

MMRambotics Code Guide

Abstract:

This document outlines code formatting and Windriver C++ IDE setup for MMRambotics. The goal of this document is to organize development and code in a consistent matter, eliminate ambiguity and confusion, and increase readability and development speed.

Git Version Control:

All projects should be under version control to prevent data loss, achieve easier collaboration and develop without worry of doing permanent damage.

By default, a new Git repository includes a *master* branch but we want to create a *development* branch and use *master* as a stable branch of deploying code. Only deployed and tested code should be in *master*.

```
# Create development branch.  
git checkout -b development  
# Switch to master and merge stable development code.  
git checkout master  
git merge development
```

Compiled files and project meta data should not be tracked in a Git repository; therefore your *.gitignore* file should contain the following:

```
*.a  
*.o  
*.cproject  
*.project  
*.wrmakefile  
*.wrproject  
PPC603gnu
```

Each time you make a change to your code, you should commit to the *development* branch with a descriptive message of your changes.

```
git add .  
git commit -m "I fixed these bugs and added these constants."
```

Commit early, commit often!

Windriver C++ Integrated Development Environment Setup:

The first thing we want to do is set up default build properties for *VxWorks Downloadable Kernel Module Project* projects. This will allow you to create a new project without reconfiguring build properties (required to properly build and deploy cRio-FRC code).

1. Open the *Windriver C++ IDE*
2. In the main menu, click *Window -> Preferences*

3. On the left side pane, there is a hierarchy of categories, select *Wind River -> Build -> Build Properties*
4. On the larger right pane there is a drop down at the top with a label “Specify default build properties for new:”; select *VxWorks Downloadable Kernel Module Project (Wind River VxWorks 6.3)*
5. Six tabs are now available, start with *Build Support*
 - a. Under *Build support* choose *Managed build*
 - b. The *Build command* should be `%makeprefix% make --no-print-directory`
 - c. Both checkmarks under *Build output passing* should be unchecked.
6. Switch to the *Build Specs* tab
 - a. Four checkmarks are available, only `PPC603gnu` should be checked.
7. Switch to the *Build Tools* tab
 - a. The drop down with the label *Build tool:* should be set to `C-Compiler`
 - b. The *Suffixes* labelled field should be set to `*.c`
 - c. Under *Build output generation* the radio button pane should be set to `Generated build output is an object`
 - d. Under *Build spec specific settings* there is a field labelled *Command:* which should be set to `echo "building %@"; %ccompilerprefix%$(TOOL_PATH) ccppc%DebugModeFlags% %ToolFlags% $(ADDED_CFLAGS) %Includes%`
 - e. The *Tool Flags...* field should be set to `$(CC_ARCH_SPEC) -ansi -Wall -MD -MP -mlongcall`
8. Switch to the *Build Macros* tab
 - a. *Build macro definitions* should be set to the following
 - i. `PROJECT_TYPE: DKM`
 - ii. `DEFINES: <nothing>`
 - iii. `EXPAND_DBG: 0`
 - b. *Build spec specific settings* should be set to the following
 - i. `VX_CPU_FAMILY: ppc`
 - ii. `CPU: PPC603`
 - iii. `TOOL_FAMILY: gnu`
 - iv. `TOOL: gnu`
 - v. `TOOL_PATH: <nothing>`
 - vi. `CC_ARCH_SPEC: -mcpu=603 -mstrict-align -mno-implicit-fp -mlongcall`
 - vii. `LIBPATH: <nothing>`
 - viii. `LIBS: <nothing>`
9. Switch to the *Build Paths* tab
 - a. Add the following *Include directories* under *Include paths*
 - i. `-I$(WIND_BASE)/target/h`
 - ii. `-I$(WIND_BASE)/target/h/WPILib`
 - iii. `-I$(WIND_BASE)/target/h/wrn/coreip`
10. Switch to the *Libraries* tab
 - a. Add the following *Library directives* under *Libraries*
 - i. `$(WIND_BASE)/target/lib/WPILib.a`

Next we need to set up some various formatting options, starting with editing the Formatting Profile for C/C++.

1. Open the *Windriver C++ IDE*
2. In the main menu, click *Window -> Preferences*
3. On the left pane click, *C/C++ -> Code Style*
4. Select the *K&R [Built in]* profile, then click *Edit*.
5. Change the *Profile name:* to *K&R*
6. Under *Indent* check on *'public', 'protected', 'private' within class body*
7. Click *Ok*.

Content Assist is very helpful for remembering function names in the WPI Library, the default wait time for auto-assist is 500ms from when a '.', '->', or '::' is typed, some people might want that changed to 200ms or other time.

1. Open the *Windriver C++ IDE*
2. In the main menu, click *Window -> Preferences*
3. On the left pane click, *C/C++ -> Editor -> Content Assist*
4. Under *Auto-Activation* replace 500 with 200 beside *delay (ms)*

I like my remote connections (cRio) to automatically connect when I start Wind River, this can be easily enabled.

1. Open the *Windriver C++ IDE*
2. In the main menu, click *Window -> Preferences*
3. On the left pane, click, *Wind River -> Target Management*
4. Check on *Create default connections on each Workbench startup*

Testing New Robot Code Quickly with the Windriver C++ IDE:

The problem with repeatedly Undeploying and Downloading code to the cRio device is that it requires you to reboot in order to reload the new code. When making repeated minor changes this can become extremely time consuming. Luckily you can load a program into the cRio's RAM and run it, still having Dashboard control.

1. Make sure there is no previous executable written to non-volatile memory on the cRio, this seems to take precedence over code in RAM. Undeploy your code by clicking *FIRST -> Undeploy*
2. Ensure your Dashboard and Driver Station are loaded onto a computer on the network, and IP addresses are configured correctly.
3. In the *Project Manager* pane, right-click on your current project (make sure you've built any changes) and click *Run Kernel Task*.
4. Select a Connection, you should have previously connected in the *Remote Systems* pane.
5. Click *Browse* beside the *Entry Point* field and label, double-click the project you're working in and then select *FRC_UserProgram_StartupLibraryInit*
6. Then click the *Downloads* tab and select the first row in the *File* table, followed by clicking the *Edit* button.
7. Check on the *Download even when the file is not modified* option.
8. Click *Advanced Option*
9. Uncheck *do not unload the existing module*
10. Click *Ok* and *Run*.
11. Your changes in the *Run Dialog* are automatically saved.

Code Styling:

Due to multiple people working on the same project, code standards should be put in place to create uniform code that looks clean and is self readable.

Naming Scheme

- Function names should be in camel case, with the first letter capitalized.
e.g. `SetCenter()`
- Variable names should be in camel case, with the first letter in undercase.
e.g. `currentState = 5;`
- Class names should be in camel case, with the first letter capitalized.
e.g. `MyRobot {};`
- File names should be in camel case, with the first letter capitalized.
e.g. `StateMachine.cpp`
- Constants should be in all caps, in snake case.
e.g. `MOTOR_PORT_A`

Formatting

- Opening curly braces (`{`) should be placed on the same line as a function prototype, flow control statement, class, or other block as to prevent an opening curly brace from being on it's own line. The only exception being when the opening curly brace follows an initialization list.
- If a conditional only contains one line to execute no curly braces should be used.
- Scope statements (`public:`, `private:`, `protected:`) should be in the same column as the class name.