

C++ conventions for objects, methods, and variables

Every sensor, actuator and operator interface component is an object in either C++ or Java programs. To use one of these you must create an instance of it using the class name. Any references to the objects such as reading values, setting values, or setting parameters is done through the reference. There are a number of utility objects in WPILib such as the RobotDrive and Compressor that don't represent a single sensor or actuator, but a larger subsystem.

Another convention used throughout the library is the case of the method names. In C++ all methods start with an upper case letter, then are camel case (intermediate words capitalized). In Java all methods start with lower case letters then camel case for the remainder of the name.

Creating objects that are connected to the roboRIO in C++

C++

```
Gyro *headingGyro = new AnalogGyro(1); // Creates a Gyro object connected to channel 1
float heading = headingGyro->GetAngle(); // Gets the current heading from the Gyro and
stores it in the variable heading
```

Generally all the objects in WPILib that connect to roboRIO have one argument in the constructor when created where you specify the channel or port number it is connected to. The above example illustrate the conventions used in WPILib for both C++ and Java.

Creating operator interface objects

C++

```
Joystick *stick = new Joystick(1); // Creates a Joystick object connected to USB on port 1
of the driverstation

double speed = stick->GetX(); // Gets the current X axis value of the joystick and stores
```

C++ conventions for objects, methods, and variables

```
it in the variable "speed"
```

Generally objects connected to the Driver station PC via USB take a single argument indicating the USB port they are connected to. A single Joystick class is provided which should provide the functionality needed to interface with any joystick or gamepad which works with the FRC Driver Station.

Class, method, and variable naming

Type of name	Naming rules	Examples
Class name	Initial upper case letter then camel case (mixed upper/lower case) except acronyms which are all upper case	Victor, SimpleRobot, PWM
Method name	Initial upper case letter then camel case	StartCompetition, Autonomous, GetAngle
Member variable	"m_" followed by the member variable name starting with a lower case letter then camel case	m_deleteSpeedControllers, m_sensitivity
Local variable	Initial lower case	targetAngle
Macro	All upper case with _ between words. Note: It's better to use <u>const</u> values and inline functions than macros.	DISALLOW_COPY_AND_ASSIGN

Names in WPILib follow the conventions shown in the table above. It makes it very easy to determine what the scope and use of a variable is just by looking at the name and is a convention that is used throughout WPILib.

C++ conventions for objects, methods, and variables

MXP IO Numbering

Pinout Designations				
C++/Java				C++/Java
DIO 25	DIO 15 / I2C SDA	34	33	+3.3V
DIO 24	DIO 14 / I2C SCL	32	31	DIO 10 / PWM6
	DGND	30	29	DIO 9 / PWM5
	DGND	28	27	DIO 8 / PWM4
DIO23/ PWM19	DIO 13 / PWM9	26	25	DIO 7 / SPI MOSI
	DGND	24	23	DIO 6 / SPI MISO
DIO22/ PWM18	DIO 12 / PWM8	22	21	DIO 5 / SPI CLK
	DGND	20	19	DIO 4 / SPI CS
DIO21/ PWM17	DIO 11 / PWM7	18	17	DIO 3 / PWM3
	DGND	16	15	DIO 2 / PWM2
	UART.TX	14	13	DIO 1 / PWM1
	DGND	12	11	DIO 0 / PWM0
	UART.RX	10	9	AI3
	DGND	8	7	AI2
	AGND	6	5	AI1
AO1	AO1	4	3	AI0
AO0	AO0	2	1	+5V

In C++ and Java the numbering for the MXP IO is a continuation of the numbering from the headers, meaning MXP DIO 0 is DIO 10, MXP DIO 1 is DIO 11 and so on. This applies to DIO, PWM and Analog Input on the MXP. The I2C and SPI buses have enumerations used to indicate which port you are using.