Tinkerer Level   Engineer Level

Installation Guide   v3.14   March 2025

The sensorCAM is an advanced concept foreign to most model railroaders and consequently deals with many unfamiliar issues. It also calls on multiple software applications to address them. The initial software setup for an inexperienced tinkerer is consequently involved and needing perseverance. The Arduino IDE, the ESP32 libraries, the Processing 4 App, the EX-Command Station and EX-rail (if used) combine to create a unique system supporting the sensorCAM's own software. Any prior knowledge of these will vastly expedite the installation.

Outline:

 This installation process is divided into 12 steps outlined below. It is recommended that it be treated as a learning exercise and that testing of progress occur after each major step. For example, familiarization with Arduino IDE should be developed before moving to the ESP32 example, and then familiarize with the ESP32- CAM before moving on to the sensorCAM loading.

As the sensorCAM is not for Conductors, and perhaps only advanced Tinkerers, it is anticipated that users will already be familiar with the Arduino IDE and maybe already have toyed with the ESP32-CAM.   We have not therefore included much detail on the first few steps but rather referred to existing documentation elsewhere (e.g. Arduino & youtube). Most detail is provided in steps 4 to 8 at this stage.

The chapters of the full manual may be referred to where additional detail is sought. The 12 steps are:

1.   Check youtube video then acquire ESP32-CAM and MB/FTDI
2.   install Arduino IDE with the ESP32 libraries (includes a WiFi CAM example)
3.   Load the CAM with the WiFi example and test.
4.   Install the *sensorCAM.ino* software and configure.
5.   Test the basic sensorCAM functions to confirm functional install
6.   Load Processing 4 app. and *sensorCAM.pde* code
7.   Use the Processing 4 app to replace the Arduino Monitor. Further familiarise and test.
8.   With a fixed camera, create test sensors and test detection with moving targets.
9.   Setup the CAM viewing the model railroad and test a virtual sensor with moving rolling stock.
10.   Optimise parameters for best performance.
11.   Connect CAM to an i2c interface (e.g. PCA9515A or better)
12.   Depending on system, integrate sensorCAM into Command Station using appropriate code.

Note: Check version of your CH340 driver in **Device Manager – Ports**. Change if issues arise**.**

USB-SERIAL CH340 (COM6)

Driver Provider:	wch.cn
Driver Date:	30/01/2019
Driver Version:	3.5.2019.1

Recommended system USB driver is 3.5.2019.1

See Drivers for the CH340 for detailed instructions.

Step 1. ESP-32 CAM

To understand the animal, watch the video at [ESP32 CAM - 10 Dollar Camera for IoT Projects - YouTube](#)

Be aware that you need a USB adapter and the ESP32-CAM-MB is preferred as it avoids hazards associated with a "wired" FTDI power where errors can easily result in CAM destruction. Programming is also easier. Acquire a CAM from your favourite source. Also be prepared for defective product so a spare CAM may be worthwhile. I have had 2 faulty ov2640 modules. ESP32-CAM version 1.9 is preferred (has GND/Reset pin) Purchasing ESP32-CAM-MB as a pair (MB & CAM) is safest. Purchasing TWO pairs is insurance.

Step 2. Install the Arduino IDE

If you are not already familiar with this IDE (Integrated Development Environment), rather than go into details that are already covered in great detail on the Arduino web page, just follow the instructions in the following link and then return here.

[Arduino IDE Guide](#)

## Step 2A. Install the IDE ESP32 libraries (including for ESP32-CAM).

If you are new to ESP32, the board manager for ESP32 must be loaded. This is explained in the video at the **10minute** mark. **Note: WE *MUST* USE esp32 by Espressif <u>Select version 2.0.17</u> & <u>Install</u>, NOT latest (3.1.x)**

[Introduction to ESP32 - Getting Started - YouTube](#)

Step 3. Load and test the ESP32-CAM example

Select the tools - board manager and the AI thinker ESP32 CAM. Then run the example as per youtube at **9:30 minutes** in. Run the ESP32 - camera - webserver example. Note: wifi uses 2.4GHz NOT 5GHz wifi.

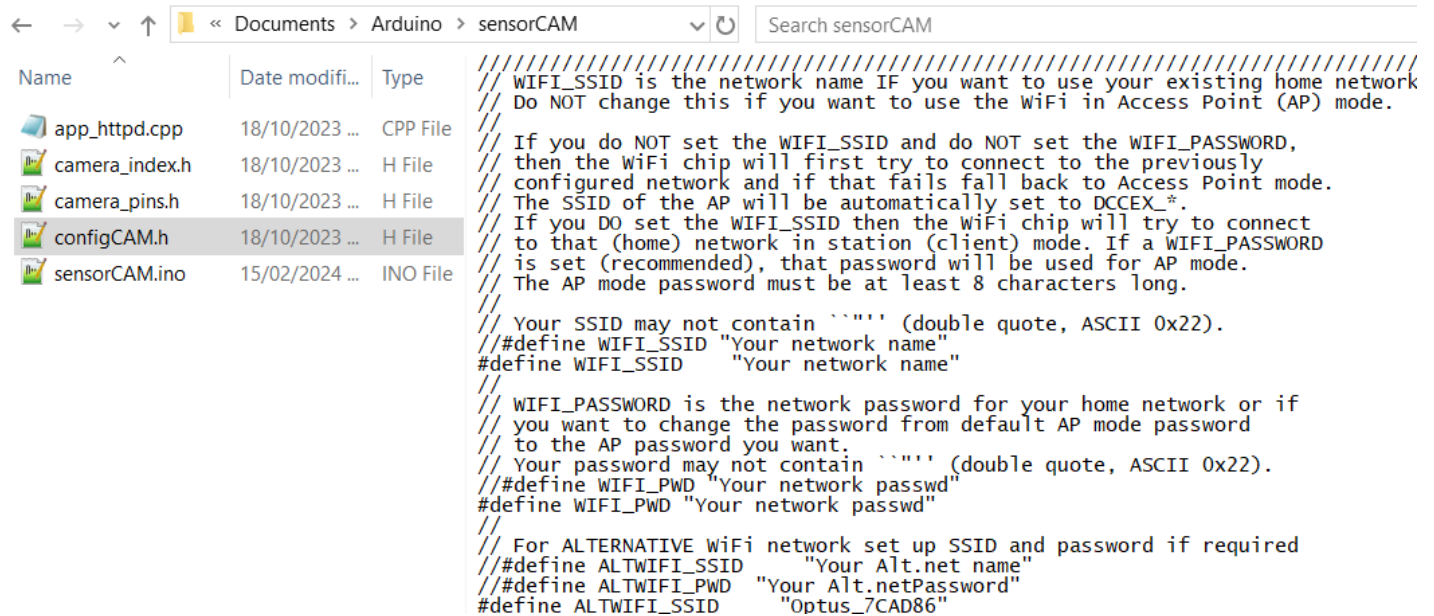[ESP32 CAM - 10 Dollar Camera for IoT Projects - YouTube](#)

Note: With the MB board there is a push button (IO0) that replaces the FTDI programming link. Hold the button in when powering up the CAM to initiate programming mode. With later versions of exp32-CAM the IO0 push button on the MB may not be needed. Ensure a Huge Partition Scheme is selected.
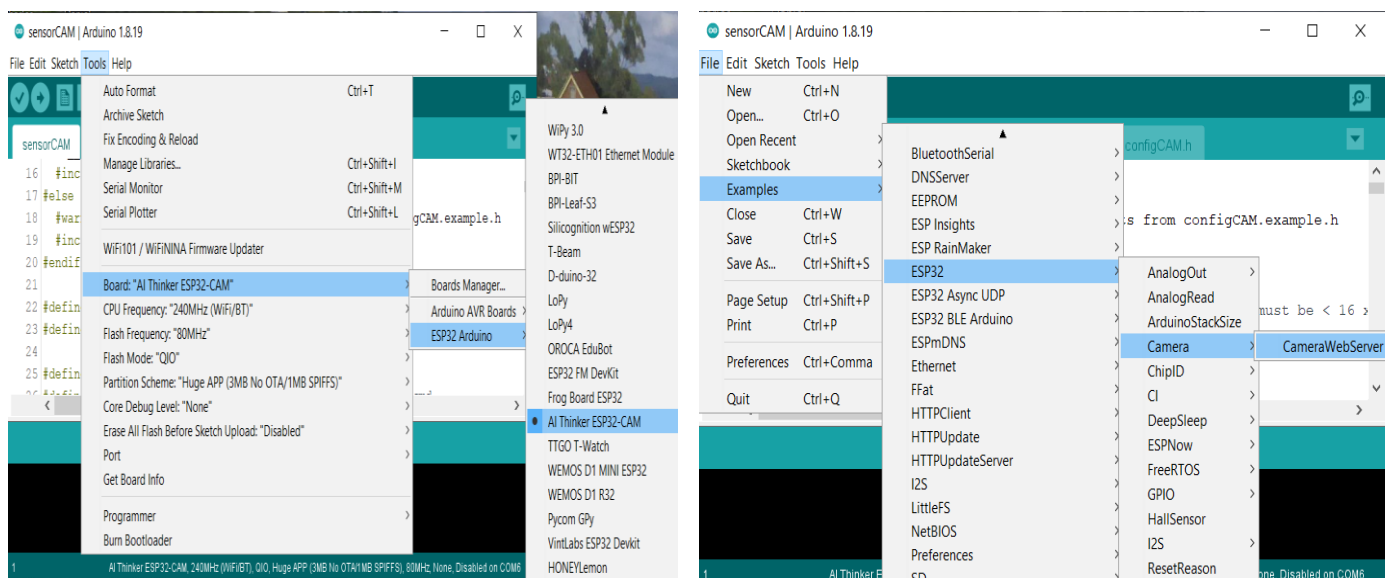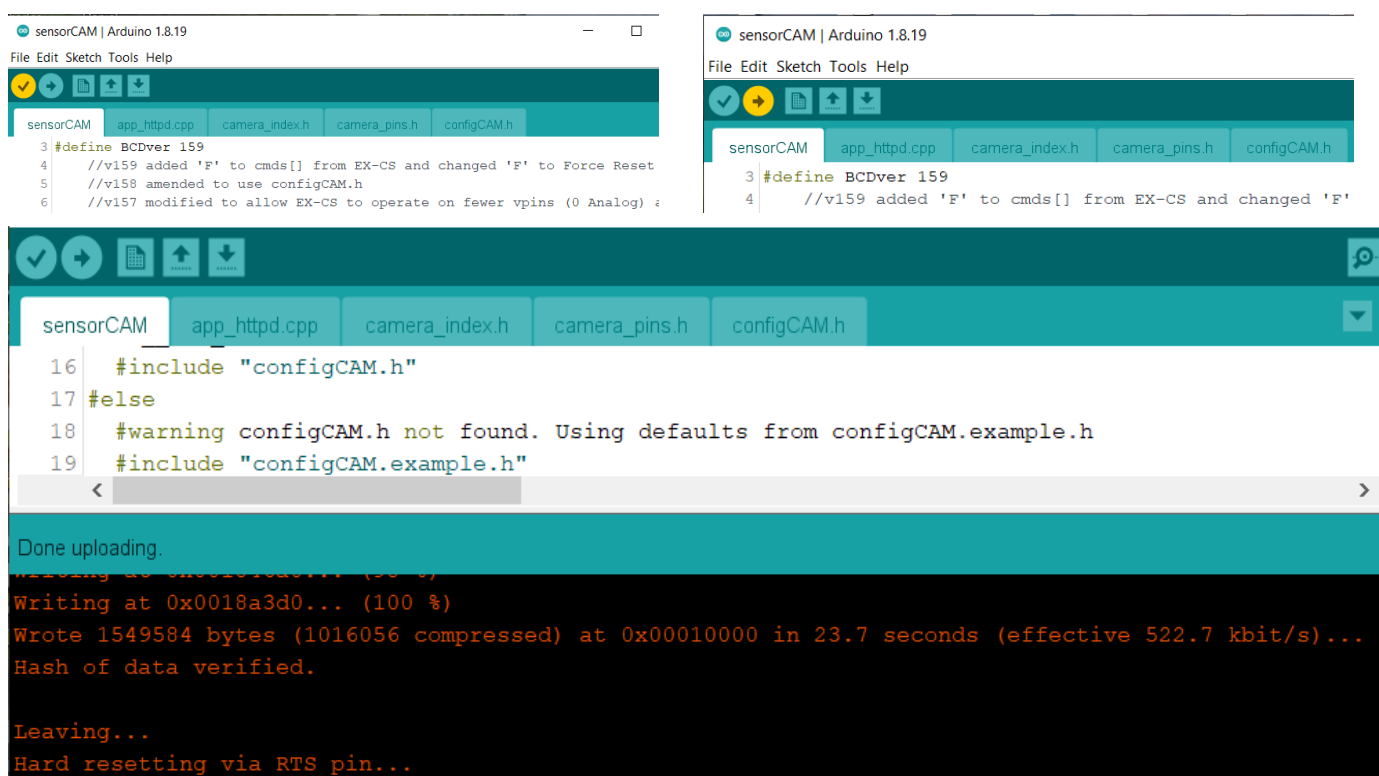
Step4. Install the sensorCAM

4.1 Download the software zip to your PC from https://github.com/DCC-EX/EX-SensorCAM
(for the latest parser version, rather than "main", select the later "devel" software v3.14 or higher)

4.2 Right click the zip and click Extract all. Browse to your Arduino sketches folder e.g. *Documents/Arduino*

4.3 Click Extract, creating new folder e.g. *Documents/Arduino/EX-SensorCAM-main*

4.4 Rename *EX-SensorCAM-main* to *sensorCAM*

The essential files for now are shown below. Other files may be used in later install steps.

Name	Date modifi...	Type
app_httpd.cpp	18/10/2023 ...	CPP File
camera_index.h	18/10/2023 ...	H File
camera_pins.h	18/10/2023 ...	H File
configCAM.h	18/10/2023 ...	H File
sensorCAM.ino	15/02/2024 ...	INO File

```
///////////////////////////////////////////////////////////////////////////////
// WIFI_SSID is the network name IF you want to use your existing home network
// Do NOT change this if you want to use the WiFi in Access Point (AP) mode.
//
// If you do NOT set the WIFI_SSID and do NOT set the WIFI_PASSWORD,
// then the WiFi chip will first try to connect to the previously
// configured network and if that fails fall back to Access Point mode.
// The SSID of the AP will be automatically set to DCCEX_*.
// If you DO set the WIFI_SSID then the WiFi chip will try to connect
// to that (home) network in station (client) mode. If a WIFI_PASSWORD
// is set (recommended), that password will be used for AP mode.
// The AP mode password must be at least 8 characters long.
//
// Your SSID may not contain ``"'' (double quote, ASCII 0x22).
//#define WIFI_SSID "Your network name"
#define WIFI_SSID    "Your network name"
//
// WIFI_PASSWORD is the network password for your home network or if
// you want to change the password from default AP mode password
// to the AP password you want.
// Your password may not contain ``"'' (double quote, ASCII 0x22).
//#define WIFI_PWD "Your network passwd"
#define WIFI_PWD "Your network passwd"
//
// For ALTERNATIVE WiFi network set up SSID and password if required
//#define ALTWIFI_SSID    "Your Alt.net name"
//#define ALTWIFI_PWD  "Your Alt.netPassword"
#define ALTWIFI_SSID    "Optus_7CAD86"
```
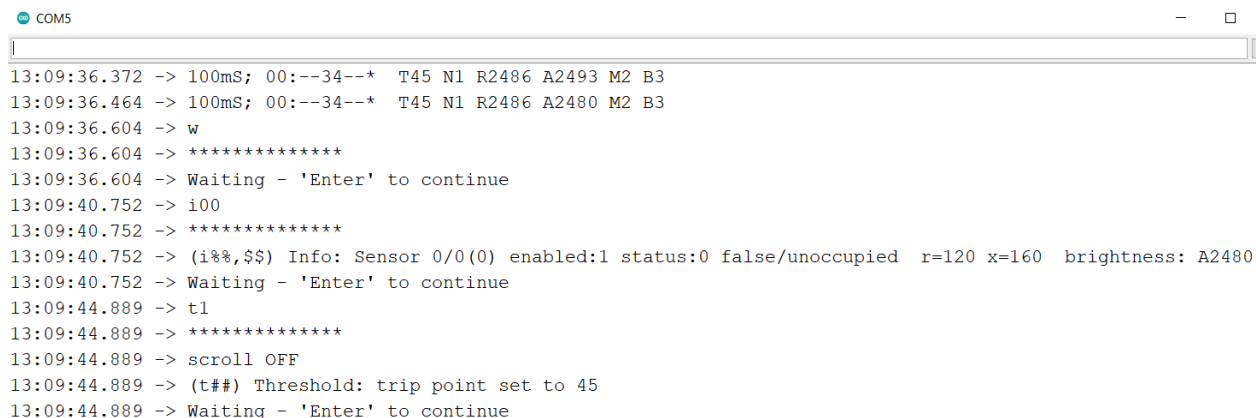
4.5 Using the *configCAM.example.h* as a guide, create and edit a *configCAM.h* to reflect your WiFi network name and password. Then with the correct com port and board selected, compile the *sensorCAM.ino* sketch. If all is well, load it into the CAM (replacing the ESP32 WiFi example)
Note: It may be necessary to use the MB IO0 button (or GPIO0 link) to get the CAM into upload mode.
(refer back to video and/or Appendix below)

```
sensorCAM | Arduino 1.8.19
File Edit Sketch Tools Help

sensorCAM   app_httpd.cpp   camera_index.h   camera_pins.h   configCAM.h
 3 #define BCDver 159
 4    //v159 added 'F' to cmds[] from EX-CS and changed 'F' to Force Reset
 5    //v158 amended to use configCAM.h
 6    //v157 modified to allow EX-CS to operate on fewer vpins (0 Analog) a
```

```
sensorCAM | Arduino 1.8.19
File Edit Sketch Tools Help

sensorCAM   app_httpd.cpp   camera_index.h   camera_pins.h   configCAM.h
 3 #define BCDver 159
 4    //v159 added 'F' to cmds[] from EX-CS and changed 'F'
```

```
sensorCAM   app_httpd.cpp   camera_index.h   camera_pins.h   configCAM.h
16    #include "configCAM.h"
17 #else
18    #warning configCAM.h not found. Using defaults from configCAM.example.h
19    #include "configCAM.example.h"
```

```
Done uploading.

Writing at 0x0018a3d0... (100 %)
Wrote 1549584 bytes (1016056 compressed) at 0x00010000 in 23.7 seconds (effective 522.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```
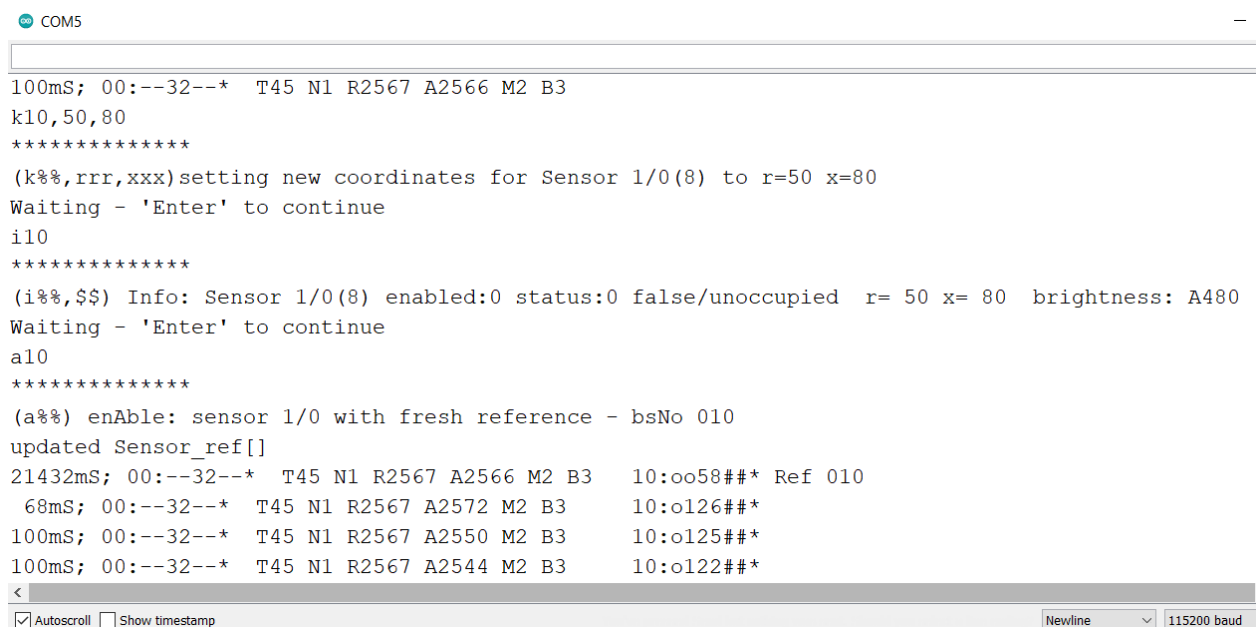
Step 5. Test the basic sensorCAM functions

The sensorCAM has a suite of command codes with which you will need familiarity. Refer to the manual summary Appendix A or <u>Chapter 5</u> (Configuration) for more detailed description. For now, open the Arduino monitor and the initialization procedure should scroll out. It requires that the monitor is set for "115200 baud" and "NewLine" (CR is unacceptable).

After about 15 seconds you can enter the 'w' (wait) command into the monitor and the scrolling should pause. Another command such as 'i00' will give a relevant command response, or **Enter** alone will resume scrolling tabular output. The sensors still need configuring before they can work. Command 't1' will toggle the scrolling status data ON or OFF.

```
● COM5                                                                    —    □

|
13:09:36.372 -> 100mS; 00:--34--*   T45 N1 R2486 A2493 M2 B3
13:09:36.464 -> 100mS; 00:--34--*   T45 N1 R2486 A2480 M2 B3
13:09:36.604 -> w
13:09:36.604 -> **************
13:09:36.604 -> Waiting - 'Enter' to continue
13:09:40.752 -> i00
13:09:40.752 -> **************
13:09:40.752 -> (i%%,$$) Info: Sensor 0/0(0) enabled:1 status:0 false/unoccupied  r=120 x=160  brightness: A2480
13:09:40.752 -> Waiting - 'Enter' to continue
13:09:44.889 -> t1
13:09:44.889 -> **************
13:09:44.889 -> scroll OFF
13:09:44.889 -> (t##) Threshold: trip point set to 45
13:09:44.889 -> Waiting - 'Enter' to continue
```

At this time a NEW CAM will have NO sensors set and is a clean slate needing meaningful configuration data. There may be a default sensor(00) mid field (120,160). If all is well, this would be a good time to familiarize oneself with some commands starting with **k10,50,80** to DEFINE your first sensor, followed by **i10** to read info. on the new sensor S10 coordinates (50,80). Try **a10** to enable sensor 10, quickly followed by **w** to "wait" scrolling of new sensor values. 't1' can be used to toggle ON/OFF the scrolling status data.

```
● COM5                                                                         —

100mS; 00:--32--*   T45 N1 R2567 A2566 M2 B3
k10,50,80
**************
(k%%,rrr,xxx)setting new coordinates for Sensor 1/0(8) to r=50 x=80
Waiting - 'Enter' to continue
i10
**************
(i%%,$$) Info: Sensor 1/0(8) enabled:0 status:0 false/unoccupied  r= 50 x= 80  brightness: A480
Waiting - 'Enter' to continue
a10
**************
(a%%) enAble: sensor 1/0 with fresh reference - bsNo 010
updated Sensor_ref[]
21432mS; 00:--32--*   T45 N1 R2567 A2566 M2 B3    10:oo58##* Ref 010
  68mS; 00:--32--*   T45 N1 R2567 A2572 M2 B3    10:o126##*
100mS; 00:--32--*   T45 N1 R2567 A2550 M2 B3    10:o125##*
100mS; 00:--32--*   T45 N1 R2567 A2544 M2 B3    10:o122##*
<
☑ Autoscroll ☐ Show timestamp                               Newline    ∨  115200 baud
```

Then try **a11,60,90** to DEFINE *and* ENABLE sensor 11. The 'p1' position pointer command should now list the new sensors and the "difference" score hopefully under 39 for any stable sensor provided the camera is fixed and pointing at a reasonably lit area.

```
a11,60,90
**************
(a%%[,rrr,xxx]) new coordinates. Sensor[1/1] to r:60 x:90
Waiting - 'Enter' to continue
(a%%) enAble: sensor 1/1 with fresh reference - bsNo 011
updated Sensor_ref[]
p1
**************
(p$) Position Pointers: TL corners for DEFINED Sensors to bank 1
Bank 0; s[0]: r=120 x=160
Bank 1; s[0]: r= 50 x= 80 s[1]: r= 60 x= 90
Waiting - 'Enter' to continue

**************
553836mS; 00:--33--*  T45 N1 R2567 A2556 M2 B3   10:o123##* 11:oo63##* Ref 011
 64mS; 00:--33--*  T45 N1 R2567 A2557 M2 B3      10:o123##* 11:o144##*
100mS; 00:--32--*  T45 N1 R2567 A2557 M2 B3      10:o121##* 11:o140##*
(r%%[,0]) Refresh Reference: for sensor[%%]. (default r = r00 for all) (MUST BE UNOCCUPIED!)
References: enable new Sensor_ref[1/1]; 1st 2 HEX rgb pixels[0-5]: 1F 21 1F  1E 1F 1D
full new ref[bsNo] from current frame only - average ref still to be calculated
116mS; SUS 00:--32--*  T45 N1 R2567 A2554 M2 B3 10:o122##* 11:?_33##* Ref 011
 84mS; 00:--32--*  T45 N1 R2567 A2556 M2 B3      10:o121##* 11:?_34##*
100mS; 00:--32--*  T45 N1 R2567 A2556 M2 B3      10:o120##* 11:--34--*
100mS; 00:--32--*  T45 N1 R2567 A2553 M2 B3      10:o121##* 11:--34--*
```

An 'r00' command will take an internal reference photo for ALL sensors (including S10). Follow with 'w'.

```
r00
**************
(r%%[,0]) Refresh Reference: for sensor[%%]. (default r = r00 for all) (MUST BE UNOCCUPIED!)
Ref: r00 refreshes ALL defined sensor references with new Cratios & brightness
240912mS; SUS 00:--33--*  T45 N1 R2553 A2555 M2 B3      10:?_33##* 11:--33--* Ref 0120
 88mS; 00:--33--*  T45 N1 R2553 A2559 M2 B3      10:?_34##* 11:--33--*
100mS; 00:--33--*  T45 N1 R2553 A2567 M2 B3      10:--33--* 11:--34--*
100mS; 00:--33--*  T45 N1 R2553 A2564 M2 B3      10:--34--* 11:--33--*
```

An **Enter** should resume scrolling with live sensor states and a hand in front of the CAM *may* trip them (depending on lighting etc.). The **w** command will again pause scrolling. To save the new sensor settings in EPROM enter the 'e' command. You can undefine(erase) a sensor with 'u', redefine/relocate a sensor with 'a', inspect/interrogate a sensor with 'i', even display the digital (hex) values of pixels in sensor frame with 'f'. Refer to manual chapter 5 for more details on parameters, but do read Section 4.1 (Notation) first.

Note: There are other (easier) ways to position new sensors. The coordinate method is best for "tweaking" positions by small amounts. Step 6 below leads to the preferable visual setting method.

```
COM6                                                                    —  ☐  ✕
                                                                          Send
a11,60,85
**************
(a##,rrr,xxx)setting new coordinates for Sensor[1/1] to r:60 x:85
Waiting for new line
(a##) Activate: sensor 1/1 with fresh reference - bsNo 011
updated Sensor_ref[] HEX bytes(0-5)(BGRBGR): 6 b 8 6 b a
i11
**************
(i##,$$) Info: Sensor 1/1(9) enabled:1 status:0 false/unoccupied  r= 60 x= 85  brightness A524
Waiting for new line
f11
**************
(f##) Frame: Print fb ref & newest sample for S011
06 0B 08  06 0B 0A  0A 09 08  0A 09 08   offsetBytes 432 bsNo=1/1 -> 06 0B 08  06 0B 0A  0A 09 08  0A 09 08
08 0C 10  06 0A 0E  08 09 0C  08 0A 0C   offsetBytes 432 bsNo=1/1 -> 08 0C 10  06 0A 0E  08 09 0C  08 0A 0C
08 0E 10  06 0D 0E  08 0C 0E  08 0C 10   offsetBytes 432 bsNo=1/1 -> 08 0E 10  06 0D 0E  08 0C 0E  08 0C 10
08 0E 12  08 0C 10  0A 0C 10  0A 0C 10   offsetBytes 432 bsNo=1/1 -> 08 0E 12  08 0C 10  0A 0C 10  0A 0C 10
Waiting for new line
```

Step 6. Load Processing 4

The Arduino monitor is not "visual". The Processing 4 software enables one to see what the CAM sees and permits one to place sensors visually. Close down the Arduino IDE to free up the sesorCAM USB com port.

Download the "processing-4.3-windows-r64.zip" file from https://processing.org

Highlight the zip file and "Extract all" files to a suitable Destination directory (process4 say)

Locate and make a shortcut to the processing-4.3/processing.exe file on the desktop.

Make a sensorCAM folder in process4 & Copy the *sensorCAM.pde* file into a *process4/sensorCAM* directory

Click the shortcut thereby opening Processing4

Click the GetStarted box on the Welcome window

Click file-open, select *sensorCAM.pde* and open.

Click the (left) circular run button to produce the test pattern shown below

Reposition and resize the Console window and test pattern to look like that below, wide enough to avoid line wrap-around. The Cmd: line space will be the text entry line to be used for the CAM commands previously used with Arduino monitor. The broad black Console space replaces the sensorCAM monitor display and the test pattern window is for image display and sensor positioning.

Note the first lines of Console output. Choosing the first [0] COM port is the default. Should you have two (or more) COM ports listed, you may have to edit the ***int comNo=0;*** line to reflect the port the sensorCAM is attached to in the list (e.g. [1] as in image below). (this needs to be automated in a future revision)
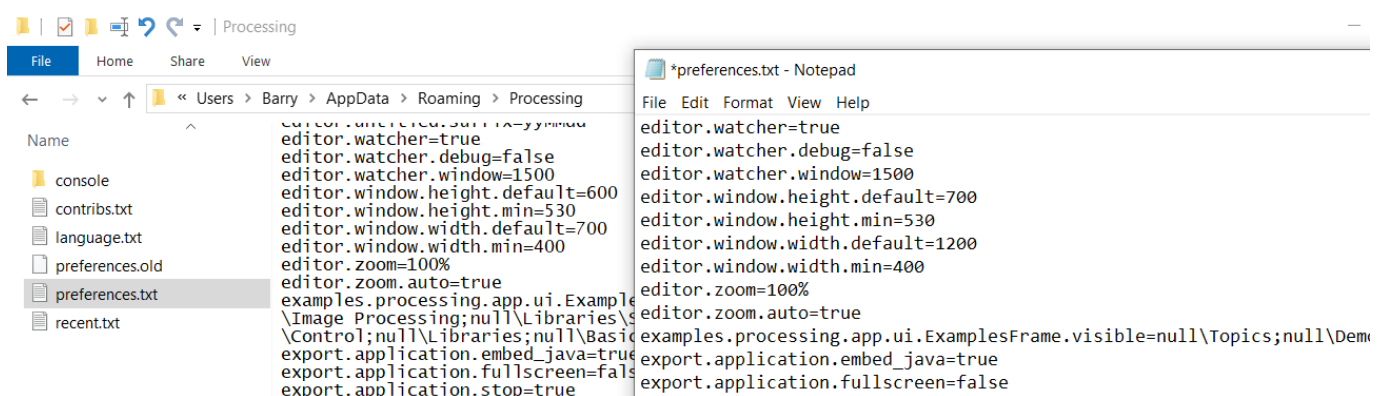
If the sensorCAM reboots and runs, enter cmd: **w Enter** or **ver Enter** which will suspend any scrolling and print the sensorCAM version. Use the scroll bar to view the top/bottom of the Console window. Note that you may have to click the mouse cursor on the Cmd: window before typing commands.

Note the dark blue round icon on the taskbar will bring the image to the front should it vanish behind the Console window. The white round button to the right of "Run" will "Stop" the **sensorCAM.pde** program.

For convenience, locate sensorCAM/sensorCAM.pde and create a shortcut to this file on the desktop.

Optionally, the file Users/[Barry]/AppData/Roaming/Processing/preferences.txt can be edited with a text editor (e.g. notepad) for more convenient startup of sensorCAM.pde. Depending on your screen resolution, for convenience, adjust the window.height.default=700 and window.width.default=1200 as below. This should ensure that the opening Processing 4 window is higher and wide enough to avoid undesirable line-wrapping of scrolling sensor state tabulation. You can adjust these settings later if 700x1200 is not optimum for your screen resolution and sensor count.
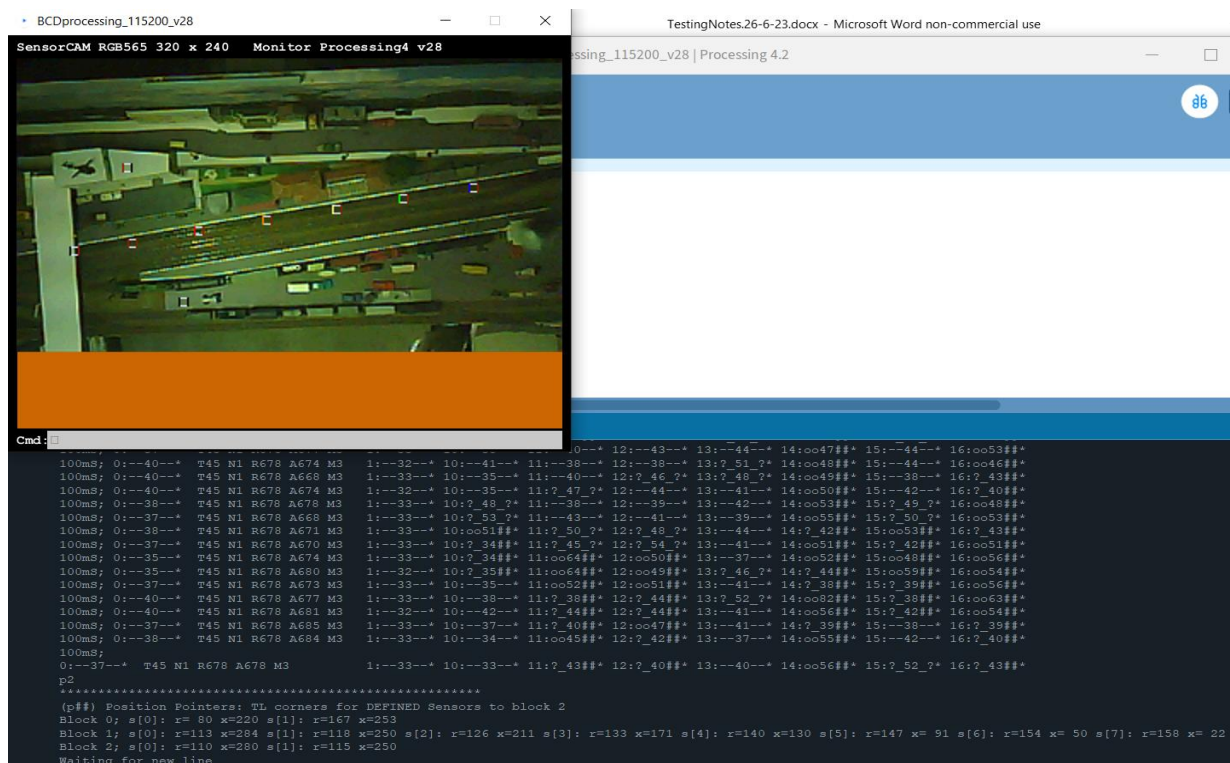
Step 7. Use the Processing 4 App to replace the Arduino monitor.

The processing app. can communicate the same commands to the sensorCAM as used in Step 5 above.

Proceed to verify the sensorCAM commands are generally working as before with Arduino IDE.

New commands can now be explored to see still images from the CAM and to see location of defined sensors as small squares. Enter the command '**Y**' at the Cmd: line and wait 15 seconds for a full image to appear. Remember the sensors after rebooting will be those last stored in EPROM, not last defined.

To explore the imaging functions (W, X, Y, Z) in depth refer to the manual Chapter 6.

Step 8. Create test sensors and test detection with moving targets

Create some test sensors using Processing 4 by typing Cmd: **a31** (no Enter!) and following with a click on the image previously obtained as above. The command line will be completed with the cursor coordinates. Follow up with an **Enter** to define a new sensor S31. Verify the new sensor with a '**p3**' command and SEE the new sensor with a new '**Y**' command. Later versions will "box" the new sensor automatically.

The sensor squares on the image are colour coded with the b/s number using the standard resistor colour codes (refer manual Appendix E).

Place a coloured object at the sensor position and observe the change in the scrolling tabulation for the sensor. Without the obstruction, after a reference command ('**r00**'), the difference score should be below 39 and with the obstruction the diff score should rise substantially. If the threshold is set ('**t42**' say) the sensor will "trip" and indicate occupied confirming successful installation. A moving hand test is an easy quick test to do.

It is advisable to define a "reference" sensor S00 that will NOT be obstructed. Any trips of this sensor will indicate an unwanted change to the environment (e.g. lighting). To define such a sensor use **a00** and a Processing 4 image "click" on a quiet location away from tracks but with "good" mid-range illumination (see Manual section 8.4). S00 is reserved and displayed separately at the left of the scrolling status table.

Step 9. Setup the CAM viewing a railroad and test a virtual sensor with moving rolling stock.

The CAM must be rigidly mounted, in a "convenient" location for testing at this stage. It may be limited by your cabling reach with USB control. Later, with buffered I2C and power, the sensorCAM may be able to be located more optimally.

The manual Chapter 5 covers many points of relevance to getting satisfactory detection. Review this chapter and test topics for clarification. Good familiarity leads to success. Initial tests should be with good contrast between track and rolling stock. Refine parameters later to handle more difficult targets.

Step 10. Optimise parameters for best performance.

Initially by trial & error users will learn the significance of parameters. This is an Alpha release and consequently the default settings may not be best. Refer sensorCAM manual Appendix B.

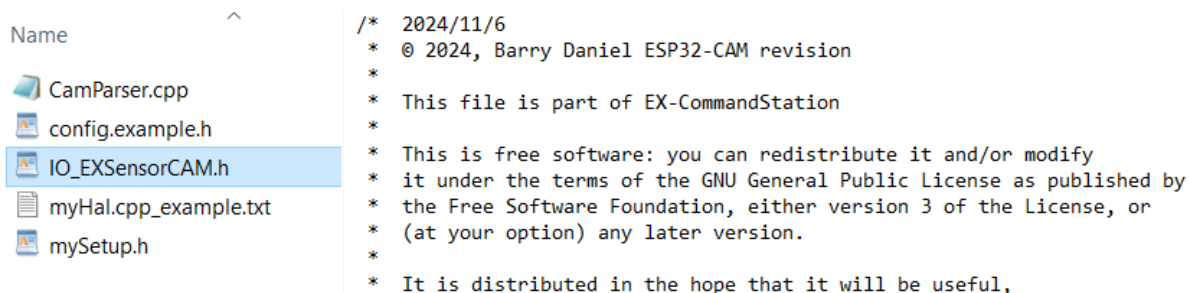## Step 11. Connect CAM to an i2c interface (e.g. PCA9515A or better)

The quickest and simplest method is to fit a PCA9515A to the ESP32-CAM-MB. Refer to the manual Chapter 3, the pictures in the manual (Figure 7 and Appendix F). Level shifting (3.3V to 5V) for a Mega is the primary function of the PCA9515A. Adjust voltages if not 5V compliant (e.g. ESP32 Command Station). It has 10k pull-up resistors but extra may be needed. If distance is an issue, as the PCA9515A is limited to about 3metres, either an LTC4311A enhancer or a buffered i2c solution will be needed. The manual Ch. 7 & AppF describes the PCA9515A used, but the Sparkfun differential Endpoint is also recommended.

Step 12. Depending on system, integrate sensorCAM into Command Station using appropriate code.

The CAM can be connected to a "Control Station" via i2c using appropriate host code (typically C++). It has been successfully interfaced to two different (Arduino Mega) systems but most focus has been on the DCC-EX-Command Station (CS). The sensorCAM can be connected to a CS and interrogated using EX-Rail.

If you are wanting to integrate with a CS, we assume you have installed EX-CS previously. If not, refer to the DCC-EX website for details. The following guide assumes you have a working v5.4.x CS in place.

12.1 The sensorCAM repository has an EX-CS directory. These files normally reside with the *CommandStation-EX.ino* beside the other standard CS files. For devel CS only, optionally, copy latest *IO_EXSensorCAM.h* to this CS directory. Do not copy to Production v5.4.xx , only to "devel" versions.


```
Name
    CamParser.cpp
    config.example.h
    IO_EXSensorCAM.h
    myHal.cpp_example.txt
    mySetup.h
```

```
/*  2024/11/6
 *  © 2024, Barry Daniel ESP32-CAM revision
 *
 *  This file is part of EX-CommandStation
 *
 *  This is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  It is distributed in the hope that it will be useful,
```

12.2 Edit the *config.h* file to include *#define SENSORCAM_VPIN 700* and *#define CAM SENSORCAM_VPIN+* IF you do NOT want to use vpins 700 to 779 for the sensorCAM virtual sensors, or use multiple sensorCAM's, adjust accordingly. Be careful to not change any other CS *config.h* code from your previous working CS installation. This suits CS Production version v5.4.0+ and versions v5.5.10+ devel versions.

```
// SENSORCAM
// ESP32-CAM based video sensors require #define to use appropriate base vpin number.
#define SENSORCAM_VPIN 700
// To bypass vPin number, define CAM for ex-rail use e.g. AT(CAM 012) for S12 etc.
#define CAM SENSORCAM_VPIN+
//
//#define SENSORCAM2_VPIN 600   //define other CAM's if installed.
//#define CAM2 SENSORCAM2_VPIN+ //for EX-RAIL commands e.g. IFLT(CAM2 020,1)
//
// For smoother power-up, define a STARTUP_DELAY to allow CAM to initialise ref images
#define STARTUP_DELAY 5000    // up to 20sec. CS delay
```

12.3 Edit your *myHal.cpp* to tell CS to include/create a driver for sensorCAM. Add the sensorCAM driver with *#include "IO_EXSensorCAM.h* along with the other "includes" at the top. If not present, add the line *I2CManager.setClock(100000);* below *void halSetup(){* . Now add the "create" directive below. If you change *SENSORCAM_VPIN* (step 12.2), then you may need to amend the "create" from 700 to desired.

```
//#include "IO_PCA9555.h"      // 16-bit I/O expander (NXP & Texas Instruments).
//#include "IO_I2CDFPlayer.h"  // DFPlayer over I2C
#include "IO_EXSensorCAM.h"    // sensorCAM driver


//=============================================================================
// The function halSetup() is invoked from CS if it exists within the build.
// The setup calls are included between the open and close braces "{ ... }".
// Comments (lines preceded by "//") are optional.
//=============================================================================

void halSetup() {

  I2CManager.setClock(100000);
```

```
// The following directive defines an ESP32-CAM instance.
//=============================================================================
// EXSensorCAM::create(VPIN, Number of VPINs, I2C Address)
//
// The parameters are:
//    VPIN=an available Vpin as start of block of consecutive sensors (up to 80)
//        #define SENSORCAM_VPIN0 #00 in config.h if not using 700.
//    Number of VPINs=pin count (must not exceed 80)
//    I2C address=an available I2C address (default 0x11)
//        #define ESP32CAP 0x13 in config.h to raise allowable ESP32 range of addresses
//        Note that the I2C address (0x11) is the default in the sensorCAM code
//
// EXSensorCAM::create(700, 80, 0x11);   //Can define SENSORCAM_VPIN in config.h
EXSensorCAM::create(SENSORCAM_VPIN, 80, 0x11);  //preference is now to use HAL(VPIN, 80, 0x11) in myA
//EXSensorCAM::create(600, 80, 0x12);   //alternate or second CAM device address creation
```

12.4 Now the individual sensor IDs need to be assigned to vPin numbers through the mySetup file. They can be added in manually later, so we don't have to have all 80 defined now. SETUP only aids debugging.

```
// setup for sensorCAM on an ESP32-CAM NUMDigitalPins <= 80
// assume 700 is first vpin (set with ...CREATE(700,80,0x11)
// the optional SETUP operations below initiate jmri monitoring of sensors for any change of state
// Mostly only useful during debug of initial system but load up CS with extra work. Use judiciousl
//        id vPin
SETUP("<Z 100 700 0>");     // set up for control OUTPUT on vpin #00
// start of up to 80 sensors numbered bsNo's 100 to 197 (OCT) (0/0 to 9/7)
SETUP("<S 100 700 0>");     // first sensor (S00) (reference)
SETUP("<S 101 701 0>");
SETUP("<S 102 702 0>");
//        as many as you want.  You can add later manually with CS native commands
SETUP("<S 107 707 0>");
SETUP("<S 110 708 0>");     // Note: suggested id is b/s format (~OCT); vpin is DEC.
SETUP("<S 111 709 0>");     // myFilter.cpp REQUIRES this relationship for bsNo to vPin conversion
SETUP("<S 112 710 0>");
SETUP("<S 113 711 0>");
```

12.5 With the release of CS Production version v5.4.0, the former sensorCAM myfilter method was made redundant. Versions of CS beyond v5.4.0 all use the CamParser.cpp method and have *IO_EXSensorCAM.h CamParser.cpp* and *CamParser.h* included by default. These files DO NOT need to be overwritten. Unless you wish to use the "bleeding edge" devel releases, proceed to step 12.6.

```
Name                        #define driverVer 305
                            // v305 less debug & alpha ordered switch
                            // v304 static oldb0;  t(##[,%%];
  CamParser.cpp             // v303 zipped with CS 5.2.76 and uploaded to repo (with debug)
  config.example.h          // v302 SEND=StringFormatter::send, remove Sp(), add 'q', memcpy( .8) -> .7);
  IO_EXSensorCAM.h          // v301 improved 'f','p'&'q' code and driver version calc. Correct bsNo calc. for 'a'
  myHal.cpp_example.txt      // v300 stripped & revised without expander functionality. Needs sensorCAM.h v300 AND CamParser
  mySetup.h                 // v222 uses '@'for EXIORDD read.  handles <NB $> and <NN $ ##>
                            // v216 includes 'j' command and uses CamParser rather than myFilter.h Incompatible with v203 s
```

12.6 Once the changes have been put in place, the *CommandStation-EX.ino* file will have to be recompiled and loaded into the CS (Mega?) using the Arduino IDE, to invoke the changes. Refer to the sensorCAM manual Appendix G for details on using the CS monitor to replace the sensorCAM USB monitor. Initially it would be helpful to maintain a USB monitor on the sensorCAM to catch and verify activity. Note that there are some differences in the capabilities and format of displays between CAM and CS. No visual imagery can be obtained via CS (except webcam) once the USB is disconnected. The CAM MUST be flashing to respond correctly to most CS **<N>** commands.

12.7 After up-loading to the Command Station, open the Arduino Monitor (at 115200 baud) to see the CS "console". If the sensorCAM i2c is connected and the CAM running (flashing), The user's <Nw> CS command should issue the CAM wait command (stops flashing) and another <Nw> should resume. Note it can be helpful to have 2 instances of the IDE, so you can separately monitor CS and CAM on two USB ports.

```
<* License GPLv3 fsf.org (c) dcc-ex.com *>
<* I2C clock speed set to 100000 Hz *>
<* I2C Device found at 0x11, ?? *>
<* I2C Device found at 0x65, ?? *>
<* EX-IOExpander device found, I2C:0x65, Version v0.0.23 *>
<* EX-SensorCAM device found, I2C:0x11, Version v0.3.0 *>
<* MCP23017 I2C:0x20 Device not detected *>
<* MCP23017 I2C:0x21 Device not detected *>
<* Found PORTB pin 13 *>
<* Pin 55 Max 1497mA (501) *>
<* Found PORTB pin 12 *>
<* Pin 54 Max 1497mA (501) *>
<* Timer 2 A=1 B=6 *>
<= A MAIN>
```

```
<* EX-SensorCAM device found, I2C:0x11,CAM Version v0.3.0 vpins 700-779 *>
<* IO_EXSensorCAM driver  v0.3.1 vpin: 700  *>
```

With CAM flashing, try <N m>, <N i 11> and <N t 46> to confirm functionality then <NM> for current min/max settings. Note <N t 0> command returns the LAST value of threshold. Repeat just to confirm changes, if necessary, or just use <NM>. Cmd <Nv> will show the Vpin usage and version of sensorCAM.

```
<* CamParser: 700 M 0 9999 *>
<@ 0 3 "Free RAM= 4568b">
<* (m$[,##]) Min/max: $ frames min2flip (trip) 0, maxSensors 041, minSensors 00, nLED 2, threshold 42, TWOIMAGE_MAXBS 030 *>
<* CamParser: 700 I 11 9999 *>
<* (i%%[,$$]) Info: Sensor 011(9) enabled:1 status:0 row=124 x=140 Twin=00 pvtThreshold=255 A~1680 *>
<* CamParser: 700 T 46 9999 *>
<@ 0 3 "Free RAM= 4566b">
<n (t[##[,%%]]) Threshold:42 sensor S00:-32--* 01:--33--* 11:--33--* 12:--33--* 13:--34--* 14:--33--* 15:--33--* 20:--37--*
<* CamParser: 700 M 0 9999 *>
<* (m$[,##]) Min/max: $ frames min2flip (trip) 0, maxSensors 041, minSensors 00, nLED 2, threshold 46, TWOIMAGE_MAXBS 030 *>
<* CamParser: 700 ^ 0 9999 *>
<* EX-SensorCAM device found, I2C:0x11,CAM Version v0.3.10 vpins 700-779 *>
<* IO_EXSensorCAM driver  v0.3.5 vpin: 700  *>
```

12.8 Powering up the CS and CAM(s) can be problematical. The bootup interactions are affected by the
following: **a) ALL devices on an i2c bus should be powered for i2c communications to be reliable.**

b) Full sensorCAM boot-up calls MyWire.begin() which FAILS if a) is incomplete.

c) CS bootup executes driver EXSensorCAM::create(); which FAILS if b) is unsuccessful.

To avoid a race between b) and c) the CS has a STARTUP_DELAY 5000 (5sec.) to allow b) time to boot.
Rebooting a sensorCAM AFTER 5 seconds (b) can break the connection the CS had established, and so it
may need to be followed by a reboot of CS (c).

Opening an Arduino IDE monitor on a sensorCAM (or uploading software) can cause a reboot of the
sensorCAM, so the above issue needs to be appreciated and catered for, possibly with a following CS
reboot.

12.9 To use EX-RAIL, refer to the EX documentation regarding general sensor usage.

Refer to the sensorCAM manual Appendix H for further tips regarding sensorCAM with EXRAIL.

Note: As there are up to 80 sensors or banks of sensors, the #define CAM .. in step 12.2 above helps. The
vPin management can be simplified by using the substitution for vPin of (CAM 0%%) which automatically
calculates the vPin for sensor S%%. For example, AT(CAM 010) will wait for S10 to trip.

12.10 To manage MULTIPLE sensorCAMs, it is advisable to follow a convention to avoid errors, as follows:

a) Choose a sequential address range (e.g. 0x11,0x12,0x13,0x14) for CAM1 CAM2 .. CAM4 respectively,
and preferably a simple sequential base_vPin series (e.g. 500,600,700 say). Then using Arduino IDE…

b) For each CAM# in turn, open *sensorCAM.ino* & edit **configCAM.h** (e.g. *#define I2C_DEV_ADDR 0x12*)
AND edit **sensorCAM.ino** *#define BCDver 315* (e.g. 315 to 12315) then File>SaveAs sensorCAM315.12
Upload to CAM# (CAM2). This saves distinct identifiable versions for each CAM#. (works for 11-14,21-24)
This fudge makes management of multiple sensorCAMs less error prone ('v' will show version as 12.3.15).

b) In CS **MyHal.cpp**, create sequential drivers for CAM,CAM2, .. (starting with CAM 0x11) (e.g.
EXSensorCAM::create(SENSORCAM2_VPIN,80,0x12);

c) In CS **config.h,** define CAM2, CAM3 & CAM4 as for CAM in **config.h** (e.g.
#define SENSORCAM2_VPIN 600
#define CAM2 SENSORCAM2_VPIN+

d) edit CS **MySetup.h** to cover all required sensors in each CAM. e.g. for CAM2 (2bs from base vpin 600)
*for(uint16_t b=1; b<=4;b++) for(uint16_t s=0;s<8;s++) Sensor::create(200+b*10+s,600+b*8+s,1);*

e) Once loaded and booted, make use of the CamParser change-cam commands by adding an [optional]
cam # to bs numbers (e.g. <N> <Ni 112> <Np 201> <NV 300> <NC vpin>. <N> shows current CAM (vPin).

f) In EXRAIL scripts, select CAM# by using ID format (CAM# 0%%) e.g. AT(CAM2 012) for sensor S[2]12.
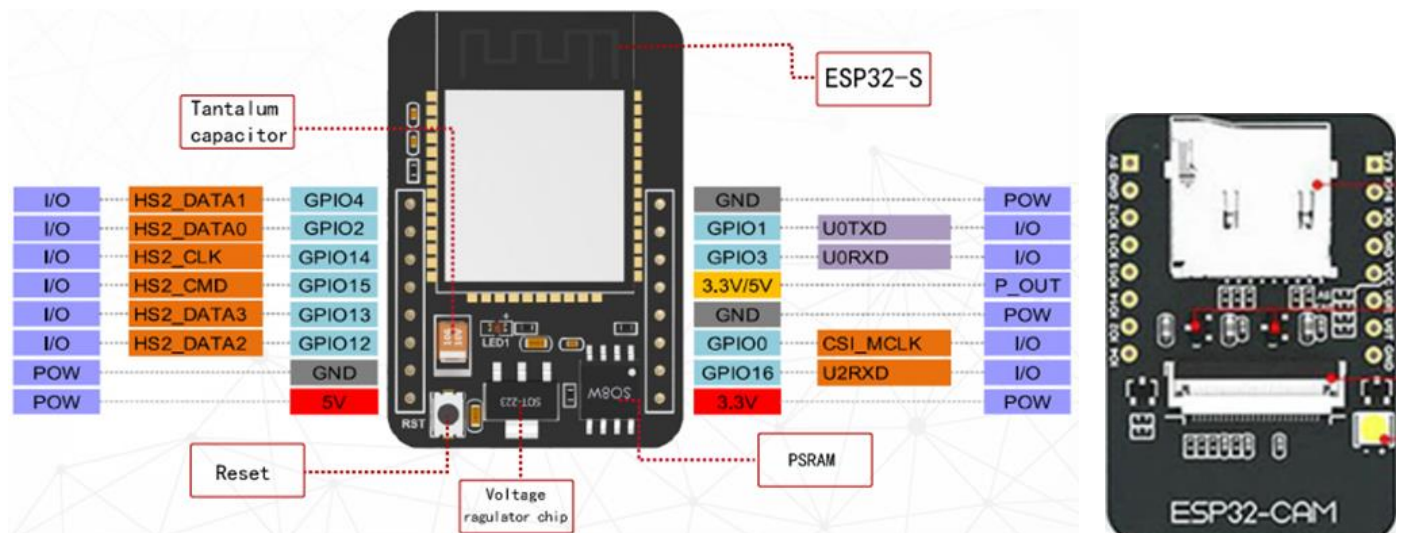
N.B. CAM v1.6 has <u>4x jumpers</u> (cam side) near U0R/U0T pins & CAM v1.9 has <u>6x jumpers</u> (including RST)

In board <u>version v1.9</u>, the "GND" pin adjacent GPIO1/U0T is used for RESET (GND/R) to ESP32-CAM and **MUST NOT be tied to GND** or CAM will remain in RESET mode. CHIP version shows at start of upload, e.g.

```
esptool.py v4.5.1
Serial port COM5
Connecting.....
Chip is ESP32-D0WD-V3 (revision v3.1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 3c:8a:1f:ae:44:68
```

The esp32-CAM-MB obtained with extra headers uses a small CH340N IC unlike the 16 pin package used on the "regular" MB devices (micro-B). The wider MB (Type C) also has an issue in that it does NOT have a reset pin because the RST/GND socket is PERMANENTLY connected to GND which will hold CAM v1.9 in permanent Reset.
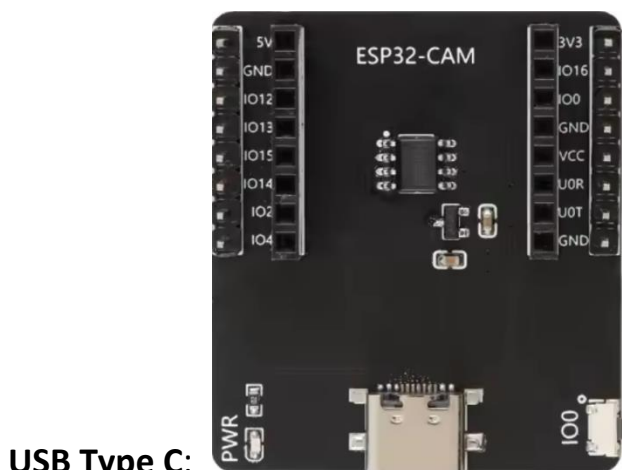
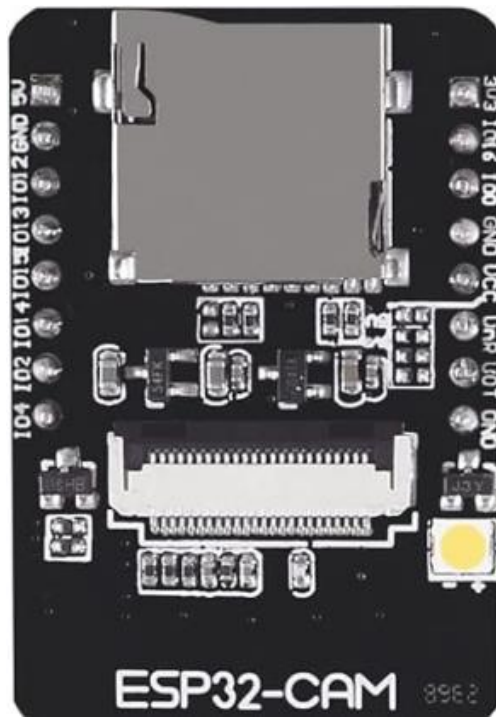**USB Type C**:   Newer **USB Type micro-B:**

So – without butchery, can only use the v1.6 CAM's on this board ☹, and even then will have to hold the IO0 button in (grounding it) on reset until upload is progressing. As a workaround for v1.9, it may be possible to slice off this MB GND socket completely. View the Camera side of the ESP32-CAM to distinguish the boards.

Identification: ESP32-CAM v1.6 has 4x jumpers beside UOR/UOT pins and v1.9 has 6x jumpers (inc. Reset).

Using v1.6 with no RTS/DTR reset, users will have to boot up holding IO0 button in to get into programming mode.

V1.6 (NO RST)

V1.9 (RST)