

# Creating Simple Commands

This article describes the basic format of a Command and walks through an example of creating a command to drive your robot with Joysticks.

## Basic Command Format

```
public class MyCommandName extends CommandBase {  
  
    public MyCommandName() {  
        super("MyCommandName");  
        requires(elevator);  
    }  
  
    public void initialize() {  
    }  
  
    public void execute() {  
    }  
  
    public boolean isFinished() {  
        return true-if-command-is-finished;  
    }  
}
```

To implement a command, a number of methods are overridden from the WPILib Command class. Most of the methods are boiler plate and can often be ignored, but are there for maximum flexibility when you need it. There a number of parts to this basic command class:

1. Constructor - Might have parameters for this command such as target positions of devices. Should also set the name of the command for debugging purposes. This will be used if the status is viewed in the dashboard. And the command should require (reserve) any devices is might use.
2. initialize() - This method sets up the command and is called immediately before the command is executed for the first time and every subsequent time it is started . Any initialization code should be here.
3. execute() - This method is called periodically (about every 20ms) and does the work of the command. Sometimes, if there is a position a subsystem is moving to, the command might set the target position for the subsystem in initialize() and have an empty execute() method.
4. isFinished() - This method returns true if the command is finished. This would be the case if the command has reached its target position, run for the set time, etc. There are other methods that might be useful to override and these will be discussed in later sections

# Creating Simple Commands

## Simple Command Example

```
public class DriveWithJoysticks extends CommandBase {  
  
    public DriveWithJoysticks() {  
        requires(drivetrain); 1  
    }  
  
    protected void initialize() {  
    }  
  
    protected void execute() {  
        drivetrain.tankDrive(oi.getLeftSpeed(), oi.getRightSpeed());  
    } 2  
  
    protected boolean isFinished() {  
        return false; 3  
    }  
  
    protected void end() {  
    }  
  
    protected void interrupted() {  
    }  
}
```

1. This example illustrates a simple command that will drive the robot using tank drive with values provided by the joysticks. The elements we've used in this command:
2. `requires(drivetrain)` - "drivetrain" is an instance of our Drivetrain subsystem. The instance is instantiated as static in Command Base so it can be referenced here. We need to require the drivetrain system as this command uses it when it executes.
3. `execute()` - In our execute method we call a `tankDrive` method we have created in our subsystem. This method takes two speeds as a parameter which we get from methods in the OI class. These methods abstract the joystick objects so that if we want to change how we get the speed later we can do so without modifying our commands (for example, if we want the joysticks to be less sensitive, we can multiply them by .5 in the `getLeftSpeed` method and leave our command the same).
4. `isFinished` - Our `isFinished` method always returns false meaning this command never completes on it's own. The reason we do this is that this command will be set as the default command for the subsystem. This means that whenever the subsystem is not running another command, it will run this command. If any other command is scheduled it will interrupt this command, then return to this command when the other command completes. For more on default commands see [Default Commands](#).