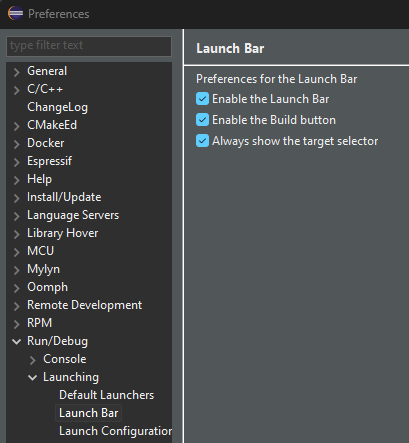
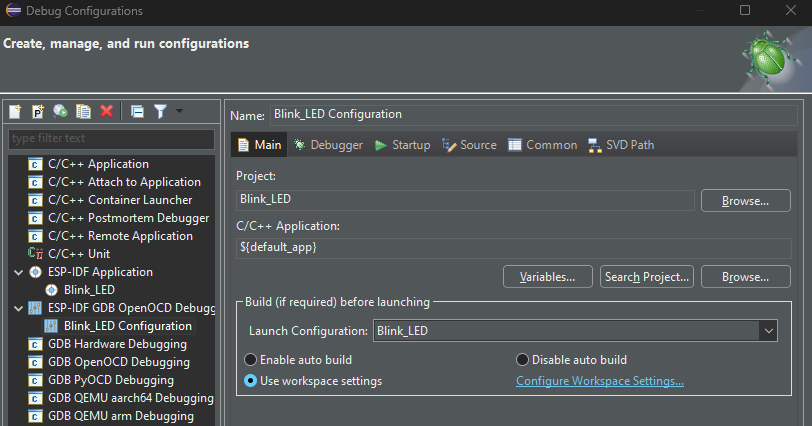
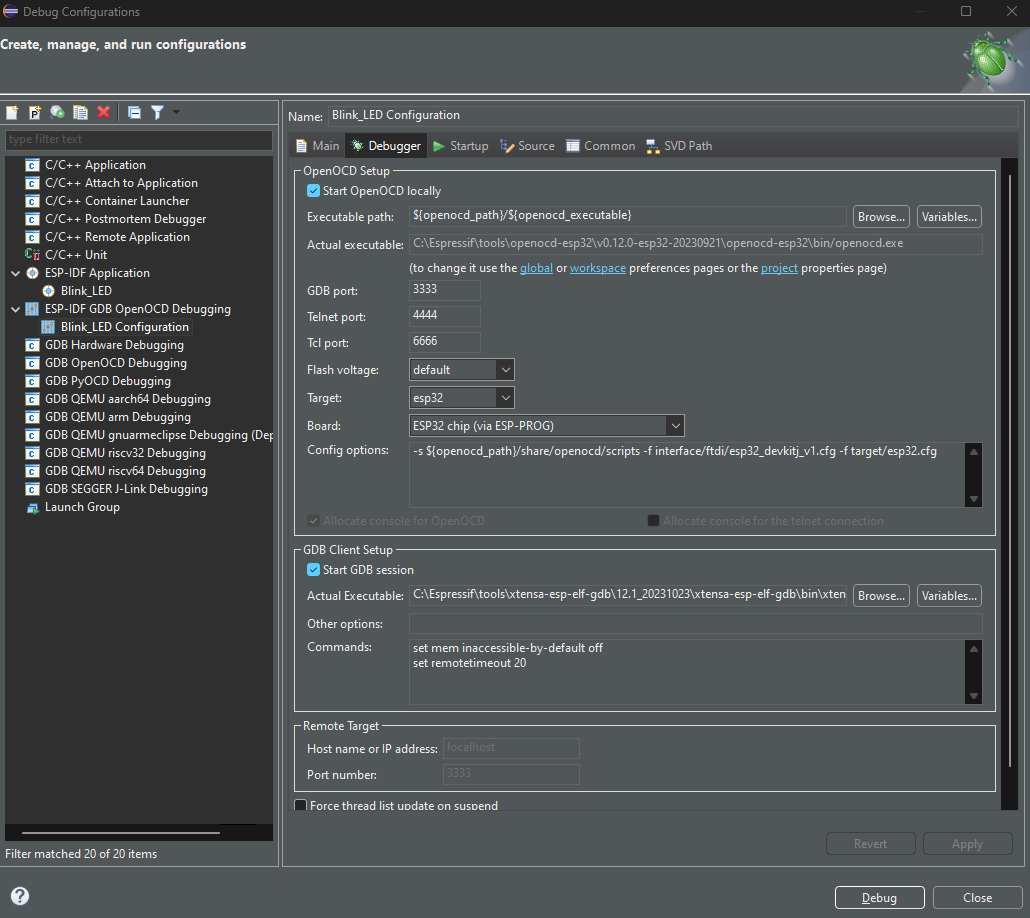
This guide results from my experiences with installing Eclipse IDE, ESP-IDF and the many problems I encountered setting them up and configuring to work with an ESP-PROG VCP and JTAG debugging board. Solutions to some of the problems I encountered were very difficult to find by searching the Internet; so hopefully others may save some time getting to a working system.

1. To uninstall Eclipse IDE:
   1. Delete "C:\Users\Steven\eclipse"
   2. Delete "C:\Users\Steven\.p2"
2. Install Eclipse IDE from: <https://www.eclipse.org/downloads/packages/installer>
   1. If the download of artifacts is slow, switch to Advanced mode (via the menu in the upper right), then you can disable the use of mirrors on the confirmation page before you hit Finish.
3. Install ESP-IDF from e.g. <https://github.com/espressif/esp-idf/releases/download/v5.2.1/esp-idf-v5.2.1.zip>
   1. I installed to C:\ESP-IDF\esp-idf-v5.2.1
   2. Set-up ESP-IDF as per here: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/windows-setup.html>
   3. Command and PowerShell shortcuts will be added to your desktop, use these (Prefer PowerShell) to run python commands without using Eclipse e.g.
      1. Idf.py set-target esp32
      2. Idf.py menuconfig (open blue config window)
      3. Idf.py build
      4. Idf.py -p COM30 flash
      5. Idf.py -p COM30 flash monitor (Serial output from ESP32 is displayed in the PowerShell window.
4. Install latest Espressif plugin for Eclipse:
   1. Help > Install New Software then click the add button
      1. Use name = ESP-IDF Eclipse Plugin v3.0.0-beta
      2. URL = https://dl.espressif.com/dl/idf-eclipse-plugin/updates/beta/
      3. Links to latest Plugins and download URLs below:
         1. https://github.com/espressif/idf-eclipse-plugin/releases
         2. https://dl.espressif.com/dl/idf-eclipse-plugin/updates/beta/
   2. From Eclipse menu select > Espressif > ESP-IDF Manager
   3. Click Add ESP-IDF button and browse to installed location
      1. E.g. C:\ESP-IDF\esp-idf-v5.2.1
      2. If requested install Python, from Microsoft Store, by running the following command from a PowerShell window: C:\Users\Steven\AppData\Local\Microsoft\WindowsApps\python3.EXE
      3. Then when running >Espressif > Product Information from Eclipse it produces output without warning messages.
5. Project Launch Bar missing in Eclipse plug-in for IDF
   1. This is vital and not easy to discover, why is it not the default??
   2. In my case I had to turn it on in Window->Preferences->Launching->Launch Bar->Enable the Launch Bar. Took me a while to find it.
   3. in eclipse 2022-09 the "Launch Bar" setting is located under Window->Preferences->Run/Debug->Launching



1. Once the Project Launch Bar is added the vital green LAUNCH IN RUN MODE button appears, which uploads the code to the target.
2.  The LAUNCH TARGET button can be used to specify the target hardware e.g. ESP32 and the associated COM port.
3. Format source code = highlight and Ctrl + Shift + F, or right click file e.g. main.c and > source > format. Or better install CppStyle by going to > Help > Eclipse Marketplace Select the Eclipse logo at the bottom and in the search box at the top put cppstyle and when found install it. Next go to Window -> Preferences -> CppStyle and set these two paths:
   1. Clang-format path: C:\Espressif\tools\esp-clang\16.0.1-fe4f10a809\esp-clang\bin\clang.exe
   2. Cpplint path: C:\Espressif\python\_env\idf5.2\_py3.11\_env\Scripts\cpplint.exe
   3. And tick the boxes: Enable cpplint and Run clang-format on file save
4. Eclipse HELP = <https://help.eclipse.org/latest/index.jsp>
5. Flashing message in info box at bottom right corner of Eclipse: “update rpm packages proposal list” and CPU usage very high with cooling fan at high speed!!
   1. It's a bug in the "Linux Tools" plugin. It seems that the plugin is unable to create the file specified at Window -> Preferences -> Specfile Editor -> RPM Completions -> Path to packages list file and therefore keeps trying to create it.
   2. Quick solution is: Go to >Windows > Preferences > RPM > Specfile Editor > RPM Completions and untick – Automatically build the RPM package proposals list
      1. Before you can do this go to >Windows > Preferences > RPM > Rpmlint and set he path to: C:\Users\Steven\.p2\pool\plugins\org.eclipse.linuxtools.rpm.rpmlint\_1.1.0.202403061658.jar the path can be found by using the Everything app to search for rpmlint.
   3. Better solution: Create a text file e.g. new.txt in C:\Users\Steven\ and rename it to .pkglist i.e. C:\Users\Steven\.pkglist and leave the box ticked.
   4. For more info see: <https://bugs.eclipse.org/bugs/show_bug.cgi?id=428302>
6. Start debugging:
   1. Check OpenOCD is installed by opening the ESP-IDF PowerShell console and typing:
      1. PS C:\ESP-IDF\esp-idf-v5.2.1> openocd –version, which should produce:
         1. Open On-Chip Debugger v0.12.0-esp32-20230921 (2023-09-21-13:40)
   2. Connect ESP-PROG to computer and configure ports using Zadig
      1. See: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/jtag-debugging/configure-ft2232h-jtag.html
   3. Connect ESP-PROG to target hardware.
   4. Go to > Run > Debug Configurations > and right click ESP-IDF GDB OpenOCD Debugging and choose New Configuration.
7. Using the Command Shell Console
   1. The built in default new ESP-IDF project outputs “Hello from app\_main!” on the COM port used to upload and run compiled code. The output can be viewed on either a separate Serial Terminal e.g. Putty, or using the Command Shell built into Eclipse, invoked from the Open Console button at the right of the output pane at the bottom of Eclipse i.e. 3 Command Shell Console. The first time you use Command Shell Console you need to give the connection a name e.g. ESP32 Module and choose a port number e.g. COM30. The created connection will be saved and can be reused next time. Remember to close the terminal e.g. disconnect Putty or click the disconnect button at the right of the output pane at the bottom of Eclipse before you upload and run the code again OR ECLIPSE WILL CRASH! There is a Connect button next to the Disconnect button, the use of which is obvious!
   2. A better way to handle serial output is to use the micro-USB connection on the ESP32 module and a separate USB cable to another USB port on your computer and a terminal app such as Putty; then there is no possibility of a conflict with the code uploading process.
   3. To delete unwanted connections, go to Menu > Window > Show View > Other then search for Connections and click the open button; all the remote connection will be listed and any unwanted can be deleted.
   4. On the MAIN tab browse for the Project e.g. Blink\_LED
   5. On the DEBUGGER tab configure as below:
   6. The other tabs can be left as default.
   7. Click the Debug button to start debugging.
   8. If a tab opens asking to locate main.c just click File Location and browse to the main.c file for your project e.g.
      1. "E:\Users\Steven\Documents\GitHub\eclipse-workspace\Blink\_LED\_by\_steve\_2\main\main.c"
      2. Without doing this the debugger will not work.
   9. If a tab opens asking for the location of additional files just browse and locate from the address shown e.g.
      1. "C:\ESP-IDF\esp-idf-v5.2.1\components\esp\_hw\_support\cpu.c"
      2. "C:\ESP-IDF\esp-idf-v5.2.1\components\freertos\FreeRTOS-Kernel\portable\xtensa\port.c"
8. Misc Problems & Solutions
   1. Debug fails with OpenOCD error
      1. Close Eclipse, disconnect and reconnect ESP-PROG and restart Eclipse.
   2. Debug running but can’t STEP OVER (single step)
      1. Click debug Restart button
   3. An internal error occurred during: "DSF Source Lookup"
      1. Click OK and ignore
   4. Lots of “function could not be resolved” errors, with red underlines.
      1. Create a new project and copy and paste the code.
   5. Python errors when launching a project in RUN mode.
      1. Ensure any serial terminal using the launch COM port is disconnected.
9. Notes about using ESP-PROG
   1. Assuming that you have used Zadig to reconfigure Dual RS232-HS (Interface 0) to be a serial bus device then you will have one COM port remaining e.g. COM3
   2. Code can be uploaded via COM3 by clicking the settings button (gear wheel) in the ESP target box in Eclipse and selecting *COM3 – User specified port* then selecting RUN as the launch mode and clicking the *Launch in Run Mode* button, in just the same way as if you were using the micro-USB socket on the ESP32 module itself.
      1. Once the code is uploaded any serial output from the running code may be viewed using a separate serial terminal app, or by opening a *Command Shell Console*, creating a *Remote Connection* for COM3 and opening the connection. My experience is that to re-upload your code you must disconnect the *Remote Connection*, upload and reconnect. There is comment on the Internet that the switch between monitor and upload should be automatic using Eclipse, but it does not work for me.
   3. When debugging code, don’t try and use the ESP-PROG VCP as it does not work properly i.e. it displays the first print statement then stops and crashes the debug process, requiring a restart of the debugging process, or better a restart of Eclipse. What does work well is using a separate terminal app like Putty, or my preference Serial Port Assistant from <https://github.com/KangLin/SerialPortAssistant>..
      1. You might think that all serial terminals are much the same, well usually, but not when connected to Eclipse ESP-IDF via the VCP of an ESP-PROG, oh no! I have tried many terminal apps and some cause the debugger to crash, or they only display the first print statement and then stop. If you have a problem with your terminal app just try another one. So far I have found Serial Port Assistant to be problem free!
10. Example code to get started:

#include "driver/gpio.h"

#include "esp\_event.h"

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/unistd.h>

#include <unistd.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

//Led is connected GPIO 2, the on-board LED on an ESP32 module

#define led 2

int count = 0; //A value to be incremented and compared to predefined\_value

int predefined\_value = 1000; // Define the predefined value

int var = 0; // Initialise the variable

void app\_main(void) {

esp\_rom\_gpio\_pad\_select\_gpio(led); // Initialise the led pin (gpio 2) //

gpio\_set\_direction(led,

GPIO\_MODE\_OUTPUT); // led pin initialise as an output //

int level = 1; //led OFF value

while (true) {

printf("Hello from app\_main!\n");

//sleep(1);

vTaskDelay(200 / portTICK\_PERIOD\_MS);

if (count == predefined\_value) {

printf("Predefined value reached. Run While loop...\n");

while (var < 10) {

printf("Running While loop...\n");

sleep(1);

// Do nothing inside the loop

}

}

count++;

printf("The value of count is: %d\n", count);

//sleep(1);

vTaskDelay(200 / portTICK\_PERIOD\_MS);

gpio\_set\_level(led, level); // led will be turn ON (level = 0) and OFF (level =1) as value of level

level = !level; // change the level value

}

}

xx