RobotPy CommandV1 Documentation

Release 2020.3.2.0

RobotPy development team

Feb 21, 2020

Robot Programming

CHAPTER 1

Commands V1 API

Objects in this package allow you to implement a robot using the legacy version of WPILib's Command-based programming. Command based programming is a design pattern to help you organize your robot programs, by organizing your robot program into components based on *Command* and *Subsystem*

Each one of the objects in the Command framework has detailed documentation available. If you need more information, for examples, tutorials, and other detailed information on programming your robot using this pattern, we recommend that you consult the Java version of the FRC Control System documentation

Note: As of 2020, the legacy command framework is distributed separately from WPILib.

1.1 wpilib.buttons Package

wpilib.buttons.Button(self)	This class provides an easy way to link commands to OI inputs.
wpilib.buttons.ButtonScheduler(self, last, ...)	
wpilib.buttons.CancelButtonScheduler(self, ...)	
wpilib.buttons.HeldButtonScheduler(self, ...)	
wpilib.buttons.InternalButton(*args, **kwargs)	Overloaded function.
wpilib.buttons.JoystickButton(self, ...)	
wpilib.buttons.NetworkButton(*args, **kwargs)	Overloaded function.
wpilib.buttons.POVButton(self, joystick, ...)	Creates a POV button for triggering commands.
wpilib.buttons.PressedButtonScheduler(self, ...)	

Continued on next page

Table 1 – continued from previous page

wpilib.buttons.ReleasedButtonScheduler(self, ...)	
wpilib.buttons.ToggleButtonScheduler(self, ...)	
wpilib.buttons.Trigger(self)	This class provides an easy way to link commands to inputs.

1.1.1 Button

class wpilib.buttons.**Button**() → None

Bases: *wpilib.command.Trigger*

This class provides an easy way to link commands to OI inputs.

It is very easy to link a button to a command. For instance, you could link the trigger button of a joystick to a "score" command.

This class represents a subclass of Trigger that is specifically aimed at buttons on an operator interface as a common use case of the more generalized Trigger objects. This is a simple wrapper around Trigger with the method names renamed to fit the Button object use.

cancelWhenPressed(*command: wpilib.command._commands_v1.Command*) → None

Cancels the specificed command when the button is pressed.

Parameters **command** – The command to be canceled

toggleWhenPressed(*command: wpilib.command._commands_v1.Command*) → None

Toggle the specified command when the button is pressed.

Parameters **command** – The command to be toggled

whenPressed(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to run when a button is first pressed.

Parameters **command** – The pointer to the command to run

whenReleased(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to run when the button is released.

The command will be scheduled a single time.

Parameters **command** – The pointer to the command to run

whileHeld(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to be scheduled while the button is pressed.

The command will be scheduled repeatedly while the button is pressed and will be canceled when the button is released.

Parameters **command** – The pointer to the command to run

1.1.2 ButtonScheduler

class wpilib.buttons.**ButtonScheduler**(*last: bool*, *button: frc::Trigger*, *orders: frc::Command*) → None

Bases: pybind11_builtins.pybind11_object

execute() → None

start() → None

1.1.3 CancelButtonScheduler

class wpilib.buttons.**CancelButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.1.4 HeldButtonScheduler

class wpilib.buttons.**HeldButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.1.5 InternalButton

class wpilib.buttons.**InternalButton**(**args*, ***kwargs*)

Bases: *wpilib.command.Button*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.InternalButton) -> None
2. __init__(self: wpilib.command._commands_v1.InternalButton, inverted: bool) -> None

get() → bool

setInverted(*inverted: bool*) → None

setPressed(*pressed: bool*) → None

1.1.6 JoystickButton

class wpilib.buttons.**JoystickButton**(*joystick: wpilib.interfaces._interfaces.GenericHID*, *buttonNumber: int*) → None

Bases: *wpilib.command.Button*

get() → bool

1.1.7 NetworkButton

class wpilib.buttons.**NetworkButton**(**args*, ***kwargs*)

Bases: *wpilib.command.Button*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.NetworkButton, tableName: str, field: str) -> None
2. __init__(self: wpilib.command._commands_v1.NetworkButton, table: _pyntcore._ntcore.NetworkTable, field: str) -> None

get() → bool

1.1.8 POVButton

class wpilib.buttons.**POVButton**(*joystick: wpilib.interfaces._interfaces.GenericHID*, *angle: int*, *povNumber: int = 0*) → None

Bases: *wpilib.command.Button*

Creates a POV button for triggering commands.

Parameters

- **joystick** – The GenericHID object that has the POV
- **angle** – The desired angle in degrees (e.g. 90, 270)
- **povNumber** – The POV number (@see GenericHID#GetPOV)

get() → bool

1.1.9 PressedButtonScheduler

class wpilib.buttons.**PressedButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.1.10 ReleasedButtonScheduler

class wpilib.buttons.**ReleasedButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.1.11 ToggleButtonScheduler

class wpilib.buttons.**ToggleButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.1.12 Trigger

class wpilib.buttons.**Trigger**() → None

Bases: wpilib._wpilib.Sendable

This class provides an easy way to link commands to inputs.

It is very easy to link a polled input to a command. For instance, you could link the trigger button of a joystick to a "score" command or an encoder reaching a particular value.

It is encouraged that teams write a subclass of Trigger if they want to have something unusual (for instance, if they want to react to the user holding a button while the robot is reading a certain sensor input). For this, they only have to write the `Trigger.Get()` method to get the full functionality of the Trigger class.

cancelWhenActive(*command: wpilib.command._commands_v1.Command*) → None

get() → bool

grab() → bool

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

toggleWhenActive(*command: wpilib.command._commands_v1.Command*) → None

whenActive(*command: wpilib.command._commands_v1.Command*) → None

whenInactive(*command: wpilib.command._commands_v1.Command*) → None

whileActive(*command: wpilib.command._commands_v1.Command*) → None

1.2 wpilib.command Package

wpilib.command.Button(self)	This class provides an easy way to link commands to OI inputs.
wpilib.command.ButtonScheduler(self, last, ...)	
wpilib.command.CancelButtonScheduler(self, ...)	
wpilib.command.Command(*args, **kwargs)	The Command class is at the very core of the entire command framework.
wpilib.command.CommandGroup(*args, **kwargs)	A CommandGroup is a list of commands which are executed in sequence.
wpilib.command.CommandGroupEntry(*args, **kwargs)	Overloaded function.
wpilib.command.ConditionalCommand(*args, ...)	A ConditionalCommand is a Command that starts one of two commands.
wpilib.command.HeldButtonScheduler(self, ...)	
wpilib.command.InstantCommand(*args, **kwargs)	This command will execute once, then finish immediately afterward.
wpilib.command.InternalButton(*args, **kwargs)	Overloaded function.
wpilib.command.JoystickButton(self, ...)	
wpilib.command.NetworkButton(*args, **kwargs)	Overloaded function.
wpilib.command.PIDCommand(*args, **kwargs)	Overloaded function.
wpilib.command.PIDSubsystem(*args, **kwargs)	This class is designed to handle the case where there is a Subsystem which uses a single PIDController almost constantly (for instance, an elevator which attempts to stay at a constant height).
wpilib.command.POVButton(self, joystick, ...)	Creates a POV button for triggering commands.
wpilib.command.PressedButtonScheduler(self, ...)	
wpilib.command.PrintCommand(self, message)	

Continued on next page

Table 2 – continued from previous page

wpilib.command.ReleasedButtonScheduler(self, ...)	
wpilib.command.Scheduler	
wpilib.command.StartCommand(self, commandToStart)	
wpilib.command.Subsystem(self, name)	Creates a subsystem with the given name.
wpilib.command.TimedCommand(*args, **kwargs)	A TimedCommand will wait for a timeout before finishing.
wpilib.command.ToggleButtonScheduler(self, ...)	
wpilib.command.Trigger(self)	This class provides an easy way to link commands to inputs.
wpilib.command.WaitCommand(*args, **kwargs)	Overloaded function.
wpilib.command.WaitForChildren(*args, **kwargs)	Overloaded function.
wpilib.command.WaitUntilCommand(*args, **kwargs)	Overloaded function.

1.2.1 Button

class wpilib.command.**Button**() → None

Bases: *wpilib.command.Trigger*

This class provides an easy way to link commands to OI inputs.

It is very easy to link a button to a command. For instance, you could link the trigger button of a joystick to a "score" command.

This class represents a subclass of Trigger that is specifically aimed at buttons on an operator interface as a common use case of the more generalized Trigger objects. This is a simple wrapper around Trigger with the method names renamed to fit the Button object use.

cancelWhenPressed(*command: wpilib.command._commands_v1.Command*) → None

Cancels the specificed command when the button is pressed.

Parameters **command** – The command to be canceled

toggleWhenPressed(*command: wpilib.command._commands_v1.Command*) → None

Toggle the specified command when the button is pressed.

Parameters **command** – The command to be toggled

whenPressed(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to run when a button is first pressed.

Parameters **command** – The pointer to the command to run

whenReleased(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to run when the button is released.

The command will be scheduled a single time.

Parameters **command** – The pointer to the command to run

whileHeld(*command: wpilib.command._commands_v1.Command*) → None

Specifies the command to be scheduled while the button is pressed.

The command will be scheduled repeatedly while the button is pressed and will be canceled when the button is released.

Parameters **command** – The pointer to the command to run

1.2.2 ButtonScheduler

class wpilib.command.**ButtonScheduler**(*last: bool*, *button: frc::Trigger*, *orders: frc::Command*) → None

Bases: pybind11_builtins.pybind11_object

execute() → None

start() → None

1.2.3 CancelButtonScheduler

class wpilib.command.**CancelButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.2.4 Command

class wpilib.command.**Command**(**args*, ***kwargs*)

Bases: wpilib._wpilib.ErrorBase, wpilib._wpilib.Sendable

The Command class is at the very core of the entire command framework.

Every command can be started with a call to Start(). Once a command is started it will call Initialize(), and then will repeatedly call Execute() until the IsFinished() returns true. Once it does,End() will be called.

However, if at any point while it is running Cancel() is called, then the command will be stopped and Interrupted() will be called.

If a command uses a Subsystem, then it should specify that it does so by calling the Requires() method in its constructor. Note that a Command may have multiple requirements, and Requires() should be called for each one.

If a command is running and a new command with shared requirements is started, then one of two things will happen. If the active command is interruptible, then Cancel() will be called and the command will be removed to make way for the new one. If the active command is not interruptible, the other one will not even be started, and the active one will continue functioning.

@see CommandGroup @see Subsystem

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.Command) -> None

Creates a new command.

The name of this command will be default.

2. __init__(self: wpilib.command._commands_v1.Command, name: str) -> None

Creates a new command with the given name and no timeout.

Parameters **name** – the name for this command

3. __init__(self: wpilib.command._commands_v1.Command, timeout: float) -> None

Creates a new command with the given timeout and a default name.

Parameters **timeout** – the time (in seconds) before this command "times out" @see IsTimedOut()

4. __init__(self: wpilib.command._commands_v1.Command, subsystem: frc::Subsystem) -> None

Creates a new command with the given timeout and a default name.

Parameters **subsystem** – the subsystem that the command requires

5. __init__(self: wpilib.command._commands_v1.Command, name: str, timeout: float) -> None

Creates a new command with the given name and timeout.

Parameters

- **name** – the name of the command
- **timeout** – the time (in seconds) before this command "times out" @see IsTimedOut()

6. __init__(self: wpilib.command._commands_v1.Command, name: str, subsystem: frc::Subsystem) -> None

Creates a new command with the given name and timeout.

Parameters

- **name** – the name of the command
- **subsystem** – the subsystem that the command requires

7. __init__(self: wpilib.command._commands_v1.Command, timeout: float, subsystem: frc::Subsystem) -> None

Creates a new command with the given name and timeout.

Parameters

- **timeout** – the time (in seconds) before this command "times out"
- **subsystem** – the subsystem that the command requires @see IsTimedOut()

8. __init__(self: wpilib.command._commands_v1.Command, name: str, timeout: float, subsystem: frc::Subsystem) -> None

Creates a new command with the given name and timeout.

Parameters

- **name** – the name of the command
- **timeout** – the time (in seconds) before this command "times out"
- **subsystem** – the subsystem that the command requires @see IsTimedOut()

assertUnlocked(*message: str*) → bool
If changes are locked, then this will generate a CommandIllegalUse error.

Parameters **message** – The message to report on error (it is appended by a default message)

Returns True if assert passed, false if assert failed.

cancel() → None
This will cancel the current command.

This will cancel the current command eventually. It can be called multiple times. And it can be called when the command is not running. If the command is running though, then the command will be marked as canceled and eventually removed.

A command can not be canceled if it is a part of a command group, you must cancel the command group instead.

clearRequirements() → None
Clears list of subsystem requirements.

This is only used by ConditionalCommand so cancelling the chosen command works properly in CommandGroup.

doesRequire(*subsystem: frc::Subsystem*) → bool
Checks if the command requires the given Subsystem.

Parameters **system** – the system

Returns whether or not the subsystem is required (false if given nullptr)

end() → None
Called when the command ended peacefully.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

execute() → None
The execute method is called repeatedly until this Command either finishes or is canceled.

getGroup() → frc::CommandGroup
Returns the CommandGroup that this command is a part of.

Will return null if this Command is not in a group.

Returns The CommandGroup that this command is a part of (or null if not in group)

getID() → int
Get the ID (sequence number) for this command.

The ID is a unique sequence number that is incremented for each command.

Returns The ID of this command

getName() → str
Gets the name of this Command.

Returns Name

getRequirements() → wpi::SmallPtrSetImpl<frc::Subsystem*>
Returns the requirements (as an std::set of Subsystem pointers) of this command.

Returns The requirements (as an std::set of Subsystem pointers) of this command

getSubsystem() → str
Gets the subsystem name of this Command.

Returns Subsystem name

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

initialize() → None
The initialize method is called the first time this Command is run after being started.

interrupted() → None
Called when the command ends because somebody called Cancel() or another command shared the same requirements as this one, and booted it out.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Generally, it is useful to simply call the End() method within this method, as done here.

isCanceled() → bool
Returns whether or not this has been canceled.

Returns whether or not this has been canceled

isCompleted() → bool
Returns whether or not the command has completed running.

Returns whether or not the command has completed running.

isFinished() → bool
Returns whether this command is finished.

If it is, then the command will be removed and End() will be called.

It may be useful for a team to reference the IsTimedOut() method for time-sensitive commands.

Returning false will result in the command never ending automatically. It may still be cancelled manually or interrupted by another command. Returning true will result in the command executing once and finishing immediately. We recommend using InstantCommand for this.

Returns Whether this command is finished. @see IsTimedOut()

isInitialized() → bool
Returns whether or not the command has been initialized.

Returns whether or not the command has been initialized.

isInterruptible() → bool
Returns whether or not this command can be interrupted.

Returns whether or not this command can be interrupted

isParented() → bool
Returns whether the command has a parent.

Parameters **True** – if the command has a parent.

isRunning() → bool
Returns whether or not the command is running.

This may return true even if the command has just been canceled, as it may not have yet called Interrupted().

Returns whether or not the command is running

isTimedOut() → bool
Returns whether or not the TimeSinceInitialized() method returns a number which is greater than or equal to the timeout for the command.

If there is no timeout, this will always return false.

Returns whether the time has expired

requires(*s: frc::Subsystem*) → None
This method specifies that the given Subsystem is used by this command.

This method is crucial to the functioning of the Command System in general.

Note that the recommended way to call this method is in the constructor.

Parameters **subsystem** – The Subsystem required @see Subsystem

run() → bool
The run method is used internally to actually run the commands.

Returns Whether or not the command should stay within the Scheduler.

setInterruptible(*interruptible: bool*) → None
Sets whether or not this command can be interrupted.

Parameters **interruptible** – whether or not this command can be interrupted

setName(*name: str*) → None
Sets the name of this Command.

Parameters **name** – name

setParent(*parent: frc::CommandGroup*) → None
Sets the parent of this command. No actual change is made to the group.

Parameters **parent** – the parent

setRunWhenDisabled(*run: bool*) → None
Sets whether or not this Command should run when the robot is disabled.

By default a command will not run when the robot is disabled, and will in fact be canceled.

Parameters **run** – Whether this command should run when the robot is disabled.

setSubsystem(*subsystem: str*) → None
Sets the subsystem name of this Command.

Parameters **subsystem** – subsystem name

setTimeout(*timeout: float*) → None
Sets the timeout of this command.

Parameters **timeout** – the timeout (in seconds) @see IsTimedOut()

start() → None
Starts up the command. Gets the command ready to start.

Note that the command will eventually start, however it will not necessarily do so immediately, and may in fact be canceled before initialize is even called.

timeSinceInitialized() → float
Returns the time since this command was initialized (in seconds).

This function will work even if there is no specified timeout.

Returns the time since this command was initialized (in seconds).

willRunWhenDisabled() → bool
Returns whether or not this Command will run when the robot is disabled, or if it will cancel itself.

Returns Whether this Command will run when the robot is disabled, or if it will cancel itself.

1.2.5 CommandGroup

class wpilib.command.**CommandGroup**(**args*, ***kwargs*)
Bases: *wpilib.command.Command*

A CommandGroup is a list of commands which are executed in sequence.

Commands in a CommandGroup are added using the AddSequential() method and are called sequentially. CommandGroups are themselves Commands and can be given to other CommandGroups.

CommandGroups will carry all of the requirements of their Command subcommands. Additional requirements can be specified by calling Requires() normally in the constructor.

CommandGroups can also execute commands in parallel, simply by adding them using AddParallel().

@see Command @see Subsystem

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.CommandGroup) -> None
2. __init__(self: wpilib.command._commands_v1.CommandGroup, name: str) -> None

Creates a new CommandGroup with the given name.

Parameters **name** – The name for this command group

addParallel(**args*, ***kwargs*)
Overloaded function.

1. addParallel(self: wpilib.command._commands_v1.CommandGroup, command: wpilib.command._commands_v1.Command) -> None

Adds a new child Command to the group. The Command will be started after all the previously added Commands.

Instead of waiting for the child to finish, a CommandGroup will have it run at the same time as the subsequent Commands. The child will run until either it finishes, a new child with conflicting requirements is started, or the main sequence runs a Command with conflicting requirements. In the latter two cases, the child will be canceled even if it says it can't be interrupted.

Note that any requirements the given Command has will be added to the group. For this reason, a Command's requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

Parameters **command** – The command to be added

2. addParallel(self: wpilib.command._commands_v1.CommandGroup, command: wpilib.command._commands_v1.Command, timeout: float) -> None

Adds a new child Command to the group with the given timeout. The Command will be started after all the previously added Commands.

Once the Command is started, it will run until it finishes, is interrupted, or the time expires, whichever is sooner. Note that the given Command will have no knowledge that it is on a timer.

Instead of waiting for the child to finish, a CommandGroup will have it run at the same time as the subsequent Commands. The child will run until either it finishes, the timeout expires, a new child with conflicting requirements is started, or the main sequence runs a Command with conflicting requirements. In the latter two cases, the child will be canceled even if it says it can't be interrupted.

Note that any requirements the given Command has will be added to the group. For this reason, a Command's requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

Parameters

- **command** – The command to be added
- **timeout** – The timeout (in seconds)

addSequential(**args*, ***kwargs*)
Overloaded function.

1. addSequential(self: wpilib.command._commands_v1.CommandGroup, command: wpilib.command._commands_v1.Command) -> None

Adds a new Command to the group. The Command will be started after all the previously added Commands.

Note that any requirements the given Command has will be added to the group. For this reason, a Command's requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

Parameters **command** – The Command to be added

2. addSequential(self: wpilib.command._commands_v1.CommandGroup, command: wpilib.command._commands_v1.Command, timeout: float) -> None

Adds a new Command to the group with a given timeout. The Command will be started after all the previously added commands.

Once the Command is started, it will be run until it finishes or the time expires, whichever is sooner. Note that the given Command will have no knowledge that it is on a timer.

Note that any requirements the given Command has will be added to the group. For this reason, a Command's requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

Parameters

- **command** – The Command to be added
- **timeout** – The timeout (in seconds)

end() → None
Can be overridden by teams.

execute() → None
Can be overridden by teams.

getSize() → int

initialize() → None
Can be overridden by teams.

interrupted() → None
Can be overridden by teams.

isFinished() → bool
Can be overridden by teams.

isInterruptible() → bool

1.2.6 CommandGroupEntry

class wpilib.command.**CommandGroupEntry**(**args*, ***kwargs*)
Bases: pybind11_builtins.pybind11_object

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.CommandGroupEntry) -> None
2. __init__(self: wpilib.command._commands_v1.CommandGroupEntry, command: wpilib.command._commands_v1.Command, state: wpilib.command._commands_v1.CommandGroupEntry.Sequence, timeout: float = -1.0) -> None

class Sequence(*arg0: int*) → None
Bases: pybind11_builtins.pybind11_object

Members:

kSequence_InSequence

kSequence_BranchPeer

kSequence_BranchChild

kSequence_BranchChild = Sequence.kSequence_BranchChild

kSequence_BranchPeer = Sequence.kSequence_BranchPeer

kSequence_InSequence = Sequence.kSequence_InSequence

name
(self: handle) -> str

isTimedOut() → bool

m_command

m_state

m_timeout

1.2.7 ConditionalCommand

class wpilib.command.**ConditionalCommand**(**args*, ***kwargs*)
Bases: *wpilib.command.Command*

A ConditionalCommand is a Command that starts one of two commands.

A ConditionalCommand uses the Condition method to determine whether it should run onTrue or onFalse.

A ConditionalCommand adds the proper Command to the Scheduler during Initialize() and then IsFinished() will return true once that Command has finished executing.

If no Command is specified for onFalse, the occurrence of that condition will be a no-op.

A ConditionalCommand will require the superset of subsystems of the onTrue and onFalse commands.

@see Command @see Scheduler

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.ConditionalCommand, onTrue: wpilib.command._commands_v1.Command, onFalse: wpilib.command._commands_v1.Command = None) -> None

Creates a new ConditionalCommand with given onTrue and onFalse Commands.

Parameters

- **onTrue** – The Command to execute if Condition() returns true
- **onFalse** – The Command to execute if Condition() returns false

2. __init__(self: wpilib.command._commands_v1.ConditionalCommand, name: str, onTrue: wpilib.command._commands_v1.Command, onFalse: wpilib.command._commands_v1.Command = None) -> None

Creates a new ConditionalCommand with given onTrue and onFalse Commands.

Parameters

- **name** – The name for this command group
- **onTrue** – The Command to execute if Condition() returns true
- **onFalse** – The Command to execute if Condition() returns false

isFinished() → bool

1.2.8 HeldButtonScheduler

class wpilib.command.**HeldButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.2.9 InstantCommand

class wpilib.command.**InstantCommand**(**args*, ***kwargs*)

Bases: *wpilib.command.Command*

This command will execute once, then finish immediately afterward.

Subclassing InstantCommand is shorthand for returning true from IsFinished().

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.InstantCommand, name: str) -> None

Creates a new InstantCommand with the given name.

Parameters **name** – The name for this command

2. __init__(self: wpilib.command._commands_v1.InstantCommand, subsystem: wpilib.command._commands_v1.Subsystem) -> None

Creates a new InstantCommand with the given requirement.

Parameters **subsystem** – The subsystem that the command requires

3. __init__(self: wpilib.command._commands_v1.InstantCommand, name: str, subsystem: wpilib.command._commands_v1.Subsystem) -> None

Creates a new InstantCommand with the given name.

Parameters

- **name** – The name for this command
- **subsystem** – The subsystem that the command requires

4. __init__(self: wpilib.command._commands_v1.InstantCommand, func: Callable[[], None]) -> None

Create a command that calls the given function when run.

Parameters **func** – The function to run when Initialize() is run.

5. __init__(self: wpilib.command._commands_v1.InstantCommand, subsystem: wpilib.command._commands_v1.Subsystem, func: Callable[[], None]) -> None

Create a command that calls the given function when run.

Parameters

- **subsystem** – The subsystems that this command runs on.
- **func** – The function to run when Initialize() is run.

6. __init__(self: wpilib.command._commands_v1.InstantCommand, name: str, func: Callable[[], None]) -> None

Create a command that calls the given function when run.

Parameters

- **name** – The name of the command.
- **func** – The function to run when Initialize() is run.

7. __init__(self: wpilib.command._commands_v1.InstantCommand, name: str, subsystem: wpilib.command._commands_v1.Subsystem, func: Callable[[], None]) -> None

Create a command that calls the given function when run.

Parameters

- **name** – The name of the command.
- **subsystem** – The subsystems that this command runs on.
- **func** – The function to run when Initialize() is run.

8. __init__(self: wpilib.command._commands_v1.InstantCommand) -> None
9. __init__(self: wpilib.command._commands_v1.InstantCommand) -> None

isFinished() → bool

1.2.10 InternalButton

class wpilib.command.**InternalButton**(**args*, ***kwargs*)

Bases: *wpilib.command.Button*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.InternalButton) -> None
2. __init__(self: wpilib.command._commands_v1.InternalButton, inverted: bool) -> None

get() → bool

setInverted(*inverted: bool*) → None

setPressed(*pressed: bool*) → None

1.2.11 JoystickButton

class wpilib.command.**JoystickButton**(*joystick: wpilib.interfaces._interfaces.GenericHID*, *buttonNumber: int*) → None

Bases: *wpilib.command.Button*

get() → bool

1.2.12 NetworkButton

class wpilib.command.**NetworkButton**(**args*, ***kwargs*)

Bases: *wpilib.command.Button*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.NetworkButton, tableName: str, field: str) -> None
2. __init__(self: wpilib.command._commands_v1.NetworkButton, table: _pyntcore._ntcore.NetworkTable, field: str) -> None

get() → bool

1.2.13 PIDCommand

class wpilib.command.**PIDCommand**(**args*, ***kwargs*)

Bases: *wpilib.command.Command*, wpilib.interfaces._interfaces.PIDOutput, wpilib.interfaces._interfaces.PIDSource

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float) -> None
2. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float, period: float) -> None
3. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float, f: float, period: float) -> None
4. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float) -> None
5. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float, period: float) -> None

6. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float, f: float, period: float) -> None
7. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None
8. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float, period: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None
9. __init__(self: wpilib.command._commands_v1.PIDCommand, name: str, p: float, i: float, d: float, f: float, period: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None
10. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None
11. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float, period: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None
12. __init__(self: wpilib.command._commands_v1.PIDCommand, p: float, i: float, d: float, f: float, period: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None

PIDGet() → float

PIDWrite(*output: float*) → None

getPIDController() → frc::PIDController

getPosition() → float

getSetpoint() → float

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

returnPIDInput() → float

setSetpoint(*setpoint: float*) → None

setSetpointRelative(*deltaSetpoint: float*) → None

usePIDOutput(*output: float*) → None

1.2.14 PIDSubsystem

class wpilib.command.**PIDSubsystem**(**args*, ***kwargs*)

Bases: *wpilib.command.Subsystem*, wpilib.interfaces._interfaces.PIDOutput, wpilib.interfaces._interfaces.PIDSource

This class is designed to handle the case where there is a Subsystem which uses a single PIDController almost constantly (for instance, an elevator which attempts to stay at a constant height).

It provides some convenience methods to run an internal PIDController. It also allows access to the internal PIDController in order to give total control to the programmer.

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.PIDSubsystem, name: str, p: float, i: float, d: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, and D values.

Parameters

- **name** – the name
- **p** – the proportional value

- **i** – the integral value
- **d** – the derivative value

2. __init__(self: wpilib.command._commands_v1.PIDSubsystem, name: str, p: float, i: float, d: float, f: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, D, and F values.

Parameters

- **name** – the name
- **p** – the proportional value
- **i** – the integral value
- **d** – the derivative value
- **f** – the feedforward value

3. __init__(self: wpilib.command._commands_v1.PIDSubsystem, name: str, p: float, i: float, d: float, f: float, period: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, D, and F values.

It will also space the time between PID loop calculations to be equal to the given period.

Parameters

- **name** – the name
- **p** – the proportional value
- **i** – the integral value
- **d** – the derivative value
- **f** – the feedfoward value
- **period** – the time (in seconds) between calculations

4. __init__(self: wpilib.command._commands_v1.PIDSubsystem, p: float, i: float, d: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, and D values.

It will use the class name as its name.

Parameters

- **p** – the proportional value
- **i** – the integral value
- **d** – the derivative value

5. __init__(self: wpilib.command._commands_v1.PIDSubsystem, p: float, i: float, d: float, f: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, D, and F values.

It will use the class name as its name.

Parameters

- **p** – the proportional value

- **i** – the integral value
- **d** – the derivative value
- **f** – the feedforward value

6. __init__(self: wpilib.command._commands_v1.PIDSubsystem, p: float, i: float, d: float, f: float, period: float) -> None

Instantiates a PIDSubsystem that will use the given P, I, D, and F values.

It will use the class name as its name. It will also space the time between PID loop calculations to be equal to the given period.

Parameters

- **p** – the proportional value
- **i** – the integral value
- **d** – the derivative value
- **f** – the feedforward value
- **period** – the time (in seconds) between calculations

PIDGet() → float

PIDWrite(*output: float*) → None

disable() → None
Disables the internal PIDController.

enable() → None
Enables the internal PIDController.

getPIDController() → frc::PIDController
Returns the PIDController used by this PIDSubsystem.

Use this if you would like to fine tune the PID loop.

Returns The PIDController used by this PIDSubsystem

getPosition() → float
Returns the current position.

Returns the current position

getRate() → float
Returns the current rate.

Returns the current rate

getSetpoint() → float
Return the current setpoint.

Returns The current setpoint

onTarget() → bool
Return true if the error is within the percentage of the total input range, determined by SetTolerance().

This asssumes that the maximum and minimum input were set using SetInput(). Use OnTarget() in the IsFinished() method of commands that use this subsystem.

Currently this just reports on target as the actual value passes through the setpoint. Ideally it should be based on being within the tolerance for some period of time.

Returns True if the error is within the percentage tolerance of the input range

returnPIDInput() → float

setAbsoluteTolerance(*absValue: float*) → None
Set the absolute error which is considered tolerable for use with OnTarget.

Parameters **absValue** – absolute error which is tolerable

setInputRange(*minimumInput: float*, *maximumInput: float*) → None
Sets the maximum and minimum values expected from the input.

Parameters

- **minimumInput** – the minimum value expected from the input
- **maximumInput** – the maximum value expected from the output

setOutputRange(*minimumOutput: float*, *maximumOutput: float*) → None
Sets the maximum and minimum values to write.

Parameters

- **minimumOutput** – the minimum value to write to the output
- **maximumOutput** – the maximum value to write to the output

setPercentTolerance(*percent: float*) → None
Set the percentage error which is considered tolerable for use with OnTarget().

Parameters **percent** – percentage error which is tolerable

setSetpoint(*setpoint: float*) → None
Sets the setpoint to the given value.

If SetRange() was called, then the given setpoint will be trimmed to fit within the range.

Parameters **setpoint** – the new setpoint

setSetpointRelative(*deltaSetpoint: float*) → None
Adds the given value to the setpoint.

If SetRange() was used, then the bounds will still be honored by this method.

Parameters **deltaSetpoint** – the change in the setpoint

usePIDOutput(*output: float*) → None

1.2.15 POVButton

class wpilib.command.**POVButton**(*joystick: wpilib.interfaces._interfaces.GenericHID*, *angle: int*, *povNumber: int = 0*) → None
Bases: *wpilib.command.Button*

Creates a POV button for triggering commands.

Parameters

- **joystick** – The GenericHID object that has the POV
- **angle** – The desired angle in degrees (e.g. 90, 270)
- **povNumber** – The POV number (@see GenericHID#GetPOV)

get() → bool

1.2.16 PressedButtonScheduler

class wpilib.command.**PressedButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.2.17 PrintCommand

class wpilib.command.**PrintCommand**(*message: str*) → None

Bases: *wpilib.command.InstantCommand*

initialize() → None

1.2.18 ReleasedButtonScheduler

class wpilib.command.**ReleasedButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.2.19 Scheduler

class wpilib.command.**Scheduler**

Bases: pybind11_builtins.pybind11_object

addButton(*button: wpilib.command._commands_v1.ButtonScheduler*) → None

addCommand(*command: wpilib.command._commands_v1.Command*) → None

Add a command to be scheduled later.

In any pass through the scheduler, all commands are added to the additions list, then at the end of the pass, they are all scheduled.

Parameters **command** – The command to be scheduled

static addToSmartDashboard(*key: str*) → None

This is equivalent to `wpilib.SmartDashboard.putData(key, Scheduler.getInstance())`. Use this instead, as SmartDashboard.putData will fail if used directly

Parameters **key** – the key

static getInstance() → wpilib.command._commands_v1.Scheduler

Returns the Scheduler, creating it if one does not exist.

Returns the Scheduler

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

registerSubsystem(*subsystem: frc::Subsystem*) → None

Registers a Subsystem to this Scheduler, so that the Scheduler might know if a default Command needs to be run.

All Subsystems should call this.

Parameters **system** – the system

remove(*command: wpilib.command._commands_v1.Command*) → None
Removes the Command from the Scheduler.

Parameters **command** – the command to remove

removeAll() → None

resetAll() → None
Completely resets the scheduler. Undefined behavior if running.

run() → None
Runs a single iteration of the loop.

This method should be called often in order to have a functioning Command system. The loop has five stages:

 - Poll the Buttons - Execute/Remove the Commands - Send values to SmartDashboard - Add Commands - Add Defaults

setEnabled(*enabled: bool*) → None

1.2.20 StartCommand

class wpilib.command.**StartCommand**(*commandToStart: wpilib.command._commands_v1.Command*) → None
Bases: *wpilib.command.InstantCommand*

initialize() → None

1.2.21 Subsystem

class wpilib.command.**Subsystem**(*name: str*) → None
Bases: wpilib._wpilib.ErrorBase, wpilib._wpilib.Sendable

Creates a subsystem with the given name.

Parameters **name** – the name of the subsystem

addChild(**args*, ***kwargs*)
Overloaded function.

1. addChild(self: wpilib.command._commands_v1.Subsystem, name: str, child: wpilib._wpilib.Sendable) -> None

Associate a Sendable with this Subsystem. Also update the child's name.

Parameters

- **name** – name to give child
- **child** – sendable

2. addChild(self: wpilib.command._commands_v1.Subsystem, child: wpilib._wpilib.Sendable) -> None

Associate a {@link Sendable} with this Subsystem.

Parameters **child** – sendable

getCurrentCommand() → wpilib.command._commands_v1.Command
Returns the command which currently claims this subsystem.

Returns the command which currently claims this subsystem

getCurrentCommandName() → str
Returns the current command name, or empty string if no current command.

Returns the current command name

getDefaultCommand() → wpilib.command._commands_v1.Command
Returns the default command (or null if there is none).

Returns the default command

getDefaultCommandName() → str
Returns the default command name, or empty string is there is none.

Returns the default command name

getName() → str
Gets the name of this Subsystem.

Returns Name

getSubsystem() → str
Gets the subsystem name of this Subsystem.

Returns Subsystem name

initDefaultCommand() → None
Initialize the default command for this subsystem.

This is meant to be the place to call SetDefaultCommand in a subsystem and will be called on all the subsystems by the CommandBase method before the program starts running by using the list of all registered Subsystems inside the Scheduler.

This should be overridden by a Subsystem that has a default Command

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

periodic() → None
When the run method of the scheduler is called this method will be called.

setCurrentCommand(*command: wpilib.command._commands_v1.Command*) → None
Sets the current command.

Parameters **command** – the new current command

setDefaultCommand(*command: wpilib.command._commands_v1.Command*) → None
Sets the default command. If this is not called or is called with null, then there will be no default command for the subsystem.

WARNING: This should **NOT** be called in a constructor if the subsystem is a singleton.

Parameters **command** – the default command (or null if there should be none)

setName(*name: str*) → None
Sets the name of this Subsystem.

Parameters **name** – name

setSubsystem(*subsystem: str*) → None
Sets the subsystem name of this Subsystem.

Parameters **subsystem** – subsystem name

1.2.22 TimedCommand

class wpilib.command.**TimedCommand**(**args*, ***kwargs*)

Bases: *wpilib.command.Command*

A TimedCommand will wait for a timeout before finishing.

TimedCommand is used to execute a command for a given amount of time.

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.TimedCommand, name: str, timeout: float) -> None

Creates a new TimedCommand with the given name and timeout.

Parameters

- **name** – the name of the command
- **timeout** – the time (in seconds) before this command "times out"

2. __init__(self: wpilib.command._commands_v1.TimedCommand, timeout: float) -> None

Creates a new WaitCommand with the given timeout.

Parameters **timeout** – the time (in seconds) before this command "times out"

3. __init__(self: wpilib.command._commands_v1.TimedCommand, name: str, timeout: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None

Creates a new TimedCommand with the given name and timeout.

Parameters

- **name** – the name of the command
- **timeout** – the time (in seconds) before this command "times out"
- **subsystem** – the subsystem that the command requires

4. __init__(self: wpilib.command._commands_v1.TimedCommand, timeout: float, subsystem: wpilib.command._commands_v1.Subsystem) -> None

Creates a new WaitCommand with the given timeout.

Parameters

- **timeout** – the time (in seconds) before this command "times out"
- **subsystem** – the subsystem that the command requires

isFinished() → bool

Ends command when timed out.

1.2.23 ToggleButtonScheduler

class wpilib.command.**ToggleButtonScheduler**(*last: bool*, *button: wpilib.command._commands_v1.Trigger*, *orders: wpilib.command._commands_v1.Command*) → None

Bases: *wpilib.command.ButtonScheduler*

execute() → None

1.2.24 Trigger

class wpilib.command.**Trigger**() → None
Bases: wpilib._wpilib.Sendable

This class provides an easy way to link commands to inputs.

It is very easy to link a polled input to a command. For instance, you could link the trigger button of a joystick to a "score" command or an encoder reaching a particular value.

It is encouraged that teams write a subclass of Trigger if they want to have something unusual (for instance, if they want to react to the user holding a button while the robot is reading a certain sensor input). For this, they only have to write the Trigger.Get() method to get the full functionality of the Trigger class.

cancelWhenActive(*command: wpilib.command._commands_v1.Command*) → None

get() → bool

grab() → bool

initSendable(*builder: wpilib._wpilib.SendableBuilder*) → None

toggleWhenActive(*command: wpilib.command._commands_v1.Command*) → None

whenActive(*command: wpilib.command._commands_v1.Command*) → None

whenInactive(*command: wpilib.command._commands_v1.Command*) → None

whileActive(*command: wpilib.command._commands_v1.Command*) → None

1.2.25 WaitCommand

class wpilib.command.**WaitCommand**(**args*, ***kwargs*)
Bases: *wpilib.command.TimedCommand*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.WaitCommand, timeout: float) -> None

Creates a new WaitCommand with the given name and timeout.

Parameters

- **name** – the name of the command
- **timeout** – the time (in seconds) before this command "times out"

2. __init__(self: wpilib.command._commands_v1.WaitCommand, name: str, timeout: float) -> None

Creates a new WaitCommand with the given timeout.

Parameters **timeout** – the time (in seconds) before this command "times out"

1.2.26 WaitForChildren

class wpilib.command.**WaitForChildren**(**args*, ***kwargs*)
Bases: *wpilib.command.Command*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.WaitForChildren, timeout: float) -> None
2. __init__(self: wpilib.command._commands_v1.WaitForChildren, name: str, timeout: float) -> None

isFinished() → bool

1.2.27 WaitUntilCommand

class wpilib.command.**WaitUntilCommand**(**args*, ***kwargs*)

Bases: *wpilib.command.Command*

Overloaded function.

1. __init__(self: wpilib.command._commands_v1.WaitUntilCommand, time: float) -> None

A WaitCommand will wait until a certain match time before finishing.

This will wait until the game clock reaches some value, then continue to the next command.

@see CommandGroup

2. __init__(self: wpilib.command._commands_v1.WaitUntilCommand, name: str, time: float) -> None

isFinished() → bool

Check if we've reached the actual finish time.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

G

H

I

T

U

W