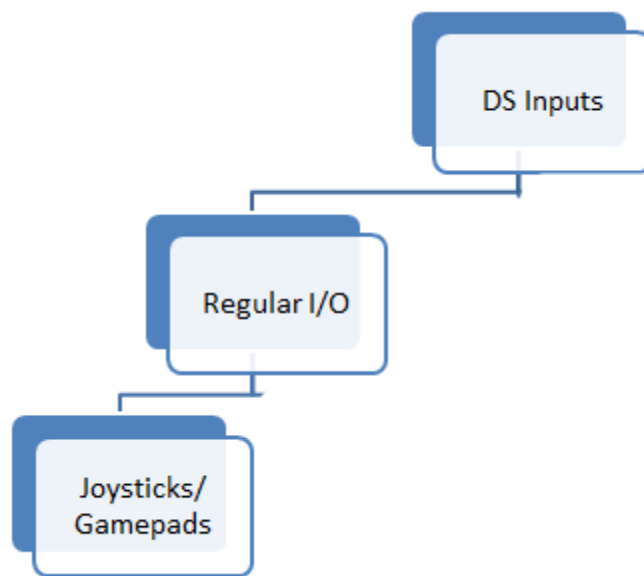


# Driver Station Input Overview

The FRC Driver Station software serves as the interface between the human operators and the robot. The software takes input from a number of sources and forwards it to the robot where the robot code can act on it to control mechanisms.

## Input types



The chart above shows the different types of inputs that may be transmitted by the DS software. The most common input is an HID compliant joystick or gamepad such as the Logitech Attack 3 or Logitech Extreme 3D Pro joysticks which have been provided in the Kit of Parts since 2009. Note that a number of devices are now available which allow custom IO to be exposed as a standard USB HID device such as the [The TI Launchpad](#) and [16 Hertz Leonardo++](#) included in your Kit of Parts.

## Driver Station Class

C++

```
DriverStation& ds = DriverStation::GetInstance();  
ds.SomeMethod();
```

```
DriverStation::GetInstance().SomeMethod();
```

# Driver Station Input Overview

## Java

```
DriverStation ds = DriverStation.getInstance();  
ds.someMethod();
```

```
DriverStation.getInstance().someMethod();
```

The Driver Station class has methods to access information such as the robot mode, battery voltage, alliance color and team number. Note that while the Driver Station class has methods for accessing the joystick data, there is another class "Joystick" that provides a much more user friendly interface to this data. The DriverStation class is constructed as a singleton by the base class. To get access to the methods of the DriverStation object constructed by the base class, call `DriverStation.getInstance()` and either store the result in a DriverStation object (if using a lot) or call the method on the instance directly.

## Robot Mode

### C++

```
bool exampleBool;  
exampleBool = IsDisabled();  
exampleBool = IsEnabled();  
exampleBool = IsAutonomous();  
exampleBool = IsOperatorControl();  
exampleBool = IsTest();
```

```
while(IsOperatorControl() && IsEnabled())  
{  
}
```

```
exampleBool = DriverStation::GetInstance()->IsDisabled();
```

### Java

```
boolean exampleBool;  
exampleBool = isDisabled();  
exampleBool = isEnabled();
```

```
exampleBool = isAutonomous();  
exampleBool = isOperatorControl();  
exampleBool = isTest();
```

```
while(isOperatorControl() && isEnabled())  
{  
}
```

# Driver Station Input Overview

```
exampleBool = DriverStation.getInstance().isDisabled();
```

The Driver Station class provides a number of methods for checking the current mode of the robot, these methods are most commonly used to control program flow when using the [SampleRobot base class](#). There are two separate pieces of information that define the current mode, the enable state (enabled/disabled) and the control state (autonomous, operator control, test). This means that exactly one method from the first group and one method from the second group should always return true. For example, if the Driver Station is currently set to Test mode and the robot is disabled the methods `isDisabled()` and `isTest()` would both return true. While the implementation of these methods is in the DriverStation class, the RobotBase class (which the templates extend from) provides proxies to these methods so they may be used without the class specification (as shown in the first 3 example groups above). To call these methods from another class, use the DriverStation instance as shown in the final example.

## DS Attached, FMS Attached and System status

C++

```
bool exampleBool;  
exampleBool = DriverStation::GetInstance().IsDSAttached();  
exampleBool = DriverStation::GetInstance().IsFMSAttached();  
exampleBool = DriverStation::GetInstance().IsSysActive();  
exampleBool = DriverStation::GetInstance().IsSysBrownedOut();
```

Java

```
boolean exampleBool;  
exampleBool = DriverStation.getInstance().isDSAttached();  
exampleBool = DriverStation.getInstance().isFMSAttached();  
exampleBool = DriverStation.getInstance().isSysActive();  
exampleBool = DriverStation.getInstance().isSysBrownedOut();
```

The DriverStation class also has methods for determining if the DS is connected to the robot, if the DS is connected to FMS querying if the FPGA outputs are enabled (`IsSysActive`), and querying if the roboRIO is in brownout protection. The FPGA outputs may be disabled for a variety of reasons including: DS is commanding Disabled or E-Stop, the system watchdog has timed out (generally because the DS is not communicating with the roboRIO), or the roboRIO is in brownout protection.

## Battery Voltage

C++

```
double voltage = DriverStation::GetInstance().GetBatteryVoltage();
```

# Driver Station Input Overview

## Java

```
double voltage = DriverStation.getInstance().getBatteryVoltage();
```

For compatibility purposes the battery voltage can be retrieved using the DriverStation class (it is now also available from the ControllerPower class as the roboRIO input voltage). This information can be queried from the DriverStation class in order to perform voltage compensation or actively manage robot power draw by detecting battery voltage dips and shutting off or limiting non-critical mechanisms,

## Alliance

### C++

```
DriverStation::Alliance color;  
color = DriverStation::GetInstance().GetAlliance();  
if(color == DriverStation::Alliance::kBlue){  
}
```

### Java

```
DriverStation.Alliance color;  
color = DriverStation.getInstance().getAlliance();  
if(color == DriverStation.Alliance.kBlue){  
}
```

The DriverStation class can provide information on what alliance color the robot is. When connected to FMS this is the alliance color communicated to the DS by the field. When not connected, the alliance color is determined by the Team Station dropdown box on the Operation tab of the DS software.

## Location

### C++

```
int station;  
station = DriverStation::GetInstance().GetLocation();
```

### Java

```
int station;  
station = DriverStation.getInstance().getLocation();
```

The getLocation() method of the Driver Station returns an integer indicating which station number the Driver Station is in (1-3). Note that the station that the DS and controls are located in is not typically related to the starting position of the robot so this information may be of limited use.

# Driver Station Input Overview

When not connected to the FMS software this state is determined by the Team Station dropdown on the DS Operation tab.

## Match Time

C++

```
double time;  
time = DriverStation::GetInstance().GetMatchTime();
```

Java

```
double time;  
time = DriverStation.getInstance().getMatchTime();
```

This method returns the approximate time remaining in the current period (auto, teleop, etc.) in seconds. Note that this time is derived from the FMS, however due to various latencies involved it is **not an official timer**. The Driver Station's Practice Match functionality will approximate the behavior of this method when connected to FMS. Running the DS directly in Autonomous or Teleop mode will behave differently with respect to this method.