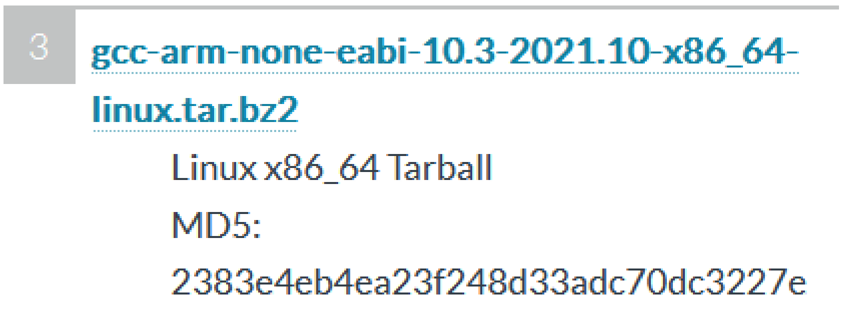
Compile the microros driver library

Note: You must use Ubuntu to compile the microros driver library. Ubuntu 22.04 is recommended.

1. Install the cross-compiler

Open the download address and download the Linux version of the cross-compiler.

```
https://developer.arm.com/downloads/-/gnu-rm
```



3 gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2

Linux x86_64 Tarball

MD5:

2383e4eb4ea23f248d33adc70dc3227e

After decompression, put the file in the following path

```
/opt/gcc-arm-none-eabi
```

And add the executable file path to the .bashrc environment variable in the user directory.

```
export PATH=$PATH:/opt/gcc-arm-none-eabi/bin
```

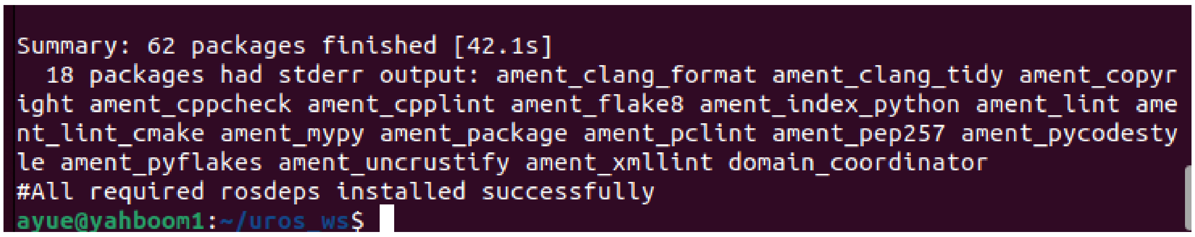
2. Get the STM32-microros library file

Download and compile the microros setup tool

```
mkdir uros_ws && cd uros_ws
git clone -b humble https://github.com/micro-ROS/micro_ros_setup.git
src/micro_ros_setup
rosdep update && rosdep install --from-paths src --ignore-src -y
colcon build
source install/local_setup.bash
```

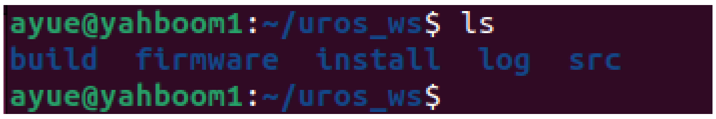
Use the microros setup tool to generate the STM32-microros library file.

```
ros2 run micro_ros_setup create_firmware_ws.sh generate_lib
```



```
Summary: 62 packages finished [42.1s]
  18 packages had stderr output: ament_clang_format ament_clang_tidy ament_copyr
ight ament_cppcheck ament_cpplint ament_flake8 ament_index_python ament_lint ame
nt_lint_cmake ament_mypy ament_package ament_pclint ament_pep257 ament_pycodesty
le ament_pyflakes ament_uncrustify ament_xmllint domain_coordinator
#All required rosdeps installed successfully
ayue@yahboom1:~/uros_ws$
```

Check the workspace. We should now have five folders, among which the firmware folder contains the STM32-microros library files. `build` `firmware` `install` `log` `src`



```
ayue@yahboom1:~/uros_ws$ ls
build  firmware  install  log  src
ayue@yahboom1:~/uros_ws$
```

3. Modify parameters

Open the toolchain.cmake file

```
vim ~/uros_ws/firmware/toolchain.cmake
```

Copy the following content into

```
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_CROSSCOMPILING 1)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

# SET HERE THE PATH TO YOUR C99 AND C++ COMPILERS
# Add the compiler path here
set(PIX /opt/gcc-arm-none-eabi/bin)
set(CMAKE_C_COMPILER ${PIX}/arm-none-eabi-gcc)
set(CMAKE_CXX_COMPILER ${PIX}/arm-none-eabi-g++)

set(CMAKE_C_COMPILER_WORKS 1 CACHE INTERNAL "")
set(CMAKE_CXX_COMPILER_WORKS 1 CACHE INTERNAL "")

# SET HERE YOUR BUILDING FLAGS
set(FLAGS "-O2 -ffunction-sections -fdata-sections -fno-exceptions -mcpu=cortex-
m7 -mfpu=fpv5-d16 -mfloat-abi=hard -nostdlib -mthumb --param max-inline-insns-
single=500 -D'RCUTILS_LOG_MIN_SEVERITY=RCUTILS_LOG_MIN_SEVERITY_NONE'" CACHE
STRING "" FORCE)
# -mcpu=cortex-m7 indicates the microcontroller architecture
# -mfpu=fpv5-d16 -mfloat-abi=hard indicates support for hardware floating-point
compilation
```

```
set(CMAKE_C_FLAGS_INIT "-std=c11 ${FLAGS} -DCLOCK_MONOTONIC=0 -
D'__attribute__(x)='" CACHE STRING "" FORCE)
set(CMAKE_CXX_FLAGS_INIT "-std=c++11 ${FLAGS} -fno-rtti -DCLOCK_MONOTONIC=0 -
D'__attribute__(x)='" CACHE STRING "" FORCE)

set(__BIG_ENDIAN__ 0)
```

Save and exit.

Open the colcon.meta file

```
vim ~/uros_ws/firmware/colcon.meta
```

Copy the following content into

```
{
    "names": {
        "tracetools": {
            "cmake-args": [
                "-DTRACETOOLS_DISABLED=ON",
                "-DTRACETOOLS_STATUS_CHECKING_TOOL=OFF"
            ]
        },
        "rosidl_typesupport": {
            "cmake-args": [
                "-DROSIDL_TYPESUPPORT_SINGLE_TYPESUPPORT=ON"
            ]
        },
        "rcl": {
            "cmake-args": [
                "-DBUILD_TESTING=OFF",
                "-DRCL_COMMAND_LINE_ENABLED=OFF",
                "-DRCL_LOGGING_ENABLED=OFF"
            ]
        },
        "rcutils": {
            "cmake-args": [
                "-DENABLE_TESTING=OFF",
                "-DRCUTILS_NO_FILESYSTEM=ON",
                "-DRCUTILS_NO_THREAD_SUPPORT=ON",
                "-DRCUTILS_NO_64_ATOMIC=ON",
                "-DRCUTILS_AVOID_DYNAMIC_ALLOCATION=ON"
            ]
        },
        "microxrcedds_client": {
            "cmake-args": [
                "-DUCLIENT_PIC=OFF",
                "-DUCLIENT_PROFILE_UDP=OFF",
                "-DUCLIENT_PROFILE_TCP=OFF",
                "-DUCLIENT_PROFILE_DISCOVERY=OFF",
                "-DUCLIENT_PROFILE_SERIAL=OFF",
                "-UCLIENT_PROFILE_STREAM_FRAMING=ON",
                "-DUCLIENT_PROFILE_CUSTOM_TRANSPORT=ON",
                "-DUCLIENT_PROFILE_SHARED_MEMORY=ON",
                "-DUCLIENT_SHARED_MEMORY_MAX_ENTITIES=20"
            ]
        },
        "rmw_microxrcedds": {
            "cmake-args": [
                "-DRMW_UXRCE_MAX_NODES=1",
                "-DRMW_UXRCE_MAX_PUBLISHERS=10",
                "-DRMW_UXRCE_MAX_SUBSCRIPTIONS=10",
                "-DRMW_UXRCE_MAX_SERVICES=1",
                "-DRMW_UXRCE_MAX_CLIENTS=1",
                "-DRMW_UXRCE_MAX_HISTORY=10",
                "-DRMW_UXRCE_TRANSPORT=custom"
            ]
        }
    }
}
```

"-DRMW_UXRCE_MAX_NODES=1", #maximum number of nodes;

"-DRMW_UXRCE_MAX_PUBLISHERS=10", #maximum number of publisher;

"-DRMW_UXRCE_MAX_SUBSCRIPTIONS=10", #maximum number of subscribers;

"-DRMW_UXRCE_MAX_SERVICES=1", #maximum number of servers;

"-DRMW_UXRCE_MAX_CLIENTS=1", #maximum number of clients;

"-DRMW_UXRCE_MAX_HISTORY=10", #history;

"-DRMW_UXRCE_TRANSPORT=custom" #custom transport interface

Save and exit.

4. Add environment variables

```
export RMW_IMPLEMENTATION=rmw_microxrcedds
```

5. Start compiling

```
cd ~/uros_ws/
ros2 run micro_ros_setup build_firmware.sh $(pwd)/firmware/toolchain.cmake
$(pwd)/firmware/colcon.meta
```

Note: Since the compilation process requires downloading many files, and most file servers are located abroad, if there is a network anomaly, please solve the network download problem yourself.

View the generated static library and header files

```
ls firmware/build
```

6. Solve the include problem

Since the generated include folder path is too long, you need to use a script to fix it.

Create a new fix_include.sh script in the firmware folder

```
cd ~/uros_ws/firmware
vim fix_include.sh
```

Copy the following content into

```
#!/bin/bash
######## Fix include paths ########

BASE_PATH=build

pushd mcu_ws > /dev/null
    INCLUDE_ROS2_PACKAGES=$(colcon list | awk '{print $1}' | awk -vd=" " '{s=
(NR==1?s:sd)$0}END{print s}')
popd > /dev/null

for var in ${INCLUDE_ROS2_PACKAGES}; do
    if [ -d "${BASE_PATH}/include/${var}/${var}" ]; then
        rsync -r ${BASE_PATH}/include/${var}/${var}/* $BASE_PATH/include/${var}
        rm -rf ${BASE_PATH}/include/${var}/${var}
    fi
done

mkdir ${BASE_PATH}/libmicroros
mv ${BASE_PATH}/libmicroros.a ${BASE_PATH}/libmicroros
```

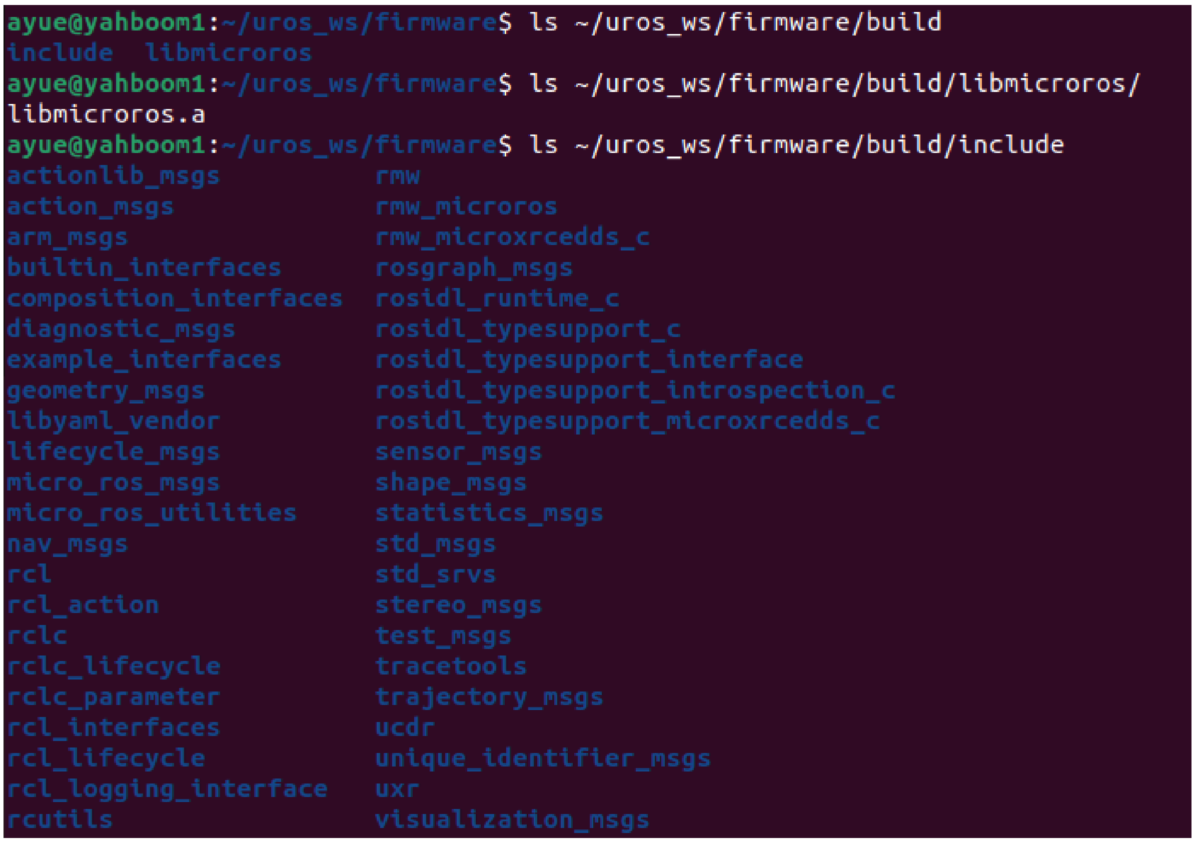
Save and exit.

Run the following command to fix the include folder problem.

```
bash fix_include.sh
```

7. The final file

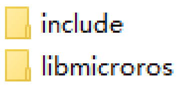
```
ls ~/uros_ws/firmware/build
ls ~/uros_ws/firmware/build/libmicroros
ls ~/uros_ws/firmware/build/include
```



```
ayue@yahboom1:~/uros_ws/firmware$ ls ~/uros_ws/firmware/build
include  libmicroros
ayue@yahboom1:~/uros_ws/firmware$ ls ~/uros_ws/firmware/build/libmicroros/
libmicroros.a
ayue@yahboom1:~/uros_ws/firmware$ ls ~/uros_ws/firmware/build/include
actionlib_msgs          rmw
action_msgs             rmw_microros
arm_msgs                rmw_microxrcedds_c
builtin_interfaces      rosgraph_msgs
composition_interfaces  rosidl_runtime_c
diagnostic_msgs         rosidl_typesupport_c
example_interfaces      rosidl_typesupport_interface
geometry_msgs           rosidl_typesupport_introspection_c
libyaml_vendor          rosidl_typesupport_microxrcedds_c
lifecycle_msgs          sensor_msgs
micro_ros_msgs          shape_msgs
micro_ros_utilities     statistics_msgs
nav_msgs                std_msgs
rcl                     std_srvs
rcl_action              stereo_msgs
rclc                    test_msgs
rclc_lifecycle          tracetools
rclc_parameter          trajectory_msgs
rcl_interfaces          ucdr
rcl_lifecycle           unique_identifier_msgs
rcl_logging_interface   uxr
rcutils                 visualization_msgs
```

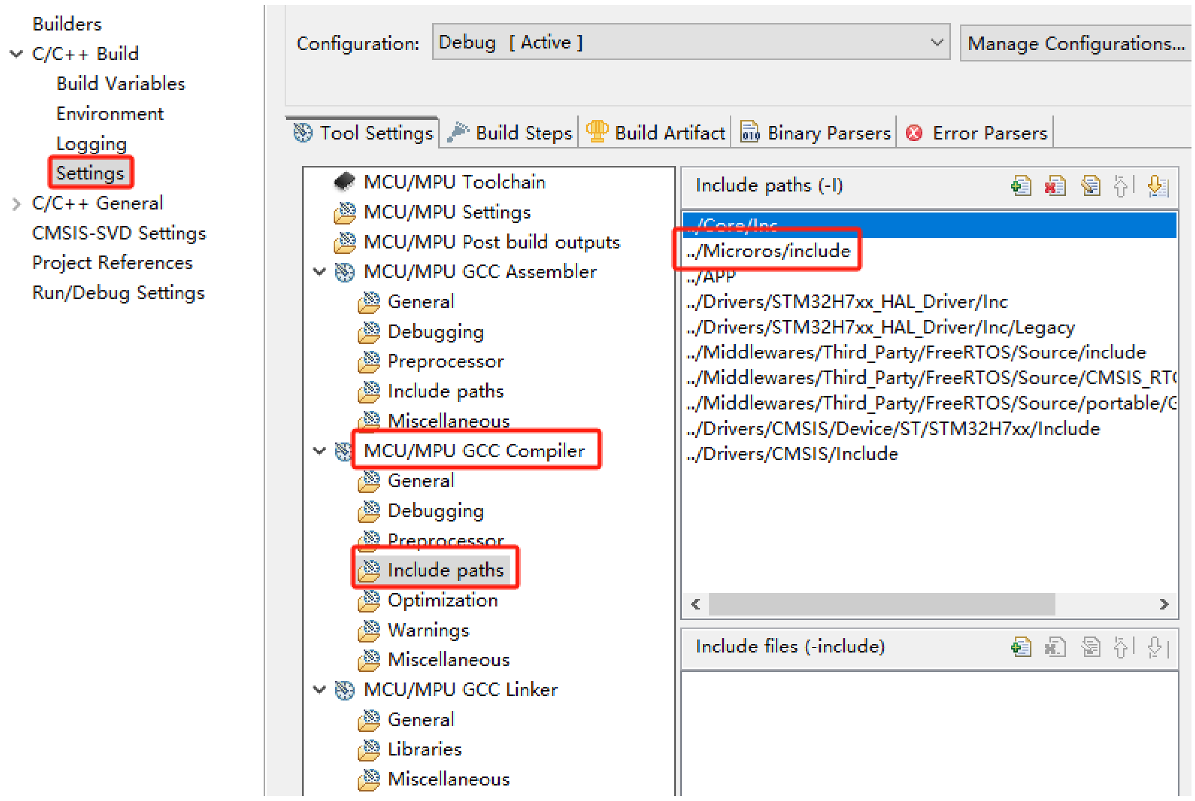
8. Import STM32CUBEIDE project

Create a new Microros folder in the project, and then copy the include and libmicroros folders generated by the previous step into Microros.



include
libmicroros

Right-click to open the project properties, then click [Settings]->[MCU/MPU GCC Compiler]->[include paths] to add the microros include directory path, and then click [Apply] to take effect.


Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
CMSIS-SVD Settings
Project References
Run/Debug Settings

Configuration: Debug [Active]   Manage Configurations...

Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

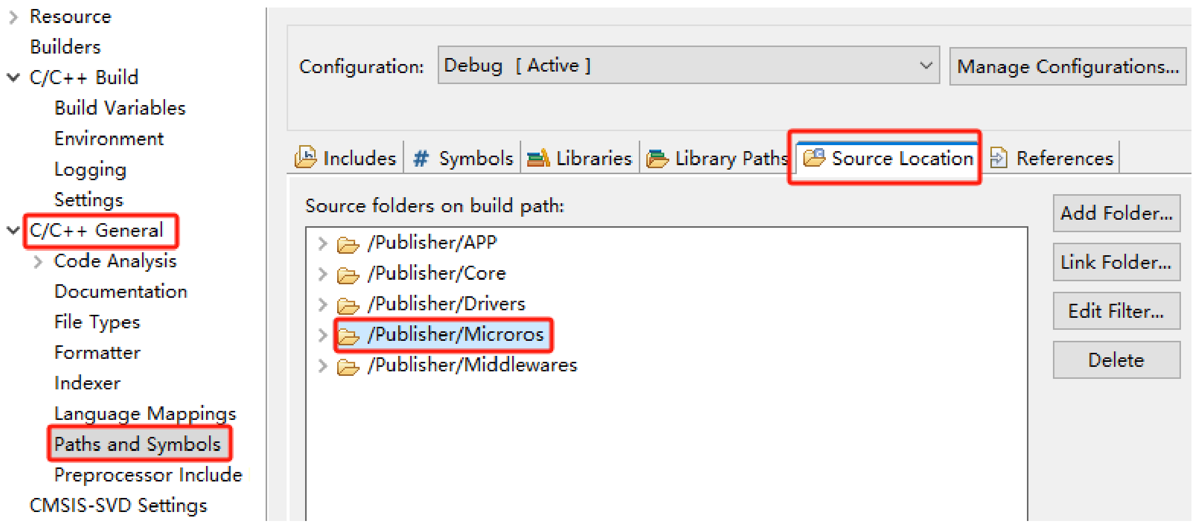
MCU/MPU Toolchain
MCU/MPU Settings
MCU/MPU Post build outputs
MCU/MPU GCC Assembler
- General
- Debugging
- Preprocessor
- Include paths
- Miscellaneous

MCU/MPU GCC Compiler
- General
- Debugging
- Preprocessor
- Include paths
- Optimization
- Warnings
- Miscellaneous

MCU/MPU GCC Linker
- General
- Libraries
- Miscellaneous

Include paths (-I)
```
../Core/Inc
../Microros/include
../APP
../Drivers/STM32H7xx_HAL_Driver/Inc
../Drivers/STM32H7xx_HAL_Driver/Inc/Legacy
../Middlewares/Third_Party/FreeRTOS/Source/include
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RT
../Middlewares/Third_Party/FreeRTOS/Source/portable/G
../Drivers/CMSIS/Device/ST/STM32H7xx/Include
../Drivers/CMSIS/Include
```
Include files (-include)

Add the microros folder as the project source code path.


Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
- Code Analysis
- Documentation
- File Types
- Formatter
- Indexer
- Language Mappings
- Paths and Symbols
- Preprocessor Include

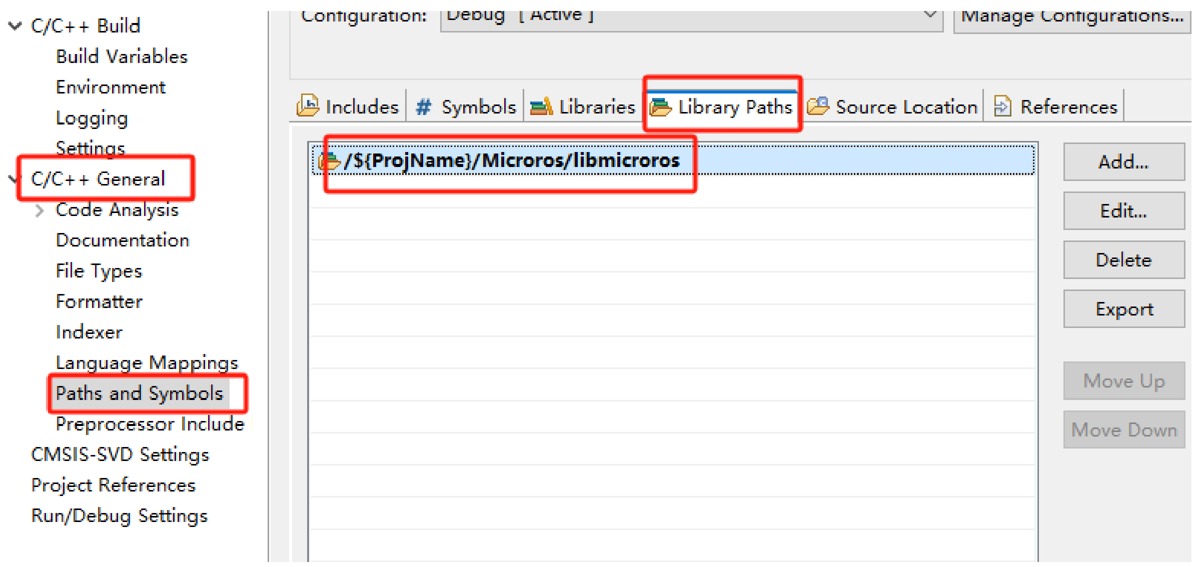
CMSIS-SVD Settings

Configuration: Debug [Active]   Manage Configurations...

Includes | Symbols | Libraries | Library Paths | Source Location | References

Source folders on build path:
```
/Publisher/APP
/Publisher/Core
/Publisher/Drivers
/Publisher/Microros
/Publisher/Middlewares
```
Add Folder... | Link Folder... | Edit Filter... | Delete

Import the microros library path


C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
- Code Analysis
- Documentation
- File Types
- Formatter
- Indexer
- Language Mappings
- Paths and Symbols
- Preprocessor Include

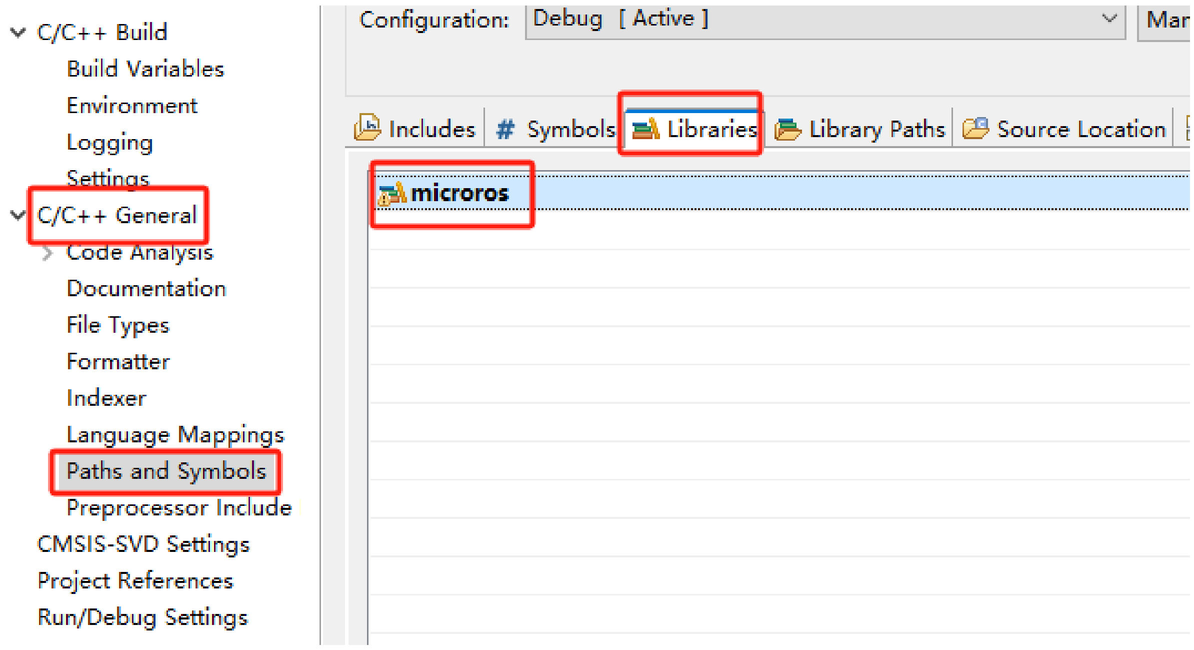
CMSIS-SVD Settings
Project References
Run/Debug Settings

Configuration: Debug [Active]   Manage Configurations...

Includes | Symbols | Libraries | Library Paths | Source Location | References

```
/${ProjName}/Microros/libmicroros
```
Add... | Edit... | Delete | Export | Move Up | Move Down

Link the microros library file to the project. Make sure the name matches the libmicroros.a static library file name (excluding the prefix and suffix "microros").



C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mappings
Paths and Symbols
Preprocessor Include
CMSIS-SVD Settings
Project References
Run/Debug Settings

Configuration: Debug [Active]

Includes | # Symbols | Libraries | Library Paths | Source Location

microros