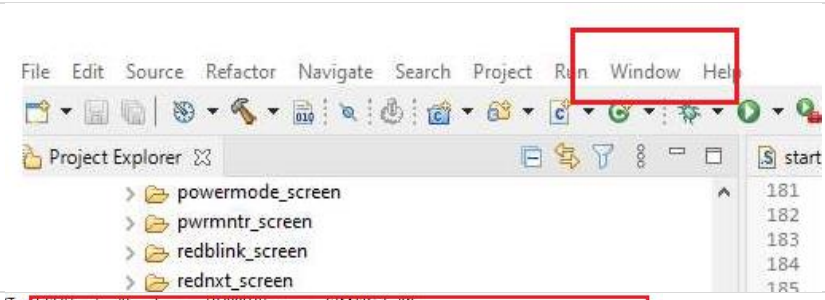
Field ID	Data Type	Permission	Size	Description	Comments
KEYPAD_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for keypad task
DIO_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for digital output task
CI_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for configurable inputs task
POWERMODE_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for power mode task
LIGHT_SENSOR_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for light sensor task
WARNINGLIGHT_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT time for warning lights task
LED_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for led task
POWER_MONITOR_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for power monitor task
USB_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for USB task
DEFAULT_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for default task
BLUETOOTH_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for Bluetooth task
RTC_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for RTC task
SW_TIMER_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for SW timer task
CAMERA_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for camera task
EEPROM_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for EEPROM task
WATCHDOG_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for watchdog task
LCD_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for LCD task
CAN_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for CAN task
J1939_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for J1939 task
TOUCHGFX_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for Touchgfx task
J1939CTRL_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for J1939 control task
IDLE_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for idle task
USERTASK1_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for user task 1
USERTASK2_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for user task 2
USERTASK3_PERTIME	DBu8	READ	4	Percentage time (PT)	Variable used to get the PT for user task 3

The below code snippet shows a sample code to get the abs time and percentage time from the DB.

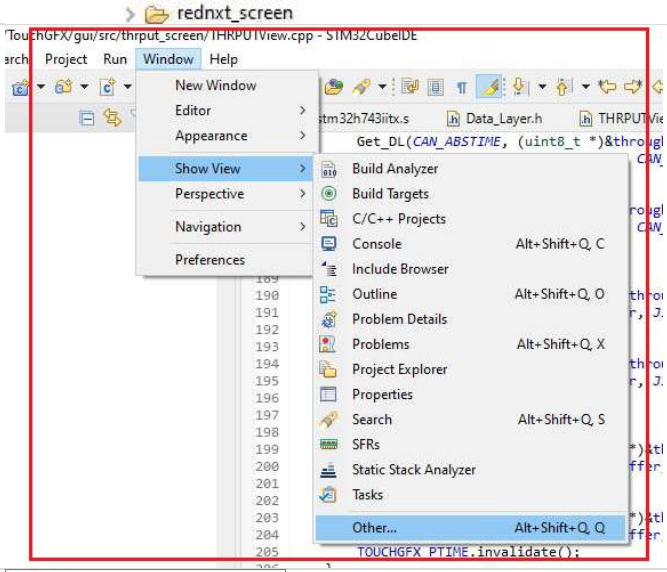
```
uint32_t throughput;
Count_val++;
if( 50 <= Count_val)
{
Count_val = 0;
throughput = 0;
Get_DL(KEYPAD_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(KEYPAD_PERTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(DIO_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(DIO_PERTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(CI_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(CI_PERTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(POWERMODE_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(POWERMODE_PERTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(LIGHT_SENSOR_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(LIGHT_SENSOR_PERTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(WARNINGLIGHT_ABSTIME, (uint8_t *)&throughput);
throughput = 0;
Get_DL(WARNINGLIGHT_PERTIME, (uint8_t *)&throughput);
```

Through put stm32CubeIDE Statistics

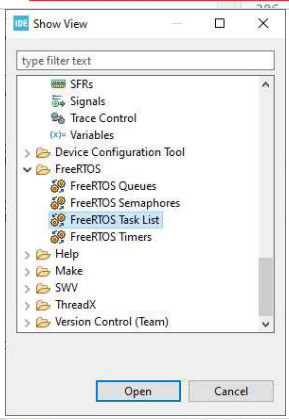
To be able to get the run time statistics from the stm32CubeIDE, the user needs to select window at the bottom of the IDE:



Then select, show view, other



Finally, select FreeRTOS, then FreeRTOS Task List.



When the user runs your application, the user should see the run time statistics as follows

Name	Priority (Base/...	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Run Time (%)
Camera	1/1	0x24006560	0x24006cac <...	DELAYED		N/A	0%
EE_Service	1/1	0x240022a8	0x24002a04 <...	DELAYED		N/A	0%
IDLE	0/0	0x24000580	0x24000924 <I...	RUNNING		N/A	99%
J1939Control	1/1	0x24007740	0x24007ea4 <...	DELAYED		N/A	0%
J1939Task	1/1	0x24006dd0	0x24007524 <...	BLOCKED	0x2400a1ec <...	N/A	<1%
Keypad	1/1	0x24002c50	0x240033ac <...	SUSPENDED		N/A	0%
LCD	1/1	0x24003f68	0x240046cc <...	DELAYED		N/A	0%
PM	1/1	0x240048d8	0x2400503c <...	DELAYED		N/A	0%
RTC	1/1	0x240035f8	0x24003d4c <...	DELAYED		N/A	0%
SWTimer	1/1	0x24005248	0x240059ac <...	DELAYED		N/A	0%
ThroughPut	1/1	0x24007fb0	0x24008724 <...	DELAYED		N/A	<1%
Tmr Svc	2/2	0x240009e0	0x24001164 <T...	BLOCKED	TmrQ	N/A	0%
USB_Service	1/1	0x24005bf0	0x24006334 <...	DELAYED		N/A	0%

Through put Sample Configuration

/**********

```
*
                                                    * Through Put Service
*
*****************************************************************************************************************************************************************/
/*!
* Through Put service (PS_ENABLE) / Disable(PS_DISABLE) Macros
*/
#define THROUGH_PUT_SERVICE PS_ENABLE
#if (THROUGH_PUT_SERVICE == PS_ENABLE)
#endif //THROUGH_PUT_SERVICE
```

Application Details

The maxAI 430 SDK S/W release package includes a sample application per module which demonstrates the functionalities of the modules and can be used for reference by the user to understand how to interact with the SDK.

It also includes a graphical demo application which can be used as a reference to understand how to use multiple modules in a single application and tie them to various graphical UI elements that are available in the touch GFX screen.

Sample Application Project Details

This sample application per module gives you a walk-through of the test procedure for each module available. The sample application can be used as a basis to understand what functionalities are available in each module and how the user can interact with the individual modules.

This section includes a brief description about sample application user interface and the minute details of each module, which includes module description, module screen navigation and the test procedures.

Introduction

To open the sample application project please follow the same procedure followed to open the blank project file as described in Section 3.

Once you have successfully compiled and flashed the Sample project on the AI430 hardware, you can reboot the hardware to run the application. Initially after the device is turned ON, A Home Screen will be displayed on the LCD screen which contains the list of all the available modules. The Home Screen sample image is given as follows:	

Home Screen Navigation

In the Home Screen you can find the list of all the modules available in the AI430 SDK i.e., Keypad, Light Sensor, Power Monitor, RTC, EEPROM, LCD, Digital o\p, WLT, USB, SW Timer, Config input, LED, BLE, Camera, Power Mode, J1939, Flash, through put.
Along with the modules there are four key navigators (i.e., Previous, Next, Enter, Back) which will allow the user to move front, back, up and down, enter into a specific module and exit from the specific module. The four key navigators are operated using the four built-in buttons, which are located at the bottom end and are represented as Key1, Key2, Key3, and Key4 respectively. Each button has a specific functionality which is mentioned in below table.

Button Name	Functionality	Button Name	Functionality
Key 1	Go to Previous	**Key 3**	Enter/Select
Key 2	Go to Next	**Key 4**	Back

Keypad Module

The keypad module sample application is shown in the next screen:

Module Description

This Keypad module is basically designed to test the functioning of all the four keys present on the device. The testing can be done for each key (i.e., All the four keys) to check whether it's working properly based on their specific functionality.

Module Navigation

To go to the Keypad module, In the Home Screen navigate to the keypad module using Key 1 and Key 2 and then select the keypad using Key3 i.e., Enter, which will take you to Keypad Functional screen.
On this Screen you will find four Keypad options i.e., Keypad1 to Keypad4. Each Keypad consists of two Blocks below them, where one block is used for short press test update and the other Block is used for long press test update.

Short Press denotes a single click on the key.
Long Press denotes click and hold the key.

Module Test Procedure

Test Case	Keypad Action	Description	Test Procedure	Results
KeyPad 1	Short Press	This key is used to turn ON the Backlight which are present on the keys	Click on key1 and check if the light on all the keys turns ON.	Light on the button Glows
KeyPad 1	Long Press	This key is used to turn OFF the Backlight which are present on the keys	Click and Hold Key1. Then check if the light on all the keys turns OFF.	Light on the button Turns OFF
KeyPad 2	Short Press	This is specifically used to check whether the key is functioning properly.	Click on Key2 and then check if the short press block below key2 gets highlighted	Keypad2 Short press block present on the LCD will be Highlighted
KeyPad 2	Long Press	This is specifically used to check whether the key is functioning properly.	Click and Hold on Key2.Then check if long press block below key2 gets highlighted	Long press block present on the LCD will be Highlighted
KeyPad 3	Short Press	This is specifically used to check whether the key is functioning properly.	Click on Key3 and then check if the short press block below key3 gets highlighted	Short press block present on the LCD will be Highlighted
KeyPad 3	Long Press	This is specifically used to check whether the key is functioning properly.	Click and Hold on Key3.Then check if long press block below Key3 gets highlighted	Long press block present on the LCD will be Highlighted
KeyPad 4	Short Press	This is specifically used to check whether the key is functioning properly.	Click on Key4 and then check if the short press block below key4 gets highlighted	Short press block present on the LCD will be Highlighted

KeyPad 4	Long Press	This key is used to Exit from the Keypad Functionality screen and go to Home Screen	Click and Hold on Key3.Then check if the Screen exited from Keypad functionality and goes to Home screen.	Back/Exit

Light Sensor

The light sensor module sample application is shown in the below screen,

Light Sensor Module

The Light Sensor module sample application is shown in the next screen:

Module Description

The light sensor module is basically designed to test the Light intensity which is observed by providing external light on to the light sensor which is present on the bottom right corner of the device.

Module Navigation

To go to the Light Sensor test screen, from Sample application screen navigate to the Light Sensor block using the Key1 and Key2. And then using Key3 enter into the Light Sensor test. This screen consists of three main divisions, they are Conversion Time, Conversion Mode, and LS value. Where, Conversion Time: There two conversion time supported by the SDK are:

100 | 800

This time ranges are used to give the sensor outcome based on the specified range.

Conversion Mode: There are three different modes supported by the SDK. They are:

LS Shutdown | Single Shot Mode | Continuous Mode

LS Value: LS value is the Light Sensor value which is calculated and updated based on two factors they are:
1) The external light which is provided at the light sensor that is located at the bottom right corner of the device.
2) Conversion Time and Conversion Mode

Module Test Procedure

Conversion Mode	Conversion Time	Description	Test Procedure	Results
LS Shutdown	100	This functionality is used to Shutdown the light sensor for 100ms	Using Key1 navigate to Conversion time 100 then Use Key3 (Enter) to set. Then again using Key1, select LS shutdown mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result.	LS value will be 0 as this is in Shutdown mode.
LS Shutdown	800	This functionality is used to Shutdown the light sensor for 800ms.	Using Key1 and Key2 navigate to Conversion time 800 then Use Key3	LS value will be 0 as this is in Shutdown mode.

			(Enter) to set. Then again using Key1, select LS shutdown mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result.	
Single Mode Conv.	100	This functionality is used provide the LS value for one single time after 100ms.	Using Key1 navigate to Conversion time 100 then Use Key3 (Enter) to set. Then again using Key1 and Key2 select Single Mode Conv. mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result	LS value will get updated once after 100ms of external light is being projected on to the sensor.
Single Mode Conv.	800	This functionality is used provide the LS value for one single time after 800ms.	Using Key1 and Key2 navigate to Conversion time 800 then use Key3 (Enter) to set. Then again using Key1 and Key2 select Single Mode Conv. mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result	LS value will get updated once after 800ms of external light is being projected on to the sensor.
Cont Mode Conv.	100	This is functionality is used to continuously update the LS Value for every 100ms.	Using Key1 and Key2 navigate to Conversion time 100 then use Key3 (Enter) to set. Then again using Key1 and Key2 select Cont Mode Conv. mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result.	LS value will get updated for every 100ms based on the external light projected on the sensor.
Cont Mode Conv.	800	This is functionality is used to continuously update the LS Value for every 800ms.	Using Key1 and Key2 navigate in Conversion time to 800 then use Key3 (Enter) to set. Then again using Key1 and Key2 select Cont Mode Conv. mode and set using Key3 (enter). Now provide the external light on the Sensor and check the result.	LS value will get updated for every 100ms based on the external light projected on the sensor.

Power Monitor

The power monitor module sample application is shown in the next screen:

Module Description

This Power Monitor test is basically used to provide the battery level and ignition status updates

Module Navigation

To go to Power Monitor Test screen, from Sample Application Screen navigate to Power Monitor block using Key1 and Key2, then by using Key3 enter into the Power Monitor Test Screen In Power Monitor Test screen there are 2 blocks available they are:

Ignition State | IGN_STATE | Battery Level | BAT_LEVEL

Module Test Procedure

Test Case	Description	Test Procedure	Results	Units
IGNITION STATE	This functionality is used for testing whether the external device supply is ON or OFF.	For checking ignition value, the user will need to provide some external supply. And then by using Key1 navigate to IGN_STATE and then click on Key3 to test	If the external supply provided is turned On then it will give ON or else OFF	-
BATTERY LEVEL	This functionality updates the battery level of the device.	Using Key1and Key2 navigate to BAT_LEVEL then click on Key3 (Enter). Now check the result that is displayed at the blank space.	The battery level of the device will be displayed.	mv

RTC

The RTC module sample application is shown in the next screen:

Module Description

RTC is the real time clock which supports the RTC configuration for the AI430 board and also supports two alarms.

Module Navigation

To go to RTC, from “Sample Application” navigate to RTC using key1 and Key2 then click on key3 to enter into RTC screen.

RTC has the three blocks as given below:

RTC Time | ALARM_A | ALARM_B

RTC Time	ALARM_A	ALARM_B

The RTC screen consist of HR: MIN:SEC: WKDAYS:DD:MM: YY blocks which is used to set the Hour's, Minutes, Seconds, Weekdays, Date, Month, Year respectively using the Key1 and key4 Short Press. And to display it on the main Screen of RTC long press Key4.	The ALARAM_A screen is routed by selecting ALARAM_A from RTC main screen. This screen consists of HR:MIN:SEC:WK_DAY, which are again for Hour's, Minutes, Seconds, Weekdays respectively and can be set using the Key1 and key4 Short Press. And there is an empty block available below which Displays ALARM_OCCURED message when ALARM_A is triggered in the platform.	The ALARAM_B screen is routed by selecting ALARAM_B from RTC main screen. This screen consists of HR: MIN:SEC: WK_DAY, which are again for Hour's, Minutes, Seconds, Weekdays respectively and can be set using the Key1 and key4 Short Press. And there is an empty block available below which Displays ALARM_OCCURED message when ALARM_B is triggered in the platform.

Module Test Procedure

Test Case	Description	Test Procedure	Results
RTC Time	This functionality is used to set the time of the device.	Using key1 navigate to RTC Time block and then enter using key3.Now in the Sub Screen, by using Key1 set all the required details then by a short press on key4 you can set the time and later give a long press exit from the present screen and go to RCT main screen.	The updated time will be displayed on the screen.
ALARM_A	This functionality is used to set alarm on the device.	Using key1 navigate to Alarm_A block and then enter using key3.Now in the Sub Screen, by using Key1 set all the required details and then by a short press on Key4 set the alarm.	When the alarm gets triggered in the platform the empty Block under ALARM_A will Display ALARM_OCCURED Message.
ALARM_B	This functionality is used to set alarm on the device	Using key1 navigate to Alarm_B block and then enter using key3.Now in the Sub Screen, by using Key1 set all the required details and then by a short press on Key4 set the alarm.	When the alarm gets triggered in the platform then the empty Block under ALARM_B will Display ALARM_OCCURED message.

LCD

The LCD module sample application is shown in the next screen:

Module Description

LCD functionality is basically designed to check if the screen is functioning properly. The verification is done based on two factors one is by turning ON and OFF the screen and the other by increasing or decreasing the brightness of the screen.

Module Navigation

To go to LCD test, from “Sample Application” screen select LCD using Key1 and Key2 and later enter into the LCD Test Screen by using Key3. In the LCD Test there is a session which is used to update the brightness value of the display.

Module Test Procedure

Test Case	Keypad Action	Description	Test Procedure	Results
Brightness Value	Increase	This functionality is used to increase the brightness of the Device	Click on key1(Brightness+) and check the results	The brightness value would increase from the previous value
Brightness Value	Decrease	This functionality is used to decrease the brightness of the Device	Click on key2(Brightness-) and check the results	The brightness value would decrease from the previous value
LCD ON\OFF	-	This functionality is used to Turn ON/OFF the LCD Screen.	Click on the ON\OFF button given on the LCD Screen.	The LCD display will turn ON in case of ON and Turn Off in case of OFF.

Digital Output

The digital output module sample application is shown in the next screen:

Module Description

There are two digital pins in the device Digital High and Digital Low. This Digital Output module is used to set this pin values

Module Navigation

To go to the Digital Output screen, from “Sample Application” Screen select Digital Output using Key1 and Key2, then enter the Digital Output screen using key3.On the Digital output screen there are 4 blocks they are as given below:

High Side	Low Side	Status ON	Status OFF

There are three configurable modes available in Digital output they

Mode	High pin	Low pin
High side	1	0
Low side	0	1
Open Drive	0	0

Module Test Procedure

Test Case	Description	Test Procedure	Results
Status Off	This functionality is used to set the system into Open Drive configuration mode.	Using Key1 and Key2 navigate to tatus OFF block and click on Key3. Now check the results.	The system will go to Open Drive state which means the high pin and low ping will be low.
Status ON	This functionality is used to set the Digital Output pin state to the previous state which was present before the system was set as OFF status.	Using Key1 and Key2 navigate to Status OFF block and click on Key3. Now check the results.	The pin status gets updated according to previous state.
High Side	This is used to show what pin state the system is operation on at present.	This will be tested along with status ON/OFF.	High side block in the display will get highlighted and the high pin will be 1 and low pin will be 0.
Low Side	This is used to show what pin state the system is operation on at present.	This will be tested along with status ON/OFF.	Low Side block in the display will get highlighted and low pin will be 1 and high pin will be 0.

Warning Light

The Warning light module sample application is shown in the next screen:

Module Description

As the name suggest the warning light module is used to provide warning signals form the device. There are a total of 20 warning signals available in our device which are from WI1 to WI20.

Module Navigation

To go to Warning light, from “Sample Application” screen navigate to WTL using Key1 and Key2 then enter into the warning light screen using Key3.In the warning light screen there are 20 functionalities available for all the 20 warning lights and it is shown in the below image. The User can use Key1 and Key2 to navigate to any one of the warning lights then by clicking on Key3 user can redirect to the next Screen.

Sub Screens

Warning light consists of three sub screens based on the functionality; each screen description is explained briefly below: Sub Screen_1(Warning light screen2): After selecting one of the warning light the second screen will be opened, in this screen there will be two blocks available they are as given below:

ON/OFF BLINK

PWM Screen: When ON/OFF button from Screen 2 is selected the PWM Screen will be displayed.

This screen provides the update as to whether the warning light is ON or OFF.

Sub Screen_3(BLINK): This screen is displayed when the user Selects BLINK option in Screen2.This screen provides the update whether the Blink option is ON or OFF and also at what range warning lights are made to blink.

Module Test Procedure

Test Case	Description	Test Procedure	Results	Range
ON/OFF	This functionality is used to turn ON/OFF the warning lights on the device.	Using Key1 and Key2 navigate to ON/OFF then click on Key3. Now it enters to PWM screen where using Key1(ON/OFF) we can turn ON or OFF the warning lights, and also, by using Key3 and Key4 we can increase or decrease the brightness of the warning lights.	Warning light status and its PWM value will be updated on PWM screen.	(0-100)
BLINK	This functionality is used to blink the warning lights	From main screen, select which warning light must blink next choose BLINK using Key1 and Key2 then enter Sub Screen_3 using Key3.Then user can set the BLINK to ON or OFF mode using Key1 and set the Blink speed using Key2 and Key3.	The selected warning light will blink at the specified speed.	(0-65535)

USB

The USB module sample application is shown in the next screen:

USB Module Description

The USB module is basically designed to provide an interaction between the specific modules and the Connected USB device. This module is implemented using two functionalities, they are as given below:

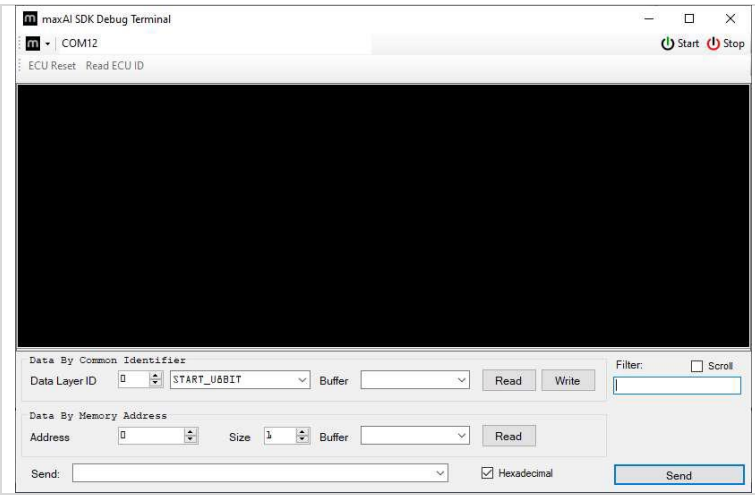
USB GUI Terminal | USB Send/Recv

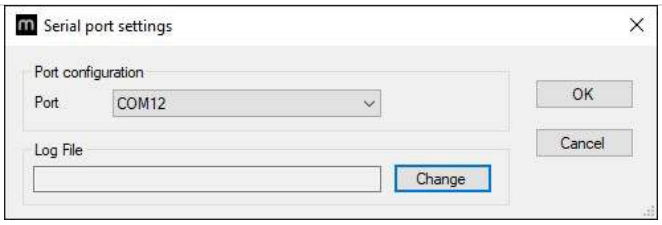
USB GUI TERMINAL

The USB terminal functionality is used to connect the maxAI 430 PC Tool and configure the device but passing the commands like START, STOP, ECU Reset, Common identifier and memory address.

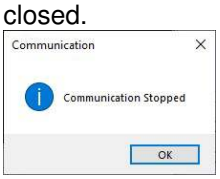
GUI Module Navigation

This test is done in maxAI SDK Debug Terminal which is present in the system that is externally connected to the device	Select the following menu and click on [Settings]





Module Test Procedure (Main Form)

Test Case	Description	Format	Test Procedure	Results	Permission
Start	Used to send a start message to open communication to GUI terminal		Click on [Start] to submit a communication start command. Command breakdown: Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x01 Frame[4]=SID=startCOM=0x81 Frame[5]=CheckSum=0x53	if communication start command is recognized a Popup will confirm it.  Response breakdown: Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x03 Frame[4]=SID=startCOM+0x40=0xC1 Frame[5]=VersionHi=0x00 Frame[6]=VersonLo=0x12 CheckSum=0xA7	Write
Stop	Used to send a stop message to close communication to GUI Terminal	Click on [Stop] to submit a communication stop command	Click on [Stop] to submit a communication stop command. Command breakdown: Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x01 Frame[4]=SID=endCOM=0x82 Frame[5]=CheckSum=0x54	communication to terminal is closed.  Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x03 Frame[4]=SID=endCOM+0x40=0xC2 Frame[5]=CheckSum=0x96	Write

ECU Reset	This button is used to reset controller board.	Click on [ECU reset]	Click on [ECU reset] Command breakdown: Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x01 Frame[4]=SID=ECUreset=0x11 Frame[5]=CheckSum=0xE3	The system will go to reset mode to restart	Write
		Data By Common Identifier			
Common Identifier	This functionality is used to reference a specific module data layer ID		Set a Data layer ID number and click [Read] Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x03 Frame[4]=SID=readDataCI=0x22 Frame[5]=ValueHi=XXXX Frame[6]=ValueLo=XXXX Frame[5]=CheckSum=0xSS	The specified data layer ID will be gotten and displayed. Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x0X Frame[4]=SID=+Resp=0x62 Frame[5]=1stValue=XX Frame[6]=2ndValue=XX Frame[N]=CheckSum=0xSS	Read
			Set a Data layer ID number and click [Write] Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x03 Frame[4]=SID=readDataCI=0x2E Frame[5]=ValueHi=XXXX Frame[6]=ValueLo=XXXX … Frame[5]=CheckSum=0xSS	The specified data layer ID will be set and updated. Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x0X Frame[4]=SID=+Resp=0x6E Frame[5]=1stValue=XX Frame[6]=2ndValue=XX … Frame[N]=CheckSum=0xSS	Write
		Data by Memory Address			
Address	This functionality is used to reference a memory address to read/write a memory section area.		Set a memory address and length to get a buffer from controller, click: [Read] Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1	Memory address section will be displayed. Frame[0]=Mode=0x80 Frame[1]=DestAddress=0x60 Frame[2]=ToolAddress=0xF1 Frame[3]=Size=0x06 Frame[4]=SID=+Resp=0x63 Frame[5]=1stValue=XX Frame[…]=2ndValue=XX Frame[N-1]=2ndValue	

			Frame[3]=Size=0x06 Frame[4]=SID=readData CI=0x23 Frame[5]=ValueHi=XX Frame[…]=ValueLo=X X Frame[N-1]=ReqSize Frame[N]=CheckSum=0 xSS	Frame[N]=CheckSum=0xSS	
			Open Communication Data to Controller		
Send	This functionality is used to send any typed data SEX/ASCII to controller		Type “80 60 F1 03 22 00 01 F7” and Click [Send]	This command will request a Data layer variable: “WARNING_LIGHT_01_STATE ” also is possible to copy/paste other commands from terminal to use as reference to type other commands.	Read

USB Functionality

This USB functionality is used to check if the packet sent from the external device is received.

The packet which is sent from externally connected device is in Hexadecimal format which is received in string format to controller.

Module Navigation

To go to USB Testing screen, from “Sample Application” screen navigate to USB block using Key1 and Key2, then enter into USB screen using Key3. In the USB Testing screen there are two functionalities available they are

Send Pckt | Rx Pckt

Module Test Procedure

Test Case	Description	Test Procedure	Expected Results
Send Pckt	This functionality is used to display the packet which has been sent form the USB terminal	When the packet is transferred from the USB terminal it will be updated at the Send Pckt space	Sent packet data will be updated.
RX Pckt	This functionality is used to display the packet which has been Received from the USB terminal	When the packet is transferred from the USB terminal it will be updated at the Send Pckt space	Received packet data in String form

Software Timer

The USB module sample application is shown in the next screen:

Module Description

The software timer is mainly designed to interact with the timer module. This will be consisting of timer reset, timer trigger and also to read the current timer counter value. There are a total of six timers available from Software Timer1 to Software timer6.

Module Navigation

To go to Software Timer, from “Sample Application” screen navigate to Software Timer Using Key1 and Key2, later click on Key3 to enter to Software Timer Test screen. In this screen there will be six blocks available which are used for six different timers.

Sub Screen

Sub Screen_1: From Software test screen when a timer is selected it will be redirected to Sub Screen_1 which has a name of the related Timer number. In this screen there are two blocks available they are:

Single Shot | Inactive

Module Test Procedure

Test Case	Keypad Action	Description	Test Procedure	Expected Results
Single/ Inactive	Short Press	This functionality is used to stop the timer	User can navigate to the timer using Key1(NEXT) or Key2(PREVIOUS) and can select the any one timer by pressing Key3(ENTER), user can go back to previous screen using Key4(BACK). Once the user navigates to the SW Timer 1, and press the Key3(ENTER), SWTIMER1 GUI Screen	Timer gets stopped

			appears, where user can set the timer according to the requirement.	
Continuous/ Active	Long Press	This functionality is used to set single shot timer and inactive state.	User can navigate to the timer using key1(NEXT) or key2(PREVIOUS) and can select the any one timer by pressing key3(ENTER), User can go back to previous screen using Key4 (BACK). Once the user navigates to the SW Timer 1, and press the key3 (ENTER), SWTIMER_1 GUI screen appears, where user can set the timer according to the requirement.	Timer will be started.

Configurable Inputs

The next screen shot shows the configurable input screen:

Module Description

The Configurable input are basically designed to configure the various input channels and then read the latest Configurable Input values according to the initial configuration done. The channels which are available for configuring are voltage, current, resistance, digital STB, digital STG, resistance, and frequency. The maxAI 430 supports 6 configurable inputs.

Module Navigation

To go to configurable Input, from “Sample Application” screen navigate to Config Input block using Key1 and Key2, now using Key3 enter to Configurable Input Test Screen. In Configurable Inputs test screen, there are six blocks available which has six AI values from AI1 Value to AI6 Value. This block represents different input channels they are as given below:

Voltage | Digital STG | Frequency
Digital STB | Resistance | Currency

From AI1 to AI5 the configurable inputs can be assigned to below mentioned channel

Voltage 32v | Frequency | Digital STG
Voltage 6v | Resistance | Digital STB

A16 configurable inputs can be assigned with:

Voltage 6v | Current

Module Test Procedure

Test Case	Description	Test Procedure	Expected Results
AI1 to AI5	AI1 to AI5 are values which are assigned to the configurable input channels 1 to 5 that are available for configurable inputs Functionality.	Using Key1(Long press) navigate to the specific block from AI1 to AI5 then by using Key1(Short press select the desired channel) and then check UI above the channel for the updated value based on the configuration. If you navigate to AI1 you will be able to configure it as resistance, voltage low and high, frequency, digital input. For example if you configure it as resistance, you will see the input resistance value on the screen.	The AI value of the specified channel will be displayed on the Screen.
AI6	AI6 is also a value which is assigned to the channels that are available for configurable inputs functionality.	Using Key1(Long press) navigate to the specific block from AI6 then by using Key1(Short press select the channel). If you navigate to AI6 you will be able to configure it as voltage 6v and current. For example, if you configure it as current, you will see the input current value on the screen.	The AI value of the specified channel will be displayed on the screen.

LED

Module Description

The LED functionality is designed to test the LED light which is In-Built in the device. The LED testing can be done in three modes they are:

Turn On | Turn Off | Blink

The maxAI 430 supports 2 LEDs. They are:

RED | AMBER

Module Navigation

To go to LED screen, from “Sample Application” screen navigate to LED using Key1 and Key2.Later enter into the LED test screen using Key3. In this screen there are two functionalities available, please see the first image below:

Sub Screen

The two functionalities which are available in LED test screen, are again having individual sub screen, they are explained in detail below:

Sub Screen_1: When the user selects from RED LED and AMB LED, Sub Screen_1 will be opened which again has two functionalities, they are:

Sub Screen_2: When the user selects the ON and OFF functionality then Sub Screen_2 will be displayed on this screen it updates if the LED is ON/OFF.

Sub Screen_3: When the user selects the Blink functionality then Sub Screen_3 will be displayed on this screen it updates if it's turned On and at what speed will the LED blink.

Module Test Procedure

Test Case	Mode	Description	Test Procedure	Expected Results
RED LED	ON/OFF	This functionality is used to Turn ON the RED LED present on the Bottom right Corner of the device	From the main Screen (LED test Screen) select RED LED Using Key1 and Key2, user will be navigated to Sub Screen_1 there select ON/OFF using Key1 and Key2, Now the user will be redirected to Sub Screen_2, now using Key1 turn On/OFF the LED based on requirement.	If the User Selects ON mode, then the LED light Glow's. And if the user selects OFF mode then the LED light which was glowing will turn OFF.
RED LED	BLINK	This functionality is used to blink the LED.	From the main Screen (LED test Screen) select RED LED Using Key1 and Key2, user will be navigated to Sub Screen_1 there select Blink using Key1 and Key2, Now the user will	If the User selects the Blink On mode, then the LED starts blinking at the specified rate.

			be redirected to Sub Screen_2, now using key2 and Key3 increase or decrease the blink speed and Using Key1 set Blink ON/OFF.	
AMB LED	ON/OFF	This functionality is used to Turn ON the AMB LED present on the Bottom right corner of the device.	From the main Screen (LED test Screen) select AMB LED Using Key1 and Key2, user will be navigated to Sub Screen_1 there select ON\OFF using Key1 and Key2, Now the user will be redirected to Sub Screen_2, now using Key1 turn On/OFF the LED based on requirement.	If the User Selects ON mode, then the LED light Glow's. And if the user selects OFF mode then the LED light which was glowing will turn OFF.
AMB LED	BLINK	This functionality is used to blink the LED.	From the main Screen (LED test Screen) select AMB LED Using Key1 and Key2, user will be navigated to Sub Screen_1 there select Blink using Key1 and Key2, Now the user will be redirected to Sub Screen_2, now user can increase or decrease the blink speed and Using Key1 set Blink ON/OFF	If the User selects the Blink On mode, then the LED starts blinking at the specified rate.

Power Mode

Module Description

The Power Mode Module is basically designed to check the low power functional mode of the device. There are three different functionality modes available for Power Mode Module which indicates the present state of the device. Currently the AI430 devices supports only the Stop mode and the user can enter/exit the stop mode functionality based on their requirement. To exit the stop mode functionality the user can configure one of the below inputs:

RTC | Keypad | Ignition

Module Navigation

To go to the Power Mode Test, from "Sample Application" screen select Power Mode Test by using Key1 and Key2, after that enter into Power Mode Test Screen by using Key3. The Power Mode Test user can test the Stop mode functionality. This would enable the user to enter the low power mode.

Module Test Procedure

Test Case	Description	Test Procedure	Expected Results

Stop Mode	This functionality is used to stop all the functions inside the system and it is waiting into the same mode until an interrupt will occur and activate the device.	Using Key1 and Key2 navigate us to Stop Mode, now by pressing Key3 will set the device into Stop mode.	The running functionality will be kept on halt, and the system will be set to Stop mode i.e., system will turn OFF.
Keypad	This is a wake-up source which is used to activate the system from the power mode.	Click on any one of the keys then the system will turn on	The device will be activated back.
RTC	This is a wake-up source which is used to activate the system from the power mode.	The user needs to set a timer(After how long the system has to restart) in the configuration file and wait till the timer reached the desired time gap	The device will be activated back.
Ignition	This is a wake-up source which is used to activate the system from the power.	When there is any activity performed in the power monitor module then the system will be activated.	The device will be activated back.
CAN	This is a wake-up source which is used to activate the system from the power mode.	The CAN must send the signals to the device in case the user wants to activate the system using CAN	The device will be activated back

Camera

The camera module sample application is shown in the below screen:

Module Description

The functionality is basically designed to test the camera capture and streaming functionality of the device. The camera is connected with the device externally which will detect the presence of any object in front of the screen of the camera

Module Navigation

To go to the Camera Test, from “Sample Application” screen, select Camera Test by using Key1 and Key2, after that enter into the Camera Test Screen by using Key3. In the Camera Test user can test the functionality in four different modes. These four modes are given below:

FULL_SCREEN_MODE_ON	RESIZE_TO_FULL_SCREEN_MODE_ON	DISPLAY_AS_ITIS_MODE_ON	MODE_OFF

The Camera Test Module provide two types of settings to the user:

The Camera Test Module provide two types of settings to the user	Display Setting: The Display Setting involves four options; these options are given below

1) Camera Setting: The camera setting involves four options; these options are given below: a) X0 : Used to set the camera resolution with respect to the x-axis. b) Y0 : Used to set the camera resolution with respect to the y-axis. c) H0 : Used to set the camera resolution with respect to the height of the image capture. d) W0 : Used to set the camera resolution with respect to the Width of the image capture.	2) Display Setting: The Display Setting involves four options; these options are given below: a) X0 : Used to set the display resolution with respect to the x-axis. b) Y0 : Used to set the display resolution with respect to the y-axis. c) VF : Used to display the captured image with respect to the vertical flip of the image capture. d) HF : Used to display the captured image with respect to the horizontal flip of the image capture

Key Description

Key	Description	SP (Short Press)	LP (Long Press)
Key1	Key1 is used for mode selection in the camera test.	Short Press of Key1 will leads the user to the mode change of the camera setting.	Long Press of the Key1 will leads the user to the previous mode.
Key2	Key2 is used to change the camera settings with respect to the x-axis and y-axis coordinates.	Short Press of Key2 will allow the user to change the coordinates of the x-axis.	Long Press of the Key2 will allow the user to change the coordinates of the y-axis.
Key3	Key3 is used to change the camera setting with respect to the width of the captured image.	Short Press of Key3 will allow the user to change the height of captured image.	Long Press of the Key3 will allow the user to change the width of the captured image.
Key4	Key4 will navigate the user to the Display Setting.	Short Press of the Key4 will allow the user to change the coordinates of the x-axis and y-axis of the Display Setting.	Long Press of the Key4 will allow the user to open the camera.

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
Full Screen Mode On	This functionality is used to enter into the full screen mode.	Using SHORT PRESS Key1, User can set the Full Screen Mode On.	The current capture will be displayed into the full screen mode.
Resize to Full Screen Mode On	This functionality is used to resize the screen to the full screen mode.	Once the user enters the camera test, user can set the camera mode by using Short Press key1 and can do the camera setting by using the key2 (Short Press for X0) and (Short Press for Y0) and camera setting for Height of the captured image (short press for H0) and camera setting for width (short press for W0).	The current capture will be resized to the full screen mode.
Display As It is Mode On	This functionality is used to display as it is mode which we have used initially.	Once the user enters into the Camera Test, user can set the Display Mode by using Short Press key1 and can do the Display Setting by using the key2 (Short Press for X0) and (Short Press for Y0) and change the display setting for vertical flip of the captured image (short press for H0) and also can change the setting for	The current capture will be displayed in as it is mode.

		horizontal flip (short press for W0).	
Mode Off	This functionality is used to display the inactive state of the system.	Using Key1 and Key2 navigate us to Off mode; now by pressing Key3 will set the device to off mode.	The current capture will be displayed in Off mode.

EEPROM

The EEPROM module sample application is shown in the next image:

Module Description

The functionality is basically designed to store the data in EEPROM memory and read the same when required. The user will provide the input value to the place holder and output will be stored into the EEPROM memory. User can provide values up to 65535 place holders.

Module Navigation

To go to the EEPROM Test, from “Sample Application” screen select EEPROM Test by using Key1 and Key2, after that enter into the EEPROM Test Screen by using Key3. In the EEPROM Test user can interact with the EEPROM module for below functionality.

To write and store the data in EEPROM memory

To read the stored data from EEPROM memory

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
Write Mode	This functionality is used to write and store the data into the EEPROM memory.	Using Key1 and Key2 navigates to Write Mode; now by pressing Key3 will set the device to the write mode.	The input data provided at the place holder will be stored at the EEPROM memory.
Read Mode	This functionality is used to read the stored data from the EEPROM memory.	Using Key1 and Key2 navigates to Read Mode; now by pressing Key3 will set the device to the read mode.	User can read the data stored at the EEPROM memory.

WatchDog

The WatchDog module sample application is shown in the next screen:

Module Description

The functionality is basically designed to monitor the state of the device. Watchdog reset depends on the Pre-scaler. The Pre-scaler value will be provided within the range of 4 to 256. Once the user provides the pre-scaler value 256, the system will reset after 50 seconds. User has to go to the config.h file to enable or disable any property on the Board.

Module Navigation

To go to the Watchdog Test, from “Sample Application” screen select Watchdog Test by using Key1 and Key2, after that enter into the Watchdog Test Screen by using Key3. The user can select any watchdog (WD0 – WD9). In the Watchdog Test user can interact with the Watchdog module for below functionality.

To enable/disable the watchdog functionality. To feed the WatchDog manually

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
WatchDog	This functionality is used to enable a property on the system.	Using Key1 and Key2 navigates to Watchdog Mode; now by pressing Key3 will select the specific Watchdog and see whether it is enabled or disabled.	Based on the selected Watchdog, whatever the default state is available (Enabled/Disabled) for the specific watchdog, that will be updated to the system.
WatchDog Ping	This functionality is used to feed or refresh the task after every second.	Once the user selects any WatchDog from WD0 to WD9 and hit stop ping, it will stop the feeding.	Hardware will go into reset mode after a few seconds.

BLE

The BLE module sample application is shown in the next screen:

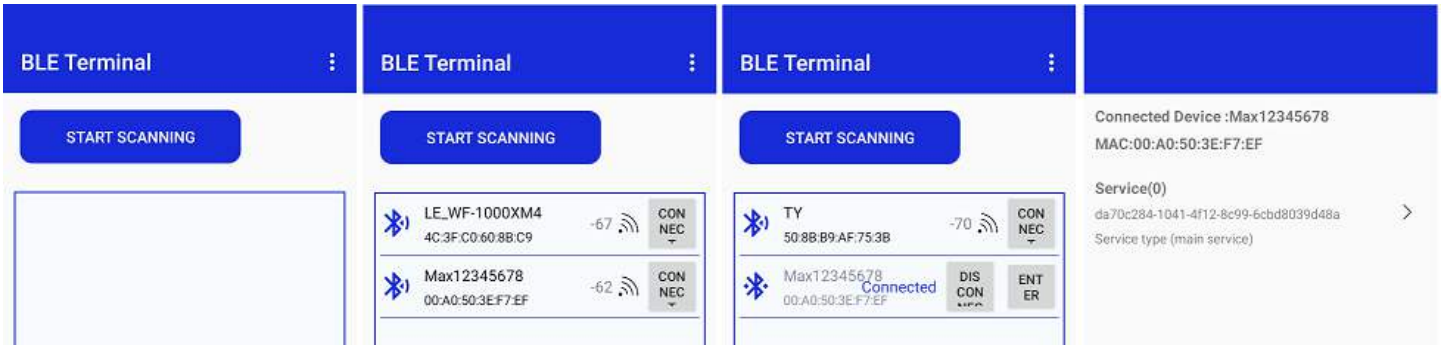
Module Description

The functionality is basically designed to establish a BLE connection between two devices and send/receive data. As a communication example using Bluetooth connection between two devices, developers can use BLE terminal to send commands and get acknowledgement on same screen.

Module Navigation

To go to the BLE Test, from “Sample Application” by scrolling using Key1 and Key2, to select BLE Test Screen then confirm by using Key3. In the Android device run application “BLE Test” once the user opens the BLE application, next step is using

button START SCANNING. Here the user will find MAX device name advertised as “maxAI12345678” as shown below and Users can connect one of these devices by select on the CONNECT option after that user will press on ENTER to move to the SERVICE screen.

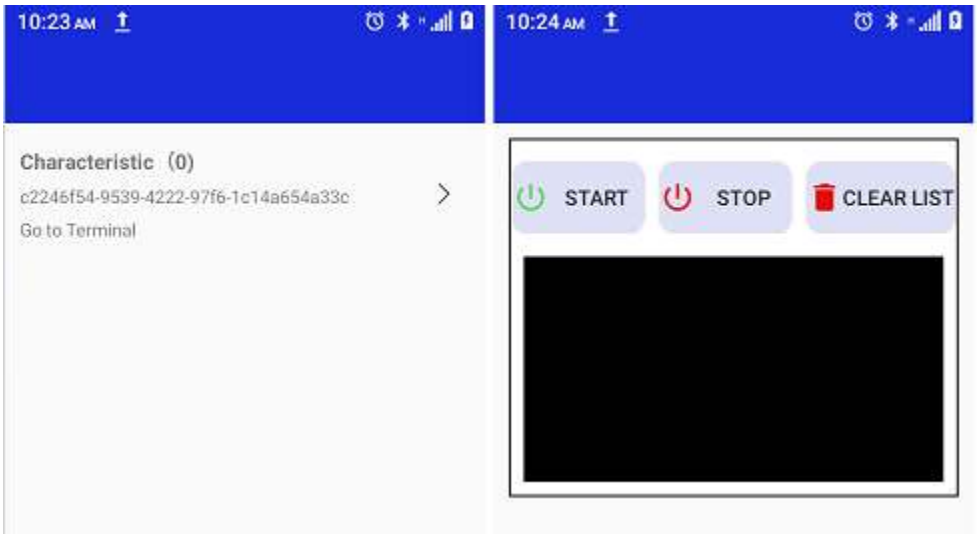


In SERVICE Screen the user will find Service (0), select it will move to the characteristic screen and In the characteristic screen user will find Characteristic (0). Select it will move to BLE terminal screen. User can interact with BLE module by selecting below options:

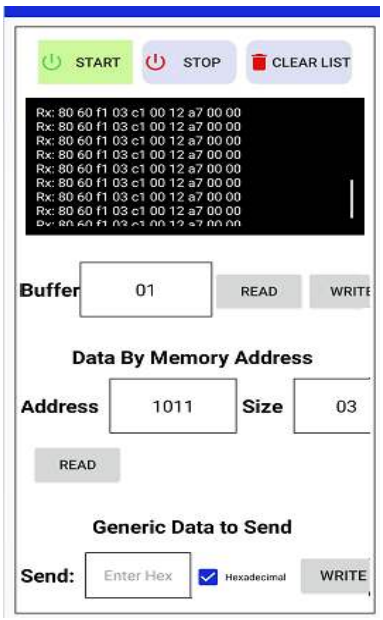
START: To establish Bluetooth connection between two devices, press [Start] to send the start request packet to the system once the connection is established, the system will send a positive response packet back to the user. Which gives the indication that the command is correct and executed successfully.

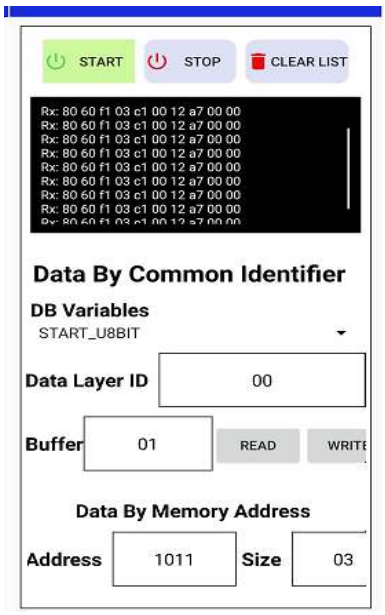
STOP: To disconnect a device, press [Stop] to send the stop request packet to the system, the system will send the positive response packet back to the user to Indicate that the BLE task has been stopped.

CLEAR LIST: To clear list box of commands sent and received



After START confirmed, Terminal will activate the following options:





1) DB Variables: Shows a selection list to choose from available Data Layer DB
a) Data Layer ID: Position index of selected DB Variable
b) Buffer: Variable content to [Read]/[Write]
2) Data by Memory Address:
a) Address: Memory address to [Read] information
b) Size: Number of bytes to [Read] from memory
3) Generic Data to Send
a) This option is used to send any typed data HEX/ASCII to controller by pressing [Write], format HEX is activated by checkbox, other case communication is ASCII

Details of buffer contents format are fully explained on USB Terminal section, so in case of need it refer to this section for further details.

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
Scan	This functionality is used to search for device to be connected.	User needs to Open the FastBLE Application in the Android device, click on START SCANNING. User will expect to detect a maxAI 430 device named “maxAI12345678”. Once it is detected then we need to click on [CONNECT] option. To establish a connection between two devices, the user needs to select [Enter] to then [Start] communication	Once the request packet [Start] is received, system will send the positive response packet back to the terminal to confirm that communication has started successfully.
Write	This functionality allows the user to write some data over the BLE channel which will be received by the device.	Select buffer to type data input in hexadecimal format to then select [WRITE] to send buffer to device	Based on the input data provided by the user, the system will provide the respective output in the hexadecimal format.
Read	This functionality is used to read the data.	Select [READ] option. User will type a Layer ID to then select [READ] option.	Layer ID input by user will get Buffer from this variable in Hexadecimal format, to then replace current data shown in control.

CAN

Module Description

The CAN is a control area network which is basically used to control the ECU (Electronic Control Unit).Can acts as a master controller and based on the functionality it sends the request and receives the messages.

Module Navigation

To go to CAN, from “Sample Application” screen navigate to CAN using Key1 and Key2 and enter into CAN test Screen using Key3. In CAN test screen there are two blocks present they are:

CAN 0 | CAN 1

Sub Screen

Sub Screen_1:When the user selects from CAN_0 or CAN_1 in CAN test screen he will be redirected to Specified functionality test screen which will be out Sub Screen_1. In this Sub Screen_1 there are two packets available as image shows:

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
CAN0	This Functionality is used to check the packet, which was sent from the externally connected CAN Analyzer.	In the CAN test screen go to CAN0 using key1 and Key2 then enter the CAN0 Testing screen using Key3. And check the result	The message packet which was sent from the CAN analyzer will be received at the Rx Packet area in string format.
CAN1	This Functionality is used to check the packet, which was sent from the externally connected CAN Analyzer.	In the CAN test screen go to CAN1 using Key1 and Key2 then enter the CAN1 testing screen using Key3 and check the result.	The message packet which was sent from the CAN Analyzer will be received at the RX Packet area in string format.
CAN State	This functionality is used to Read the state of the CAN.	When the user click on Key1 for once then the CAN state will be Read.	This was just written as an example for the user to understand the usage of CAN State DB variable.
Filter Index/CAN Mode	This functionality is used to update the Filter details and the mode in which is operation on.	When the user double clicks on Key1 then the CAN State will be updated.	The user can check if the Can Filter is enabled and the filter index is 20 and the standard mode is set in the DB for CAN Channel1.
CAN Baud Rate	This functionality is used to set the Device Baud Rate.	When the user clicks on Key1 for three times, then the CAN Baud Rate function will be updated.	The user can check if the CAN1 Baud Rate is updated to 250K.
CAN Drive Reset	This functionality is used to reinitialize the applicant.	When the user clicks on Key1 for four times, then the CAN Drive Reset function will be implemented.	The user can check this update on the DB variable.
CAN Reset	This functionality is used to set the Device into Power Down Mode.	When the user clicks on Key1 for five times, then the CAN Reset function will be implemented.	The user can check this update on the DB variable.

J1939

Module Description

J1939 module is used to interface with the J1939 stack and receive the PGN functionality values and update the values to the GUI. J1939 is also used to provide Diagnosis message to the user.

There are three Diagnosis Message available they are:

DM1 | DM2 | DM3

Module Navigation

To go to JI939 functionality, from “Sample Application” screen navigate to J1939 using Key1 and Key2, later enter into J1939 screen using Key3.In J1939 test screen there are different PGN present they are:

1) EngFuelDeliveryPresss
2) EngineOilLevel
3) EngineOilPressure
4) Coolant Pressure

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
DM1, DM2, DM3	DM as in Diagnosis message are provided for the user to send messages to the DB through CAN.	User can set any of the one DM using Key1 and Key3 (For example DM1) to enable the SDK to capture Diagnostic Message to DB through CAN.	The SDK will start capturing the Diagnostic Message that’s enabled and this can be viewed in DB.
EngFuelDeliveryPress	This functionality is used to provided data about the Engine Fuel	For testing purpose, the device can be connected to the external CAN analyzer to receive this PNG values	The Fuel delivery pressure value will be displayed on the specified space in the UI.
EngineOilLevel	This functionality is used to provided data about the Engine Oil Level.	For testing purpose, the device can be connected to the external CAN Analyzer to receive this PGN values.	The Oil Level value will be displayed on the specified space in the UI.
EngineOilPressure	This functionality is used to provided data about the Oil Pressure.	For testing purpose, the device can be connected to the external CAN Analyzer to receive this PGN values.	The Oil Pressure value will be displayed on the specified space in the UI.
CoolantPressure	This functionality is used to provided data about the Coolant Pressure.	For testing purpose, the device can be connected to the external CAN Analyzer to receive this PGN values.	The Coolant Pressure value will be displayed on the specified space in the UI.

Throughput

Module Description

The main functionality of through put is to constantly update the absolute time and percentage time used by each module until the device is working

Module Navigation

To go to Throughput, from “Sample Application” screen navigate to Throughput using Key1 and Key2. Later enter the throughput test screen using Key3. In this throughput test screen, all the modules are listed for which there are two functionalities which are being updated, they are:

Absolute Time | Percentage Time

Module Test Procedure

Test Case	Description	Test Procedure	Expected Result
Absolute Time	This functionality gives the total 'time' that the task has been executing (the total time that the task has been in the Running state). It is up to the user to select a suitable time base for their application.	This functionality is designed to constantly update the Absolute Time used by the individual module in the UI without any intervention from the user.	Absolute time will be constantly updated.
Percentage Time	This functionality will provide essentially the same information but as a percentage of the total processing time rather than as an Absolute Time	This functionality is designed to constantly update the Absolute Time used by individual module in the UI without any intervention from the user.	Percentage will be constantly updated.

Details of Demo Application

The Demo application is a combined TouchGFX application which will provide insight into how we can combine the different services of the SDK and write a wholesome application.

Difference between Sample Application and Demo Application

The Sample Application was written to help the AI430 SDK User to understand the functionalities of the individual modules and use them as per their requirement. The home screen helps navigate to all the available modules present on the “Sample Application” screen, which can be tested by entering into a specific module whereas in case of Demo Application there are

five screens available which has all the modules integrated within the screens based on their functionality. And the screens can be switched using the panel button functionality mentioned below in 7.2.2.

Panel Button Functionality

Initially when the device is turned ON, the main interface is displayed which would be the screen1, now to shift from one screen to another screen and to interact with each screen the below keys are available:

Block Name	Function	Key Press Instructions	Description
Key1	Back	Short Press	This key is used to go back to the previous screen.
Key2	Inc++	Short Press	This key is used to increment the value of specific module.
Key2	SET	Long Press	This key is used to update the changes.
Key3	Dec--	Short Press	This key is used to decrement the value of specific module.
Key3	SEL NEXT	Long Press	This key is used to select the next module.
Key4	Next SCR	Short Press	This key is used to go to the next screen.

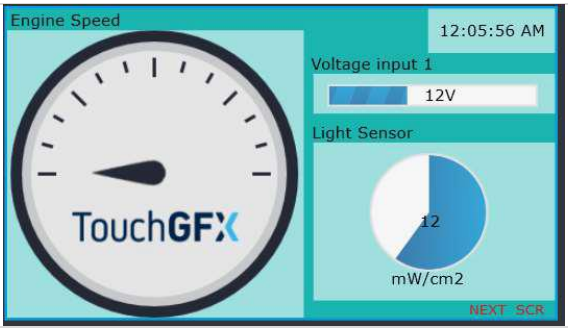
Demo App Screen 1

The below display is the integrated UI screen from which the user can verify the following software modules.

1) J1939
2) RTC
3) Configurable Inputs
4) Light Sensor

Screen1 Description

Screen-1 can be explained by dividing the screen into two halves, where left half consist of a gauge which has a pointer value that will vary based on Engine speed and engine speed will be updated based on CAN and J1939.
On the other half (right half) there are three functionalities available they are for Light Sensor, RTC and Configurable Inputs. Where light sensor has a Circle progress, Configurable Input has Image progress and RTC has a digital Clock to display the Data



In the Above image all the sections of the screen are shown using the arrows, as each section is functioning for different test case such as RTC is for time, J1939 for Engine Speed, Digital Output for Voltage, and Light Sensor for sensor intensity.

Screen1 Test Procedure

Test Case	Screen Section	Description	Test Procedure	Results	Range
Light Sensor	Light Sensor	The light sensor module is basically designed to test the light intensity which is observed by providing external light on the light sensor which is present on the bottom right corner of the device.	Project light externally on the light sensor which is present at the bottom right of the hardware and check the results.	The shaded region in the circle progress which is present at the light sensor will increase or decrease based on the intensity of the light.	0-4914
RTC	Time	This is the real time clock value which will be displayed on the top right corner of screen.	There is no specific test, rather the time will be updated based on the real time data.	The current time will be displayed on the Digital Clock.	-

Configurable Input	Voltage Input 1	The Configurable input is basically designed to read the latest Configurable Input values and to configure the various input channels. The channels which are available for configuring are voltage, current, resistance, digital STB, digital STG, resistance, and frequency.	Connect our device with external device and later check the results.	Whatever voltage value is present in the external device that will be updated on the image progress block and the shaded portion will increase or decrease accordingly.	-
J1939	Engine Speed	This functionality is used to update the Engine Speed based on J1939 which gets updated through CAN.	When the system is connected to an external device via CAN, the CAN Channel will send the speed details to J1939 and that value will be updated on the UI without any intervention from the user.	The gauge value varies based on the speed of the system.	-

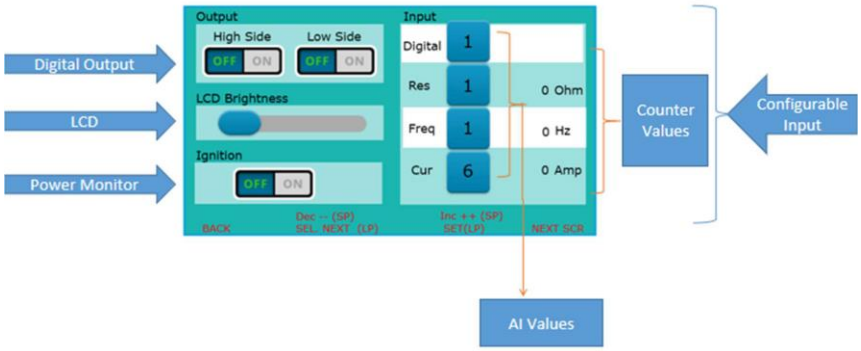
Demo App Screen 2

The below display is the integrated UI screen from which the user can verify the following software modules.

1) LCD
2) Configurable Input
3) Digital Output
4) Ignition Indication

Screen2 Description

The below illustration provides details about screen 2. The screen has two partitions the first partition contains three modules namely Digital Output, LCD and Power monitor. The other partition contains the configurable input. The digital output comprises of two pin values which is HIGH side and LOW side, these pins decide the value of the output. The value 1 indicates the HIGH side and value 0 indicates the LOW side. The LCD brightness module determines the brightness of the screen and it ranges from (0-100). The user can control the LCD brightness by increasing or decreasing it between the provided range. The power monitor comprises the ignition indicator which indicates whether the ignition is ON/OFF. When the indicator is turned ON the power monitor is enabled and when it is turned OFF the power monitor is disabled. The configurable input is designed to configure various input channels. It is used to update the latest configurable input values based upon the previously configured channel properties. The different configurable inputs that are currently available are Digital, resistance, current and frequency. The blocks in the figure represents different configurable inputs.



In the image, the user can see Output block which has two functionality one is for High Side and the other is for Low Side, in this both we have ON and OFF options. Next is the LCD brightness which has a slider that is used to show the brightness level. Next to LDC brightness is Ignition which has OFF/ON option. On to the other side of the Screen there is Input block available which has four configuration properties and their respective counter values. The user can verify the below functionality on Screen 2 of the sample app.

Screen2 Test Procedure

Test Case	Screen Section	Description	Test Procedure	Results	Units
Digital Output	Output	The digital output comprises of two pin values which is HIGH side and LOW side, these pins decide the value of the output. The value 1 indicates the HIGH side and value 0 indicates the LOW side.	High Side:- User can probe the digital output pins by verifying whether the Digital Output high pin should be 1 and Digital Output low Pin should be 0. User can navigate to High side by long pressing Key2 and set it using Key3(long press)	The High Side Functionality present in the Output block will be Turned ON	-
			Low Side:- User can probe the digital output pin to by verifying the Digital Output High pin should be 0 and Digital Output low Pin should be 1.User can navigate to Low side by long pressing Key2 and set it using Key3(long press)	The LOW side functionality present in the output block will be turned ON.	-
LCD	LCD Brightness	The LCD brightness module determines the brightness of the screen, it ranges from (0-100) and depending upon the user convenience the LCD brightness can be increased or decreased between the provided range.	User can increase or decrease the LCD brightness using Key2 (Short Press) and Key3 (Short Press). And set the final LCD brightness using Key3 (Long Press).	The slider slides depending up on the brightness.	0-100
Configurable Input	Input	The configurable input is designed to set various input channels. It is used to update the latest configurable input values based upon the previously	User can configure each property for a particular channel and then view the data in the adjacent location. **Digital**: User can configure the Digital property for AI1 to AI5 Channel using Key2 (Short Press) and Key3	The value of digital input will be updated on the GUI counter.	0-5

		configured channel properties. The different configurable inputs that are currently available are Digital, Resistance, Current and Frequency.	(Short Press) and set the specific AI value using Key3.		
			Resistance: User can configure the resistance property for AI1 to AI5 Channel using Key2 (Short Press) and Key3 (Short Press) and set the specific AI value using Key3.	The value of Resistance input will be updated on the GUI counter.	Ohms
			Frequency: User can configure the frequency property for AI1 to AI5 Channel using Key2 (Short Press) and Key3 (Short Press) and set the specific AI value using Key3.	The counter value of the Frequency input will be updated.	Hz
			Current: User can set the current only for AI6 which is set by default.	The counter value of the Current input will be updated.	Amp
Power Monitor	Ignition Indication	The power Monitor comprises the indicator mode which provides two basic functionalities ON/OFF mode. When the indicator is turned ON the Power Monitor is enabled and when it is turned OFF the Power Monitor is disabled.	User can navigate the ignition ON/OFF.	The ignition will switch modes to ON/OFF based on Power Monitor.	-

Demo App Screen 3

The below display is the integrated UI screen from which the user can verify the following software modules.

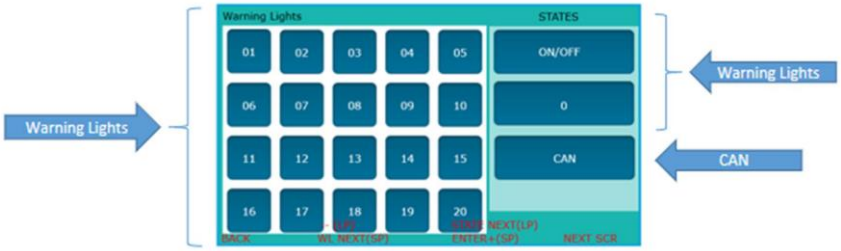
1) Warning Lights 2) CAN

Screen3 Description

Screen 3 is designed to provide the functionality of warning light and CAN where the screen can be partitioned into two divisions, one phase has all the available warning lights and the other has the two functionality of warning lights that are:

ON/OFF BLINK

And also, the second half consist of CAN functionality.



Screen3 Test Procedure

Module Name	Screen Section	Description	Test Procedure	Results
Warning Light	Warning Light	The Warning Light module is used to provide warning signals from the device. There are a total of 20 warning signals available in our device which are from WL1 to WL20.	User can switch to one warning light to next warning light by long pressing of Key2. Also, user can navigate the Warning Light to different state by long pressing of the Key3. User can turn OFF/ON a Warning Light by Short Press of Key2. User can Increase/Decrease blinking count of a particular Warning Light. User can increase the blinking count to 65535 by short pressing of Key2 and to decrease the blinking count up to 0 by long pressing of Key2.	The specific Warning Light which is being selected will glow or blink accordingly.
CAN	CAN	The CAN is a Control Area Network which is basically used to control the ECU. CAN acts as a master controller and based on the functionality it sends the request and receives the messages	When external CAN is connected to the device. This CAN data received will update the Warning Lights without any manual operation performed by the user. For testing purpose, the user can operate the WL ON/OFF functionality with the help of CAN Analyzer terminal by changing PGN value	The Warning Light will glow based on the signals sent from CAN.

Demo App Screen 4

The below display is the integrated UI screen from which the user can verify the following software modules.

1) Camera

Screen4 Description

Screen 4 can be explained by dividing the screen into two halves, the first half will be consisting of the Camera section where the video that is captured from the external camera will be projected and the second part of the screen will be having two functionalities namely:

1) Flip VERT (Flip Vertical) 2) Flip HOR (Flip Horizontal)

Screen4 Test Procedure

Module Name	Screen Section	Description	Test Procedure	Results
Camera	Camera	The functionality is basically designed to test the camera capture and streaming functionality of the device. The camera is connected with the device externally and the video will be displayed on the device screen.	When the camera is externally connected to the device it captures the video of all the activates that are being preformed in front of the camera and that video will be displayed on the screen present below the camera functionality.	In the display which is available on the screen will show the video that is captured by the externally attached camera.
Flip Vertical	FLIP VERT	This functionality is used to flip the video into vertical angle.	When the user selects the vertical flip state by using Key2 (SELNEXT) video will be flipped vertical.	The video will be flipped vertically.
Flip Horizontal	FLIP HOR	This functionality is used to flip the video into horizontal angle.	When the user selects the vertical flip state by using Key2 (SELNEXT) video will be flipped horizontally.	The video will be flipped horizontally.

Demo App Screen 5

The below display is the integrated UI screen from which the user can verify the following software modules.

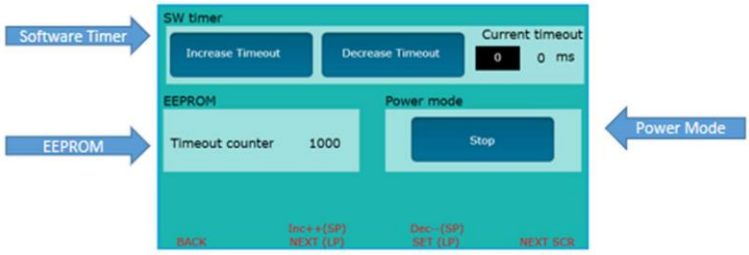
SW Timer EEPROM Power Mode

Screen5 Description

Screen 5 is designed using three software module which includes Software timer, EEPROM and Power Mode.
The topmost block which is available on the screen is for Software Timer which has three functionalities:

Increase Timeout | Decrease Timeout | Current Timeout

In the later part there are two sections, on the left part EEPROM block is available and to the right Power mode is available. In the EEPROM block the Timeout counter is read from the EEPROM and displayed and for Power mode the Stop functionality is available



Screen5 Test Procedure

Screen Section	Functionality	Description	Test Procedure	Results
SW Timer	Increase Timeout	Used to increase the Timeout value.	User can increase the Timeout value by Short Press of the Key2	Timeout value will be increased and displayed under Current Timeout.
SW Timer	Decrease Timeout	Used to decrease the Timeout value.	User can decrease the Timeout value by Short Press of the Key3	Timeout value will be decrease and displayed under Current Timeout.
SW Timer	Current Timeout	Used to show the current Timeout value.	The current value is auto generated based on Increment or decrement.	The current Timeout value will be updated.
EEPROM	Timeout Counter	EEPROM stores the software timer data and the same can be read back.	The Timeout counter value will be directly fetched from the EEPROM DB which will be dependent on the software Timeout.	The Timeout counter value will be displayed.
Power Mode	Stop Mode	Used to stop all the functions inside the system and it is waiting into the same mode until an interrupt will occur and activate the device.	User can set the Power Mode as STOP by Long Press of the Key3 the board will switch to the Reset Mode. User can come back to the wake-up state by using any of the sources like: Keypad RTC Ignition CAN	The running functionality will be kept on halt and the system will be set to STOP mode i.e., system will turn OFF

BLE Mobile Test Application

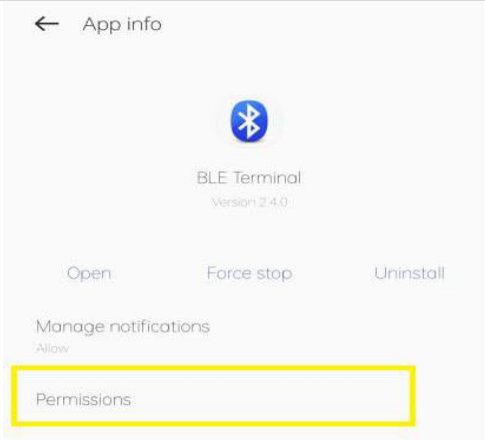
The BLE test application is an android based mobile application which can be used to communicate with the maxAI 430 hardware via BLE for testing/debugging purpose. It supports the below functionalities.

1) Read/Write to all the DB variables support by the SDK
2) Direct Memory location Read

In this section we will walk you through the BLE App screens and how to use the functionalities of the BLE App.

Installing the application

The application used in this test is an apk file that works on android phones. After downloading the apk file to an android phone, it is installed by locating it in the downloads folder and clicking on it. Once installed the BLE App, we need to give permission to storage and to location as shown in the below image



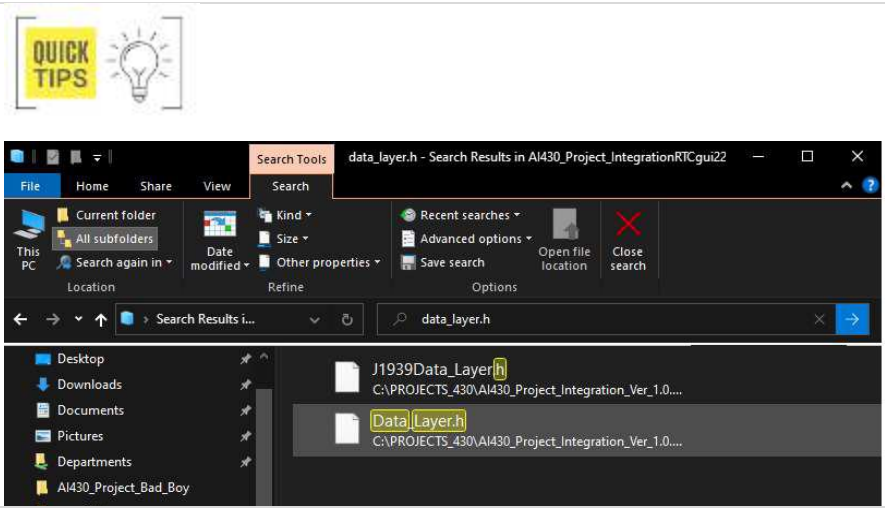
After following the above step, by default the following files will be generated:

Config.h	Data_Layer.h
EE_PH_DB.h	J1939Data_Layer.h

The above files will be generated in the following path:

Internal storage/Documents/VariableFiles/ (Config.h,Data_Layer.h,EE_PH_DB.h, J1939Data_Layer.h)

These files need to be replaced in your phone with the most recent ones from the project of the MAXAI 430. They are obtained by searching for them through the project shown in the next below. After finding the files, download them to your phone.

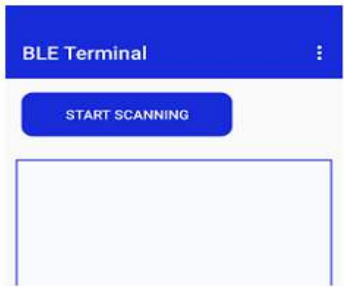


Scan Screen

Open the app, where the app home screen will appear as next image:

Click on the “3 dots” button where you can load the previously downloaded files. You need to select each of the 4 files in the following fields and press the OK button shown in the image below.

If the name of any of the files is modified the app will not work properly. If the files are not located on the Internal storage/Documents/VariableFiles patch, the app will not work properly.



After loading the files, you can go back to the home screen by clicking on the close button, then press the Start Scanning button. The list of maximatecc AI430 devices available will be displayed on the screen as shown below. If Bluetooth is turned OFF on your mobile phone you will get a notification to turn on Bluetooth and location before using the BLE functionality. Once the scanning starts you will see the below screen with the list of devices.

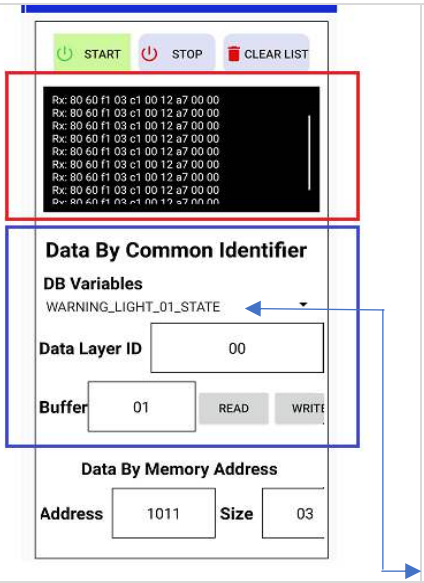
Connect Screen

Press the Start Scanning Button to connect to the maxAI 430 device. Once the device is connected, Connected button status will be shown. Press the enter button to go to the next screen. To disconnect the device from BLE communication, the user can press the Disconnect button. After the connection is successful, the connected device names and services will be shown on the display. Press the service UUID on the list and it will move to the next screen. The screen below shows the characteristic UUID on the list. Press the characteristics on the list and it will move to the GUI Screen to communicate with the device:

GUI Screen

The main GUI Screen for the device is shown below. Press the START button to start open communication port. The second half of the screen will be populated with the options once the communication has started.

Read/Write DB Variable Screen



The screen section marked in **RED** shows the Terminal for RX/TX communications of the DB variables.

The screen marked in **BLUE** shows the Read/Write DB Variable Screen.

To read the data from the AI430 SDK DB, select the appropriate SDK module for accessing the module's DB variables on the dropdown list.

Select the DB variables in the DB dropdown list.

Please click the **READ** Button for the selected DB variable. The terminal will reflect the communication between the device and the Mobile App. The values present on the device shall also be reflected on the Terminal.

To send the updated data to the device, manually type the value in the buffer text box and then click the **WRITE** Button. The Terminal will reflect the communication between the device and the Mobile App.

Please see the image, variable WARNING_LIGHT_01_STATE is selected.

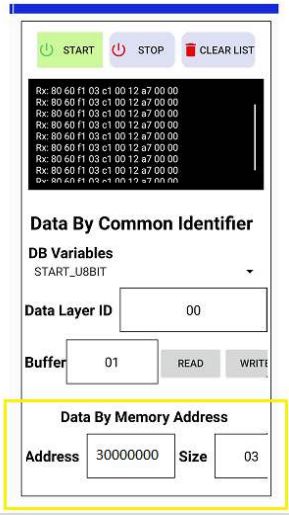
In the Warning light module, the Warning light 19 State DB variable is selected. This variable as defined in the section Warning Light Module is used to turn on the Warning light. Once the DB variable is selected, the appropriate DB field ID value will be displayed in the Data Layer ID text box available in the area marked by Data by Common Identifier.

To enable the warning light type 1 in the Buffer text box and hit the write button. The value gets written to the Data base in the AI430 module and the warning light 19 turns ON. Please see the below device screen shot which shows the warning light is ON.

Read/Write by Memory Address Screen

The screen marked by yellow shows the Memory Access Area. If the user needs to read any
memory location, he can directly provide the address in the address field and the size value in the size text field and then click read. The data received from the device would be listed in the TX/RX area.
In the above illustration the Data By Memory address section has an address and size field.
To read the data of the Memory Location, enter the Address and the Size of the variable under
consideration. Click the READ Button for the Selected Memory Address. The terminal will reflect the communication between the device and the Mobile App. The values present on the device shall also be reflected on the Terminal.

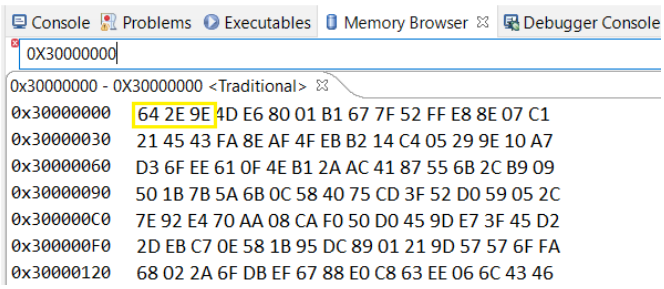
For example, if we want to read 3 bytes from the address 0x30000000, we will update the address and size as shown below and then click read. The result can be got from the TX/RX area.



The first 4 bytes in the Tx/Rx data carries the header field and the 5th byte in each variable represents the request packet. The next few bytes represent the Data, the last byte denotes the checksum value.

TX : 80 60 f1 06 23 30 00 00 00 03
RX : 80 60 f1 04 63 64 2e 9e 68 00

Please see the next screen which shows that the value at memory location 0x30000000 is “64 2e 9e” as received in our response packet in **ST32IDECube**.



Generic Data to Send

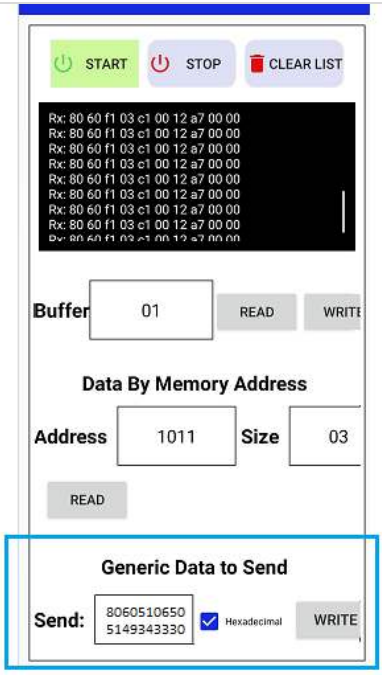
The next image screen marked by blue shows the Generic Data to send section where any generic BLE hex data can be sent to the device.
For example , we will send a hex data to the BLE device .

TX : 80 60 51 06 50 41 49 34 33 30

Hex Value:

80 60 51 06 50
41 49 34 33 30

The first 4 bytes are header , the next is the request packet followed by the data.
When this data has been received by the MAXAI430, it is translated to text and then displayed in the UI by the Bluetooth App as shown in the below image by converting this value into a string.



The Text AI430 is the packet value received in the unit.

Clear list and Stop Testing

Selecting clear list would clear the RX/TX terminal so that the user can see the latest data. Press the STOP button to stop the testing as shown in the below image.

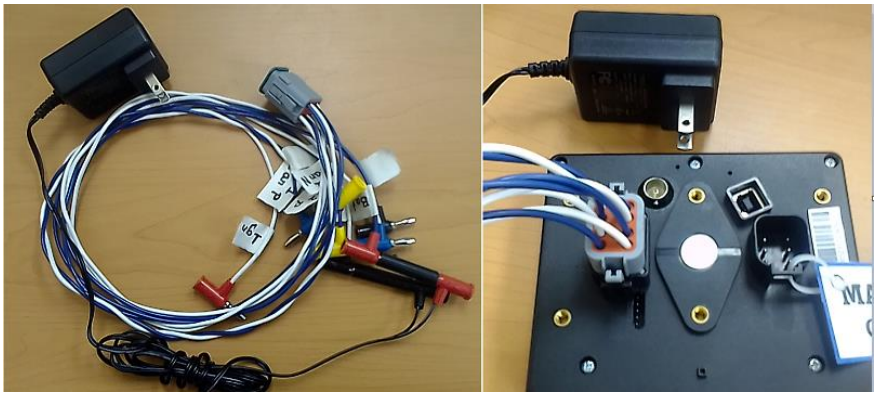
Flashing Guide

This section will contain the instructions to flash a maxAI unit via CAN, including erasing the memory of the unit, loading a new bootloader, and writing the application. For that, the required materials are:

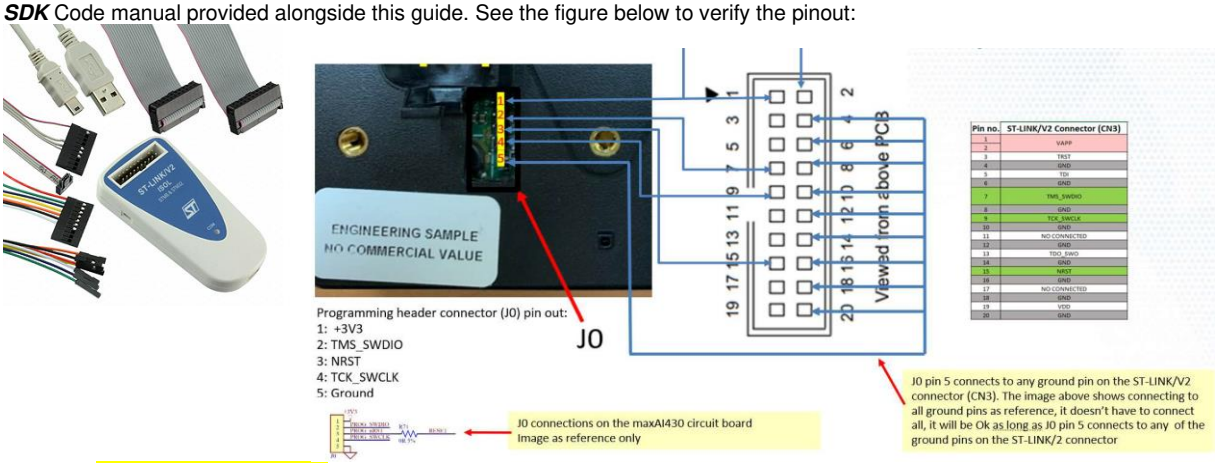
maxAI 430 Unit	Harness to 12v Power Supply
12v Power Supply	ST-LINK ISO Debugger
Debugger-Unit Harness or Dupont Cables	CAN Input Harness
CAN to USB Cable	Computer

Instructions

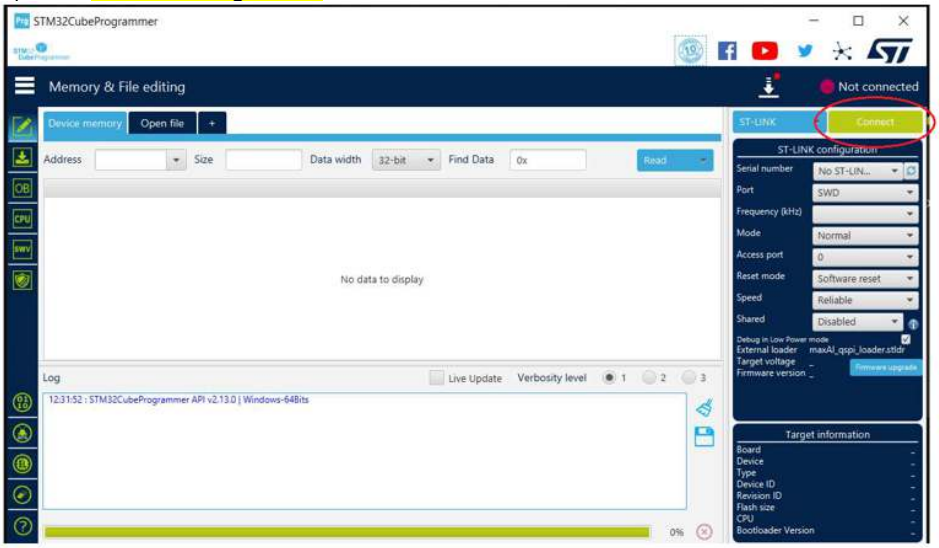
1. Connect to the maxAI 430 unit to the Power Supply of 12v via harness:



2. Connect the unit the computer through the debugger. This will be done using the ***ST-LINK/V2 Debugger Connections into a maxAI 430 with SDK*** Code manual provided alongside this guide. See the figure below to verify the pinout:



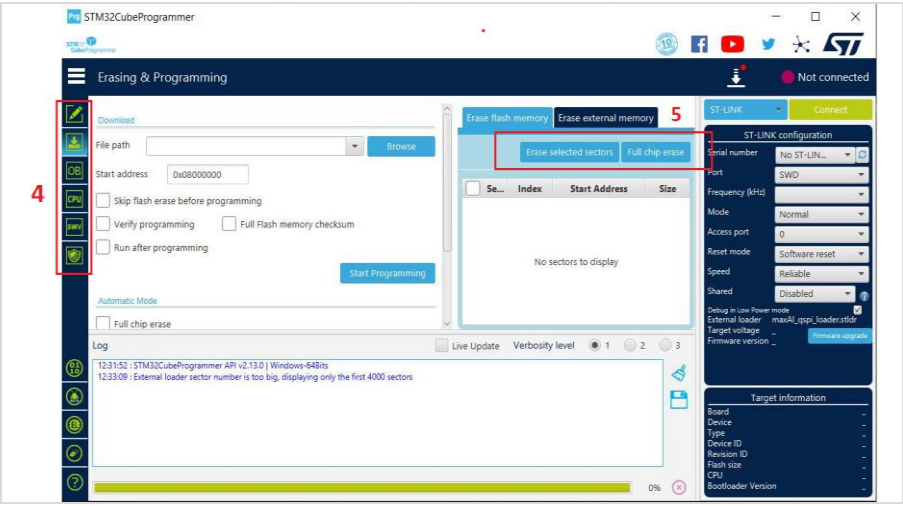
3. Open the **ST32CubeProgrammer** Software and click on the connect button to ensure the cables are connected properly.



4. Go to the Erasing & Programing section located in the tab menu at the left side of the software.

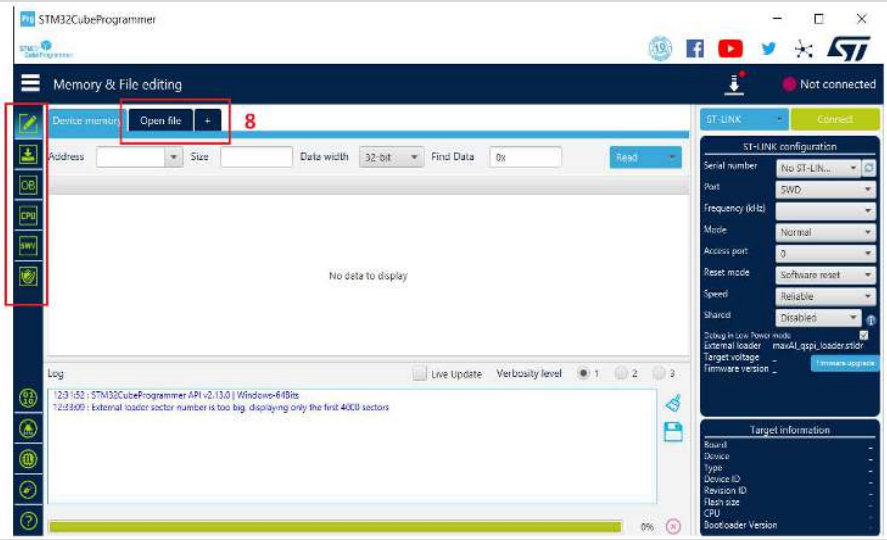
5. While connected, click on the Full chip erase from the Erase Flash Memory window and wait for the program to complete the task.

6. Do the same process for the external memory and wait for the program to complete the task.



7. Go to the Memory & File editing section located on the tab menu at the left side of the software and click on the Open File button located on the tab menu at the left side and look for the .srec file of the bootloader, then open it on the system explorer.

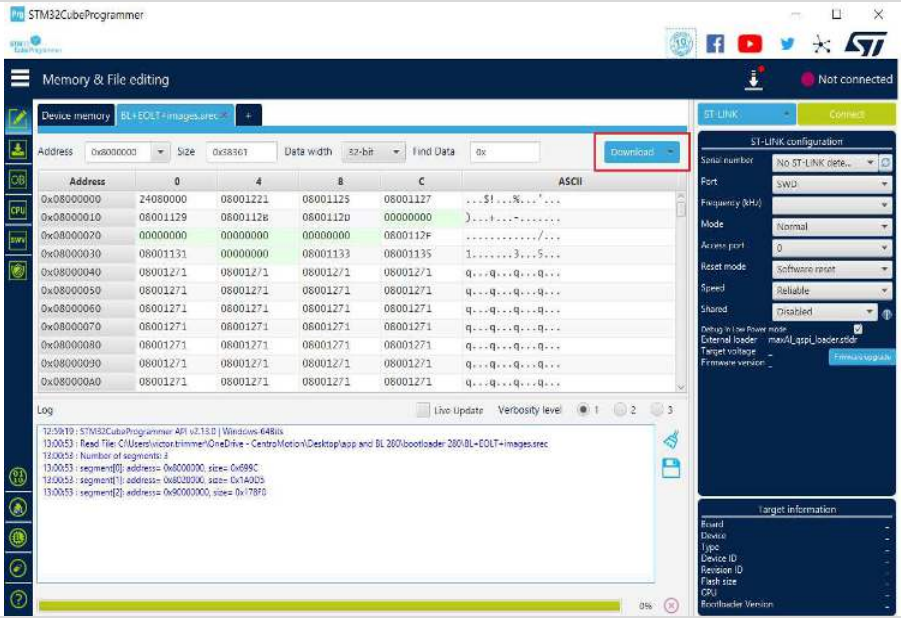
8. Once the bootloader has been selected, click the Download button from the right and wait for the program to complete the task.



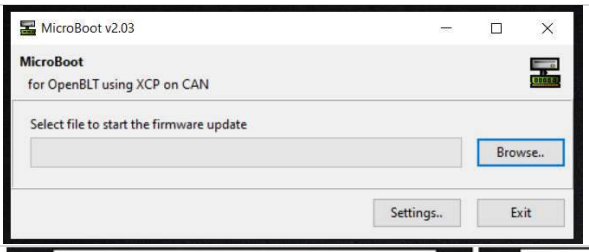
9. Click on the disconnect button and disconnect the unit from the debugger.

10. Connect the unit to the computer using the CAN to USB cable and the CAN input harness.

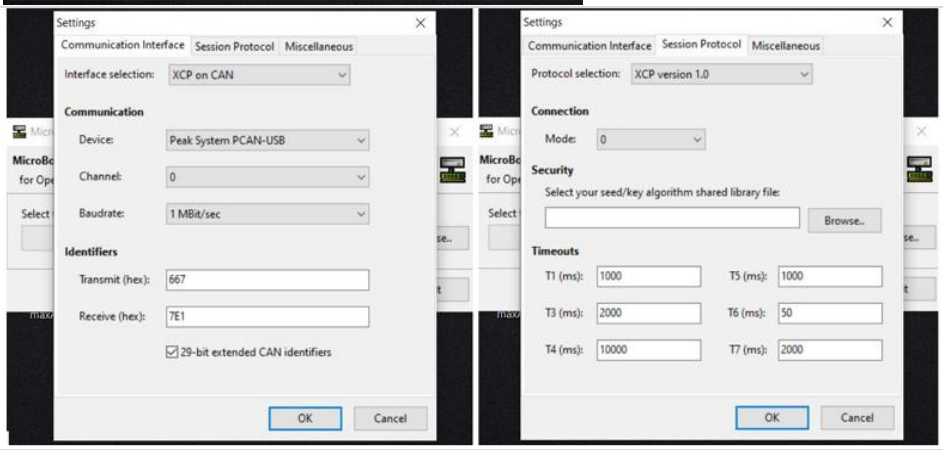
11. Enter the bootloader mode by pressing the button and the right button when powering on the unit. If done correctly, the keypad's backlight will start to blink rapidly.



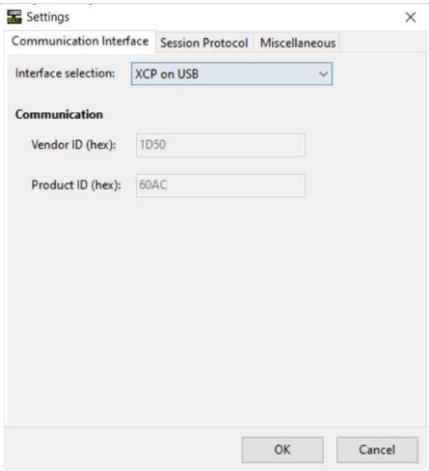
12. Open the MicroBoot software



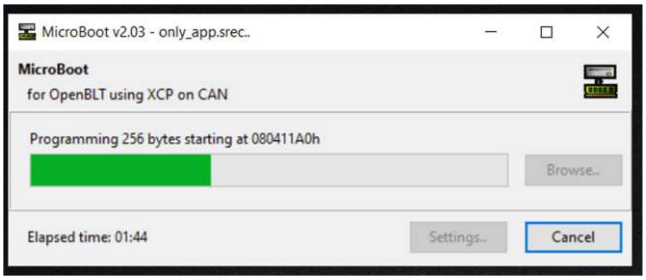
13. Click the Setting button and make sure to select the correct settings and have installed the CAN-USB drivers.



In case the unit is being flashed via USB, just connect the USB/A-USB/B cable to the unit and select the following settings



14. After that, click OK and it will return to the menu seen before, in which the Browse button shall be clicked for a "System Explorer" window to open to which the .srec file of the application should be selected. Afterwards, the program will automatically attempt to write the application onto the flash internal memory, skipping the bootloader section (from address 0x080000000 to 0x080200000).



15. In case MicroBoot doesn't start writing the file immediately, it means there is something wrong with the setup, check the steps above again and make sure all of them are correctly fulfilled.

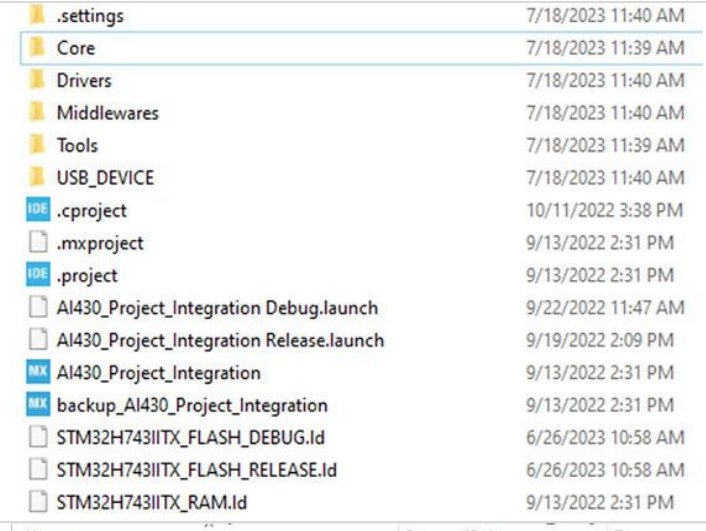
Project Migration

This section will review the steps necessary to migrate any project involving the maxAI SDK to the newer version of the SDK template, which among other functional changes includes version upgrades:

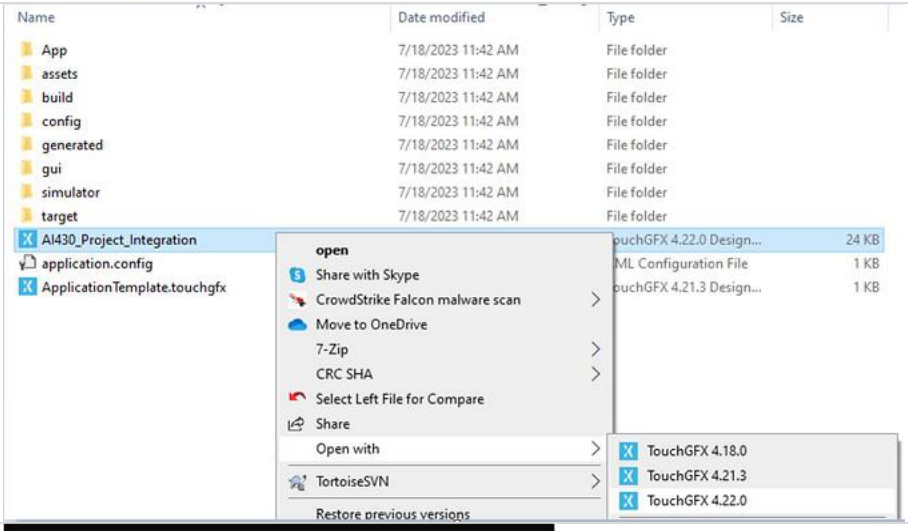
Transition from TouchGFX v4.18.1 to 4.22.0	Transition from STM CUBE IDE v1.8.0 to 1.11.0

TouchGFX Migration

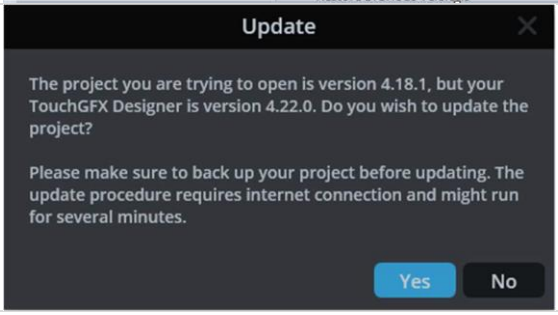
First, the new SDK template shall be downloaded to a directory path that doesn't contain any spaces, or special characters. See the figure below for reference as to what the template should contain.



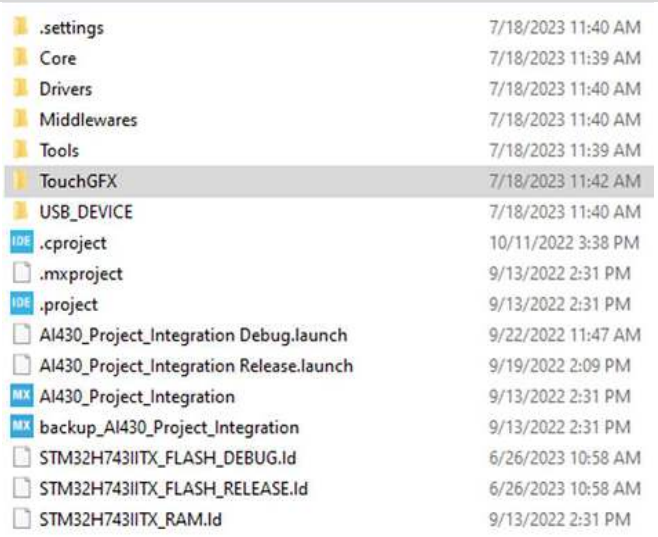
Afterwards, to migrate the version of TouchGFX in the previous project on its separate directory (making a copy of said project is recommended as backup) by opening it with the TouchGFX Designer version 4.22.0 directly from the folder of the project



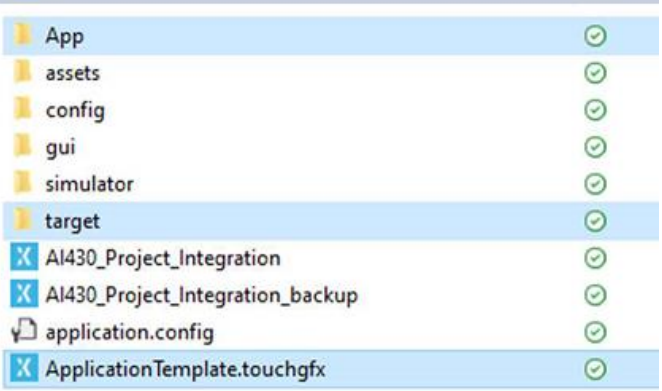
In doing so, a pop-up will appear asking if the project should be migrated from the present version (4.18.0) to the new version (4.22.0), to which "Yes" shall be selected.



After that, copy and paste the complete TouchGFX folder from the separate project where the migration was made, to the folder with the new SDK template.

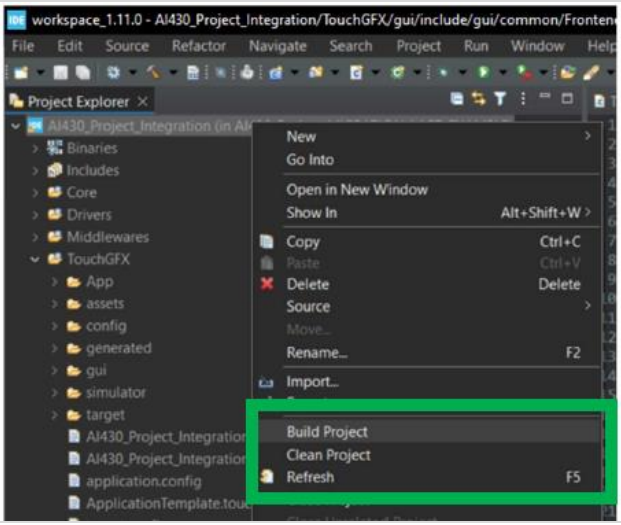


With the project migrated, 2 folders and 1 file must be copied and pasted overwriting the previous files on the new TouchGFX folder. These files/folders are obtained with the .zip App+Target+Template, which contains the folder with the same names as the ones they should replace. When trying to paste the folder over at the new TouchGFX folder a prompt from the system will ask if the files should be replaced, to which Replace all files should be selected



Once the correct files are on the project, it is a good practice to open the AI430_Project_Integration (or 280) file with the TouchGFX Designer 4.22.0 to make sure that no migration prompts appear again. Then, new code should be generated again, in case any file wasn't generated by the first step, see the next figure for it: (Remainder, if the directory path has any spaces or special characters, the Designer won't allow generating new code).

Afterwards, proceeding to STM32CUBE IDE v1.11.0, a full clean, refresh and build with the Debug configuration is necessary for the project. If no errors occur, the project is ready.



Steps Resume

Resuming the steps on a list, it would be as follows:

1. Import SDK Template to a directory path without spaces or special characters.
2. Migrate the TouchGFX component of your project on previous project (make a backup if necessary) by opening it with the TouchGFX design v4.22.0 and answering yes to prompt.
3. Copy and Paste the whole TouchGFX folder from the migrated project to the new template.
4. Copy and Paste the zipped files “App, Target and Template” into the new template’s TouchGFX folder.
5. Open the interface with TouchGFX v4.22.0 Designer and generate code again.
6. Open the project with STM32CubeIDE v1.11.0 and Refresh, Clean then Build the Debug Configuration.
7. (Depend on the application) if any errors occur, it means that header files, references or declarations were present on the files altered with the new template, all those sections of the code should be reintroduced manually.