C++ Programming For 2010-2011 FRC Teams

Mike Anderson
(manderson13@cox.net)
and
Greg Smith
(gnsmith@cox.net)

Herndon High School
FRC Team #116

What We'll Talk About

- Goals
- Why C/C++?
- The development environment
- Talking to the cRIO
- Making it move
- Resources
- Summary

Goals

- The goal of this presentation is to help you understand how to use C/C++ in the development of your robot
- We clearly can't explain all of the aspects because we only have 75 minutes
- But, you should leave here with a better understanding of the process

Why C/C++?

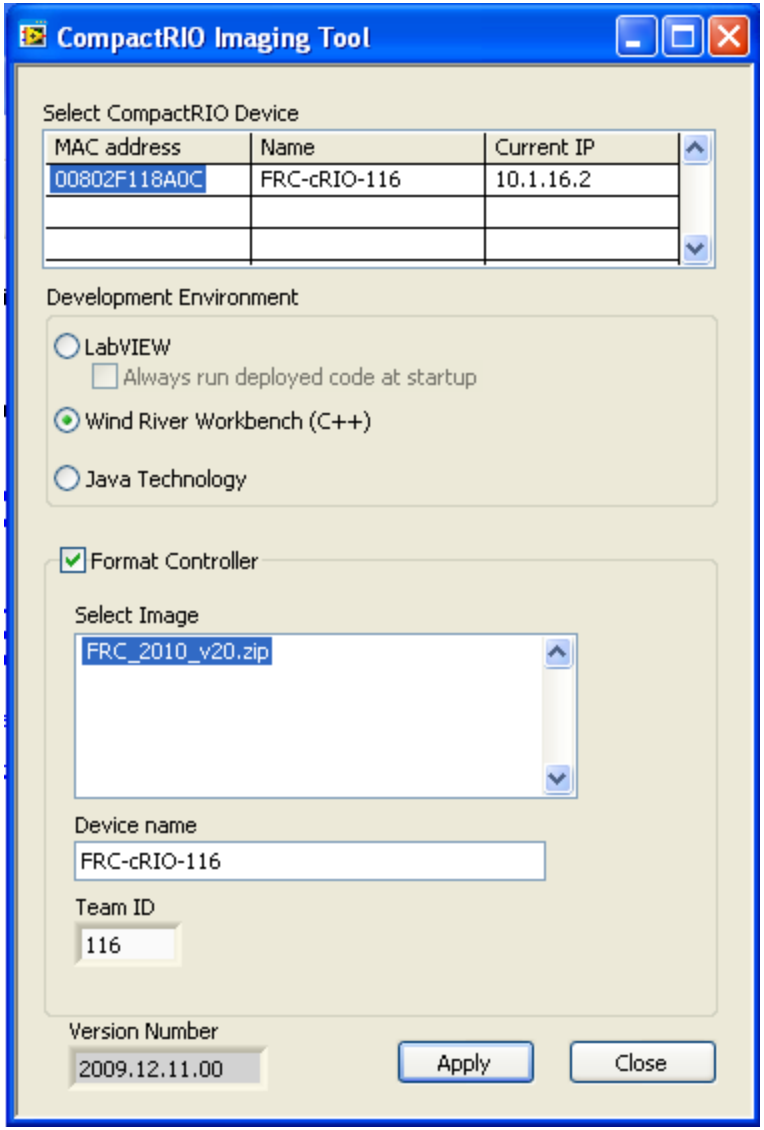
- C/C++ is a standard in embedded systems programming for over 30 years
  - It's still the most predominant language in the embedded, real-time operating system (RTOS) world
    - This gives your team valuable real-world experience
- It's compiled to native machine code
  - No virtual machine interpreters
    - No pausing due to garbage collection
  - It's fast
- It's the native language of the VxWorks RTOS
  - The environment is written in C and Assembler
  - You get easy, direct access to the underlying O/S
- C++ is object oriented
  - Full support from WPILib

Why Not C/C++?

- C/C++ is compiled
  - This adds complexity to the build
- C/C++ is textual
  - There are no cutesy GUIs with lots of obscure symbols and squiggly lines ☺
- There is no hand-holding VM to catch your mistakes
  - The syntax is similar to Java, but it's definitely not Java
- C/C++ has pointers
  - Objects can be referenced in many different ways
  - This concept can be too much for some developers

Getting Your cRIO Ready

- Before you can start development, you'll need to make sure that your cRIO has the proper operating system image on it
  - This is accomplished using the cRIO imaging tool



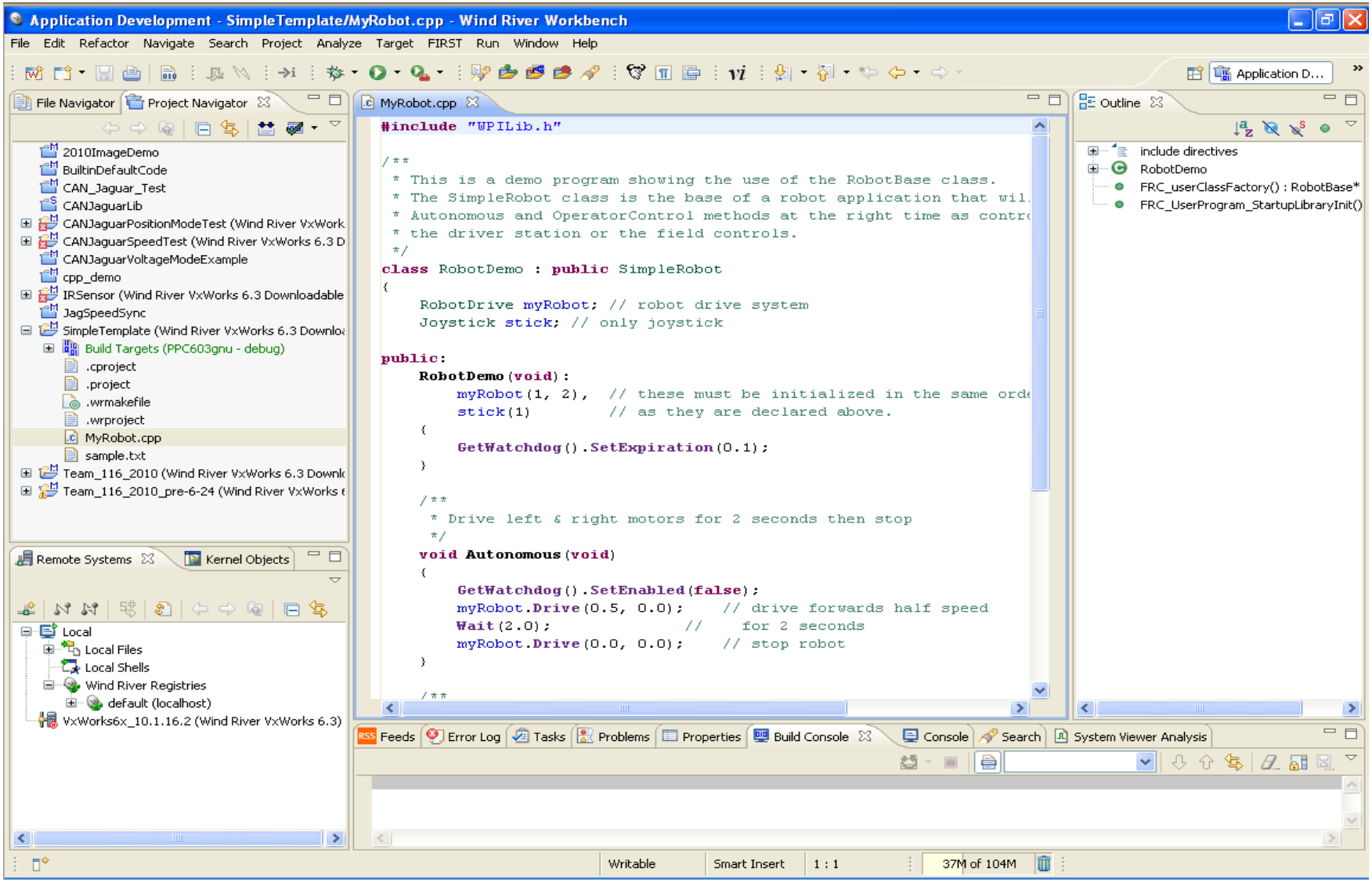
The Development Environment

- The FIRST provided platform is the Wind River Systems Workbench tool
  - IDE is based on the open-source Eclipse tool
  - The compiler is the open-source GNU compiler
    - But, the front-end is licensed and requires a key
      - Provided by FIRST
- The compiler is actually a *cross-compiler*
  - We are building on an x86 for a PowerPC
    - Again, this is a standard approach for commercial, embedded development

Development Environment #2

- Workbench runs under Windows and Linux
  - FIRST only supplies the Windows installation
- Workbench runs under Windows XP, Vista or Windows 7 (2011 season)
  - You can also run it in a virtual machine
    - For all you Mac OS/X and Linux fans
  - When you install it, there will be a key file that needs to be placed in c:\windriver\license

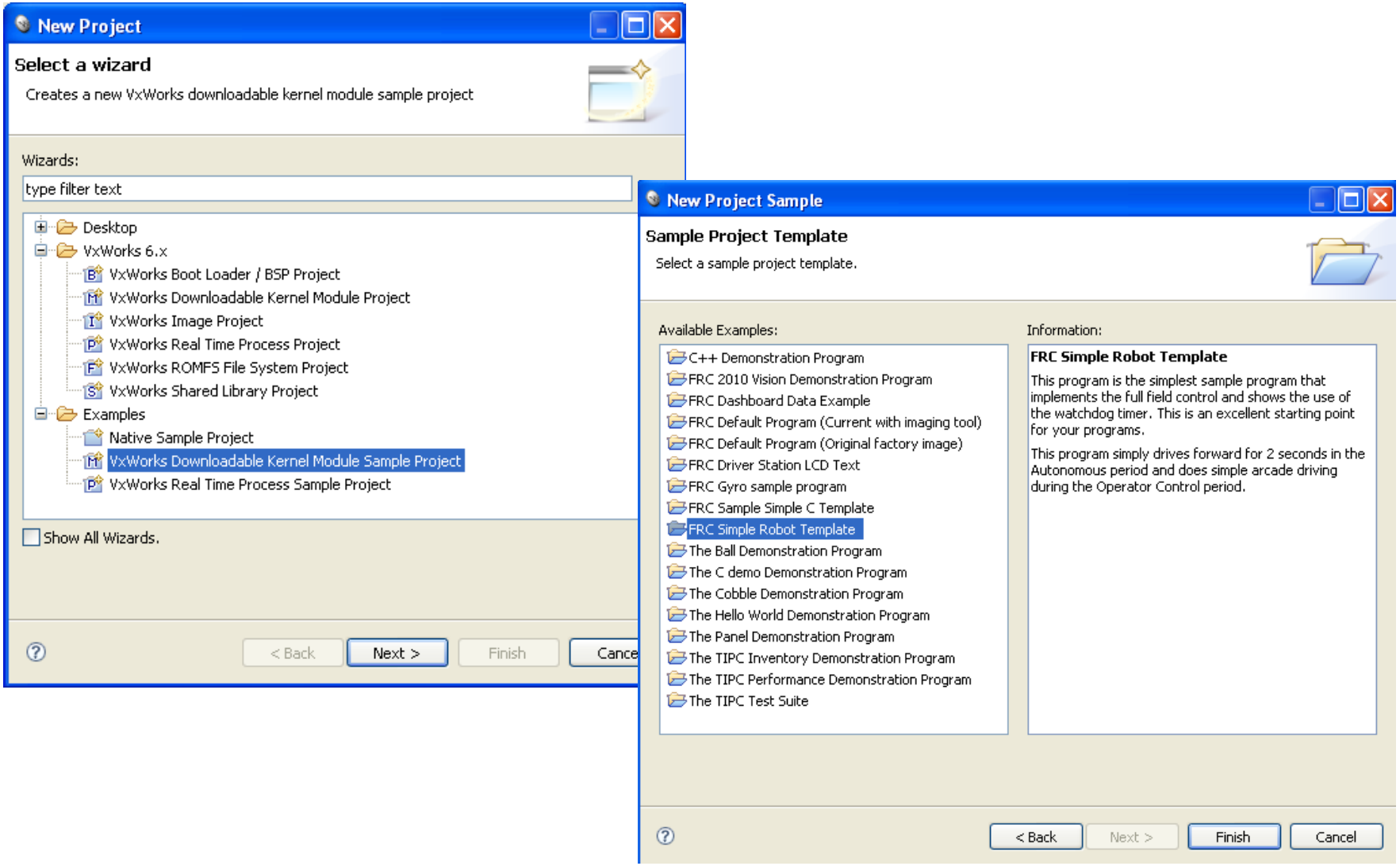
The WRS Workbench



Creating A Project

- Workbench collects all of the files related to building a piece of code into a subdirectory called a project folder
  - It's normally stored under the c:\windriver\workspace directory
    - You can put the project elsewhere when you open the workbench tool
- You can also import and export projects
  - This allows you to create a .zip of the project for archival purposes

New Project -- Simple Robot

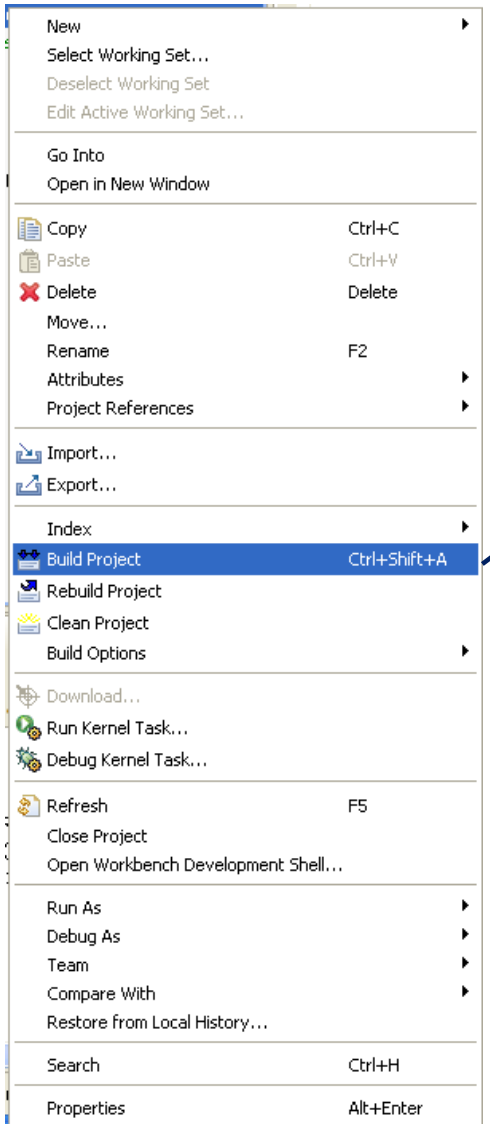


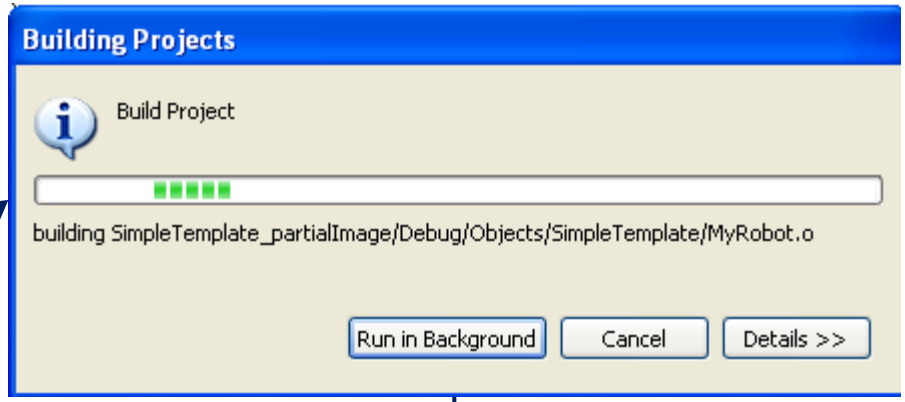
New Project Result

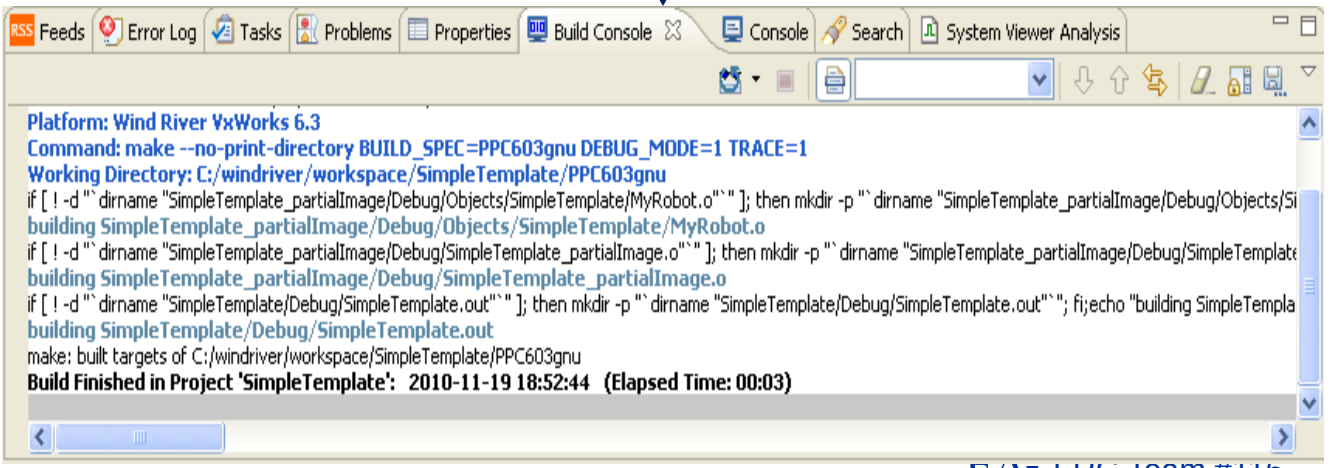
MyRobot.cpp | sample.txt

```
 */
class RobotDemo : public SimpleRobot
{
    RobotDrive myRobot; // robot drive system
    Joystick stick; // only joystick

public:
    RobotDemo(void):
        myRobot(1, 2),  // these must be initialized in the same order
        stick(1)        // as they are declared above.
    {
        GetWatchdog().SetExpiration(0.1);
    }

    /**
     * Drive left & right motors for 2 seconds then stop
     */
    void Autonomous(void)
    {
        GetWatchdog().SetEnabled(false);
        myRobot.Drive(0.5, 0.0);    // drive forwards half speed
        Wait(2.0);                  //    for 2 seconds
        myRobot.Drive(0.0, 0.0);    // stop robot
    }

    /**
     * Runs the motors with arcade steering.
     */
    void OperatorControl(void)
    {
        GetWatchdog().SetEnabled(true);
        while (IsOperatorControl())
        {
            GetWatchdog().Feed();
            myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
            Wait(0.005);                // wait for a motor update time
        }
    }
};
```

Build the Project

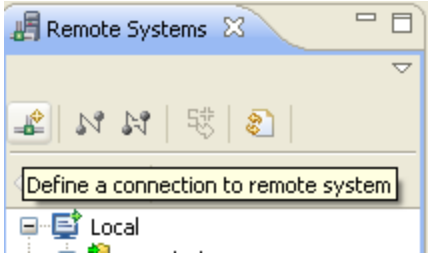




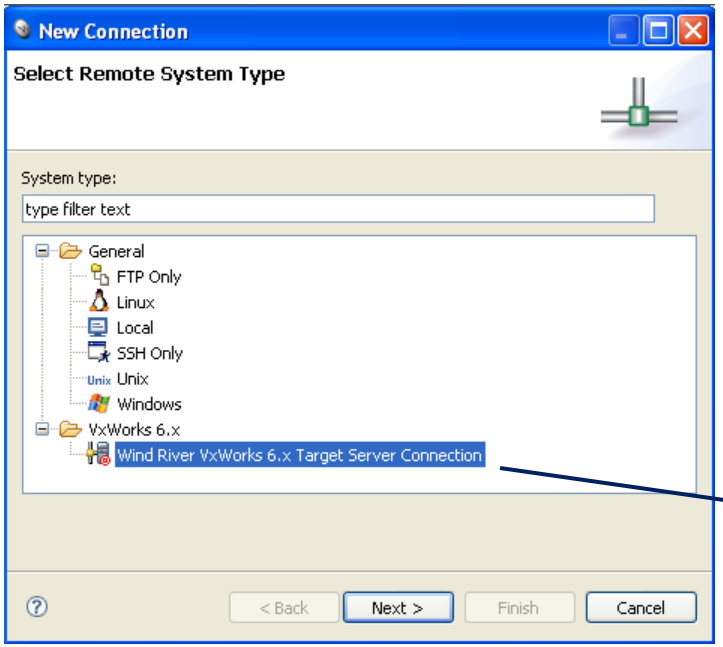


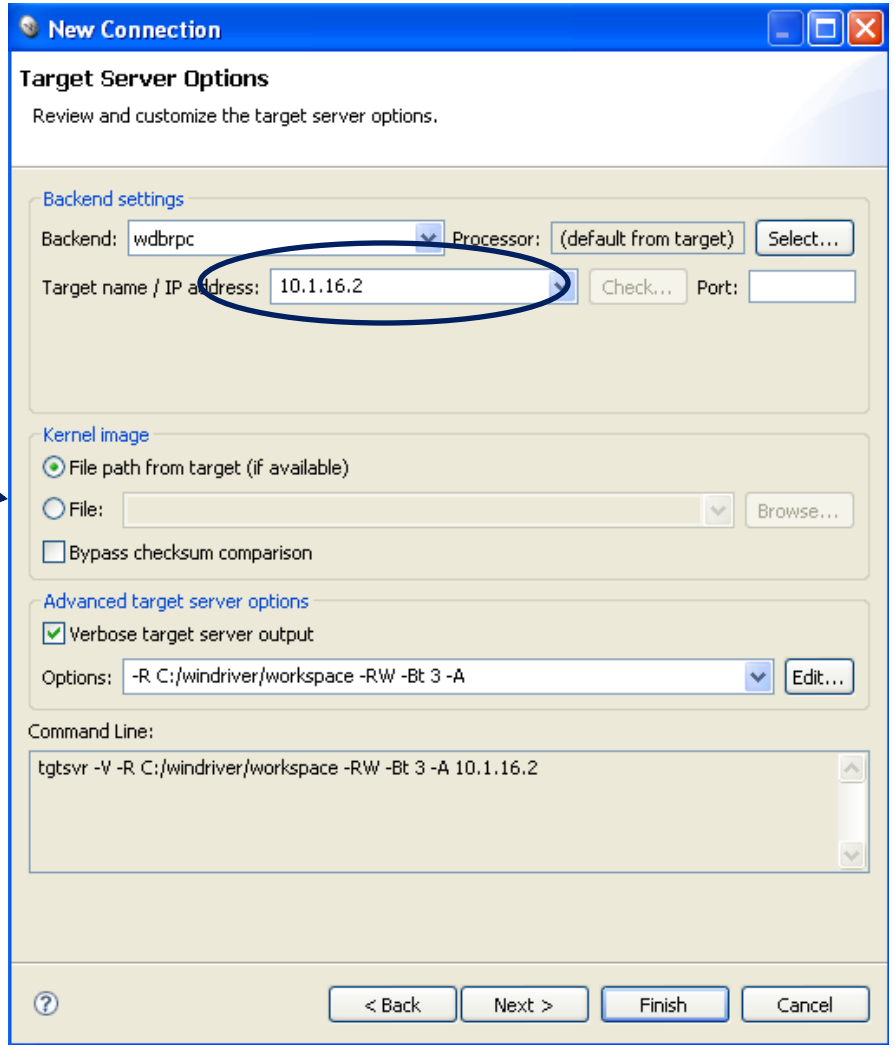
Attach to the Target

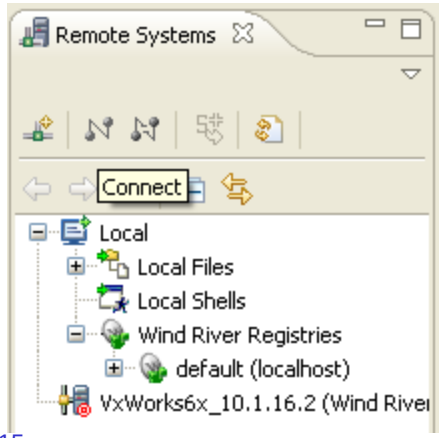
- Before you can load code to the robot, you need to create a target connection
  - This will also create a “target server” and “registry” instance
- It is possible to have multiple users attached to the same target
  - Not recommended if you’re both running robot code though ;-)



Attach to the Target #2





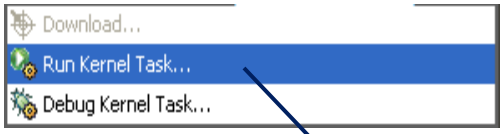


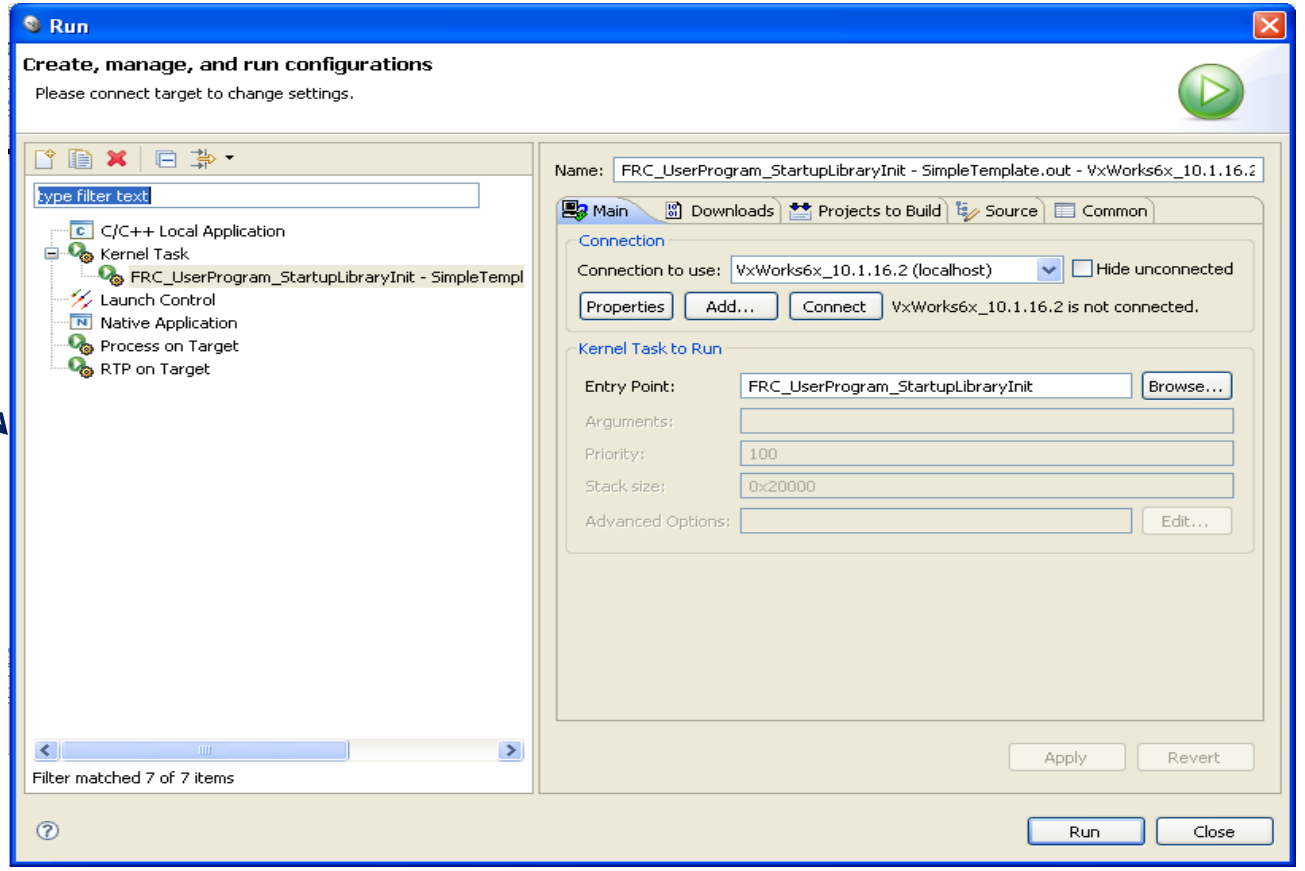
Attached Target

- Once the robot is attached to the target server, you'll see all of the tasks running on the target in the "Remote Systems" window
- If you create your own threads, they'll show up in the list as well

Running Code

- In order to run your robot code, you'll need to have the target attached
  - You'll need to create a "run config"

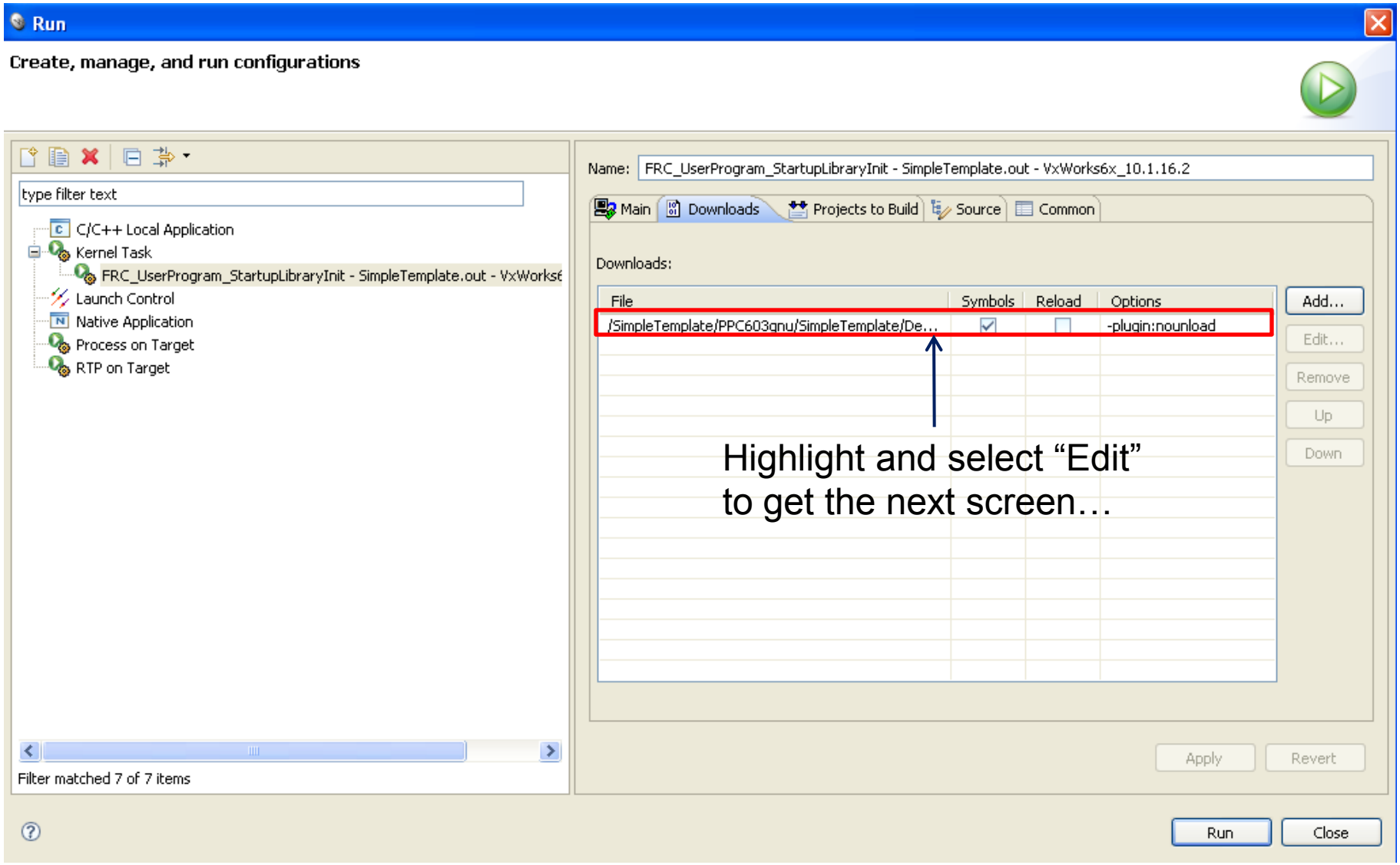




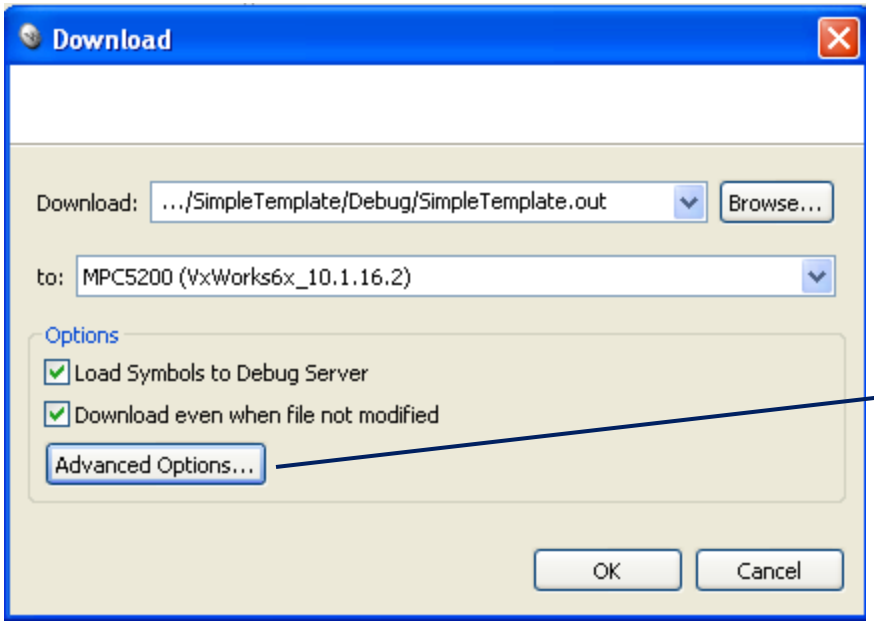
Handling Unloading

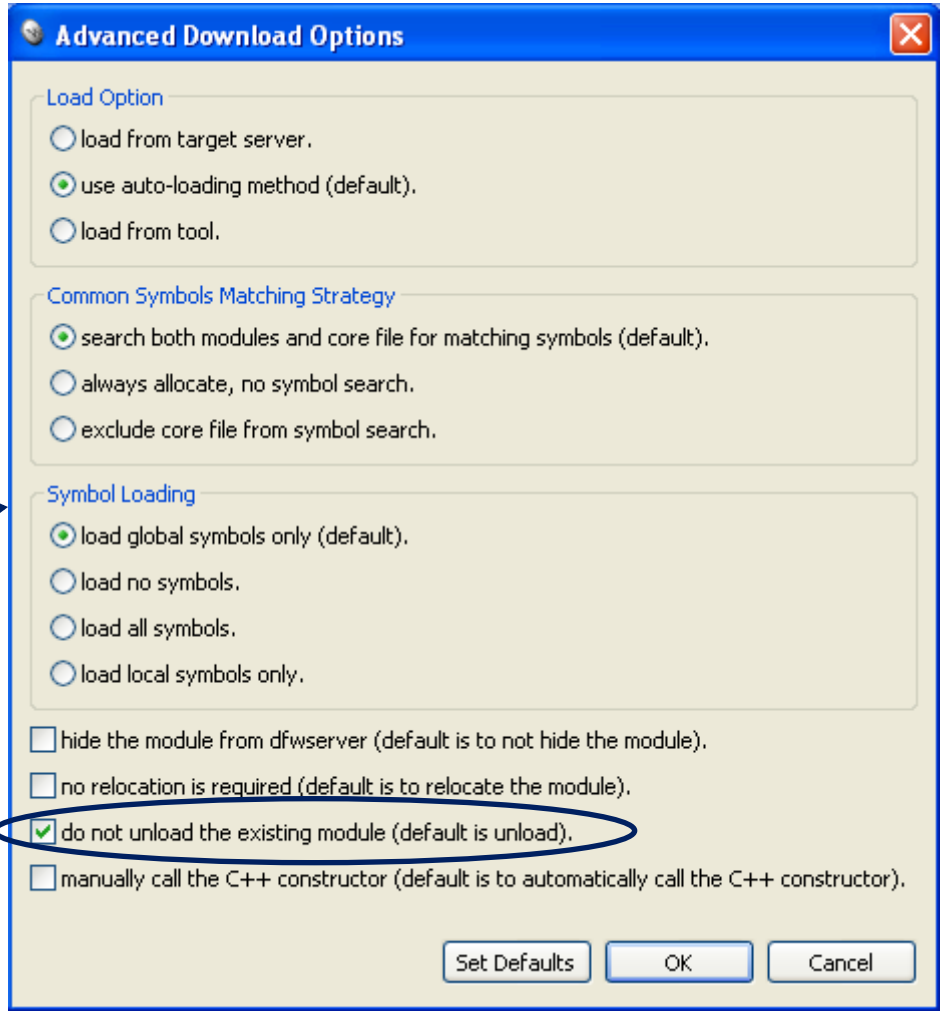
- Workbench has the option to automatically unload previously loaded code and replace it with the new code
  - Not the default behavior though
- There are a series of changes that need to be made to the run configuration to support automatic unloading

Unload Settings



Unload Settings #2



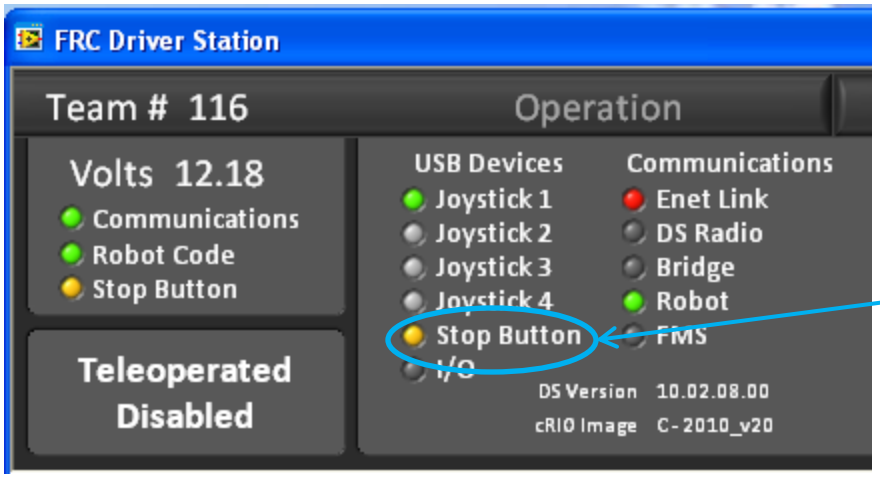


Change this setting to auto unload

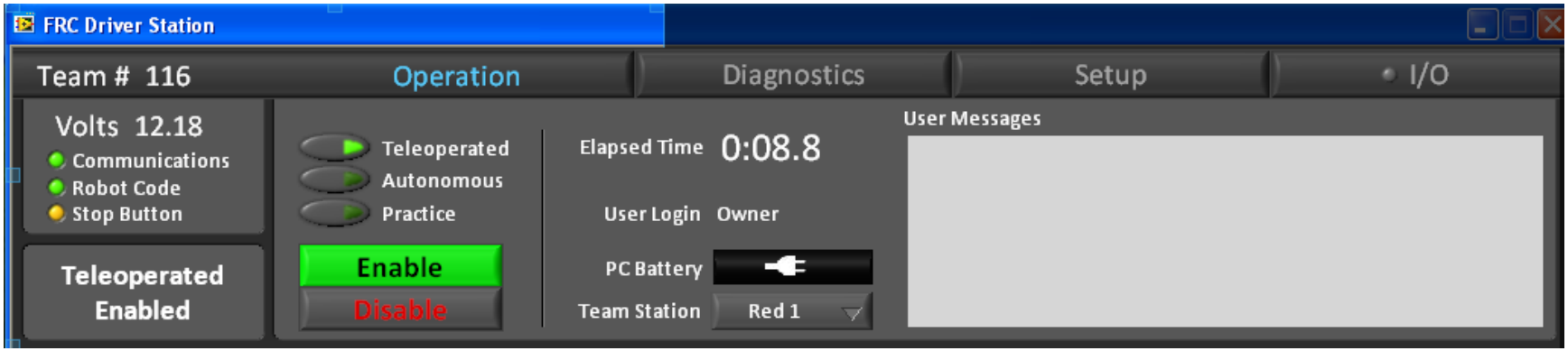
Running Code on the Robot

- Once your code is loaded, the entry point (`FRC_UserProgram_StartupLibraryInit`) will be called and your code will start running
  - Make sure that your Classmate is attached and you should be able to control the robot
- There is an option to disable the “STOP” button on the UI
  - Use with caution…
- You can then test Autonomous or Teleop code segments

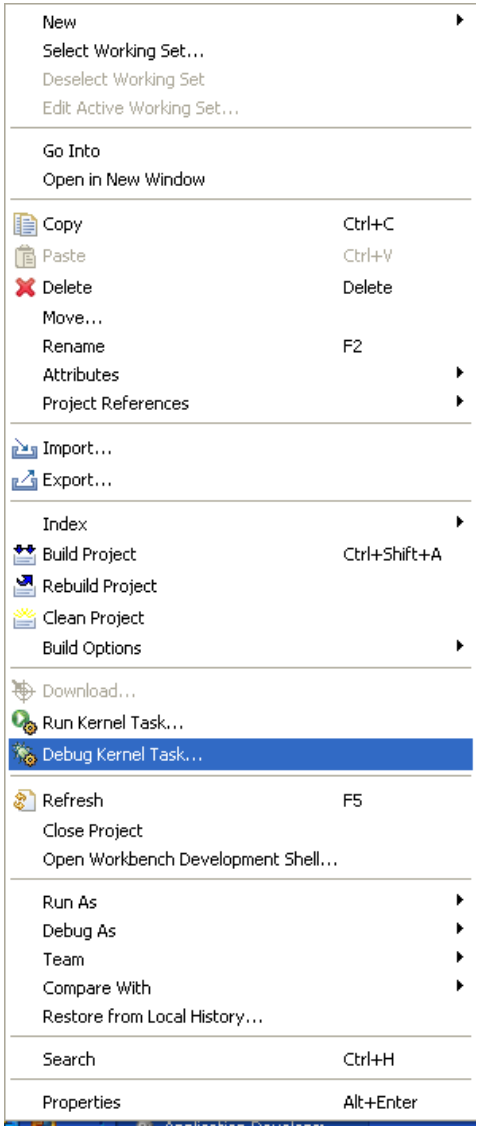
Disabling the Stop Button



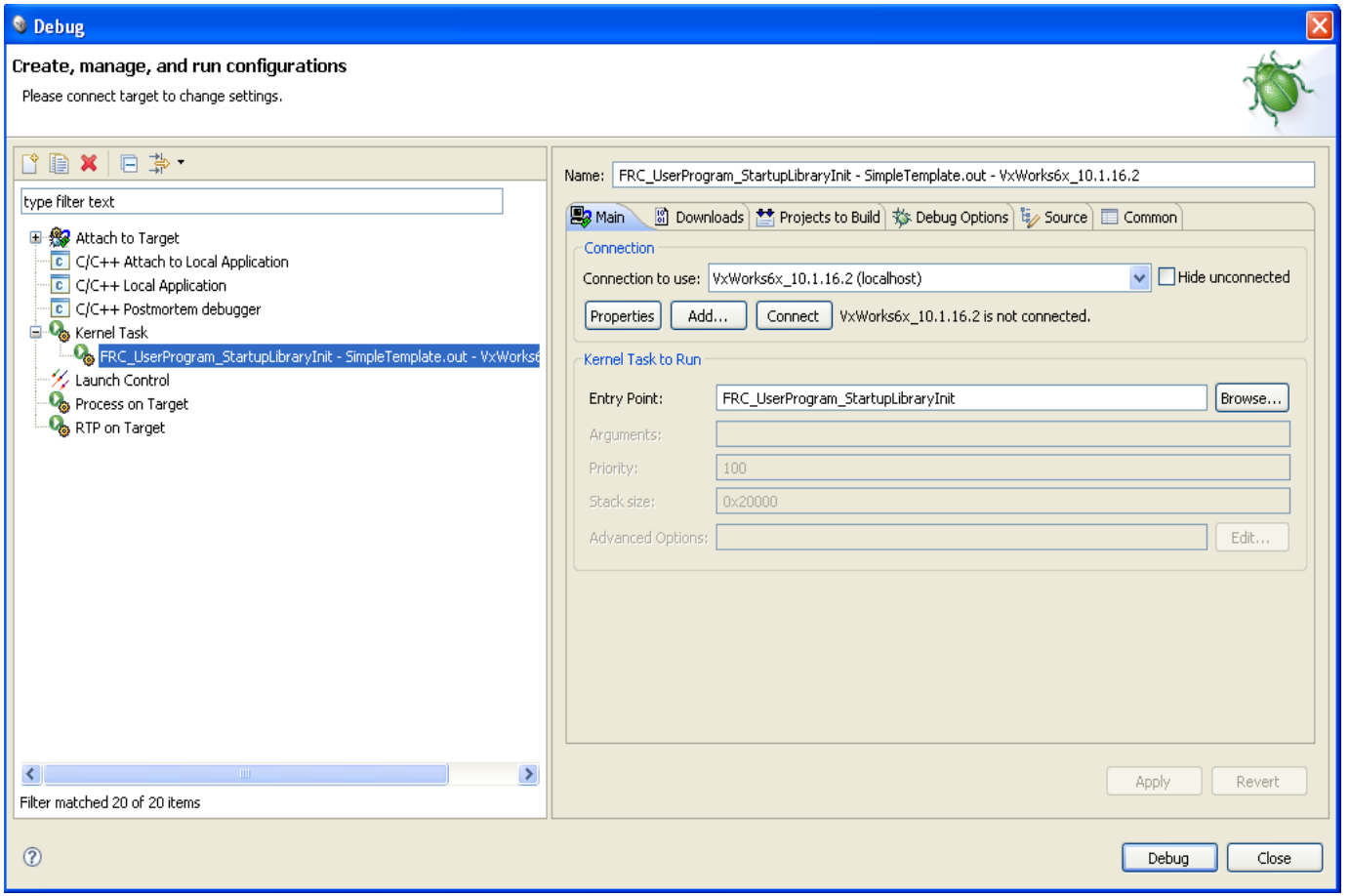
Click the LED and you'll get a dialog that allows you to disable the STOP button



Debugging Code

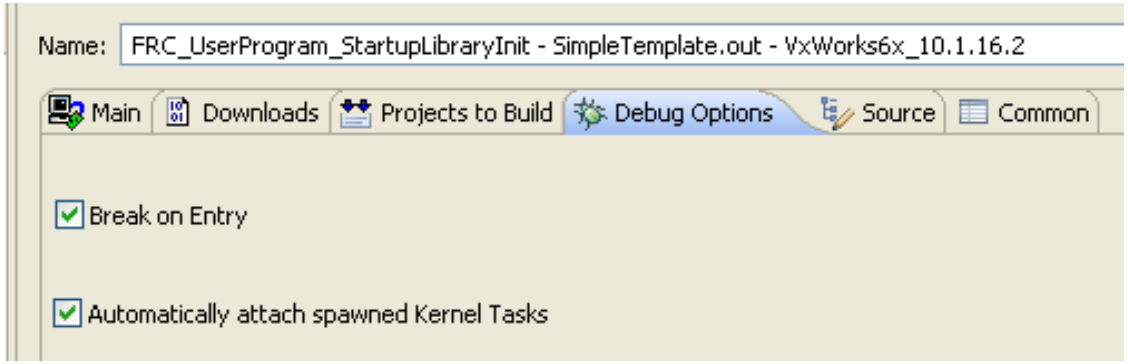


- In order to debug code, you'll need to create a debug launcher
- Check the debug option for automatically connect to spawned tasks



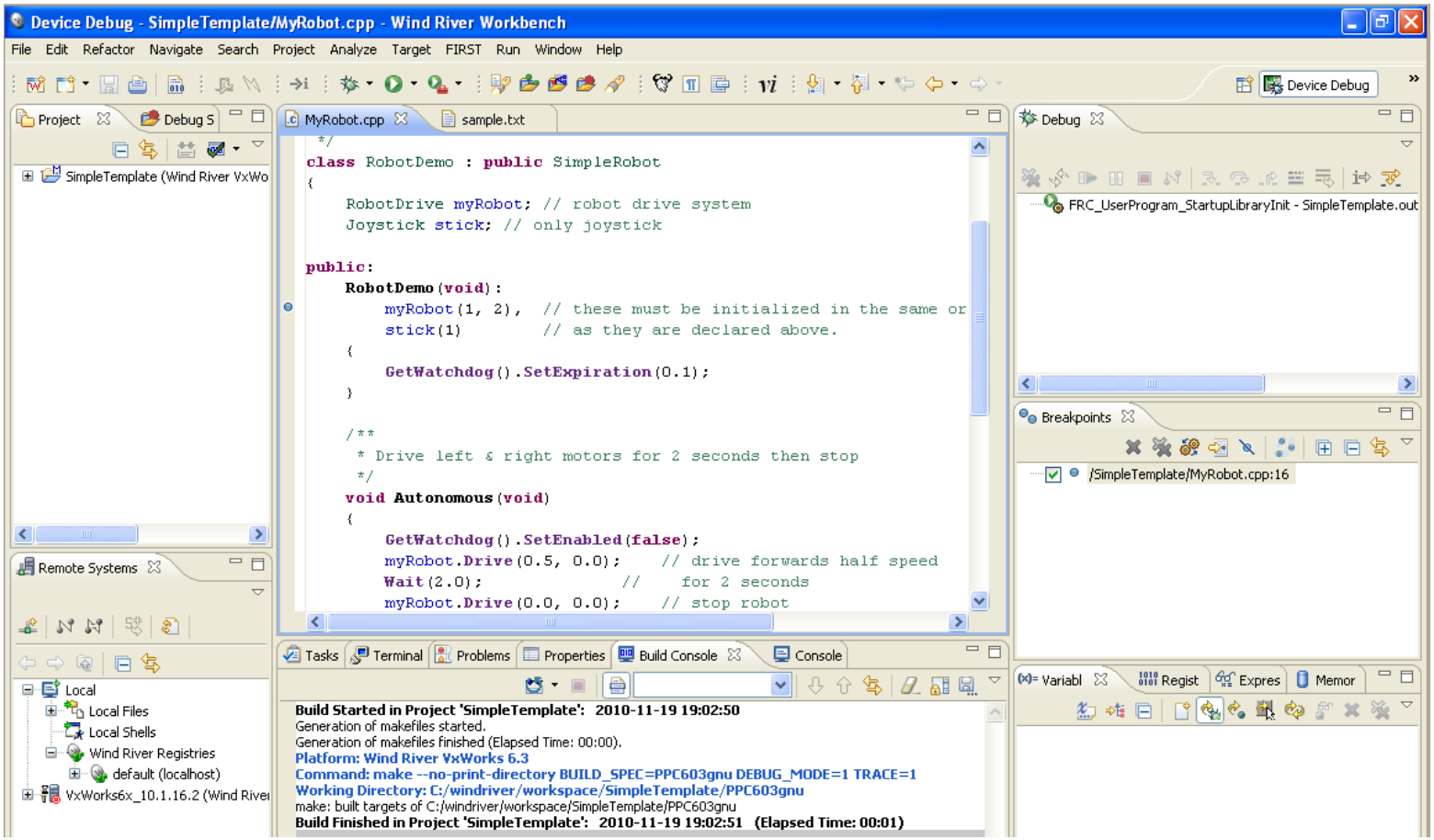
Attaching to Threads

- In order to debug your iterative robot, you need to make sure you automatically attach to spawned tasks
  - Otherwise you won't be able to debug the autonomous/teleop code
- In the debug launcher configuration, set the auto attach feature in the debug configuration:



Debugging Code #2

- Once debugging as started, Workbench will automatically switch to the debug perspective:

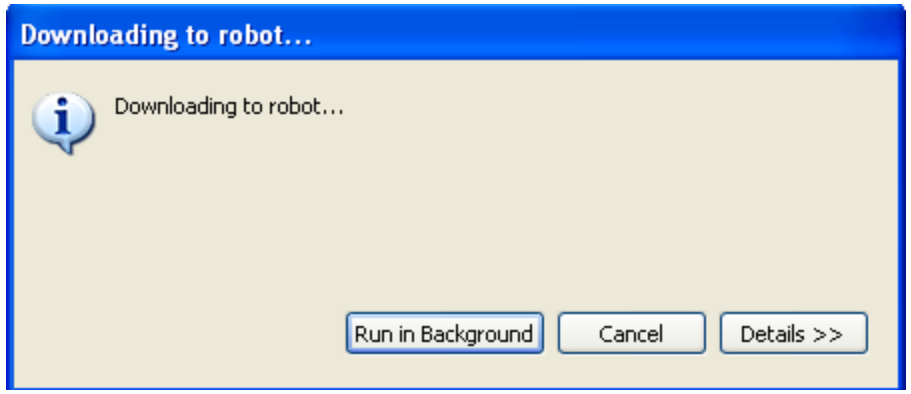


Debugging Code #3

- By default, the code will stop at the public constructor entry-point
- You can right-click in the code view “gutter” to toggle a breakpoint
  - Then tell the system to step or continue
- This is using the GDB debugger under the covers
  - If you understand GDB, you’re in familiar territory
    - If not, we’ll show you some examples during the demo

Deploying Code on the Robot

- Once you code is working, you can deploy the code to the robot
  - Only one program can be resident at a time



- Once the code is downloaded, you can reboot the robot and your code will run!
  - You can also do this with Filezilla (it's just FTP ☺)

Some Hints

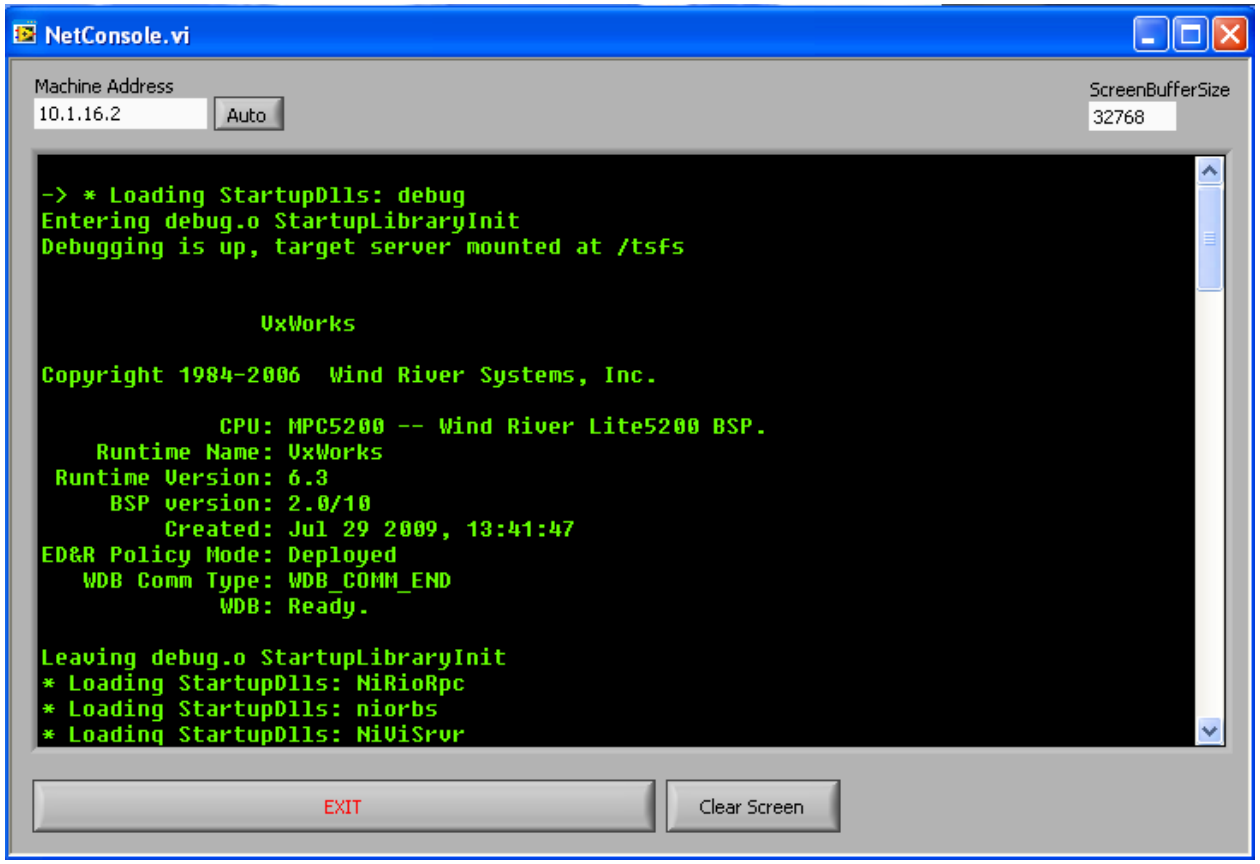
- The environment for C++ can be a bit tricky to deal with at first
  - Especially if you want to use CAN bus
- For CAN bus, you need to use a Black Jaguar and disable the standard console output on the cRIO (or use the 2CAN Ethernet bridge)
  - For the Black Jaguar, you need exclusive access to the serial port
  - Look on Chief Delphi for write-ups on CAN use

Some Hints #2

- To make the serial console available while still being able to monitor the cRIO console, load the network console application to the target and run the network console application on your host
  - All of the console I/O of the cRIO comes across the network now into a new window
- We'll give you a link to the network console application in the links

The Network Console

- Useful as an option to using the RS-232 port on the cRIO



Resources

- Chief Delphi
  - http://www.chiefdelphi.com
- FIRST forums
  - http://forums.usfirst.org
- NI Community Forums
  - http://ni.com/FIRST
- WPI / *FIRST* NSF Community site (ThinkTank)
- These sites are monitored members of:
  - WPI
  - NI
  - *FIRST*
- All source code available for team-team assistance
- Phone support through NI
  - 866-511-6285 (1PM-7PM CST, M-F) ?

Important Links

- WPILib updates and documentation
  - http://first.wpi.edu/FRC/frccupdates.html
  - http://first.wpi.edu/Images/CMS/First/WPI_Robotics_Library_Users_Guide.pdf
  - http://first.wpi.edu/Images/CMS/First/WPILibSource20100107.zip
  - http://first.wpi.edu/Images/CMS/First/CProgrammingReference.chm
    - doxygen – old
- Net console
  - http://first.wpi.edu/Images/CMS/First/NetConsoleClient_1.0.0.4.zip
- FIRST software resource page
  - http://www.usfirst.org/roboticsprograms/frc/content.aspx?id=1093

Summary

- C/C++ can be very challenging to new developers
  - C/C++ is similar enough to Java that Java developers can adapt to it quickly
    - However, pointers will require some explaining
- The rewards include:
  - Faster operation of the robot
  - Fine-grain control of the robot's behavior
  - The ability to leverage native VxWorks™ facilities
  - Training in techniques that will enable most of the "green" technology of the future