FTC TEAM 6133
THE NUTS!

Walnut FTC SDK API b v1.0

The Acronyming

Yan Vologzhanin

1/15/2016

A document that explains how to use this API and what methods do. This API will allow coders to implement code last minute (as I know you guys like to do :P) without having to worry about getting the details wrong or without copy-pasting code and making typos in the process. In addition, bugs that appear in this API can be fixed without impacting any of the code that other coders make, allowing for a cleaner development cycle. Please use it ;-;

This booklet assumes you know the FTC SDK and Java 7 (in Android studio) and how to use them.

For documentation of this SDK, go to http://ftckey.com/apis/ftc/

If you don't know Java, well…. Go to Lynda.com and use your library card, I guess, for java, and ask me about OOP

Table of Contents and Installation

Contents

Table of Contents and Installation 1
TeleOp Methods 2
  How to Implement 2
  Motors 2
    Walnut Motor 2
    IncMotor 3
    DigMotor 3
    Control Methods 4
Auto Methods 5
    LinearMotor 5
    TimedMotor 5
    DistanceMotor 5

In order to implement this, you will need to use the following import statement at the start of all your docs

```
import com.walnutHillsEagles.WalnutLibrary.*
```

The '*' character is a wild card character and it means you will import the entire library so that it is usable.

The SDK can be downloaded on GitHub here - https://github.com/yanvolo/walnut_ftc_app

Download the zip and open it as a project in android studio

Take a look at the sample programs in the "opmodes" folder for examples of how this can be used.

NOTICE AS OF v1.0: Docs have all been removed from this folder to save download bandwidth. If you would like tutorial docs, please see the original, ftc-app on GitHub here- https://github.com/ftctechnh/ftc_app

TeleOp Methods

How to Implement

In order to implement Walnut TeleOp objects, you need to start background processing with a command in the start() method like so

```
public void start(){
    WalnutMotor.GamepadUpdater.startProcessing(this);
}
```

Feel free to add anything else inside the start() method, but that line is needed to turn on a parallel thread to keep this API working

NOTICE AS OF VERSION 1.0: This line is still necessary, but I discovered method references which might make this whole thing irrelevant in future versions. Including the line won't break anything in the future, so feel free to include it.

Motors

These are simply a single motor used during teleop. They can either be given a single controller stick or an unlimited number of buttons to control them. See below to see how.

Walnut Motor

An abstract class used as a framework for the other two motors.

Constructors

WalnutMotor(DcMotor myMotor, String myName, boolean encoderCheck)

Only used by child classes. **NOTICE:** *Pls don't use. The compiler will hate you :P* **OTHER NOTICE AS OF V1.0:** *I might remove myName parameter because I never use it and there is no reason for you to use it either*

Getters and Setters

Public DcMotor getMotor()

Method used to access motor

public boolean checkEncoders()

Allows user to check if this motor has encoders (**NOTICE AS OF V1.0**: *Might be removed because I can't find a reason to use this :/*)

public void setName(String myName)

Allows user to rename motor (**NOTICE AS OF V1.0:** *Probably will be removed because I can think of use of changing name midway through*)

public void toString()

Allows user to get name of motor for the sake of Telemetry.

Operator Methods

Methods the user will use

`public void operate()`

Method that is called on a motor object to run the motor. Takes care of calculations and checks necessary to run the motor without any prompting by user.

`public void stop()`

Method used to stop motor

`public void power(double pow)`

Method used to directly assign power to motor

IncMotor

Motor that uses the control sticks

Constructors

```
public IncMotor(DcMotor myMotor, String myName, boolean checkEncoders,
            String myControl, boolean reverse, double myDeadzone)
```

reverse tells if control stick value should be inverted and myDeadzone allows you to assign a deadzone value that overrides the one on the gamepad. See “Control Methods” below to see what values you can input.

Other Methods

`public void operate()`

Operates dah motor.

DigMotor

NOTICE: If you would like to use a Servo instead, make this a “WalnutServo” object and give the constructor a servo. It holds the exact same methods, it just is a servo instead.

Constructors

```
  public DigMotor(DcMotor myMotor, String myName, boolean encoderCheck,
            String button, double power, boolean toggle)
```

See below in “Control Methods” to see what values for button you can put in. Power is the power value associated with pressing that button, and toggle tells if the button toggles the power value (true) or if it must be held (false).

`public void addButton(String button, double power, boolean toggle)`

Allows the user to add additional buttons along with all other previous buttons to be associated with this motor. Note that multiple motors can be assigned to the same button. It still works! :D

Control Methods

So DigMotor and IncMotor have Strings associated with them that tells what button or stick to use. These are the Strings you can use (they are not case sensitive).

Analog

LEFTX1 - Gamepad 1, X axis, left stick
LEFTY1 - Gamepad 1, Y axis, left stick
RIGHTX1 - Gamepad 1, X axis, right stick
RIGHTY1 - Gamepad 1, Y axis, right stick
LEFTZ1 - Gamepad 1, Left Trigger
RIGHTZ1 - Gamepad 1, Right Trigger
LEFTX2 - Gamepad 2, X axis, left stick
LEFTY2 - Gamepad 2, Y axis, left stick
RIGHTX2 - Gamepad 2, X axis, right stick
RIGHTY2 - Gamepad 2, Y axis, right stick
LEFTZ2 - Gamepad 2, Left Trigger
RIGHTZ2 - Gamepad 2, Right Trigger

Digital

A1 - Gamepad 1, a button
B1 - Gamepad 1, b button
X1- Gamepad 1, x button
Y1- Gamepad 1, y button
BACK1 - Gamepad 1, back button
START1- Gamepad 1, start button
GUIDE1 - Gamepad 1, guid button
LEFT1 - Gamepad 1, dpad left
RIGHT1 - Gamepad 1, dpad right
DOWN1 - Gamepad 1, dpad down
UP1- Gamepad 1, dpad up
LBUMP1 - Gamepad 1, Left bumper
RBUMP1 - Gamepad 1, right bumper
LSTICK1 - Gamepad 1, left stick button
RSTICK1 - Gamepad 1, right stick button
A2- Gamepad 2, a button
B2 - Gamepad 2, b button
X2- Gamepad 2, x button
Y2- Gamepad 2, y button
BACK2 - Gamepad 2, back button
START2 - Gamepad 2, start button
GUIDE2- Gamepad 2, guid button
LEFT2- Gamepad 2, dpad left
RIGHT2- Gamepad 2, dpad right
DOWN2- Gamepad 2, dpad down
UP2 - Gamepad 2, dpad up
LBUMP2 - Gamepad 2, Left bumper
RBUMP2- Gamepad 2, right bumper
LSTICK2 - Gamepad 2, left stick button
RSTICK2 - Gamepad 2, right stick button

Auto Methods

These motors are initialized without a control method, and their operate method takes two values instead of being null. Note that all operate()'s happen in parallel, so that means you can do multiple actions at once easily, but if you want everything to wait, you will need to pause your program, probably with something like Thread.sleep().

LinearMotor

Constructors

```
LinearMotor(DcMotor myMotor, String myName, boolean encoderCheck, boolean isReversed)
```

isReversed simply reverses the orientation of the motor so that you don't have to type in negative values for no reason.

Other Methods

```
public void stopMotor()
```

Stops the motor. Useful for emergency stops.

TimedMotor

Motor that uses time with a power value to go. Has same constructor as LinearMotor, and its operate() takes a time in double and a power value in float.

DistanceMotor

Constructor

```
public DistanceMotor(DcMotor myMotor, String myName, boolean encoderCheck,boolean isReveresed, double myDiameter,double myGearRatio, int myEncoder)
```

Same as linear motor with notable additions. You also need to give the diameter of the wheel in inches, the gear ratio, and the value for a full rotation on your encoder (it is 1440 for optical encoders).
Then you can use operate to travel a certain number of inches.