VIRTUOS

XPOS XP-36xx USER MANUAL

Autorské právo

TABLE OF CONTENTS

1. INTRODUCTION

XP-3685 Basic Introduction 005 - 009
Dimensions 010 - 010
Specification 011 - 011
Packing List 012 - 012

2. SOFTWARE INSTALLATION AND SETUP

Installation Order 012 - 012
Chipset 013 - 014
Audio 014 - 014
Graphics 015 - 015
Intel ME 016 - 016
Serial IO 017 - 017
USB LAN 018 - 018
Windows Driver 020 - 020
Set COM 021 – 021
Sensor 020 - 020
Intel® Rapid Storage Technology 022 - 023
ADB Android Debug Bridge 023 - 023

3. Hardware

Access Storage Device 023 - 024
Adding or Removing M.2 PCBA from M.2 Tray 025 - 025
Access the Motherboard 026 - 027
Install or Remove Memory 027 - 028

4. IO Board SDK Instruction

Overview 028 - 028
System Requirements & Installation 029 - 030
DLL Interface for FEC Io Board SDK 030 - 032
Example of DLL Interface 033 - 033
EXE Interface for FEC Cash Drawer SDK 033 - 034

5. Software

Developers Guide for Serial Communications 034 - 041
Ambient Light and Proximity Sensor 041 - 043
Control the Sensor Timing 044 - 044
Cash Drawer Command 045 - 045

XP-3685 Basic Introduction

XPOS XP-3685 is a fifteen inch all in one fanless point of sales system powered by Intel® 7th generation Celeron, i3 and i5 processors. By utilizing slim key components and enclosing them in an aluminum die-casting chassis, XP-3685 is able to achieve both a slim and strong design. The IO interface is connected with an USB cable and installed inside the stand base.

The dual hinge stand provides the user the ability to adjust the stand and display angles making it optimal for all environments.

On the bottom of the display is a standard 2in1 sensor. The proximity sensor can detect a presence infront of the sensor and wake the system up from an S1 and S3 sleep state allowing users to save on power consumption when the store traffic is low.

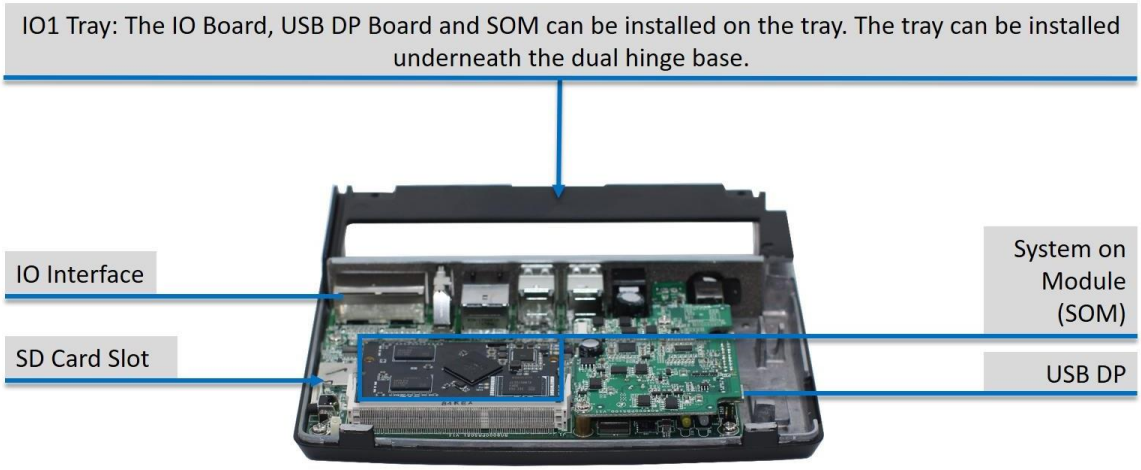
- Slim Panel PC 2.4cm
- Fanless Design
- 2in1 Sensor: Ambient Light Sensor & Proximity Sensor
- Dual Hinge Stand: High Profile and Low Profile
- Integrated and Extended 2nd Display Options

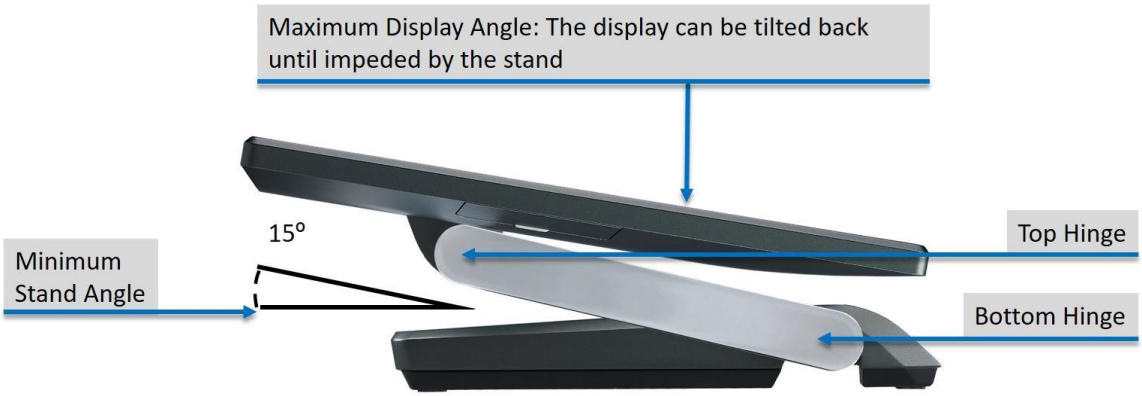

Displej
Integrovaná základní deska
Stabilní základna
I/O porty


Volitelná 2 MP kamera
Senzor přiblížení a senzor okolního světla
Kryt kabelu
Kryt kabelu
I/O porty


Horní kryt
USB 2.0
Type A
pro přídavná
zařízení
Levý
a pravý
kryt
Kryt USB-C
kabelu
Otvor pro kabel


Minimální úhel naklopení LCD
30°
Horní kloub
60°
Maximální úhel stojanu
Dolní kloub





Power Button with LED:

Red = Power adaptor connected but system is off
Blue = System is on

- Storage LED
  Orange = Flashing during storage activity

Galaxy Gray with Black
Silver with Black

Dimensions

Display Dimension

		XP-3685	XP-3685W
A	Display Height	273.3mm	235.7mm
B	Display Length	342.5mm	368.2mm

Side Profile Dimension

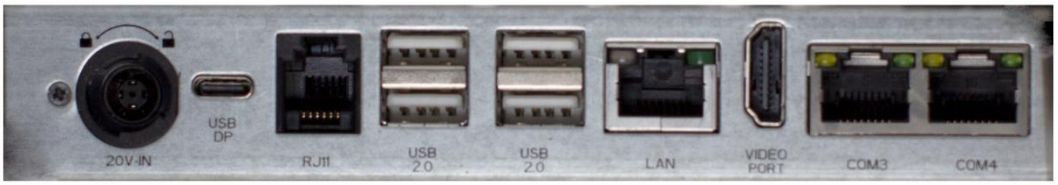
		XP-3685	XP-3685W
A	Panel PC Thickness	24mm	24mm
B	System	332mm	293mm
C	Base Depth	231.75mm	231.75mm

SPECIFICATION

Panel PC

Description	XP-3685	
Processor	Intel® Celeron® Processor 3965U, 2M Cache, 2.20 GHz Intel® Core™ i3-7100U Processor, 3M Cache, 2.40 GHz Intel® Core™ i5-7300U Processor, 3M Cache, up to 3.50 GHz	
System Memory	4GB Standard, Maximum 32GB (2 x 260-pin DDR4)	
Storage Device	Celeron: 1 x M.2 (B+M Key) SATA III and PCIE, 1 x M.2 (BM Key) SATA III i3 & i5: 2 x M.2 (B+M Key) SATA III and PCIE	
Speaker	2 x 2W Internal Speaker	
Construction	Aluminum Die-casting + Plastic + Glass	
Housing Color	Galaxy Gray + Black Silver + Black	
Touch LCD Display		
Size / Resolution	15" TFT-LCD / 1024 x 768	15.6" TFT-LCD / 1920 x 1080
Brightness / Backlight	400nits (LED) PCAP	400nits (LED) PCAP
Panel PC IO		
USB Port	1 x USB 1.0 Type A, 1 x USB 2.0 Type A 1 x USB DP (Reserved for 2nd Display)	
FEC DP Port	1 x FEC DP Port (Customer Display or Integrated 2nd Display)	

IO Options



IO1	
System on Module (SOM) Optional	ARM Cortex A7 Quad Core 1G DDR3 On Board 8GB eMMC 5.0 On Board
SD Card Slot	1 x SD Card Slot (Must Have SOM)
Video Port	1 x Video Port (Must Have SOM)

USB DP	2 x USB DP Port (1Reserved to Connect to Panel PC)
USB Port	4 x USB 2.0 Type A
LAN Port	1 x LAN (Green Light Mega LAN, Orange Light Giga LAN)
Serial Port	2 x RJ45 (RS232)
DC-in	1 x DC-in for 90W (20V/4.5A) Adaptor

Packing List

Standard	Optional
XP-3685 x 1 M.2 Tray x 1 COM RJ45 to DB9 Cable x 2 90W Adaptor x 1 Power Cord x 1 Driver and Manual CD x 1	WiFi Module x 1 M.2 Tray x 1 Add-on Device x1 Customer Display x 1 2nd Display x 1

Software Installation and Setup: Motherboard

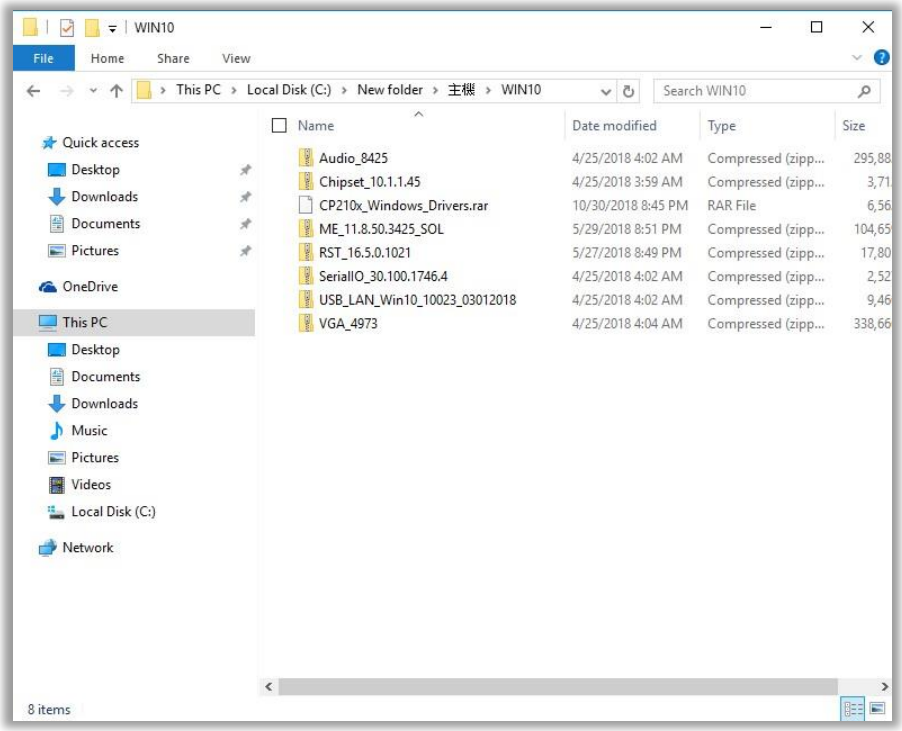
Follow the below order if installation for the Motherboard:

O1 Chipset O2 Audio O3 Graphics O4 Intel® ME

O5 Setup Serial IO O6 USB LAN O7 Windows Drivers O8 Set COM

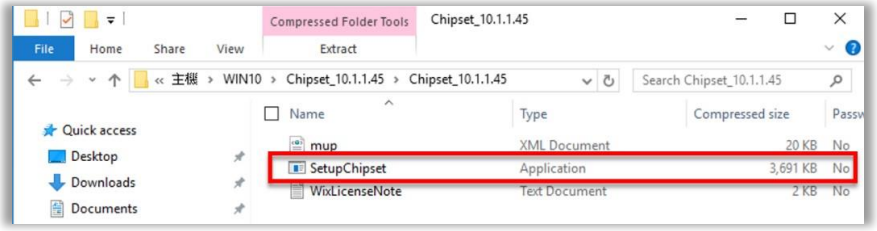
O9 Intel® Rapid Storage Technology (RST) (Applicable to i3 & i5) O10 ADB Driver (Applicable to SOM)

 Un-compress files



1. Chipset

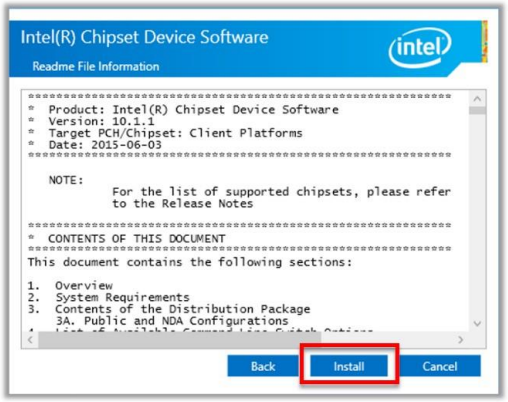
 Locate chipset folder and double click on **[SetupChipset]**



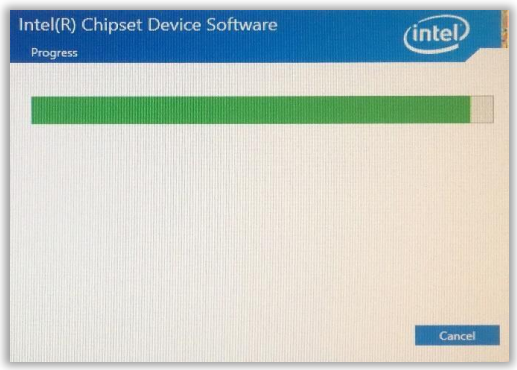
 Click **[Next]**

 Click **[Accept]**

 Click **[Install]**



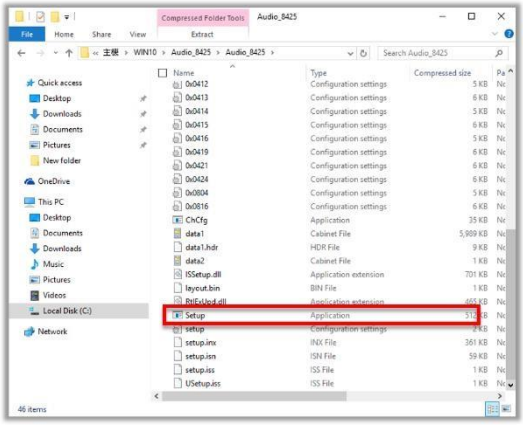
 Click **[Yes]**



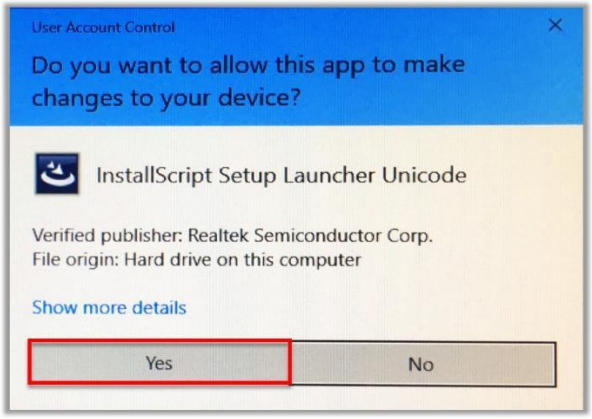
 Click **[Restart Now]**

Audio

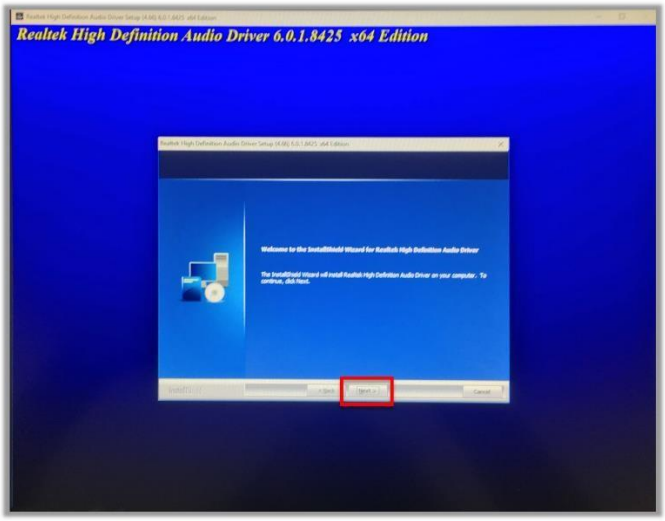
 Double click **[Setup]**



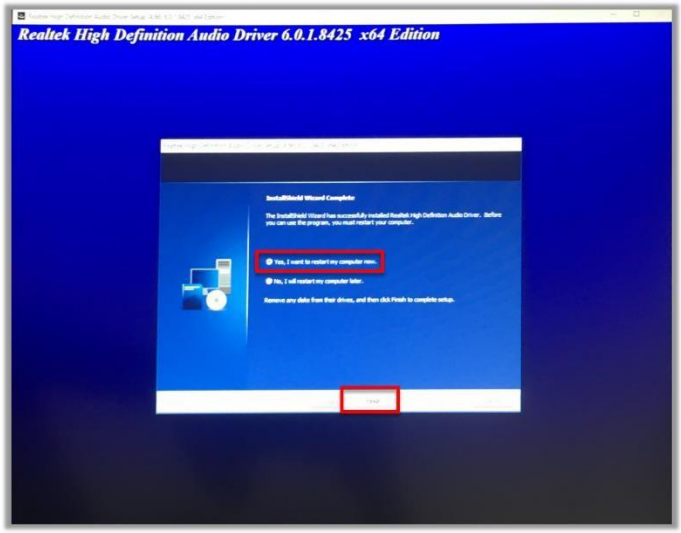
 Click **[Yes]**



 Click **[Next]**

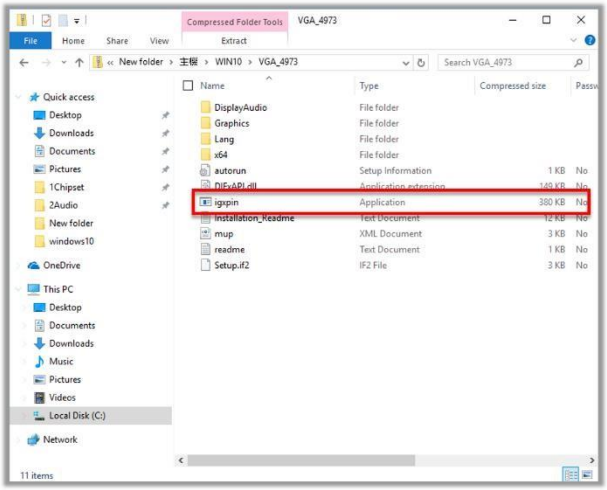


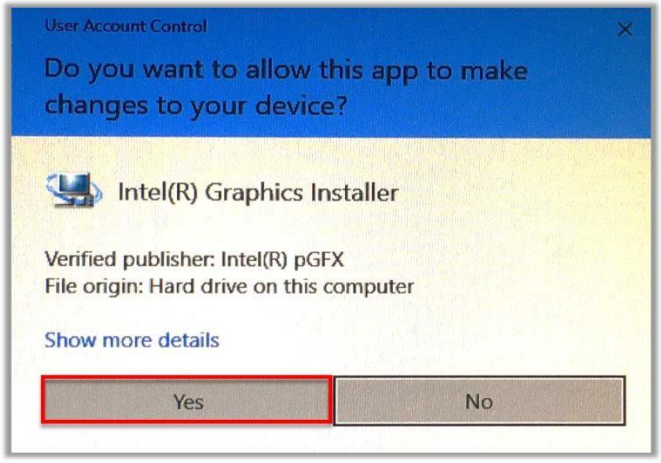
 Click **[Yes, I want to restart my computer now], [Finish]**



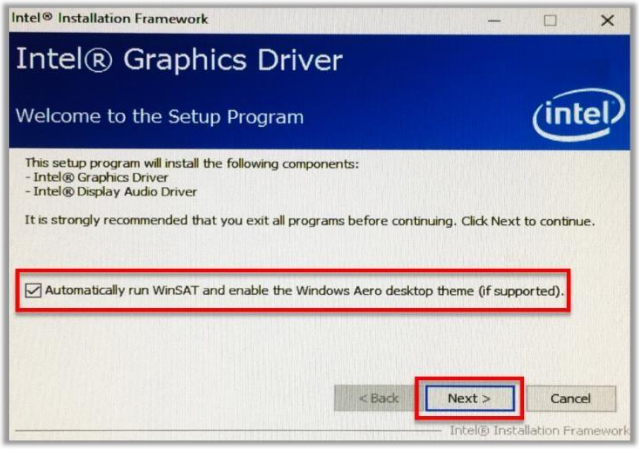
Graphics

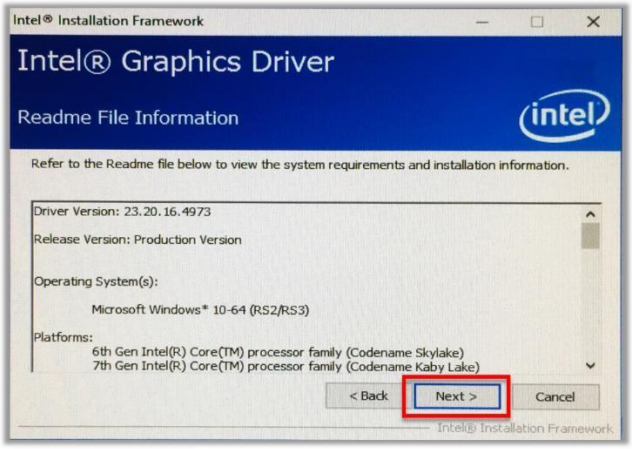
- Double Click **[igxpin]** Click **[Yes]**



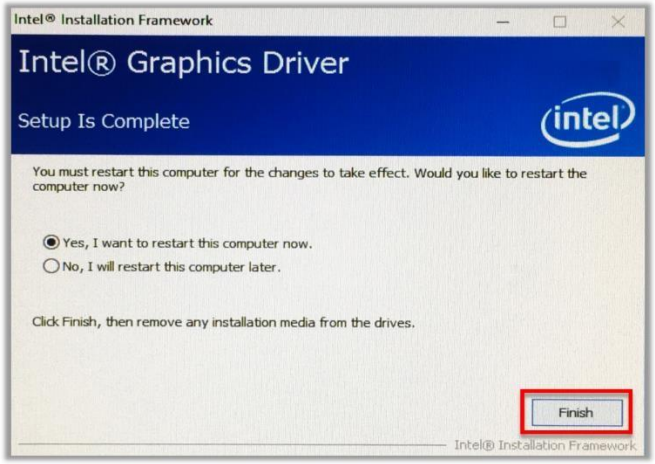


- Click **[Next]** Click **[Next]**



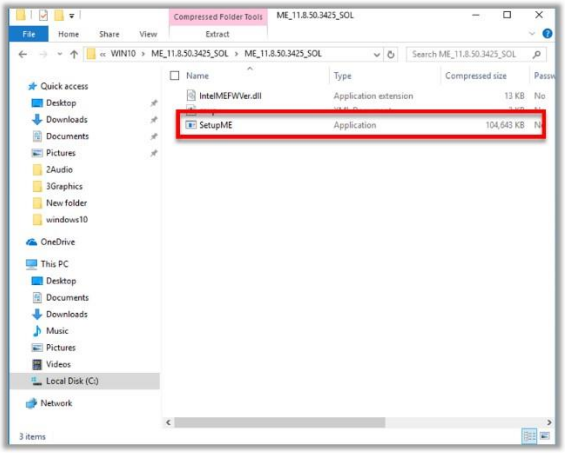


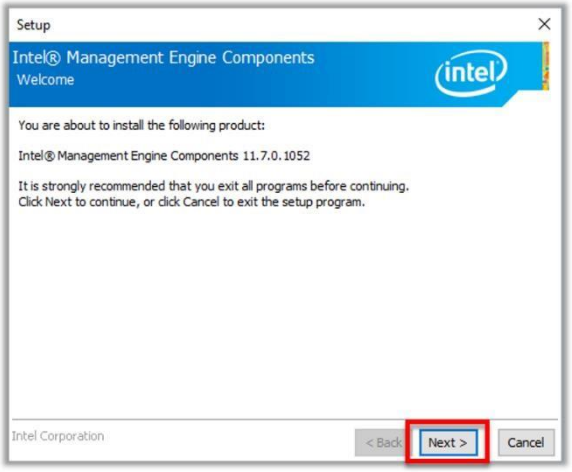
- Click **[Finish]**



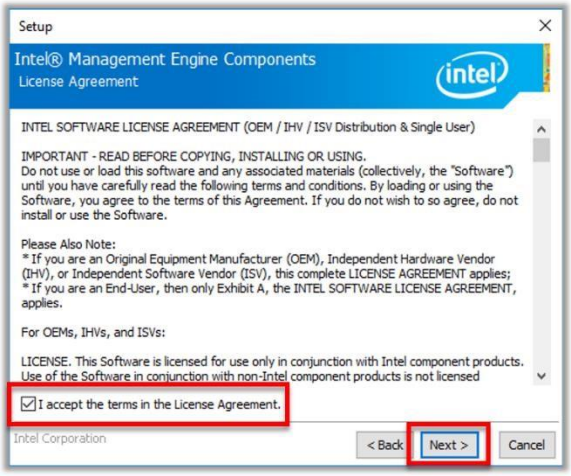
Intel® ME

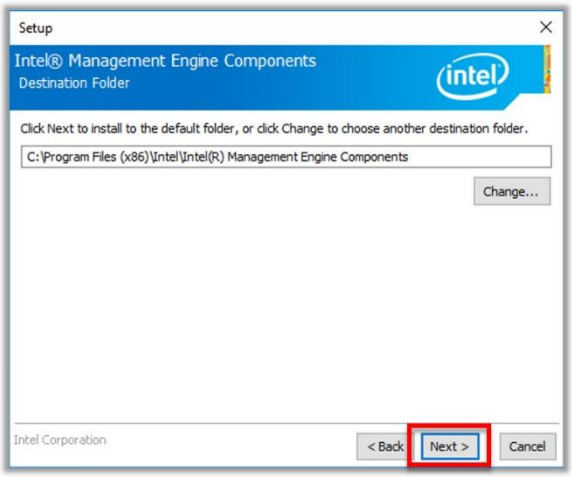
- Double Click **[SetupME]** → Click **[Next]**





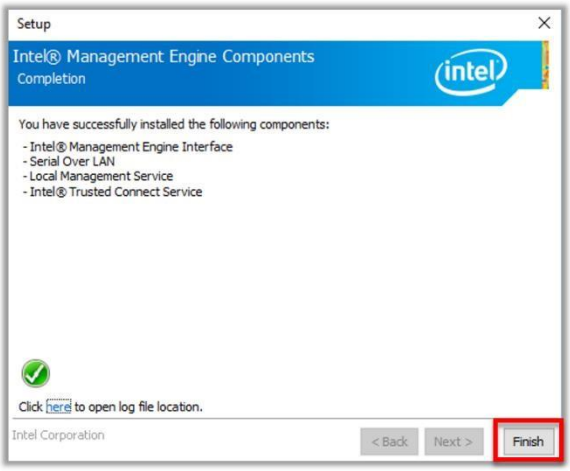
- Accept Terms then click **[Next]** → Click **[Next]**





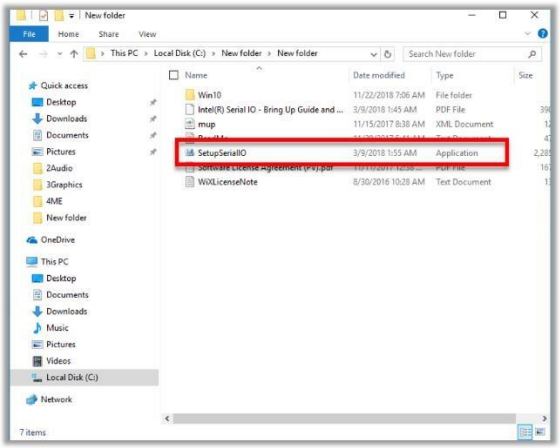
→ Click **[Yes]**

→ Click **[Finish]**

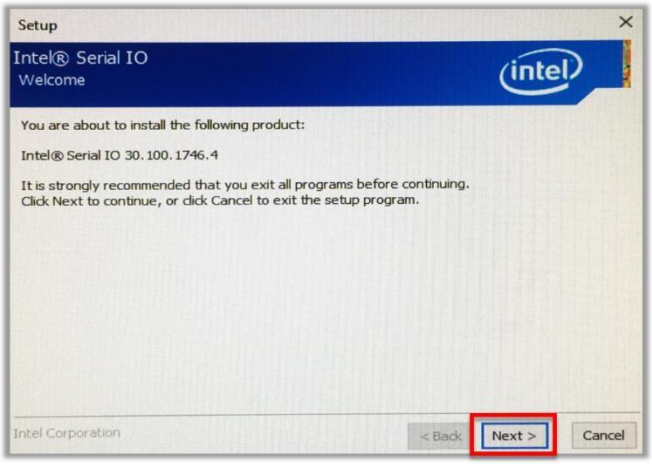


Serial IO

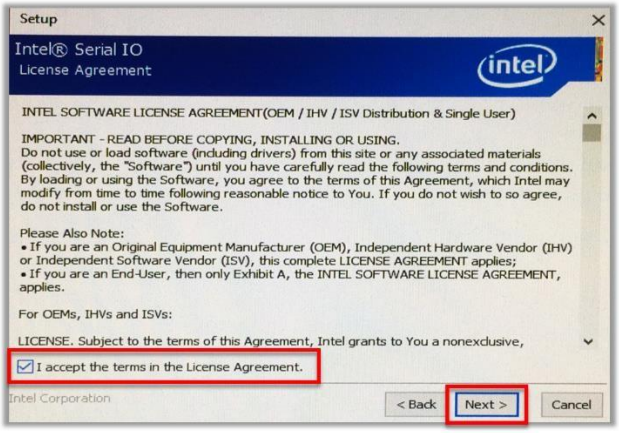
 Double Click **[SetupSerialIO]**



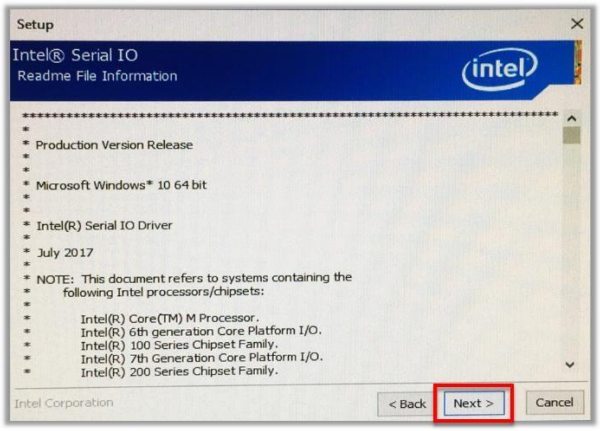
 Click **[Next]**



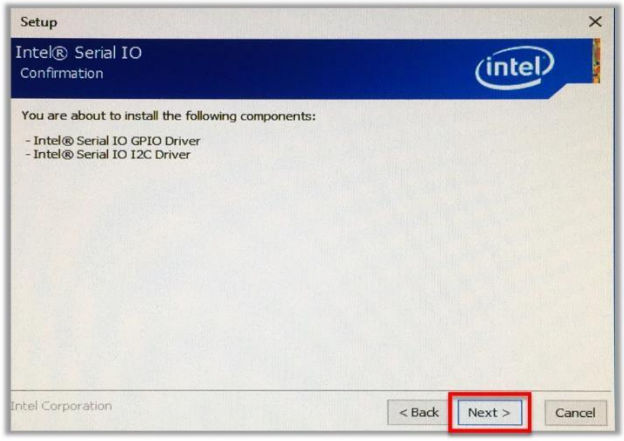
 Accept terms then click **[Next]**



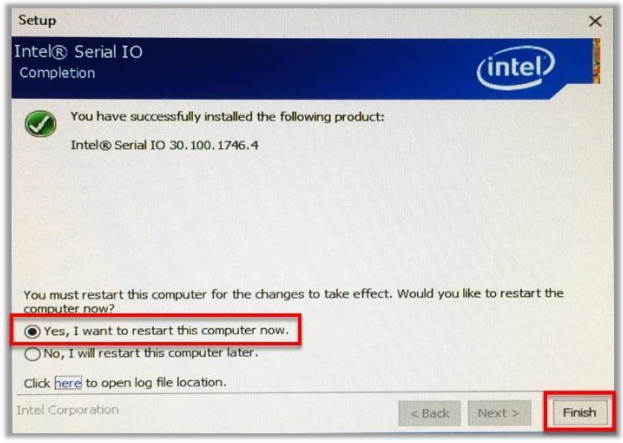
 Click **[Next]**



 Click **[Next]**

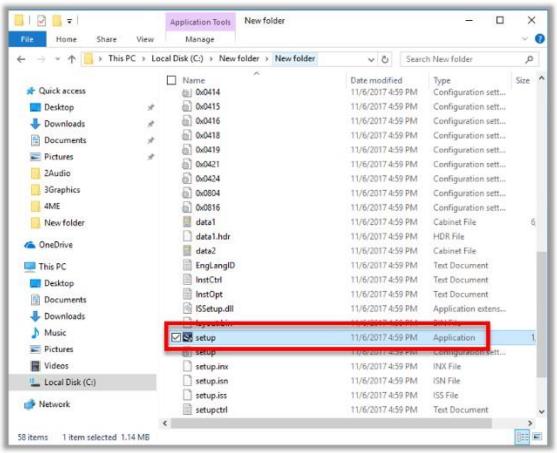


 Select Yes, I want to restart this computer now the click **[Finish]**

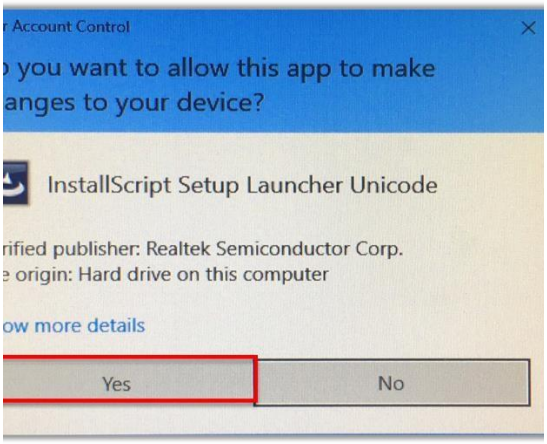


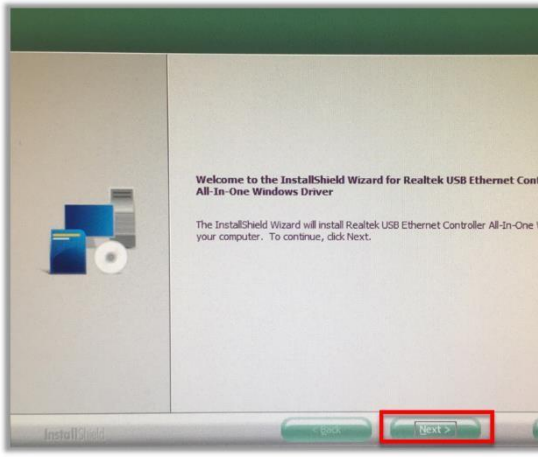
6. USB LAN

 Double Click **[Setup]**



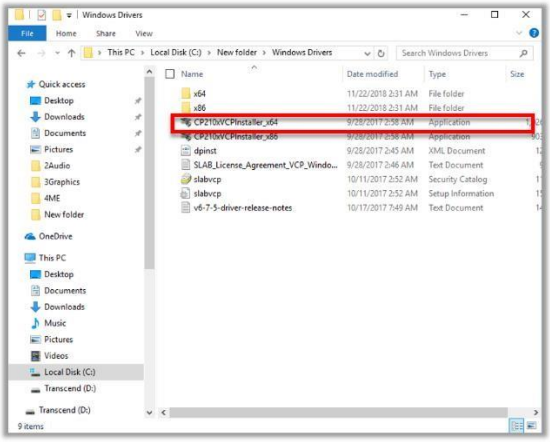
Click **[Yes]**



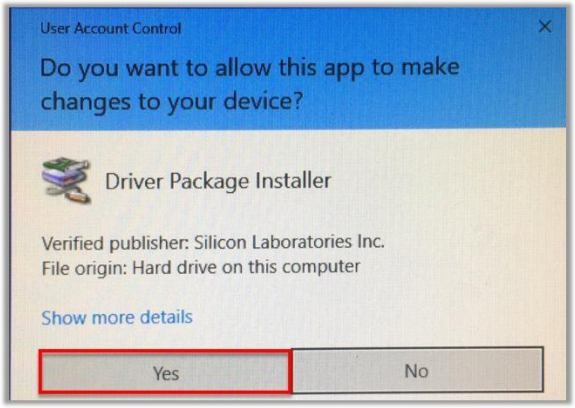


7. Windows Driver

 Double Click **[CP210XVCPInstaller_x64]**



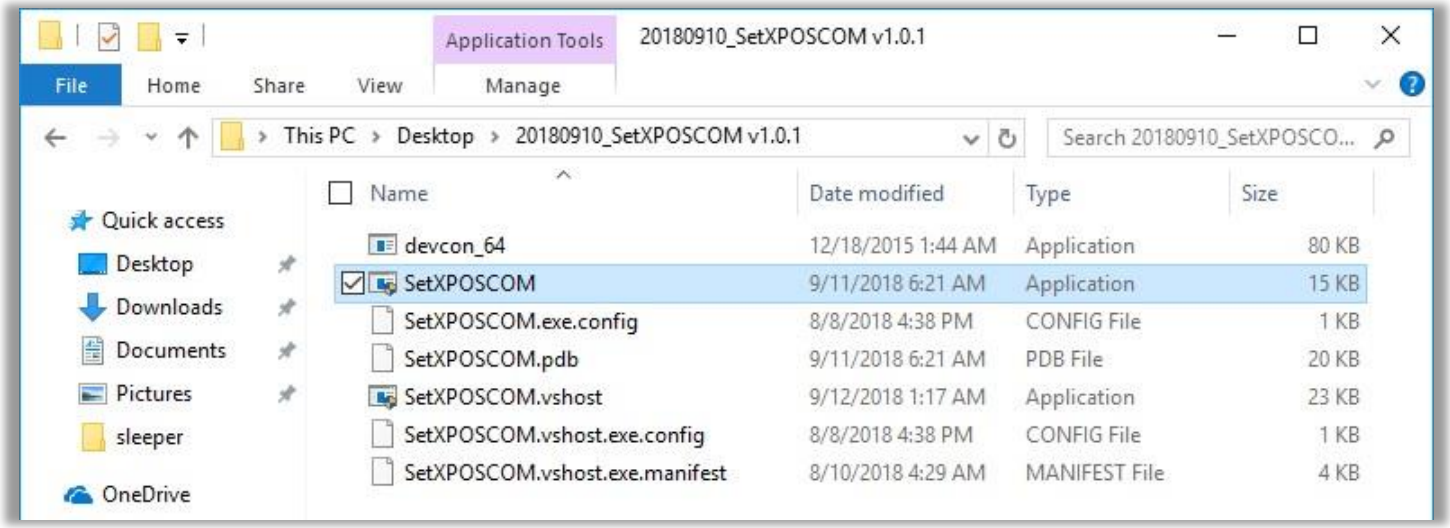
 Click **[Yes]**



8. Set COM and COM Address

1. Double click **[SetXPOSCOM]**

 You will see the COM is set in Device Manager



COM Address

You can check Ports (COM & LPT) in the Device Manager

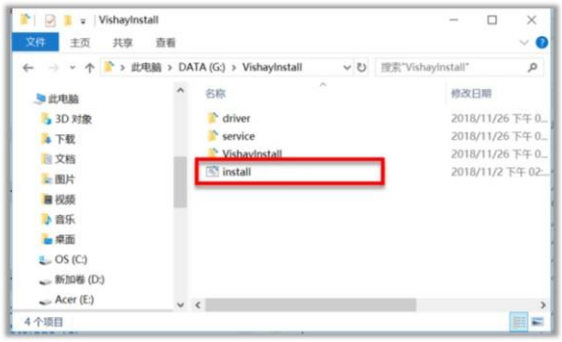
After the SET COM you should see the below items in the Device Manager:

- Communications Port (COM1): FEC DP interface reserved for customer display
- Communications Port (COM2): Reserved on motherboard

- Silicon Labs Quad CP2108 USB to UARD Bridge: Interface 0 (COM98): GPIO Control for IO1
- Silicon Labs Quad CP2108 USB to UARD Bridge: Interface 1 (COM3): RJ45 Interface on IO for Devices
- Silicon Labs Quad CP2108 USB to UARD Bridge: Interface 1 (COM4): RJ45 Interface on IO for Devices
- Silicon Labs Quad CP2108 USB to UARD Bridge: Interface 1 (COM3): RJ45 Interface on IO for Devices
- Intel iAMT: Available on i3 and i5 models

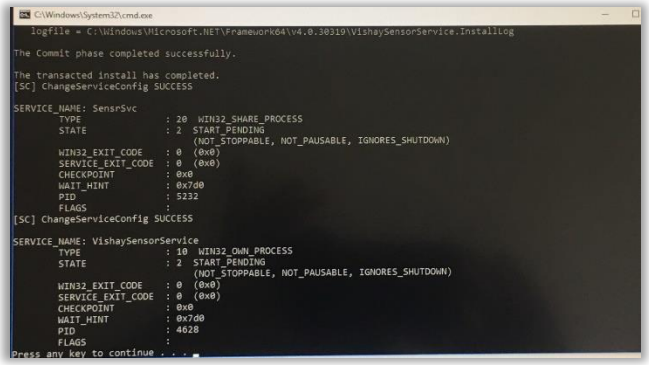
9. Sensor

Double Click **[install]**



Click **[Yes]**

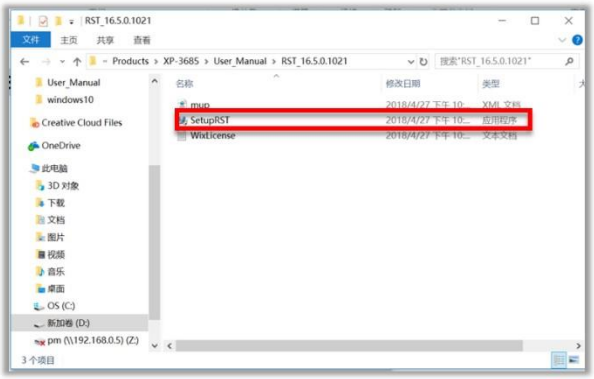
Press any key to continue ...



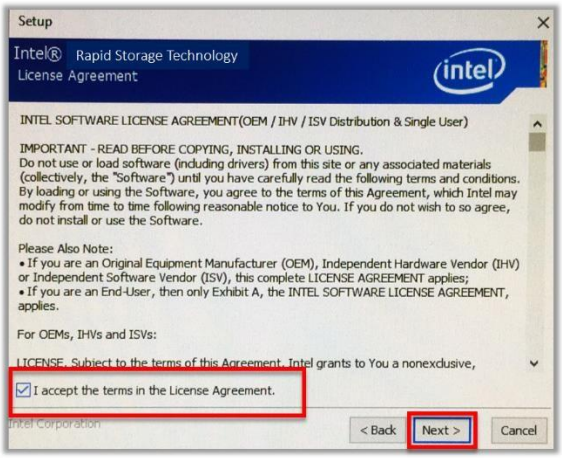
9. Intel® Rapid Storage Technology (RST)

For additional information about Intel RST: https://downloadcenter.intel.com/product/55005/IntelRapid-Storage-Technology-Intel-RST-

- Double Click **[SetupRST]** Click **[Yes]**

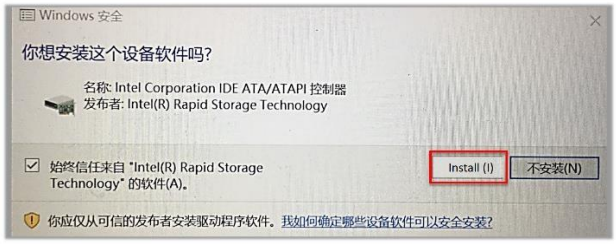


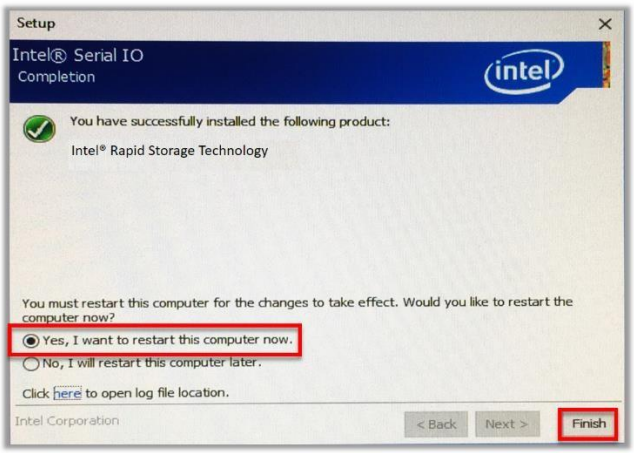
- Accept terms then click **[Next]** Click **[Next]**



- Choose save location and click **[Next]** Click **[Next]**

- Click **[Install]** Choose to restart then click **[Finish]**

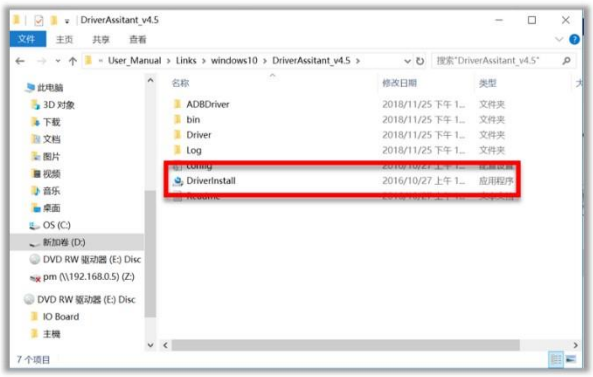




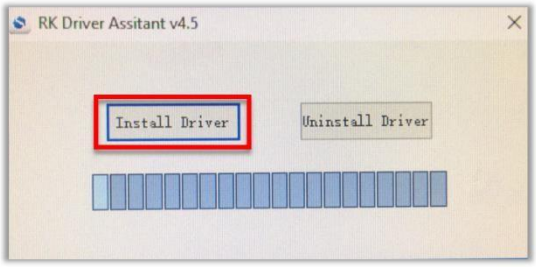
10. ADB (Android Debug Bridge)

This driver is only applicable the system on module (SOM) is installed

□ Double Click on **[DriverInstall]**



□ Click **[Install Driver]**



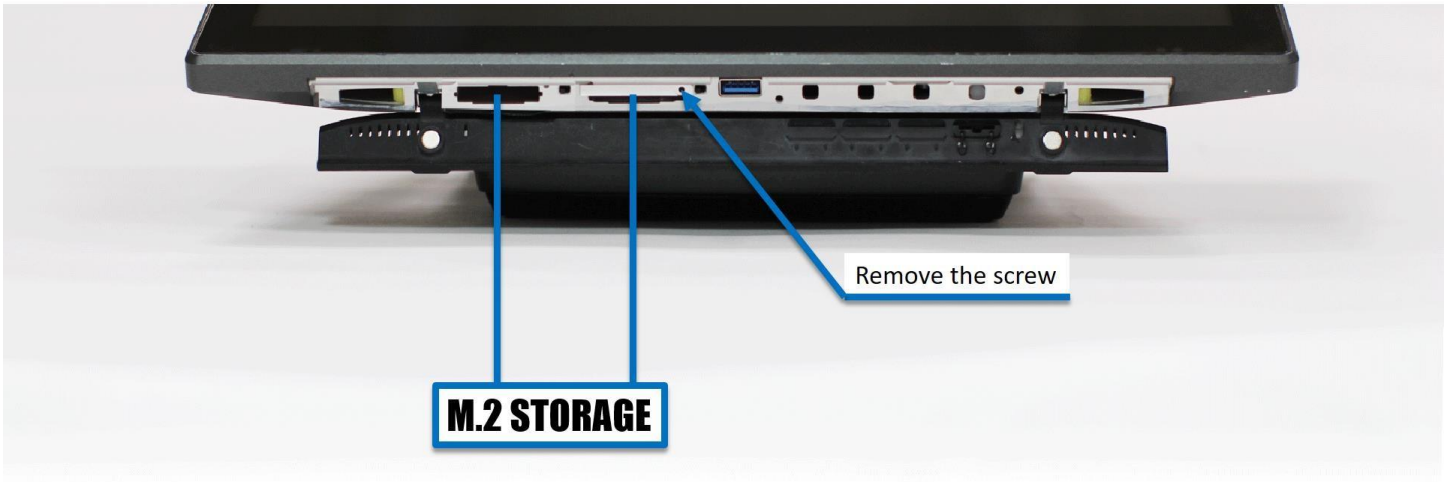
□ Click **[OK]**

Hardware

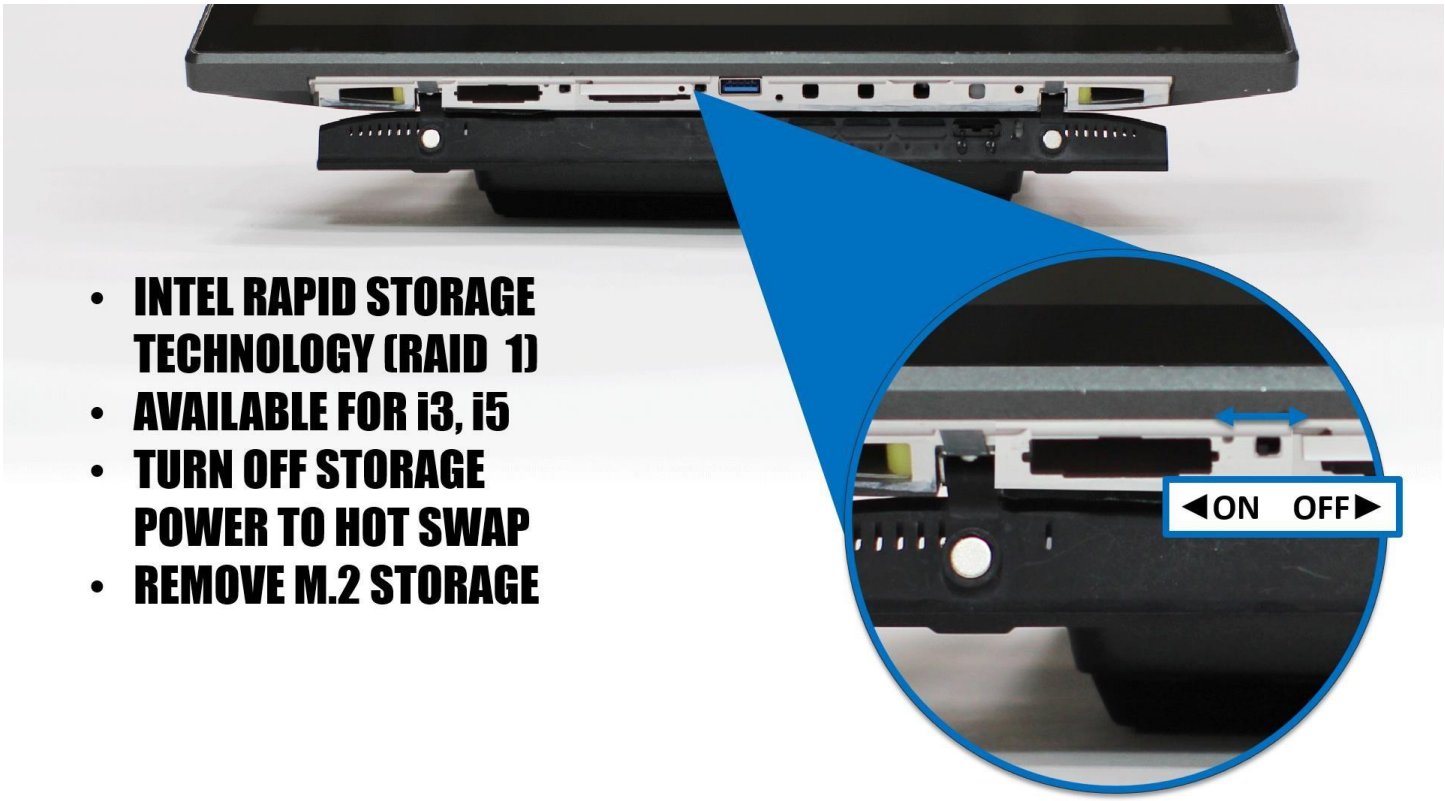
Access Storage Device

1. Make sure the system is turned off (If using RAID system can be on)
2. Flip open the cover which is held together by magnets

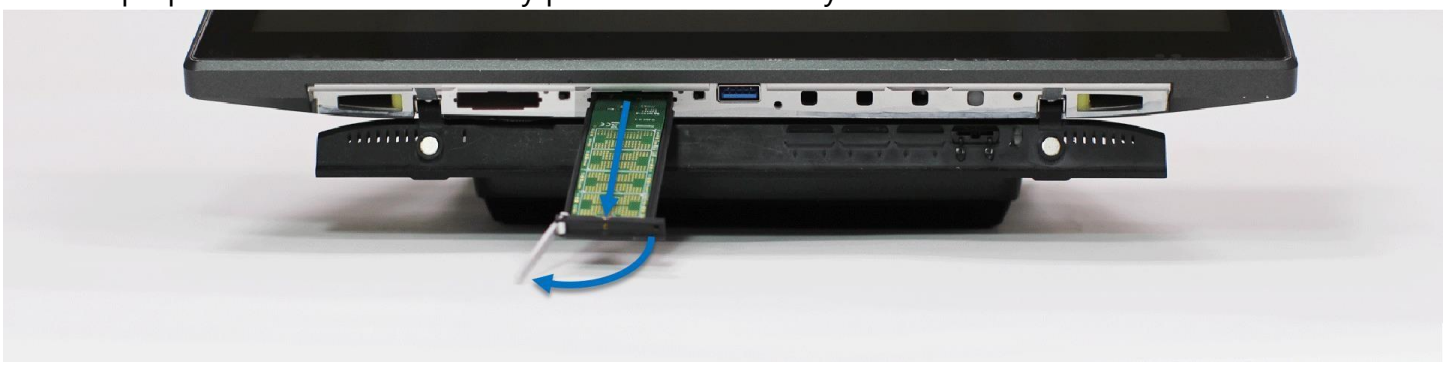
3. Remove the screw(s)



4. If you are using RAID, follow the below step. If not, skip this step. Make sure to flip the small switch to the right ► to turn off the power to the storage device. After the M.2 is inserted, flip the switch back to on ◄.

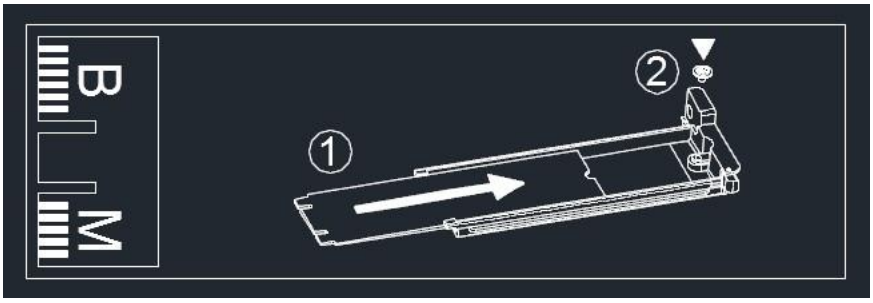


5. Flip open the handle and slowly pull out the M.2 Tray

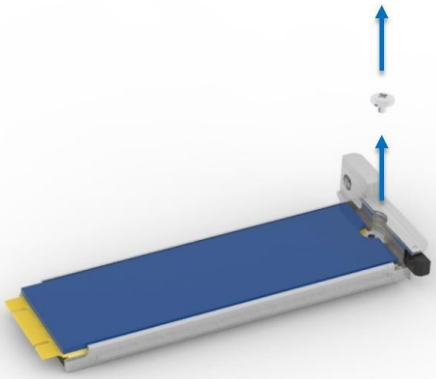


Adding or removing M.2 from tray

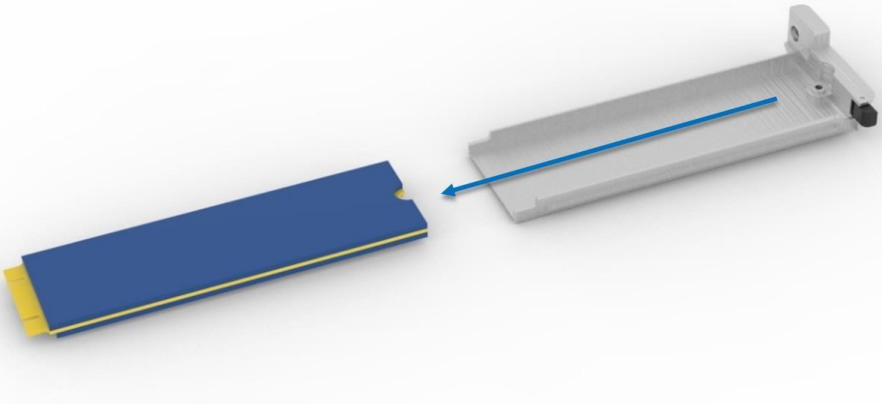
Note: This image is taped on the M.2 tray for users to identify which direction to place the M.2 module as well as how to install the M.2.



1. Remove Screw



2. Slowly slide out the M.2 module



Accessing the Motherboard

1. Remove the 3 screws as shown below

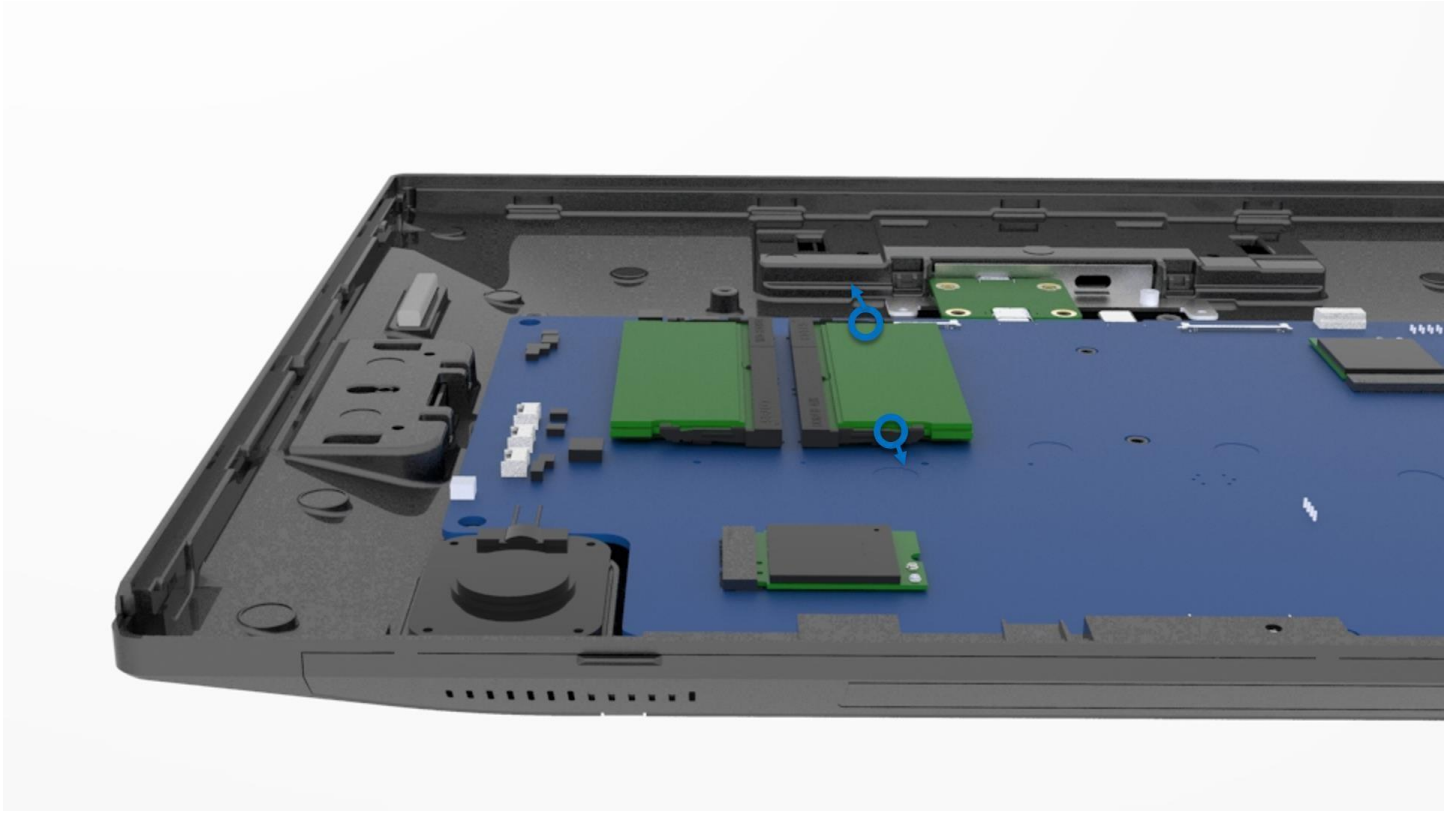
2. Remove the bottom plastic away from the aluminum chassis.

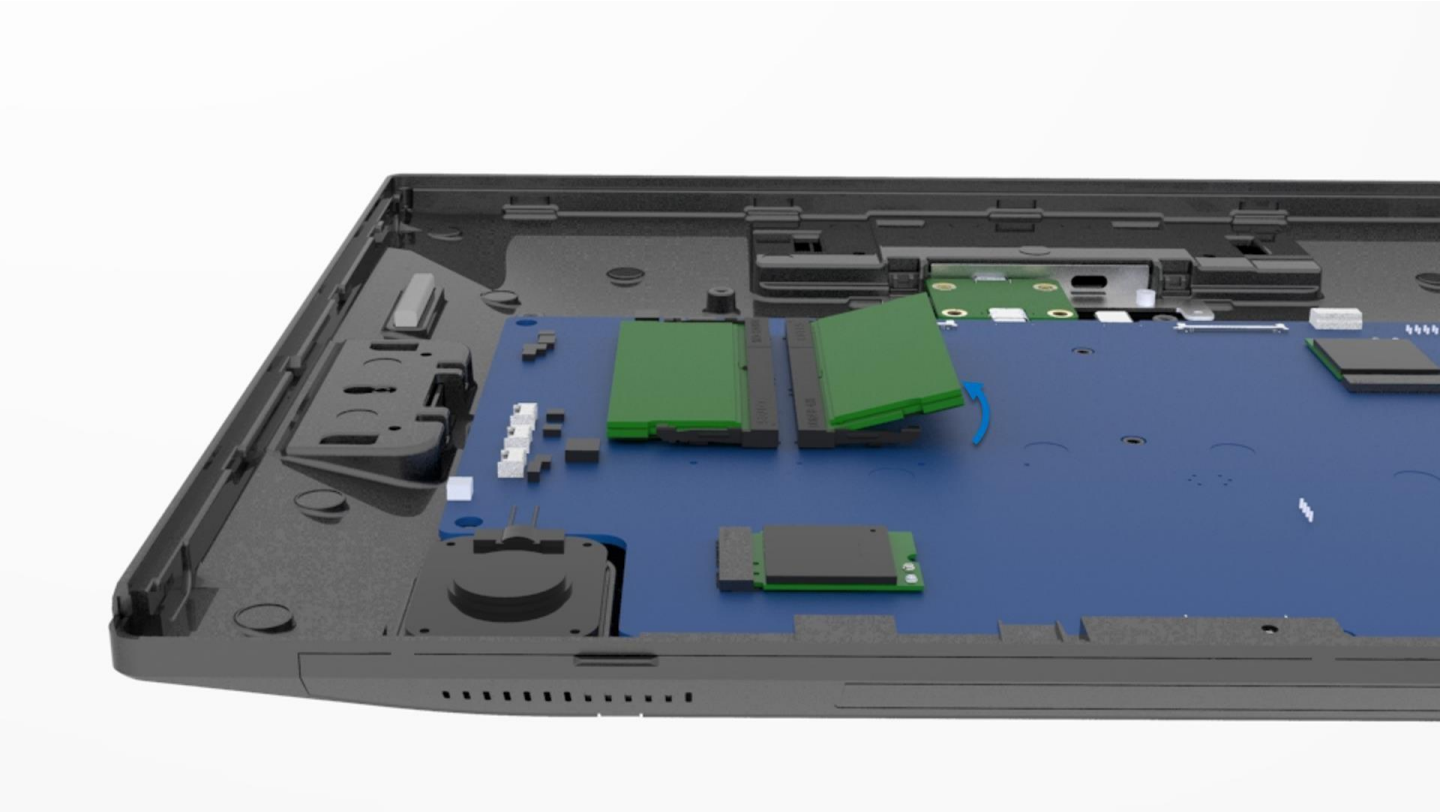
3. Slide the display module ~1cm as shown below

4. Lift the display up. There will be cables between the panel and motherboard. Reach your hand in and disconnect the cables on the motherboard.

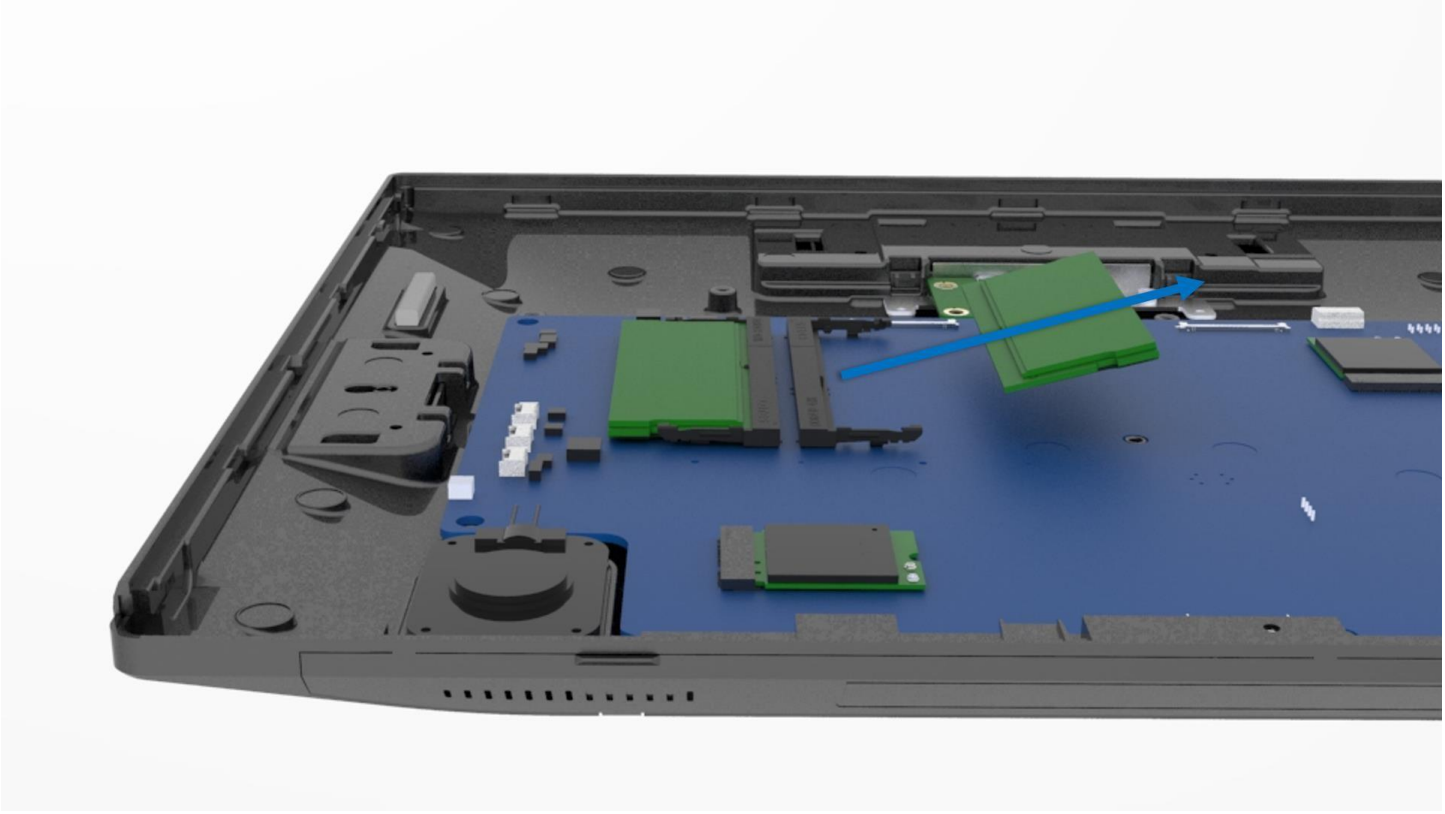
Memory

1. Slightly pull to the memory socket to the left and right. The memory will pop up





2. Remove the memory



3. IO Board SDK Instruction

1. Overview

This document describes about how to use the FEC IO Board SDK to control IO functions on FEC IO Board via serial interface. The FEC IO Board SDK support the DLL and EXE interface controlling IO on windows application

The IO functions are:

- COM A: RS232 Port A Enable/Disable, 5V/12V Setting
- COM B: RS232 Port B Enable/Disable, 5V/12V Setting
- Cash Drawer: Open, Get status
- USB Smart COM: Enable/Disable
- Reset SOM (Android System)
- Reset CCG4 USB Type-C Control IC

2. System Requirement & Installation

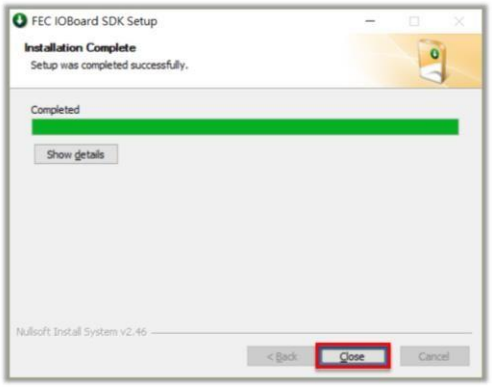
Supported Operating System(OS)
Microsoft® Windows 10 IOT Enterprise LTSB

Installation

 Click **[FECIOBoardSDKSETUP.exe]** Click **[Install]**

 Once completed, click **[Close]**

● FEC IO Board SDK will installed in C:\Program Files(x86)\FEC\IOBoardSDK



3. DLL Interface for FEC IO Board SDK

FEC IO Board SDK provide DLL interface to control IO Board, the DLL name is fec_xpos_ioboard_dll.dll.

API Functions & definitions

```
#define CTL_COM_MODE_RI                 0x00
#define CTL_COM_MODE_DC                 0x01
#define CTL_COM_PWR_5V                  0x02
#define CTL_COM_PWR_12V                 0x03
#define CTL_CASH_OUT_LOW                0x04
#define CTL_CASH_OUT_HIGH               0x05
#define CTL_CASH_OUT_HIGH_200MS         0x06
#define CTL_CASH_PWRSEL_12V             0x07
#define CTL_CASH_PWRSEL_24V             0x08
#define CTL_SOM_RST_LOW                 0x09
#define CTL_SOM_RST_HIGH                0x10
#define CTL_TYPEC_FW_RS_LOW             0x11
#define CTL_TYPEC_FW_RS_HIGH            0x12
#define CTL_SMART_COM_BYPASS            0x13
#define CTL_SMART_COM_SMART_COM         0x14
```

- DLLExport int SetComAMode(int mode)

This function enable/disable the COM Port A power supply

```
Parameter: int
mode:
CTL_COM_MODE_RI (0x00): Disable COM A Power supply
CTL_COM_MODE_DC (0x01): Enable COM A Power supply Return
Value:
Fail: 1
Success: 0
```

- DLLExport int SetComAPwr(int pwrmode)

This function set the COM Port A power level to 5V or 12V

Parameter: int
pwrmode:
CTL_COM_PWR_5V (0x02): Set to 5V
CTL_COM_PWR_12V (0x03) : Set to 12V
Return Value:
Fail: 1
Success: 0

- DLLExport int SetComBMode(int mode)

This function enable/disable the COM Port B power supply

Parameter: int
mode:
CTL_COM_MODE_RI (0x00): Disable COM B Power supply
CTL_COM_MODE_DC (0x01): Enable COM B Power supply Return
Value:
Fail: 1
Success: 0

- DLLExport int SetComBPwr(int pwrmode)

This function set the COM Port B power level to 5V or 12V

Parameter: int
pwrmode:
CTL_COM_PWR_5V (0x02): Set to 5V
CTL_COM_PWR_12V (0x03) : Set to 12V
Return Value:
Fail: 1
Success: 0

- DLLExport int SetCashDrawer1(int mode) This function opens the Cash Drawer GPIO 1

Parameter: int
mode:
CTL_CASH_OUT_HIGH_200MS (0x06): Open the cash drawer
Return Value:
Fail: 1
Success: 0

- DLLExport int SetCashDrawer2(int mode)

This function opens the Cash Drawer GPIO 2

Parameter: int
mode:
CTL_CASH_OUT_HIGH_200MS (0x06): Open the cash drawer
Return Value:
Fail: 1
Success: 0

- DLLExport int SetCashDrawerPwrSel(int pwrmode)

This function set the cash drawer power level

Parameter: int
pwrmode:
CTL_CASH_PWRSEL_12V (0x07): Set to 12V
CTL_CASH_PWRSEL_24V (0x08) : Set to 24V
Return Value:
Fail: 1
Success: 0

- DLLExport int GetCashDrawerStatus(BYTE *byStatus) This function gets the cash drawer status (Open or Close)

Parameter:
BYTE *byStatus:
*byStatus = 0: Close
*byStatus = 1: Open Return Value:
Fail: 1
Success: 0

- DLLExport int SetSomReset(int mode) This function reset the SOM Android system

Parameter:
int mode: Don't care (Set to 0~255) Return Value:
Fail: 1
Success: 0

- DLLExport int SetSmartCom (int mode)

This function set the Smart COM enable or bypass

Parameter: int
mode:
CTL_SMART_COM_BYPASS (0x13): Set Smart COM to bypass
CTL_SMART_COM_SMART_COM (0x14): Set Smart COM enable
Return Value:
Fail: 1
Success: 0

- DLLExport int GetAllStatus(BYTE *byStatus) This function gets all the GPIO pin status on IO Board

Parameter:
BYTE *byStatus: 0 = low, 1 = high byStatus [0], bit0 --> COMA_MODE byStatus [0], bit1 --> COMA_PWR byStatus [0], bit2 --> COMB_MODE byStatus [0], bit3 --> COMB_PWR byStatus [0], bit4 --> CashDrawer_GPIO0 byStatus [0], bit5 --> CashDrawer_GPIO1 byStatus [0], bit6 --> CASH_PWRSEL byStatus [0], bit7 --> CASH_IN byStatus [1], bit0 --> SOM_RST byStatus [1], bit1 --> TYPEC_FW_RS byStatus [1], bit2 --> SMART_COM Return Value:
Fail: 1
Success: 0

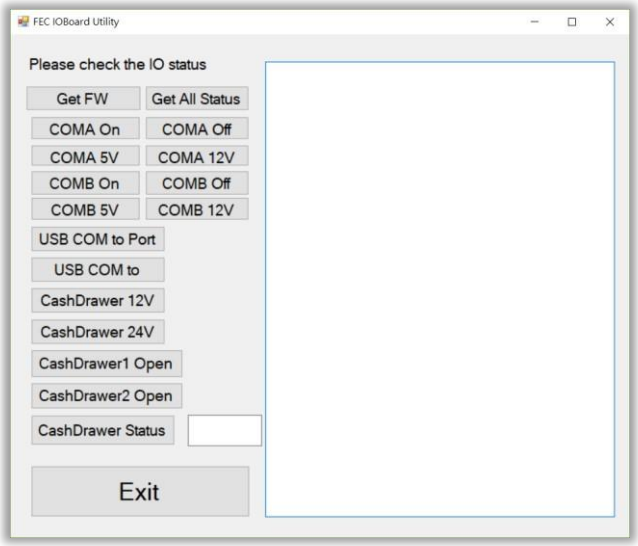
- DLLExport int GetFwVersion(BYTE * byVersion) This function gets the FW version for IO Board FW.

byStatus [0] → High byte of the FW version number byStatus [1] → Low byte of the FW version number Return Value:
Fail: 1
Success: 0

4. Example for DLL Interface

FEC_IOBoard_Utility: Please refer the sample workspace: “FEC_IOBoard_Utility” create by Visual Studio 2015 for the sample code.

The screenshot for FEC_IOBoard_Utility:



C#:
[DllImport("fec_xpos_ioboard_dll", CharSet = CharSet.Unicode)] public static extern int SetComAMode(int mode); const int CTL_COM_MODE_RI = 0x00; const int CTL_COM_MODE_DC = 0x01;
SetComAMode(CTL_COM_MODE_RI);
SetComAMode(CTL_COM_MODE_DC);

5. EXE Interface for FEC Cash Drawer SDK

Enable COM Port A Power

Set the COM Port A Power Enable with the parameter 1: “SetComAMode” & parameter 2: “enable” or “disable”

```
> FEC_XPOS_IOBoard_Tester.exe SetComAMode enable
> FEC_XPOS_IOBoard_Tester.exe SetComAMode disable
```

Enable / Disable the COM Port B Power

Set the COM Port B Power Enable / Disable with the parameter 1: “SetComBMode” & parameter 2: “enable” or “disable”

```
> FEC_XPOS_IOBoard_Tester.exe SetComBMode enable
> FEC_XPOS_IOBoard_Tester.exe SetComBMode disable
```

Set the COM Port A Power level

Set the COM Port A Power to 5v / 12v with the parameter 1: “SetComAPwr” & parameter 2: “5v” or “12v”

```
> FEC_XPOS_IOBoard_Tester.exe SetComAPwr 5v
> FEC_XPOS_IOBoard_Tester.exe SetComAPwr 12v
```

Open the Cash Drawer 1

Open the Cash Drawer 1 with the parameter 1: “SetCashDrawer1” & parameter 2: “activate”

```
> FEC_XPOS_IOBoard_Tester.exe SetCashDrawer1 activate
```

Open the Cash Drawer 2

Open the Cash Drawer 2 with the parameter 1: “SetCashDrawer2” & parameter 2: “activate”

```
> FEC_XPOS_IOBoard_Tester.exe SetCashDrawer2 activate
```

Set the Cash Drawer Power Level

Set the Cash Drawer Power Level to 12v/24v with the parameter 1: “SetCashDrawerPwrSel” & parameter 2: “12v” or “24v”

```
> FEC_XPOS_IOBoard_Tester.exe SetCashDrawerPwrSel 12v
> FEC_XPOS_IOBoard_Tester.exe SetCashDrawerPwrSel 24v
```

Get the Cash Drawer Status

Get the Cash Drawer Status with the parameter : “GetCashDrawerStatus”

```
> FEC_XPOS_IOBoard_Tester.exe GetCashDrawerStatus
CashDrawer Status  = 0 means close
CashDrawer Status  = 1 means open
```

Reset the SOM

Reset the SOM Android system with the parameter : “SetSomReset”

```
> FEC_XPOS_IOBoard_Tester.exe SetSomReset
```

Set the Smart COM enable/bypass

Set the Smart COM enable/bypass with the parameter 1: “SetSmartCom” & parameter 2: “enable” or “disable”

```
> FEC_XPOS_IOBoard_Tester.exe SetSmartCom enable
> FEC_XPOS_IOBoard_Tester.exe SetSmartCom disable
```

4. Software

Developers Guide for Serial Communications

This document is intended for developers creating products based on the CP210x USB to UART Bridge Controller. It provides information about serial communications and how to obtain the port number for a specific CP210x device. Code samples are provided for opening, closing, configuring,

reading, and writing to a COM port. Also included is a Get PortNum function that can be copied and used to determine the port number on a CP210x device by using its Vendor ID (VID), Product ID (PID), and serial number.

Opening a COM Port

Before configuring and using a COM port to send and receive data, it must first be opened. When a COM port is opened, a handle is returned by the CreateFile() function that is used from then on for all communication. Here is example code that opens COM3:

```
HANDLE hMasterCOM = CreateFile("\\\\.\\COM3",
GENERIC_READ | GENERIC_WRITE,
0,
0,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
0);
```

The first parameter in the **CreateFile()** function is a string that contains the COM port number to use. This string will always be of the form **\\\\.\\COMX** where 'X' is the COM port number to use. The second parameter contains flags describing access, which will be **GENERIC_READ** and **GENERIC_WRITE** for the example in this document, and allows both read and write access. Parameters three and four must always be 0, and the flag in parameter five must always be **OPEN_EXISTING** when using **CreateFile()** for COM applications. The sixth parameter should always contain the **FILE_ATTRIBUTE_NORMAL** flag. In addition, the **FILE_FLAG_OVERLAPPED** is an optional flag that is used when working with asynchronous transfers (this option is used for the example in this document). If overlapped mode is used, functions that read and write to the COM port must specify an OVERLAPPED structure identifying the file pointer, which is demonstrated in the sections **Purging the COM Port** and **Saving the COM Port's Original State** (more information on overlapped I/O is located at https://msdn.microsoft.com/en-us/library/windows/desktop/ms686358(v=vs.85).aspx). The seventh, and last, parameter must always be 0.
If this function returns successfully, then a handle to the COM port will be assigned to the HANDLE variable. If the function fails, **then INVALID_HANDLE_VALUE** will be returned. Upon return, check the handle and if it is valid, then prepare the COM port for data transmission.

Preparing an Open COM Port for Data Transmission

Once a handle is successfully assigned to a COM port, several steps must be taken to set it up. The COM port must first be purged and its initial state should be retrieved. Then the COM port's new settings can be assigned and set up by a device control block (DCB) structure (more information is provided on the DCB structure in the section **Setting up a DCB Structure to Set the New COM State** and at https://msdn.microsoft.com/en-us/library/windows/desktop/aa363214(v=vs.85).aspx).

Purging the COM Port

First, the COM port should be purged to clear any existing data going to or from the COM port using the **PurgeComm()** function:

```
PurgeComm(hMasterCOM, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR);
```

The first parameter in the **PurgeComm()** function is a handle to the open COM port that will be purged. The second parameter contains flags that further describe what actions should be taken. All four flags**, PURGE_TXABORT, PURGE_RXABORT, PURGE_TXCLEAR,** and **PURGE_RXCLEAR** should always be used. The first two flags terminate overlapped write and read operations, and the last two flags clear the output and input buffers.

If this function returns successfully then a non-zero value is returned. If the function fails, then it returns 0. Upon return, check the return value; if it is non-zero, continue to set up the COM port (more information on the PurgeComm() function is located at https://msdn.microsoft.com/enus/library/windows/desktop/aa363428(v=vs.85).aspx).

Saving the COM Port's Original State

Since the COM port settings can be modified to meet different needs, it is good practice to obtain the COM port's current state and store it so that when the COM port is closed, the COM port can be restored back to its original state. This can be done using the **GetCommState()** function:

```
DCB dcbMasterInitState;
GetCommState(hMasterCOM, &dcbMasterInitState);
```

The first parameter in the **GetCommState()** function is a handle to the open COM port to obtain settings from. The second parameter is an address to a DCB structure to store the COM port's settings. This DCB structure should also be used as the initial state when specifying new settings for the COM port (see section **Setting up a DCB Structure to Set the New COM State**). If this function returns successfully then a non-zero value is returned. If the function fails, then it returns 0. Upon return, check the return value; if it is non-zero, continue to set up the COM port (more information on

the **GetCommState()** function is located at
https://msdn.microsoft.com/enus/library/windows/desktop/aa363260(v=vs.85).aspx).

Setting up a DCB Structure to Set the New COM State

All of a COM port's settings are stored in a DCB structure. In section Saving the COM Port's Original State a DCB structure was retrieved that contained the initial settings of the COM port by using the **GetCommState()** function. To change a COM port's settings, a DCB structure must be created and filled out with the desired settings. Then the **SetCommState()** function can be used to activate those settings:

```
DCB dcbMaster = dcbMasterInitState;

dcbMaster.BaudRate = 57600; dcbMaster.Parity = NOPARITY; dcbMaster.ByteSize = 8;
dcbMaster.StopBits = ONESTOPBIT;

SetCommState(hMasterCOM, &dcbMaster);

Delay(60);
```

Here a new DCB structure dcbMaster has been initialized to **dcbMasterInitState**, which are the current settings of the COM port. After it has been initialized to the current settings, new settings can be assigned.

Baud Rate
The baud rate property is set to 57600 bps, but can be set to any of the baud rates supported by the CP210x. (See the current datasheet for the list of supported baud rates for the CP210x.)

Parity
The parity is set to NOPARITY, however it can also be set to ODDPARITY, EVENPARITY, SPACEPARITY, and MARKPARITY if supported by the CP210x. (See the current data sheet for the list of supported parities for the CP210x.)

Byte Size
The byte size is set to 8, so there are 8 data bits in every byte of data sent. This can also be set to 5, 6, or 7 if supported by the
CP210x. (see the data sheet for the list of supported byte sizes for the CP210x.)

Stop Bits

The stop bits are set to ONESTOPBIT, but could also be set to TWOSTOPBITS or ONE5STOPBITS (1.5). (See the current data sheet for the list of supported stop bits for the CP210x.) All combinations of data and stop bits can be used except for the combination of 5 data bits with 2 stop bits and the combination of 6, 7, or 8 data bits with 1.5 stop bits. After each of these settings is set to the desired value, the **SetCommState()** function can be called to set up the COM port. The first parameter in the **SetCommState()** function is a handle to the open COM port to change the settings on. The second parameter is an address to a DCB structure containing the COM port's new settings (more information on serial settings using DCB structures is located at https://msdn.microsoft.com/enus/library/windows/desktop/aa363214(v=vs.85).aspx

If this function returns successfully, a non-zero value is returned. If the function fails, it returns 0. Upon return, check the return value; if it is non-zero, delay for 60 ms to allow time for the settings to change and then continue to set up the COM port. This delay is not required; however, a conservative time of 60 ms is good practice to ensure that the settings are changed before any other operations take place.

Transmitting Data Across the COM Port

Once the COM port is successfully opened and configured, data can be written or read.

Writing Data

There are several things that need to happen in a write, so it is a good idea to create a function for the writes to be called whenever a write must occur. Here is an example of a write function:

```
bool WriteData(HANDLE handle, BYTE* data, DWORD length, DWORD* dwWritten)
{
	bool success = false;
	OVERLAPPED o = {0};

	o.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

	if (!WriteFile(handle, (LPCVOID)data, length, dwWritten, &o))
	{
		if (GetLastError() == ERROR_IO_PENDING)		if
		(WaitForSingleObject(o.hEvent, INFINITE) == WAIT_OBJECT_0)
			if (GetOverlappedResult(handle, &o, dwWritten, FALSE))
				success = true;
	}
	Else
		success = true;

	if (*dwWritten != length)
		success = false;

	CloseHandle(o.hEvent);

	return success;
}
```

The parameters passed in to this function are the handle to an open COM port, a pointer to an array of bytes that will be written, the number if bytes that are in the array, and a pointer to a variable to store and return the number of bytes written. Two local variables are declared at the beginning of the function: a bool named success that will store the success of the write (this is initialized to false, and only set true when the write succeeds) and an overlapped object o which is passed to the **WriteFile()** function and alerts if the transfer is complete or not (this is always initialized to {0} before the **hEvent** is assigned). Creating an event with the **CreateEvent (NULL, FALSE, FALSE, NULL)** function sets the **hEvent** property of o to prepare it to be passed to the **WriteFile()** function (more information on CreateEvent() is located at https://msdn.microsoft.com/enus/library/windows/desktop/ms682396(v=vs.85).aspx).

Next, the **WriteFile()** function is called with the handle, data, length of the data, and variable to store the amount of data that was written (more information on **WriteFile()** is located at https://msdn.microsoft.com/en-us/library/windows/desktop/aa365747(v=vs.85).aspx). If this function returns successfully, a non-zero value is returned. If the function fails, it returns 0. The if statement will determine if the write succeeded and if it did not, the last error is retrieved to see if there really was an error or the write just wasn't finished. If **ERROR_IO_PENDING** is returned then object o is then

waited on until either the write finishes or fails (if something other than **ERROR_IO_PENDING** is returned by the **GetLastError()** function, then there is the possibility of surprise removal; see section **Application Design Notes** for comments on surprise removal). When the wait is over, the result is obtained so that the amount of bytes written is updated. The success variable will then be assigned with the appropriate value, and the handle of **o.hEvent** is closed. Then the amount of bytes written is checked, and finally the function returns the success of the write, which will be true if the write successfully completed.

Reading Data

There are several things that need to happen in a read, so it is a good idea to create a function for the reads to be called whenever a read must occur. Here is an example of a read function:

```
bool ReadData(HANDLE handle, BYTE* data, DWORD length, DWORD* dwRead, UINT timeout)
{
	bool success = false;
	OVERLAPPED o = {0};

	o.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);



	if (!ReadFile(handle, data, length, dwRead, &o))
	{
		if (GetLastError() == ERROR_IO_PENDING)				if
(WaitForSingleObject(o.hEvent, timeout) == WAIT_OBJECT_0)
					success = true;
				GetOverlappedResult(handle, &o, dwRead, FALSE);
	}
	else
	success = true;

	CloseHandle(o.hEvent);

	return success;
}
```

The parameters passed in to this function are the handle to an open COM port, a pointer to an array of bytes that will be read, the number if bytes that are in the array, a pointer to a variable to store and return the number of bytes read, and a timeout value. Two local variables are declared at the beginning of the function: a bool named success that will store the success of the read (this is initialized to false, and only set true when the read succeeds), and an overlapped object o which is

passed to the **ReadFile()** function and alerts if the transfer is complete or not (this is always initialized to {0} before the **hEvent** is assigned). Creating an event with the **CreateEvent(NULL, FALSE, FALSE, NULL)** function sets the **hEvent** property of o to prepare it to be passed to the ReadFile() function (more information on **CreateEvent()** is located at https://msdn.microsoft.com/enus/library/windows/desktop/ms682396(v=vs.85).aspx).
Next, the **ReadFile()** function is called with the handle, data, length of the data, and variable to store the amount of data that was written (more information on the **ReadFile()** function is located at https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467(v=vs.85).aspx). If this function returns successfully then a non-zero value is returned. If the function fails, then it will return 0.
The if statement will determine if the write succeeded and if it didn't, the last error is retrieved to see if there really was an error or the write just wasn't finished. If **ERROR_IO_PENDING** is returned then object o is then waited on until either the write finishes or fails (if something other than **ERROR_IO_PENDING** is returned by the **GetLastError()** function, then there is the possibility of surprise removal; see section **Application Design Notes** for comments on surprise removal). When the wait is over, the result is obtained so that the amount of bytes read is updated. The success variable will then be assigned with the appropriate value, and the handle of **o.hEvent** is closed. Finally, the function returns the success of the read, which will be true if the read successfully completed.

Closing the COM Port

After all communication is finished, then the COM port should then be closed. First, the COM port should be set back to its initial state, and then the handle to the COM port should be closed and set to an invalid handle. Example code is shown below:

```
SetCommState(hMasterCOM, &dcbMasterInitState);

Delay(60);

CloseHandle(hMasterCOM); hMasterCOM
= INVALID_HANDLE_VALUE;
```

The SetCommState() function works the same as described in section **Setting up a DCB Structure to Set the New COM State**. A delay of 60 ms is used to make sure the settings have time to be set. Finally the device is closed using the **CloseHandle()** function. This function just takes in the handle of the COM port. After this function is called, it is important to set the variable to an **INVALID_HANDLE_VALUE**.

Sample Program to Demonstrate Serial Communications

Included in the AN197 software package is a directory named CP210xSerialTest which contains the source code and executables for a Visual Studio project that makes use of all the serial communication functions described in section **Preparing an Open COM Port for Data Transmission**, section **Transmitting Data Across the COM Port**, and section **Closing the COM Port**. The program is a basic dialog based application that accepts two COM port numbers, and then will send a test array of 64 bytes of data back and forth between them.

Application Design Notes

The functions used in sections Preparing an Open COM Port for Data Transmission, Transmitting Data Across the COM Port, and section Closing the COM Port are Windows COMM API functions. The examples provided are just samples of the recommended way of dealing with serial communication. For more specific information on these functions, see the MSDN website at: https://msdn.microsoft.com/enus/library/ff802693.aspx.
It should also be noted that the **SetCommState()** function does not save the settings between opening and closing the COM port. As stated before, it is good practice to get the current settings after the COM port is opened, and then restore them before it is closed. All of the functions here will return an error code. It is a good idea to nest these functions in order to catch errors if they occur by using the **GetLastError()** function. This will also solve any surprise removal problems by allowing the discovery of an invalid handle to be found and dealt with. The example application (CP210xSerialTest) has several cases that will detect surprise removal. In this example, there are checks on every function to make sure that the return code is true. If it is not, then it will display where the error occurred in the output window. As long as correct and supported settings are passed to the functions they should execute normally. Most failures can occur from having an **INVALID_HANDLE_VALUE**, however, the handles must be set to this value after a surprise removal occurs. Because regular COM ports will always be visible, then data can always be written to them successfully, even if there is no way to read it. However, because the CP210x is a virtual COM port, if the device is removed, then the handle that it uses becomes invalid when trying to write to it. If for some reason the CP210x device is unplugged the write will fail and **ERROR_OPERATION_ABORTED** will be returned by **GetLastError()**. When this happens, the handle needs to be closed and then set to **INVALID_HANDLE_VALUE**. Alternatively, a regular COM port can always be read from, but if there is no data then it will time out. When using the CP210x as the virtual COM port and it is removed before a read occurs, then the read will fail and **ERROR_ACCESS_DENIED** will be returned by **GetLastError()**.

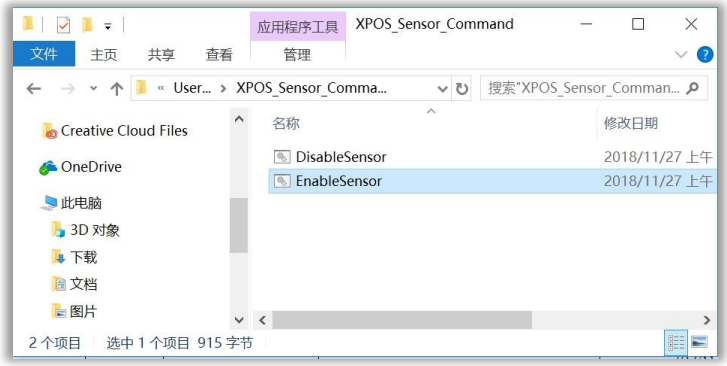
Again when this happens, the handle needs to be closed and then set to **INVALID_HANDLE_VALUE**.

Enable and Disable Ambient Light and Proximity Sensor

There are three ways to enable or disable the 2in1 sensor.

1. Utility Batch File

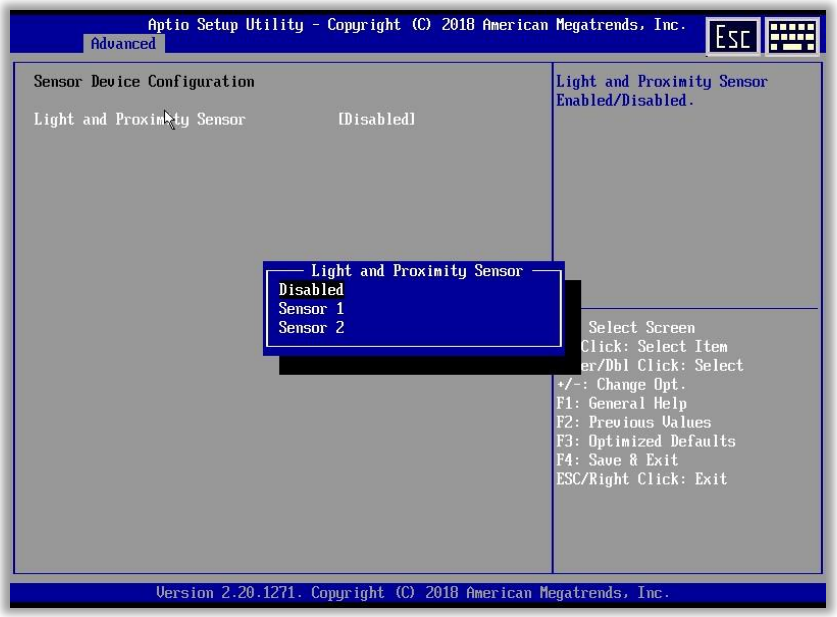
File is located in your Driver CD



2. BIOS

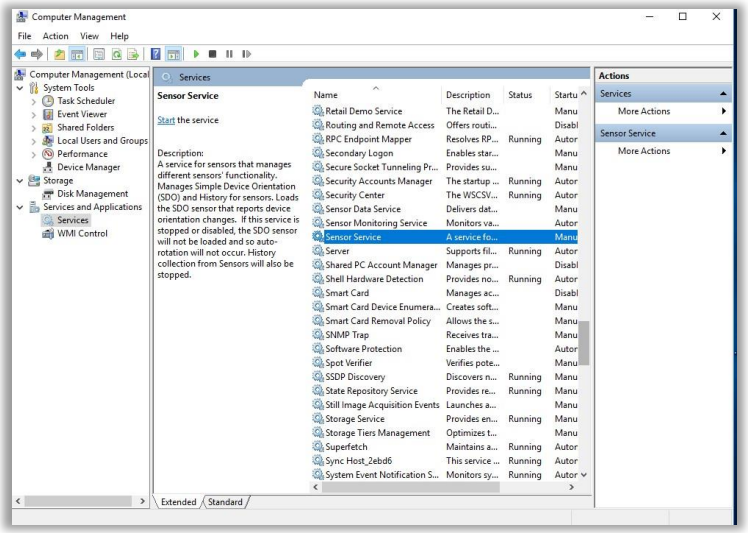
1. Start up the system
2. Press **[Delete]** during startup to enter BIOS
3. Under Advanded > Sensor Device Configuration click Disabled to turn off, Sensor 1 to turn on proximity and anmbient light sensor. Sensor 2 has no function.

Note: Sensor Service and BIOS Sensor settings both need to be on to work



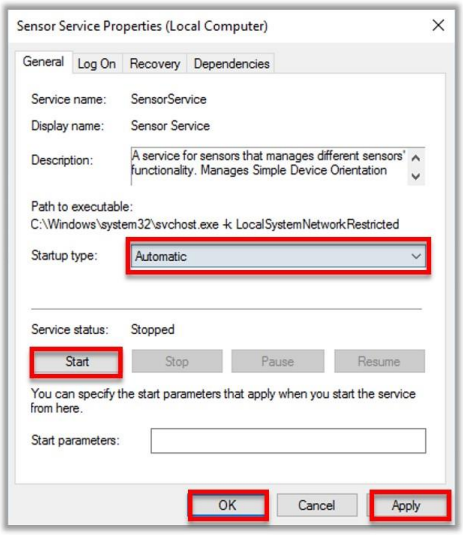
3. Windows 10 OS

1. Under Computer Management go under Services and Applications to click on **[Services]** 2. Click on **[Sensor Service]**



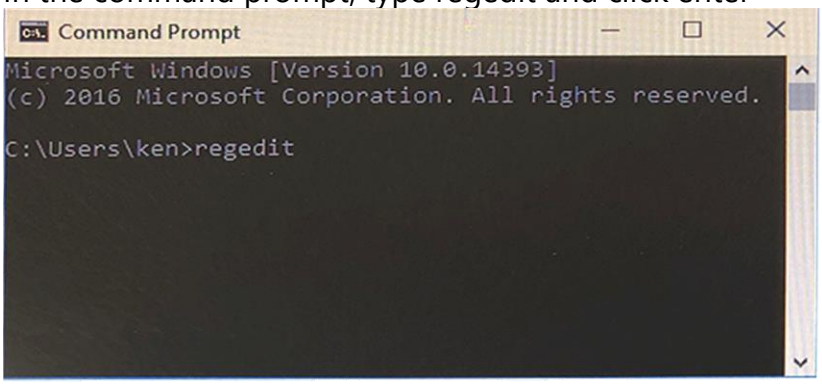
3. Sensor Services Properties make sure the "Startup type" is set to Automatic. Click

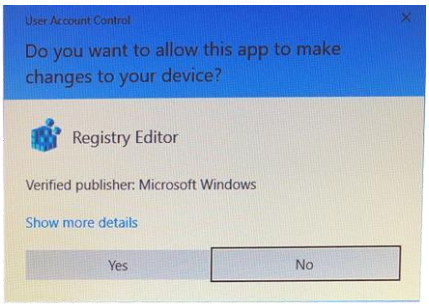
[Start] [Apply] [OK]



Control the Sensor Timing

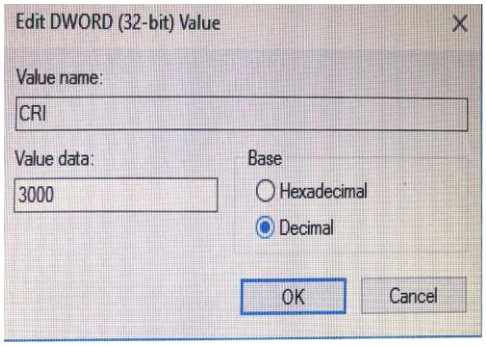
1. In the command prompt, type regedit and click enter



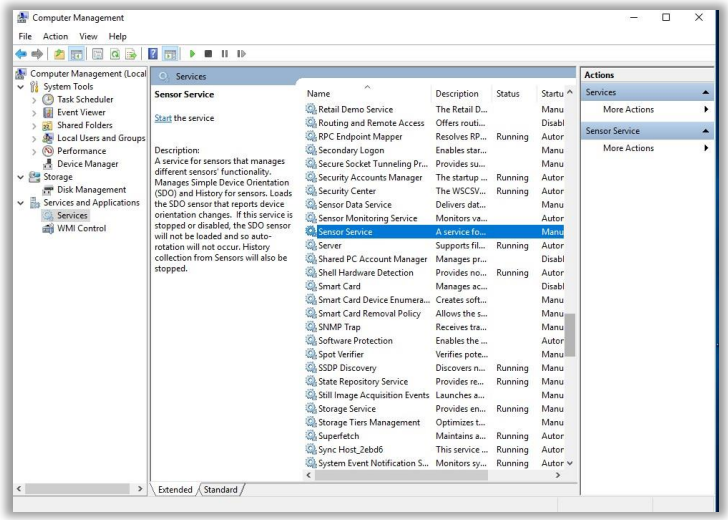


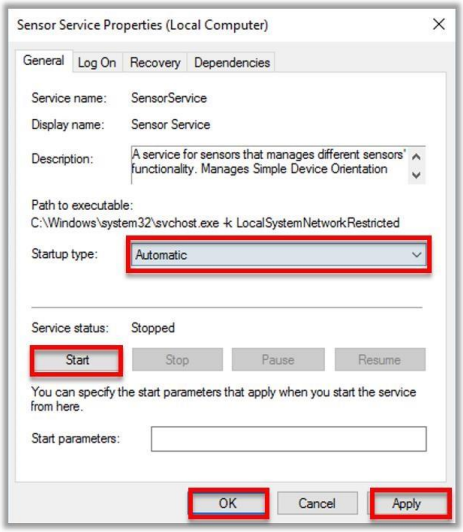
2. Locate the following folder through this path **HKEY_LOCAL_MACHINE >SOFTWARE > Microsoft > Windows NT > CurrentVersion > AdaptiveDisplayBrightness > {23B44AF278CE-4943-81DF-89817E8D23FD}**

3. Click on CRI then the radio head decimal and change the number to the desired timing (ex 3000 is approximately 3 seconds, 10000 is around 10 seconds)



4. The sensor needs to be deactivated then reactivated under Computer Management > Sensor Services





Cash Drawer Command

Note: It is recommended that developers use the chapter 3: IO Board SDK Instruction

Command:

CashDrawer output 1: A5 01 05 02 6B
Return
A5 01 08 00 5B is Open
A5 01 08 01 5B is Closed

CashDrawer output 2: A5 01 06 02 6B
Return
A5 01 08 00 6B is Open
A5 01 08 01 6B is Closed