## VSCODE installation and extensions

Once VSCODE is installed you need to install extensions. Here is the list of extensions to make the VSCODE attach to a running ROS node on Windows (using instructions from here: <https://code.visualstudio.com/docs/editor/extension-marketplace#_command-line-extension-management> ):

c:\opt\ros\noetic\gztaskboard\bin>code --list-extensions

cheshirekow.cmake-format

Compulim.compulim-vscode-closetag

hbenl.vscode-test-explorer

mikeburgh.xml-format

**ms-iot.vscode-ros**

ms-python.python

ms-python.vscode-pylance

ms-toolsai.jupyter

ms-toolsai.jupyter-keymap

**ms-vscode.cmake-tools**

**ms-vscode.cpptools**

ms-vscode.test-adapter-converter

smilerobotics.urdf

twxs.cmake

xaver.clang-format

Of importance to ROS development, the ms-iot.vscode-ros, the ms-vscode.cpptools, ms-vscode.cmake-tools are important extensions.

Note, the **cppvsdbg** that is used to attach and break into a running process is a windows extension that is manually configured (see instructions and good luck but it worked for me!).

## VSCODE to attach to running ROS node:

1. First, add looping code to break into main: Note bBreak MUST BE A GLOBAL. (not sure how to access local variables).

bool bBreak = 1;

int main()

{

// if no break then delay for Gazebo visual to finish loading.

while (bBreak)

{

// sleep 1 second

Sleep(1000);

}

1. Next, in all CMakelist.txt add /DEBUG to build filed while doing Release build (NOTE! I had too many problems mixing Debug and Release builds linking properly). So you can add Debug symbols to Release build:

set\_property(TARGET ${target} APPEND\_STRING PROPERTY LINK\_FLAGS " /DEBUG")

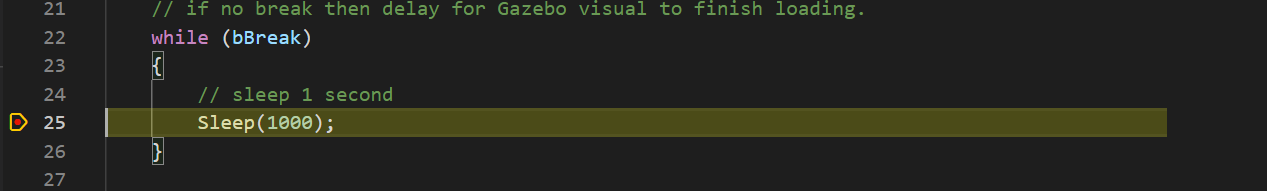
1. Build with catkin\_make in Windows Terminal configured for ROS noetic. (Instructions for installing Windows Terminal for ROS found online and in separate document).
2. In Windows terminal after successful build:

devel\setup.bat

rosrun test\_gazebo\_package test\_gazebo\_package

In VSCODE:

1. Put a breakpoint in the main.cpp at the Sleep(1000); statement (which will stop the attach).



1. Code up (Windows) attach in Launch.json, which works because catkin\_make uses vs2019 to build the cpp code and we enabled .pdb (program db) file using the /DEBUG linker flag.

{

"version": "0.1.0",

"configurations": [

**{**

**"name": "(Windows) Attach",**

**"type": "cppvsdbg",**

**"request": "attach",**

**"processId": "${command:pickProcess}"**

**},**

{

"name": "ROS: Attach",

"type": "ros",

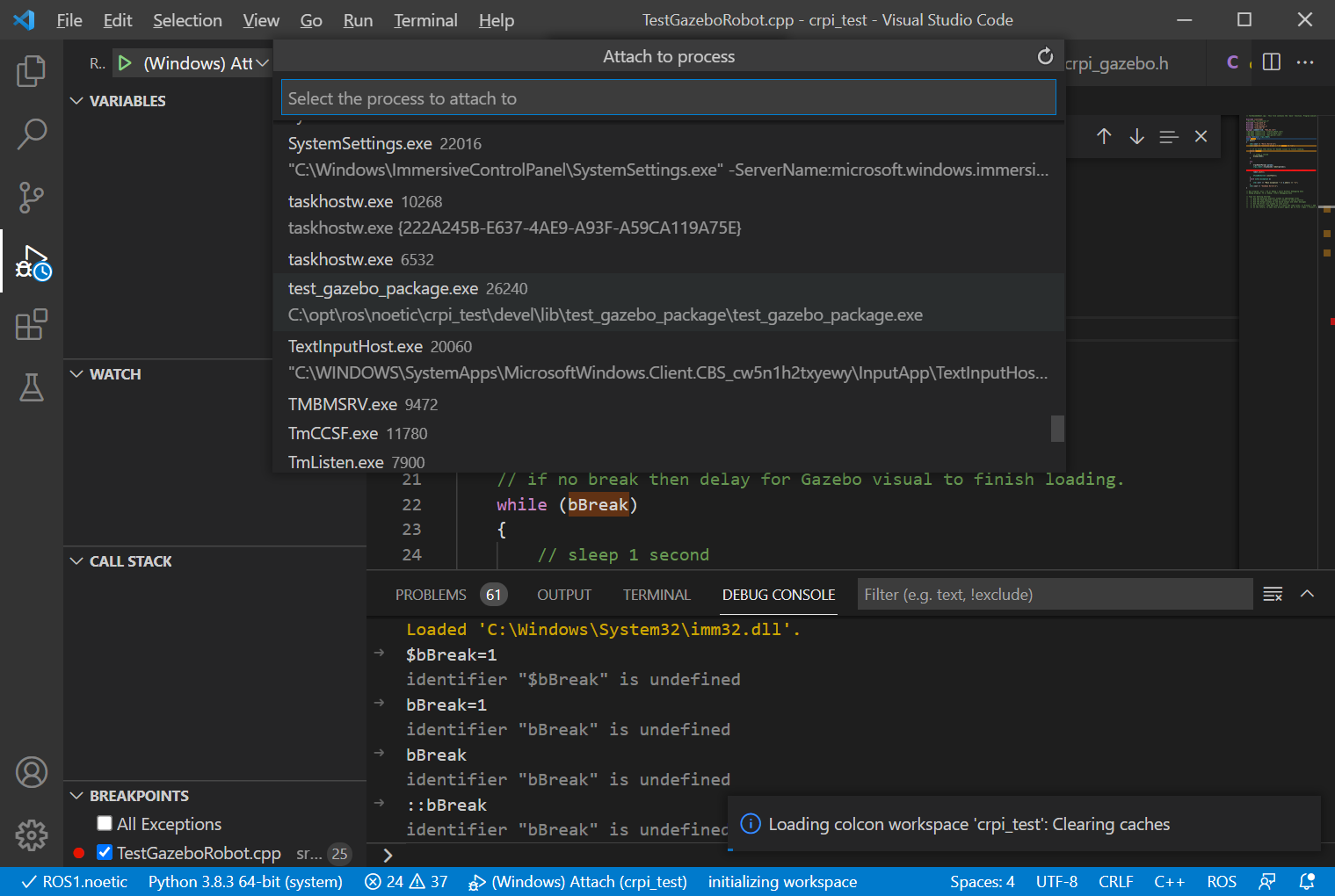
"request": "attach"

}

]

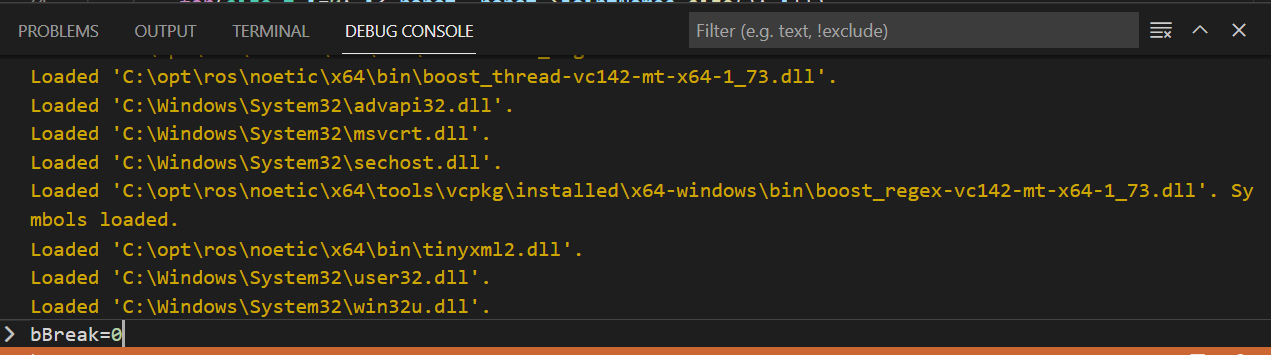
}

1. Select  which is the Debug Panel. Then, hit the green Arrow  for start debug attach command.
2. Select the process to attach to (the exe in rosrun i.e., in our case test\_gazebo\_package).



1. Switch to the DEBUG CONSOLE, and put the variable and the value:

bBreak=0



1. It should attach and then stop Use the debugger symbols to step through the code!

## Attaching to running program, or let program run

For debugging, the rosrun command is used to run a ROS program. In order to break or pass through the attach program infinte loop code, the "rosbreak" parameter is passed as part the command line arguments while running the "rosrun" shell command given as:

rosrun <package> <node> <parameter>

If we want to allow Visual code to attach and then continue, we use the following rosrun command line:

c:\opt\ros\noetic\crpi\_test>rosrun test\_gazebo\_package test\_gazebo\_package rosbreak:=1

so that rosbreak:=1 is a command line argument (described next) that is read and can be used as part of a roslaunch command line argument also.

Below, is a command line to ignore the break command loop, and the program will continue without the opportunity to break.

c:\opt\ros\noetic\crpi\_test>rosrun test\_gazebo\_package test\_gazebo\_package rosbreak:=0

The code to enable attaching to program before it reaches the running part, an infinite waiting loop is appended to the start of the program. First we read all the command line parameters as parsed and issued by rosrun (or roslaunch):

First, we declare a global declaration of break to simply changing:

bool bBreak = 1;

The Main function accepts command line arguments, and for our case parses the command line by saving all the args and then using getCmdOption function to see if the 'rosbreak:=' string has been passed in. The 'rosbreak:=' command string has either a 0 following to signal DON’T BREAK or a 1 following to signal BREAK.

int main(int argc, char\*\* argv)

{

std::vector<std::string> args;

for(size\_t i=0; i< argc; ++i)

args.push\_back(argv[i]);

Using the getCmdOption function, the command line arguments are searched for a match. If a match is found, the trailing portion of the arg string is returned, else the default value (i.e., "1") is returned. The returned string is converted into an integer Boolean using the atoi function.

bBreak = (bool) atoi( getCmdOption(args, "rosbreak:=", "1").c\_str());

Depending on the passed in command line break option, either the bBreak loop repeats until VSCODE attaches to the running process and turns off

// if no break then delay for Gazebo visual to finish loading.

while (bBreak)

{

// sleep 1 second

Sleep(1000);

}

Note, to enable the process to stop gracefully when VSCODE attaches to the running process a breakpoint is set at the 'Sleep(1000);' statement.

The getCmdOption function parses a vector of string looking for an option string match. If it finds a match, it returns the trailing portion of the arg string. If the option is NOT found then the szDefault default string value is returned.

std::string getCmdOption(std::vector<std::string> args, const std::string &option, std::string szDefault = "")

{

std::string cmd;

for (int i = 0; i < args.size(); ++i)

{

std::string arg = args[i];

if (0 == arg.find(option))

{

//std::size\_t found = arg.find(option);

cmd = arg.substr(option.size());

boost::algorithm::trim(cmd);

return cmd;

}

}

return szDefault;

}