

# Creating more robust commands with timeouts

Often you might write a command that has an "isFinished" condition that is based on a sensor reaching a particular value or a switch closing. For example, a command might rotate a robot 45 degrees by turning until the heading from a gyro reaches 45. Suppose the gyro fails or a wire pulls loose. Now the gyro will always report the same heading, probably zero degrees, and the robot will keep turning for ever. One way of making your commands more robust is to have them "time out" or quit after some time has elapsed. In the case of the turn to 45 degrees, you might know that this will always complete within 2 seconds. You could set a timeout for 3 or 4 seconds so the command will finish even if the robot never reaches the desired heading. This will allow the program to continue running rather than be stuck waiting for the roboRIO to read a heading that will never happen.

## A command with no timeout

C++

```
TurnTo45Degrees::TurnTo45Degrees() : Command("TurnTo45Degrees") {  
}  
  
void TurnTo45Degrees::Initialize() {  
}  
  
void TurnTo45Degrees::Execute() {  
    Robot::drivetrain->turn(0.5); // start turning  
}  
  
bool TurnTo45Degrees::IsFinished() {  
    return Robot::driveTrain->getHeading() >= 45.0; // wait for 45 degrees  
}  
  
void TurnTo45Degrees::End() {  
    Robot::driveTrain->turn(0); // stop turning  
}  
  
void TurnTo45Degrees::Interrupted() {  
    End();  
}
```

# Creating more robust commands with timeouts

## Java

```
public class TurnTo45Degrees extends Command {
    public TurnTo45Degrees() {
    }

    protected void initialize() {
    }

    protected void execute() {
        Robot.chassis.turn(0.5); // start turning
    }

    protected boolean isFinished() {
        return Robot.chassis.getGyroHeading() >= 45.0; // wait for 45 degrees
    }

    protected void end() {
        Robot.chassis.turn(0); // stop turning
    }

    protected void interrupted() {
        end();
    }
}
```

The code above shows a command that will never time out, and will keep running until the desired heading is reached. Just a note, this is an over-simplified example to illustrate the use of timeouts and not as a way to necessarily get your robot to turn to headings - a PID controller would make the robot performance better.

## Adding a timeout to the command

## C++

```
TurnTo45Degrees::TurnTo45Degrees() : Command("TurnTo45Degrees") {
}

void TurnTo45Degrees::Initialize() {
    SetTimeout(4); // set 4 second timeout
}
```

# Creating more robust commands with timeouts

```
void TurnTo45Degrees::Execute() {
    Robot::drivetrain->turn(0.5); // start turning
}

bool TurnTo45Degrees::IsFinished() {
    // turn until 45 degrees OR timed out
    return Robot::driveTrain->getHeading() >= 45.0 || IsTimedOut();
}

void TurnTo45Degrees::End() {
    Robot::driveTrain->turn(0); // stop turning
}

void TurnTo45Degrees::Interrupted() {
    End();
}
```

## Java

```
public class TurnTo45Degrees extends Command {
    public TurnTo45Degrees() {
    }

    protected void initialize() {
        setTimeout(4); // set 4 second timeout
    }

    protected void execute() {
        Robot.chassis.turn(0.5); // start turning
    }

    protected boolean isFinished() {
        // wait for 45 degrees OR timed out
        return Robot.chassis.getGyroHeading() >= 45.0 || isTimedOut();
    }

    protected void end() {
        Robot.chassis.turn(0); // stop turning
    }
}
```

## Creating more robust commands with timeouts

```
protected void interrupted() {  
    end();  
}  
}
```

This example shows the same command but with a 4 second timeout. This guarantees that the robot will stop turning and the command will end in 4 seconds regardless of the gyro heading. Hopefully the gyro or its wiring will never fail, but at least you know that if it does, the command will eventually end.

## Adding a command to a group with a timeout

The timeout feature can also be used when adding a command to a command group even if the command itself has no timeout. To do this, add a second parameter to the `AddSequential()` or `AddParallel()` methods (`addSequential()` and `addParallel()` in Java) that is the timeout in seconds. This will cause the sequential command to finish either when the command itself finishes from the `IsFinished` (C++) or `isFinished` (Java) method returns true OR the command runtime reaches the timeout period. When using this feature, there is no indication in the command that the timeout is happening. If the command already has a timeout, the results are undefined.