

UDG_Create User Guide

Omar Alejandro Rodríguez Rosas

7 de febrero de 2015

Índice general

1. Introduction	9
2. Setup	11
2.1. Conexiones físicas	11
2.2. Dependencias de software	12
2.3. Recompilación de la biblioteca con g++	12
2.4. Uso de UDG_Create de manera local	15
2.5. Uso de UDG_Create como biblioteca estándar	16
3. La Clase Create	17
3.1. Tipos de datos y enumeraciones	17
3.1.1. enum errorCode	17
3.1.2. enum mode	17
3.1.3. enum baudCode	18
3.1.4. enum chargingstates	18
3.1.5. enum infraredbytechars	18
3.1.6. enum sensorPackets	19
3.1.7. enum VerbosityLevels	20
3.1.8. t_verbosity	20
3.1.9. enum BoolSigned	20
3.1.10. boolSigned	20
3.1.11. NUMBER_OF_SENSORS	20
3.1.12. int16	20
3.2. Miembros (privados)	21
3.2.1. std::string portName	21
3.2.2. int mode	21
3.2.3. bool charging	21
3.2.4. int portDescriptor	21
3.2.5. int baudRate	21
3.2.6. bool bumpRight	21
3.2.7. bool bumpLeft	22
3.2.8. bool wheelDropRight	22
3.2.9. bool wheelDropLeft	22
3.2.10. bool wheelDropCaster	22

3.2.11. bool wall	22
3.2.12. bool cliffLeft	22
3.2.13. bool cliffFrontLeft	22
3.2.14. bool cliffFrontRight	22
3.2.15. bool cliffRight	23
3.2.16. bool virtualWall	23
3.2.17. bool ld0	23
3.2.18. bool ld1	23
3.2.19. bool ld2	23
3.2.20. bool rightWheel	23
3.2.21. bool leftWheel	23
3.2.22. unsigned char infraredbyte	23
3.2.23. bool advancebtn	23
3.2.24. bool playbtn	24
3.2.25. int distance	24
3.2.26. int angle	24
3.2.27. unsigned char chargingstate	24
3.2.28. int voltage	24
3.2.29. int current	24
3.2.30. unsigned char batterytemperature	24
3.2.30.0.1. int batterycharge	24
3.2.31. int batterycapacity	24
3.2.31.0.2. int wallsignal	25
3.2.32. int cliffls	25
3.2.33. int cliffls	25
3.2.34. int clifffrs	25
3.2.35. int cliffrs	25
3.2.36. bool digitalinput0	25
3.2.37. bool digitalinput1	25
3.2.38. bool digitalinput2	25
3.2.39. bool digitalinput3	26
3.2.40. bool baudchangerate	26
3.2.41. int cargoanalogsignal	26
3.2.42. bool homebase	26
3.2.43. bool internalcharger	26
3.2.44. unsigned char oimode	26
3.2.45. unsigned char songnumber	26
3.2.46. bool songplaying	26
3.2.47. unsigned char strempackets	27
3.2.48. int reqvelocity	27
3.2.49. int reqradius	27
3.2.50. int reqrvelocity	27
3.2.51. int reqlvelocity	27
3.2.52. bool streamingState	27
3.2.53. bool externalSensorsEnabled	27
3.2.54. t_verbosity robotVerbosity	27

3.3. Funciones	28
3.3.1. Publicas	28
3.3.1.1. Create()	28
3.3.1.2. Create(std::string _portName, t_verbosity verbosityLevel)	28
3.3.1.3. ~Create()	28
3.3.1.4. std::string getPortName()	28
3.3.1.5. void start()	28
3.3.1.6. void baud(unsigned char baudRate)	28
3.3.1.7. void control()	29
3.3.1.8. void safe()	29
3.3.1.9. void full()	29
3.3.1.10. void spot()	29
3.3.1.11. void cover()	29
3.3.1.12. void coverAndDock()	29
3.3.1.13. void demo(unsigned char demo)	29
3.3.1.14. void drive(int velocity, int radius)	30
3.3.1.15. void driveDirect(int rightVelocity, int leftVelocity)	30
3.3.1.16. void leds(unsigned char bit,unsigned char color, unsigned char intensity)	30
3.3.1.17. void digitalOutputs(unsigned char outputBits)	31
3.3.1.18. void pwmLowSideDrivers(unsigned char dirver1, unsigned char driver1, unsigned char driver1, unsigned char driver0)	31
3.3.1.19. void lowSideDrivers(unsigned char bits)	31
3.3.1.20. void sendIr(unsigned char byteValue)	31
3.3.1.21. void song(unsigned char,unsigned char,...)	32
3.3.1.22. void playSong(unsigned char songNumber)	32
3.3.1.23. int sensors(unsigned char idPacket)	32
3.3.1.24. int getSizePacket(int idPacket)	32
3.3.1.25. void stream(unsigned char* destinationBuffer,void* thread,int n,...)	33
3.3.1.26. void pauseResumeStream(bool streamState)	33
3.3.1.27. void script(unsigned char n,...);	33
3.3.1.28. void playScript()	34
3.3.1.29. void showScript()	34
3.3.1.30. void waitTime(unsigned char time)	34
3.3.1.31. void waitDistance(int distance)	34
3.3.1.32. void waitAngle(int angle)	34
3.3.1.33. void waitEvent(unsigned char event)	35
3.3.1.34. char* charMode(int mode)	35
3.3.1.35. int getBaudCode(int baudCode)	35
3.3.1.36. bool getBumpRight()	35
3.3.1.37. bool getBumpLeft()	35
3.3.1.38. bool getWheelDropRight()	35
3.3.1.39. bool getWheelDropLeft()	36

3.3.1.40. bool getWheelDropCaster()	36
3.3.1.41. bool getWallSeen()	36
3.3.1.42. bool getCliffLeft()	36
3.3.1.43. bool getCliffFrontLeft()	36
3.3.1.44. bool getCliffFrontRight()	36
3.3.1.45. bool getCliffRight()	36
3.3.1.46. bool getVirtualWall()	37
3.3.1.47. bool getLd0()	37
3.3.1.48. bool getLd1()	37
3.3.1.49. bool getLd2()	37
3.3.1.50. bool getRightWheel()	37
3.3.1.51. bool getLeftWheel()	37
3.3.1.52. unsigned char getInfraredByte()	37
3.3.1.53. bool getAdvanceBtn()	38
3.3.1.54. bool getPlayBtn()	38
3.3.1.55. int getDistance()	38
3.3.1.56. int getAngle()	38
3.3.1.57. unsigned char getChargingState()	38
3.3.1.58. int getVoltage()	39
3.3.1.59. int getCurrent()	39
3.3.1.60. unsigned char getBatteryTemperature()	39
3.3.1.61. int getBatteryCharge()	39
3.3.1.62. int getBatteryCapacity()	39
3.3.1.63. int getWallSignal()	39
3.3.1.64. int getCliffLS()	39
3.3.1.65. int getCliffFLS()	40
3.3.1.66. int getCliffFRS()	40
3.3.1.67. int getCliffRS()	40
3.3.1.68. bool getDigitalInput0()	40
3.3.1.69. bool getDigitalInput1()	40
3.3.1.70. bool getDigitalInput2()	40
3.3.1.71. bool getDigitalInput3()	41
3.3.1.72. bool getBaudRateChange()	41
3.3.1.73. int getCargoAnalogSignal()	41
3.3.1.74. bool getHomeBase()	41
3.3.1.75. bool getInternalCharger()	41
3.3.1.76. unsigned char getOIMode()	41
3.3.1.77. unsigned char getSongNumber()	42
3.3.1.78. bool getSongPlaying()	42
3.3.1.79. unsigned char getStreamPackets()	42
3.3.1.80. int getRequestedVelocity()	42
3.3.1.81. int getRequestedRadius()	42
3.3.1.82. int getRequestedRVelocity()	42
3.3.1.83. int getRequestedLVelocity()	42
3.3.1.84. bool getStreamingState()	43

3.3.1.85. void getExternalSensors(int[NUMBER_OF_SENSORS] sensors)	43
3.3.1.86. bool getExternalSensorsEnabledStatus()	43
3.3.1.87. void setVerbosity(t_verbosity)	43
3.3.1.88. int getExternalNthSensor(int n)	43
3.3.1.89. int toLittleEndian(unsigned char* source,int nbytes,int* destination,boolSigned sign)	44
3.3.2. Privadas	44
3.3.2.1. void error(int error,void* info)	44
3.3.2.2. void printRobotMessage(const char* message,...)	44
3.3.2.3. int readBumpsAndWheelDrops(unsigned char *data)	44
3.3.2.4. int readWall(unsigned char *data)	45
3.3.2.5. int readCliffLeft(unsigned char *data)	45
3.3.2.6. int readCliffFrontLeft(unsigned char *data) . . .	45
3.3.2.7. int readCliffFrontRight(unsigned char *data) . .	45
3.3.2.8. int readCliffRight(unsigned char *data)	45
3.3.2.9. int readVirtualWall(unsigned char *data)	46
3.3.2.10. int readLSDriverAndWheelO(unsigned char *data)	46
3.3.2.11. int readInfraredByte(unsigned char *data) . . .	46
3.3.2.12. int readButtons(unsigned char *data)	46
3.3.2.13. int readDistance(unsigned char *data)	47
3.3.2.14. int readAngle(unsigned char *data)	47
3.3.2.15. int readChargingState(unsigned char *data) . .	47
3.3.2.16. int readVoltage(unsigned char *data)	47
3.3.2.17. int readCurrent(unsigned char *data)	47
3.3.2.18. int readBatteryTemperature(unsigned char *data)	48
3.3.2.19. int readBatteryCharge(unsigned char *data) . .	48
3.3.2.20. int readWallSignal(unsigned char *data)	48
3.3.2.21. int readCliffLS(unsigned char *)data	48
3.3.2.22. int readCliffFLS(unsigned char *data)	49
3.3.2.23. int readCliffFRS(unsigned char *data)	49
3.3.2.24. int readCliffRS(unsigned char *data)	49
3.3.2.25. int readDigitalInputs(unsigned char *data) . .	49
3.3.2.26. int readCargoAnalogSignal(unsigned char *data)	49
3.3.2.27. int readChargingSources(unsigned char *data) .	50
3.3.2.28. int readOIMode(unsigned char *data)	50
3.3.2.29. int readSongNumber(unsigned char *data)	50
3.3.2.30. int readSongPlaying(unsigned char *data)	50
3.3.2.31. int readStreamPackets(unsigned char *data) . .	51
3.3.2.32. int readReqVelocity(unsigned char *data)	51
3.3.2.33. int readReqRadius(unsigned char *data)	51
3.3.2.34. int readReqRVelocity(unsigned char *data)	51
3.3.2.35. int readReqLVelocity(unsigned char *data)	51
3.3.2.36. int updateSensor(unsigned char packetID, int size)	52

3.3.2.37. void commonInitializationProcedures(string port,bool auto)	52
3.3.2.38. int16 toInt16(int integer)	52
4. Code Samples	53

Capítulo 1

Introduction

Capítulo 2

Setup

2.1. Conexiones físicas

Para la correcta ejecución de los programas creados con la biblioteca UDG_Create es necesario realizar algunas conexiones físicas entre los distintos componentes de hardware.

1. Conecte el cable DB-25 macho de la placa fenólica al puerto DB-25 hembra en la bahía de carga del Create® como se muestra en la figura 2.1.
2. Conecte un cable USB al puerto correspondiente de la placa fenólica como se muestra en la figura 2.2.
3. Conecte un cable serial al conector DB-9 de la placa fenólica como se ilustra en la figura 2.3. Es posible utilizar un convertidor serial/usb si su PC no posee un puerto serial.
4. Para conectar sensores externos, tanto analógicos como digitales, se recomienda que estos tengan terminales acopladas a las puntas. Por seguridad, la terminal VCC del sensor debe ser macho y el resto hembras (ver figura 2.4). Conecte las terminales de alimentación de los sensores a la placa fenólica. El identificador de sensor en el software será interpretado de acuerdo a la posición de la conexión física de la terminal de señal. Esto se ilustra en la figura 2.5.

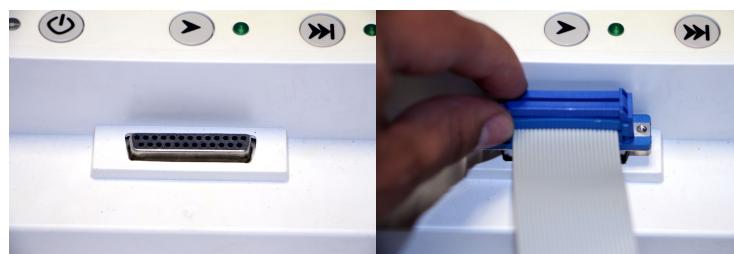


Figura 2.1: Conector DB-25 en la bahía de carga del Create®

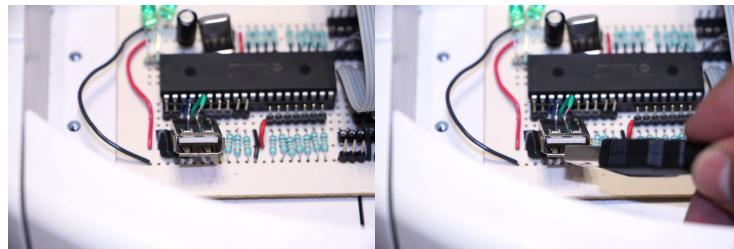


Figura 2.2: Conexión de los sensores externos mediante un cable USB

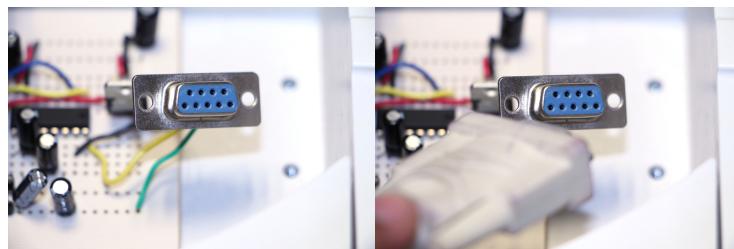


Figura 2.3: Conector DB-9 para la comunicación serial

5. Conecte el otro extremo de los cables serial y USB a la PC (ver figura 2.6).
6. Una vez realizadas estas conexiones, el robot está listo para usarse. Para aprovechar al máximos las capacidades de la biblioteca UDG_Create se recomienda utilizar una computadora portátil y montarla junto con los sensores sobre el robot como se muestra en la figura 2.7, sin embargo, otras técnicas como la utilización de hardware intermedio para la comunicación Wi-Fi o Zigbee son también posibles.

2.2. Dependencias de software

Para utilizar la biblioteca UDG_Create es necesario contar con un sistema operativo basado en linux para la arquitectura x86. Durante la ejecución se requieren permisos de root para la lectura de los sensores externos USB. Para recompilar la biblioteca es necesario contar con un compilador que cumpla con el estándar c++ 11. En esta guía de usuario se utilizan Ubuntu 13.10 y g++ 4.7.

2.3. Recompilación de la biblioteca con g++

La biblioteca UDG_Create se distribuye como una paquete precompilado. Si necesita recompilarla puede hacerlo de manera automática ejecutando el script

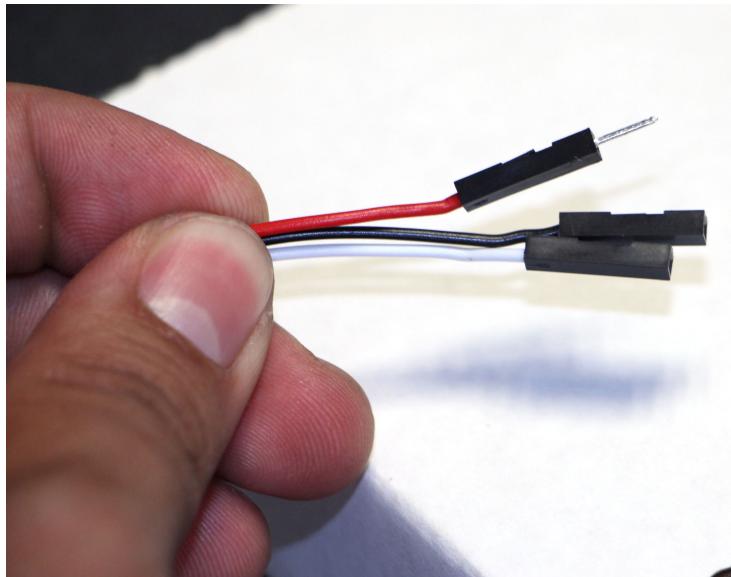


Figura 2.4: Terminales para cables recomendadas

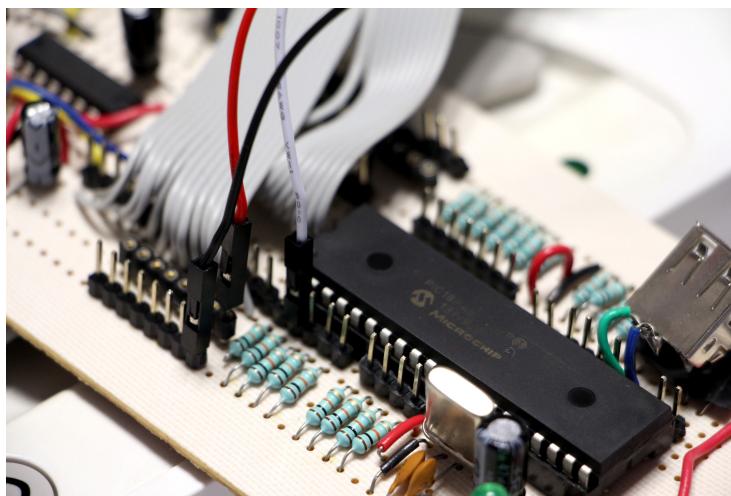


Figura 2.5: Conexión de sensores externos



Figura 2.6: Conexión de la placa fenólica a la PC



Figura 2.7: Modo de uso recomendado

rebuildLibrary.sh contenido en la carpeta del código fuente. Se recomienda utilizar los siguientes comandos:

```
cd <directorio_del_codigo_fuente>
sh rebuildLibrary.sh
```

Para compilaciones personalizadas se requiere llevar a cabo los siguientes pasos:
Compilar los archivos fuente (sin ligado).

```
g++ -c -std=c++11 USBLayer.cpp ExternalSensors.cpp UDG_Create.cpp SerialPort.c1
```

Empaqueado del código objeto.

```
ar rvs UDG_Create.a ExternalSensors.o UDG_Create.o SerialPort.o USBLayer.o
```

De manera opcional, para utilizar UDG_Create.h como archivo de encabezado estandar debe agregar el prefijo lib al nombre del paquete.

```
ar rvs libUDG_Create.a ExternalSensors.o UDG_Create.o SerialPort.o USBLayer.o
```

Por último, si así lo desea, puede eliminar los archivos intermedios generados durante el proceso.

```
rm *.o
```

2.4. Uso de UDG_Create de manera local

Copie los archivos UDG_Create.h y UDG_Create.a al directorio del código fuente de su proyecto.

Asegurese de que su programa incluya a la biblioteca como `#include "UDG_Create.h"` (con comillas).

Incluya UDG_Create.a a su comando de compilación para realizar el ligado utilizando la biblioteca. Por ejemplo, para el programa:

Listing 2.1: main.cpp

```
#include "UDG_Create.h"
int main()
{
    Create robot;
    robot.demo(9);
    return 0;
}
```

Puede utilizar:

```
g++ main.cpp UDG_Create.a
```

¹Para versiones de g++ 4.3 y posteriores pero inferiores a 4.7 utilizar -std=c++0x en lugar de -std=c++11

2.5. Uso de UDG_Create como biblioteca estándar

Copiar UDG_Create.h a uno de los directorios por defecto para archivos de encabezado (por ejemplo /usr/include/) o usar la opción -I de g++ para especificar la ruta al momento de compilar.

Copiar libUDG_Create.a a uno de los directorios por defecto de c++ para bibliotecas (por ejemplo /usr/lib/) o usar la opción -L de g++ para especificar la ruta en el momento de la compilación. La biblioteca debe incluirse en el código fuente como `#include <UDG_Create.h>` (con paréntesis angulares).

Agregue -lUDG_Create a su comando de compilación para realizar el ligado utilizando la biblioteca. Por ejemplo para el programa

Listing 2.2: main.cpp

```
#include <UDG_Create.h>
int main()
{
    Create robot;
    robot.demo(9);
    return 0;
}
```

puede usar: `g++ main.cpp -lUDG_Create`

Capítulo 3

La Clase Create

La clase de c++ Create, esta contenida en el archivo UDG_Create.h y en ella se describe la abstracción de los sensores, actuadores y otros componentes del robot además de todas las operaciones necesarias para la inicialización y operación de los módulos RS-232 y USB.

3.1. Tipos de datos y enumeraciones

3.1.1. enum errorCode

Define los códigos de error para distintas condiciones de falla durante la ejecución de un programa.

INVALID_DEMO (0x00): El número de modo de desmostración seleccionado no existe.

INVALID_BAUDRATE (0x01): Se ha seleccionado una velocidad de transmisión no válida.

INVALID_INSTRUCTION_MODE (0x02): Se ha intentado ejecutar una instrucción en un modo que no lo permite.

OUT_OF_RANGE (0x03): El parámetro seleccionado está fuera del rango permitido.

INVALID_MODE (0x04): Se ha seleccionado un modo de operación inexistente.

3.1.2. enum modes

Enlista los distintos modos en los que puede operar el robot.

OFF: Robot apagado.

PASSIVE: En el modo pasivo es posible ejecutar los programas de demostración así como leer y escribir información de sensores, pero no se permite ejecutar comandos de actuadores.

SAFE: En el modo seguro se tiene control total del robot, pero este regresa a modo pasivo si se detecta un borde (por el que el robot pudiera caer), se solicita

un radio de giro menor al radio del robot o se conecta el cargador.
FULL: Se tiene control absoluto del robot.

3.1.3. enum baudCode

Enumera las velocidades de transmisión soportadas por el Create®. Las constantes tienen un nombre de la forma BAUDX en donde X representa la velocidad de transmisión en baudios.

- BAUD300 (0x00).
- BAUD600 (0x01).
- BAUD1200 (0x02).
- BAUD2400 (0x03).
- BAUD4800 (0x04).
- BAUD9600 (0x05).
- BAUD14400 (0x06).
- BAUD19200 (0x07).
- BAUD28800 (0x08).
- BAUD38400 (0x09).
- BAUD57600 (0x0A).
- BAUD115200 (0x0B).

3.1.4. enum chargingstates

Describe los posibles estados de carga en los que pudiera encontrarse el robot. La documentación del fabricante no proporciona información adicional respecto a estas condiciones, se asumen autoexplicativas.

- NOT_CHARGING (0x00).
- RECONDITIONING_CHARGING (0x01).
- FULL_CHARGING (0x02).
- TRICKLE_CHARGING (0x03).
- WAITING (0x04).
- CHARGING_FAULT_CONDITION (0x05).

3.1.5. enum infraredbytechars

Define los posibles valores que pudieran recibirse a través del receptor infrarrojo provenientes del control remoto, la base de carga, otros robots o aplicaciones externas. La documentación del fabricante no proporciona información adicional por lo que los valores se asumen autoexplicativos.

- IRLEFT (0x81).
- IRFORWARD (0x82).
- IRRIGHT (0x83).
- IRSPOT (0x84).

IRMAX (0x85).
 IRSMALL (0x86).
 IRMEDIUM (0x87).
 IRLARGE (0x88).
 IRPAUSE (0x89).
 IRPOWER (0x8A).
 IRARC_FORWARD_LEFT (0x8B).
 IRARC_FORWARD_RIGHT (0x8C).
 IRDRIVE_STOP (0x8D).
 IRSENDALL (0x8E).
 IRSEEKDOCK (0x8F).
 IRRESERVED (0x90).
 IRRED (0x91).
 IRGREEN (0x92).
 IRFORCEFIELD (0x93).
 IRREDGREEN (0x94).
 IRREDFORCEFIELD (0x95).
 IRGREENFORCEFIELD (0x96).
 IRREDGREENFORCEFIELD (0x97).

3.1.6. enum sensorPackets

Define los el numero asociado a cada paquete de los disponibles como argumento de la funcion GetSensors().

S_PACKET0(0x00)
 S_PACKET1(0x01)
 S_PACKET2(0x02)
 S_PACKET3(0x03)
 S_PACKET4(0x04)
 S_PACKET5(0x05)
 S_PACKET6(0x06)
 S_BUMPS_AND_WHEEL_DROPS(0x07)
 S_WALL(0x08)
 S_CLIFF_LEFT(0x09)
 S_CLIFF_FRONT_LEFT(0x0A)
 S_CLIFF_FRONT_RIGHT(0x0B)
 S_CLIFF_RIGHT(0x0C)
 S_VIRTUAL_WALL(0x0D)
 S_OVERCURRENTS(0x0E)
 S_UNUSED1(0x0F)
 S_UNUSED2(0x10)
 S_IR_BYT(0x11)
 S_BUTTONS(0x12)
 S_DISTANCE(0x13)
 S_ANGLE(0x14)

```
S_CHARGING_STATE(0x15)
S_VOLTAGE(0x16)
S_CURRENT(0x17)
S_BATTERY_TEMPREATURE(0x18)
S_BATTERY_CHARGE(0x19)
S_BATTERY_CAPACITY(0x1A)
```

3.1.7. enum VerbosityLevels

VERBOSITY_NORMAL (0x00): Los mensajes del robot se escriben a la salida estándar.
 VERBOSITY_FILE (0x01): Los mensajes del robot se escriben a un archivo.
 VERBOSITY_OFF (0x02): No se escribe ningún mensaje.
 VERBOSITY_NUMBER_OF_LEVELS (0x03): El número total de niveles.

3.1.8. t_verbosity

Tipo de dato enumerado que corresponde a los valores de VerbosityLevels.

3.1.9. enum BoolSigned

SIGNED (0x01).
 UNSIGNED (0x02).

3.1.10. boolSigned

Tipo de dato enumerado que corresponde a los valores de BoolSigned y se utiliza para indicar si los elementos en un arreglo de bytes debe ser considerado como un número con o sin signo.

3.1.11. NUMBER_OF_SENSORS

Describe el máximo número de sensores externos que pueden conectarse a la interfaz USB.

3.1.12. int16

Entero de 16 bits, utilizado para garantizar su tamaño independientemente del compilador. Su definición exacta es:

```
typedef struct int_16
{
    unsigned char H; //High byte
    unsigned char L; //Low byte
} int16;
```

3.2. Miembros (privados)

3.2.1. std::string portName

Una cadena que indica el nombre del enlace simbólico asociado a la comunicación serial con el robot.

3.2.2. int mode

Un valor definido en la enumeración *oimodes* que indica el modo de operación actual del robot.

3.2.3. bool charging

Una variable lógica que indica si el robot está siendo cargado.

3.2.4. int portDescriptor

El descriptor del puerto asociado a la comunicación serial con el robot.

3.2.5. int baudRate

Un valor perteneciente a la enumeración *baudCode* que indica la velocidad de transmisión utilizada para la comunicación serial.

3.2.6. bool bumpRight

Una variable lógica que indica si el lado derecho del parachoques está siendo presionado.

3.2.7. bool bumpLeft

Una variable lógica que indica si el lado izquierdo del parachoques está siendo presionado.

3.2.8. bool wheelDropRight

Una variable lógica que indica si el sensor de caída de la rueda derecha fué activado.

3.2.9. bool wheelDropLeft

Una variable lógica que indica si el sensor de caída de la rueda izquierda fué activado.

3.2.10. bool wheelDropCaster

Una variable lógica que indica si el sensor de caída de la rueda caster fué activado.

3.2.11. bool wall

Una variable lógica que indica si una pared ha sido detectada.

3.2.12. bool cliffLeft

Una variable lógica que indica si un borde ha sido detectado del lado izquierdo.

3.2.13. bool cliffFrontLeft

Una variable lógica que indica si el sensor de bordes frontal izquierdo ha sido activado.

3.2.14. bool cliffFrontRight

Una variable lógica que indica si el sensor de bordes frontal derecho ha sido activado.

3.2.15. bool cliffRight

Una variable lógica que indica si un borde ha sido detectado del lado derecho.

3.2.16. bool virtualWall

Una variable lógica que indica si una pared virtual ha sido detectada.

3.2.17. bool ld0

Una variable lógica que indica si el *low side driver* 0 está encendido.

3.2.18. bool ld1

Una variable lógica que indica si el *low side driver* 1 está encendido.

3.2.19. bool ld2

Una variable lógica que indica si el *low side driver* 2 está encendido.

3.2.20. bool rightWheel

Una variable lógica que indica si el sensor de sobrecorriente de la rueda derecha ha sido activado.

3.2.21. bool leftWheel

Una variable lógica que indica si el sensor de sobrecorriente de la rueda izquierda ha sido activado.

3.2.22. unsigned char infraredbyte

Una variable de un byte que almacena la información leída a través del receptor infrarrojo.

3.2.23. bool advancebtn

Una variable lógica que indica si el botón *advance* del robot está siendo presionado.

3.2.24. bool playbtn

Una variable lógica que indica si el botón *play* del robot está siendo presionado.

3.2.25. int distance

Una variable entera que indica la distancia recorrida desde la última vez que se consultó.

3.2.26. int angle

Una variable entera que indica el ángulo rotado desde la última vez que se consultó.

3.2.27. unsigned char chargingstate

Un miembro de la enumeración *chargingstates* que describe las condiciones actuales de la carga de la batería.

3.2.28. int voltage

Una variable entera que representa el voltaje en mV en la batería del robot.

3.2.29. int current

La corriente en mA que entra (valores positivos) o sale (valores negativos) de la batería del robot.

3.2.30. unsigned char batterytemperature

La temperatura de la batería del robot en grados celsius.

3.2.30.0.1. int batterycharge

La carga de la batería del robot en mAh.

3.2.31. int batterycapacity

La capacidad aproximada de la batería en mAh.

3.2.31.0.2. int wallsignal

Una variable entera que representa la magnitud de la lectura en el sensor de paredes.

3.2.32. int cliffls

Una variable entera que representa la magnitud de la lectura en el sensor de bordes izquierdo.

3.2.33. int clifffls

Una variable entera que representa la intensidad de la lectura en el sensor de bordes frontal izquierdo.

3.2.34. int clifffrs

Una variable entera que representa la intensidad de la lectura en el sensor de bordes frontal derecho.

3.2.35. int cliffrs

Una variable entera que representa la intensidad de la lectura en el sensor de bordes derecho.

3.2.36. bool digitalinput0

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 0.

3.2.37. bool digitalinput1

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 1.

3.2.38. bool digitalinput2

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 2.

3.2.39. bool digitalinput3

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 3.

3.2.40. bool baudchangerate

Una variable lógica que indica si hay un nivel de voltaje alto en la terminal correspondiente al cambio de velocidad de transmisión (terminal 15).

3.2.41. int cargoanalogsignal

Una variable entera que representa el valor de la señal analógica en la terminal 4 del robot.

3.2.42. bool homebase

Una variable lógica que indica si existe una conexión entre el robot y la base de carga.

3.2.43. bool internalcharger

Una variable lógica que indica si el cargador interno está conectado.

3.2.44. unsigned char oimode

Representa el modo de operación actual del robot como un miembro de la enumeración *oimodes*.

3.2.45. unsigned char songnumber

Indica el número de la canción actualmente seleccionada.

3.2.46. bool songplaying

Indica si hay una canción siendo tocada actualmente.

3.2.47. unsigned char streampackets

El número de sensores o paquetes de sensores solicitados como parte de un flujo.

3.2.48. int reqvelocity

La velocidad de las ruedas solicitada en la última llamada a una instrucción *drive*.

3.2.49. int reqradius

El radio de giro solicitado en la última llamada a una función *drive*.

3.2.50. int reqrvelocity

La velocidad de la rueda derecha solicitada en la última llamada a una función *drive*.

3.2.51. int reqlvelocity

La velocidad de la rueda izquierda solicitada en la última llamada a una función *drive*.

3.2.52. bool streamingState

Una variable lógica que indica si un flujo de paquetes de sensores se está transmitiendo actualmente.

3.2.53. bool externalSensorsEnabled

Una variable lógica que indica si los sensores externos conectados por usb están actualmente habilitados.

3.2.54. t_verbosity robotVerbosity

Indica el tipo de mensajes de depuración que se mostrarán en pantalla.

3.3. Funciones

3.3.1. Publicas

3.3.1.1. Create()

Es el método constructor por defecto, inicializa el puerto serial utilizando los primeros enlaces simbólicos para puerto serial y USB disponibles.

3.3.1.2. Create(std::string _portName, t_verbosity verbosityLevel)

_portName: El nombre del enlace simbólico a utilizar para la comunicación serial.

verbosityLevel: El tipo de mensajes de depuración a recibir durante la ejecución. Los valores disponibles para este parámetro son

VERBOSITY_NORMAL: Los mensajes se imprimen a la salida estándar.
VERBOSITY_OFF: No se imprimen mensajes.

3.3.1.3. ~Create()

Es el método destructor por defecto, libera todos los recursos solicitados durante la ejecución.

3.3.1.4. std::string getPortName()

Regresa un valor de tipo std::string conteniendo el nombre del enlace simbólico utilizado para la inicialización del puerto serial.

3.3.1.5. void start()

Prepara al robot para recibir instrucciones por el puerto serial y lo pone en modo pasivo (*passive*).

3.3.1.6. void baud(unsigned char baudRate)

baudRate: La nueva velocidad de transmisión de acuerdo a los valores establecidos en la Create Open Interface.

Modifica por software la velocidad de transmisión (baud rate) utilizado para la comunicación serial. Se recomienda utilizar los miembros de la enumeración

baudRate de la forma BAUDx en donde x representa la velocidad.

3.3.1.7. void control()

Pone al robot en modo seguro (safe). Esta función se conserva para mantener compatibilidad con Roomba, se recomienda el uso de la función safe() en su lugar para el Create®.

3.3.1.8. void safe()

Pone al robot en modo seguro (safe).

3.3.1.9. void full()

Pone al robot en modo íntegro (full).

3.3.1.10. void spot()

Ejecuta la demostración “spot demo” en la que el robot se mueve formando una espiral. Al terminar de ejecutar la instrucción el Create® pasa a modo pasivo (passive).

3.3.1.11. void cover()

Ejecuta la demostración “cover” de la Create Open Interface en el que se cubre el área de una habitación.

3.3.1.12. void coverAndDock()

Ejecuta la demostración “cover and dock” de la Create Open Interface.

3.3.1.13. void demo(unsigned char demo)

demo: El número de la demostración a ejecutar.

Ejecuta la demostración indicada por *demo*. El número de demo debe estar entre 1 y 9.

3.3.1.14. void drive(int velocity, int radius)

speed: La velocidad promedio a la que girarán las ruedas del robot en mm/s.
radius: Radio en milímetros del giro del robot medido desde el centro del círculo de giro hasta el centro del Create®.

Controla las ruedas del robot y lo hace avanzar usando la velocidad y radio especificados. Un radio negativo genera un giro en el sentido de las manecillas del reloj y uno positivo hacia el lado contrario. Valores positivos de velocidad impulsan al robot hacia adelante mientras que los negativos lo hacen hacia atrás. Solo los 16 bits menos significativos de los parámetros de entrada son considerados.

3.3.1.15. void driveDirect(int rightVelocity, int leftVelocity)

rightVelocity: La velocidad de la rueda derecha.
leftVelocity: La velocidad de la rueda izquierda.

Hace al robot avanzar utilizando las velocidades especificadas para las ruedas izquierda y derecha. Valores positivos representan movimiento hacia adelante mientras que los negativos lo hacen hacia atrás. Solo los 16 bits menos significativos del entero con signo son considerados.

3.3.1.16. void leds(unsigned char bit,unsigned char color, unsigned char intensity)

bit: Indica el LED a encender. Los valores posibles son:

0 = ninguno.

2 = Play.

8 = Advance.

10 = Play y Advance.

color: Indica el color que mostrará el LED *Power*. El 0 representa color verde y 255 rojo. Valores intermedios mostrarán tonos intermedios (amarillo, naranja, etc.).

intensity: La intensidad con la que encenderá el LED Power.

Enciende o apaga los diodos LED seleccionados del Create® permitiendo modificar el color e intensidad del LED Power.

3.3.1.17. void digitalOutputs(unsigned char outputBits)

outputBits: El valor de salida deseado.

Establece el valor indicado en las 3 terminales de salida del conector DB-25.

Precaución: Cuando se enciende el robot, las salidas digitales permanecen en nivel alto durante 3 segundos.

3.3.1.18. void pwmLowSideDrivers(unsigned char dirver1, unsigned char driver1, unsigned char driver0)

driver2: El ciclo de trabajo para la salida PWM en la terminal 24 del conector DB-25 con un valor de entre 0 y 128.

driver1: El ciclo de trabajo para la salida PWM en la terminal 22 del conector DB-25 con un valor de entre 0 y 128.

driver0: El ciclo de trabajo para la salida PWM en la terminal 23 del conector DB-25 con un valor de entre 0 y 128.

Proporciona el ciclo de trabajo especificado en las salidas de PWM del robot. Este debe ser indicado como un número entero entre 0 y 128, por ejemplo, para indicar un ciclo de trabajo de 25 % en alguna de las salidas se escoje un 32.

3.3.1.19. void lowSideDrivers(unsigned char bits)

bits: Indica los bits a activar.

Activa los bits seleccionados de los *low side drivers*.

3.3.1.20. void sendIr(unsigned char byteValue)

byteValue: El valor del byte a enviar.

Esta función envía a un LED infrarrojo conectado a la terminal 23, el byte

indicado en el formato esperado por el receptor infrarrojo del robot.

3.3.1.21. void song(unsigned char,unsigned char,...)

Permite almacenar una canción en la memoria del robot para ser tocada posteriormente. La lista de parametros de entrada es de tamaño variable y debe apegarse al siguiente formato:

song(numeroDeCancion,numeroDeNotas, nota1,duracion1, nota2, duracion2,...,notaN, duracionN)

numeroDeCancion debe tener un valor de entre 0 y 15. *numeroDeNotas* debe estar entre 1 y 16. La altura de *notaX* se apega al esquema de numeración MIDI, pero solo en el rango de entre 31 y 127, cualquier otro valor se considerará como un silencio. El valor de *duracionX* puede estar entre 0 y 255, representando incrementos de 1/64 de segundo.

3.3.1.22. void playSong(unsigned char songNumber)

songNumber: El número de canción a tocar.

Toca la canción previamente grabada mediante la instrucion *song*. Se debe esperar a que termine una canción antes de poder tocar otra.

3.3.1.23. int sensors(unsigned char idPacket)

idPacket: El identificador del paquete solicitado.

Permite obtener el valor del sensor o paquete de sensores seleccionado y actualizar su valor en la variable correspondiente. El valor de retorno es el del último sensor leído.

3.3.1.24. int getSizePacket(int idPacket)

packet: El identificador del paquete solicitado.

Regresa el número de bytes que se esperan del sensor o paquete de sensores especificado.

3.3.1.25. void stream(unsigned char* destinationBuffer,void* thread,int n,...)

destinationBuffer: Una apuntador a un buffer de bytes en donde se almacenará la respuesta a la petición de paquetes.

thread: Este parametro se conserva para mantener compatibilidad con una versión anterior de la biblioteca y será eliminado a corto plazo.

n: El número de sensores o paquetes de sensores solicitados. Debe estar entre 0 y 42.

... : Una lista de tamaño variable contenido los identificadores (entre 0 y 42) de los sensores o paquetes de sensores solicitados.

Esta función indica al robot que debe enviar un flujo de bytes con las lecturas de los sensores solicitados cada 15 ms a través de la conexión serial. Esta información es almacenada en el buffer proporcionado en el formato:

19 nbytes IDPaquete1 DatosPaquete1 IDPaquete2 DatosPaquete2 ... IDPaqueteN DatosPaqueteN Checksum

Para más información consulte la instrucción “stream” de la Create Open Interface.

3.3.1.26. void pauseResumeStream(bool streamState)

streamState: una variable lógica que indica si el flujo debe estar activado o desactivado.

Permite detener o reiniciar el flujo de bytes contenido el valor de los sensores solicitados mediante la función stream, sin tener que reiniciar la lista.

3.3.1.27. void script(unsigned char n,...);

n: El número de instrucciones en el script. Debe estar en el rango entre 1 y 100. Permite almacenar una serie de instrucciones a manera de un script para ser ejecutadas posteriormente. Recibe como primer parametro el número de instrucciones que contendrá el script seguido de una lista de longitud variables en los que se especifican los bytes correspondientes a los códigos de operación de las instrucciones y bytes de datos de acuerdo a la Create Open Interface.

script(4, instrucion1, datos1, instrucion2, datos2);

3.3.1.28. void playScript()

Ejecuta el último script guardado mediante la instrucción `script()`.

3.3.1.29. void showScript()

Imprime en pantalla el ultimo script almacenado mediante la instrucción `script()` en su representación como numeros enteros.

3.3.1.30. void waitTime(unsigned char time)

time: El tiempo a esperar en décimas de segundo con una resolución de 15ms.

Instruye al robot para esperar el tiempo especificado durante el cual no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

3.3.1.31. void waitDistance(int distance)

distance: La distancia a esperar expresada en milímetros entre -32767 y 32768.

Instruye al robot para esperar a que la distancia indicada haya sido recorrida. Si avanza hacia adelante o las ruedas son giradas de manera pasiva en cualquier dirección la distancia se incrementa, al avanzar hacia atrás se decremente. Durante este tiempo el robot no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

3.3.1.32. void waitAngle(int angle)

angle: El ángulo a girar en grados entre -32767 y 32768.

Hace al robot esperar hasta haber rotado el ángulo especificado. Cuando el giro se hace en el sentido de las manecillas del reloj el ángulo se decrementa mientras que en la dirección contraria se incrementa. Durante este tiempo el robot no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

3.3.1.33. void waitEvent(unsigned char event)

event: Identificador del evento esperado como se especifica en la Create Open Interface, con una rango de -20 a -1 y 1 a 20.

Instruye al robot para esperar a que el evento especificado suceda. Durante este tiempo no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

3.3.1.34. char* charMode(int mode)

mode: Representación como número entero del modo actual (ver enum modes). Obtiene una representación como cadena de caracteres del modo de operación actual.

3.3.1.35. int getBaudCode(int baudCode)

baudCode: Velocidad de transmisión en su representación como número entero como se especifica en la Create Open Interface (ver enum baudCode). Regresa el valor entero correspondiente a la velocidad de transmisión.

3.3.1.36. bool getBumpRight()

Regresa el estado del parachoques derecho como un valor lógico. Verdadero significa que el parachoques ha sido presionado.

3.3.1.37. bool getBumpLeft()

Regresa el estado del parachoques izquierdo como un valor lógico. Verdadero significa que el parachoques ha sido presionado.

3.3.1.38. bool getWheelDropRight()

Indica con una variable lógica si la rueda derecha ha caído.

3.3.1.39. bool getWheelDropLeft()

Indica con una variable lógica si la rueda izquierda ha caído.

3.3.1.40. bool getWheelDropCaster()

Indica con una variable lógica si la rueda caster ha caído.

3.3.1.41. bool getWallSeen()

Regresa el estado del sensor de paredes como un valor lógico. Verdadero significa que una pared ha sido detectada.

3.3.1.42. bool getCliffLeft()

Regresa el estado del sensor de bordes izquierdo como un valor lógico. Verdadero significa que un borde ha sido detectado.

3.3.1.43. bool getCliffFrontLeft()

Regresa el estado del sensor de bordes frontal izquierdo como un valor lógico. Verdadero significa que un borde ha sido detectado.

3.3.1.44. bool getCliffFrontRight()

Regresa el estado del sensor de bordes frontal derecho como un valor lógico. Verdadero significa que un borde ha sido detectado.

3.3.1.45. bool getCliffRight()

Regresa el estado del sensor de bordes derecho como un valor lógico. Verdadero significa que un borde ha sido detectado.

3.3.1.46. bool getVirtualWall()

Regresa el estado del detector de pared virtual como un valor lógico. Verdadero significa que una pared virtual ha sido detectada.

3.3.1.47. bool getLd0()

Regresa el estado del *lowside driver 0* como un valor lógico. Verdadero significa que está activado.

3.3.1.48. bool getLd1()

Regresa el estado del *lowside driver 1* como un valor lógico. Verdadero significa que está activado.

3.3.1.49. bool getLd2()

Regresa el estado del *lowside driver 2* como un valor lógico. Verdadero significa que está activado.

3.3.1.50. bool getRightWheel()

Regresa el estado del sensor de sobrecorriente de la rueda derecha como un valor lógico. Verdadero significa que existe sobrecorriente.

3.3.1.51. bool getLeftWheel()

Regresa el estado del sensor de sobrecorriente de la rueda izquierda como un valor lógico. Verdadero significa que existe sobrecorriente.

3.3.1.52. unsigned char getInfraredByte()

Regresa un byte que representa la información recibida a través del sensor infrarrojo. Un valor de 255 significa que no se ha recibido información.

3.3.1.53. bool getAdvanceBtn()

Regresa el estado del botón “advance” como un valor lógico. Verdadero significa que el botón está presionado.

3.3.1.54. bool getPlayBtn()

Regresa el estado del botón “play” como un valor lógico. Verdadero significa que el botón está presionado.

3.3.1.55. int getDistance()

Regresa la distancia recorrida desde la última vez que se ejecutó la instrucción como un valor entero de 16 bits con signo (de -32768 a 32767). Los movimientos hacia adelante se manifiestan como números positivos y hacia atrás como negativos. El resultado final será la suma de todos los movimientos.

3.3.1.56. int getAngle()

Regresa el ángulo que se ha rotado desde la última vez que se ejecutó la instrucción como un valor entero de 16 bits con signo (de -32768 a 32767). Los movimientos en el sentido de las manecillas del reloj se manifiestan como números negativos y hacia el lado contrario como positivos. El resultado final será la suma de todos los movimientos.

3.3.1.57. unsigned char getChargingState()

Regresa el estado de carga del robot como un byte con uno de los siguientes valores ¹:

- NOT_CHARGING (0x00).
- RECONDITIONING_CHARGING (0x01).
- FULL_CHARGING (0x02).
- TRICKLE_CHARGING (0x03).
- WAITING (0x04).
- CHARGING_FAULT_CONDITION (0x05).

¹La documentación de la Create Open Interface no especifica detalles sobre estos estados

3.3.1.58. int getVoltage()

Regresa el voltaje de la batería en mV con un rango de entre 0 y 65535.

3.3.1.59. int getCurrent()

Regresa la corriente del robot en mA como un entero en un rango de -32768 a 32767. Las corrientes negativas indican un flujo desde la batería (como en la operación normal) y las positivas hacia la batería (como durante la carga).

3.3.1.60. unsigned char getBatteryTemperature()

Regresa un byte correspondiente a la temperatura de la batería en grados centígrados en un rango de -128 a 127.

3.3.1.61. int getBatteryCharge()

Regresa un entero correspondiente a la carga de la batería en mAh en un rango de 0 a 65535.

3.3.1.62. int getBatteryCapacity()

Regresa un entero correspondiente a la capacidad aproximada de la batería en mAh con un rango de 0 a 65535. Esta estimación puede no ser precisa cuando se utilizan baterías alcalinas.

3.3.1.63. int getWallSignal()

Regresa un entero correspondiente a la intensidad de la señal del sensor de paredes en un rango de 0 a 4095.

3.3.1.64. int getCliffLS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bor-

des izquierdo en un rango de 0 a 4095.

3.3.1.65. int getCliffFLS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes frontal izquierdo en un rango de 0 a 4095.

3.3.1.66. int getCliffFRS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes frontal derecho en un rango de 0 a 4095.

3.3.1.67. int getCliffRS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes derecho en un rango de 0 a 4095.

3.3.1.68. bool getDigitalInput0()

Regresa el valor de la entrada digital 0 como una variable lógica. Verdadero representa un nivel alto de voltaje.

3.3.1.69. bool getDigitalInput1()

Regresa el valor de la entrada digital 1 como una variable lógica. Verdadero representa un nivel alto de voltaje.

3.3.1.70. bool getDigitalInput2()

Regresa el valor de la entrada digital 2 como una variable lógica. Verdadero representa un nivel alto de voltaje.

3.3.1.71. bool getDigitalInput3()

Regresa el valor de la entrada digital 3 como una variable lógica. Verdadero representa un nivel alto de voltaje.

3.3.1.72. bool getBaudRateChange()

Regresa el valor de la terminal 15 en el conector DB-25 del robot como una variable lógica. Verdadero representa un nivel alto de voltaje.

3.3.1.73. int getCargoAnalogSignal()

Regresa un valor entero de 10 bits (entre 0 y 1023) representando el nivel de voltaje de 0 a 5v presente en la terminal 4 del puerto DB-25 del Create®.

3.3.1.74. bool getHomeBase()

Regresa un valor lógico que indica si la base del Create® está disponible como fuente de carga.

3.3.1.75. bool getInternalCharger()

Regresa un valor lógico que indica si el cargador interno está disponible como fuente de carga.

3.3.1.76. unsigned char getOIMode()

Regresa uno de los siguientes valores de la enumeracion *oimodes*:

OIOFF (0x00): El robot se encuentra apagado.

OIPASSIVE (0x01): El robot se encuentra en modo pasivo.

OISAFE (0x02): El robot se encuentra en modo seguro.

OIFULL (0x03): El robot se encuentra en modo íntegro.

3.3.1.77. `unsigned char getSongNumber()`

Regresa el número de la canción actualmente seleccionada.

3.3.1.78. `bool getSongPlaying()`

Regresa verdadero si hay una canción tocando o falso si no la hay.

3.3.1.79. `unsigned char getStreamPackets()`

Regresa el número de sensores o paquetes de sensores solicitados para un flujo. Este valor de retorno estará entre 0 y 43.

3.3.1.80. `int getRequestedVelocity()`

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada mediante un comando *drive* en mm/s.

3.3.1.81. `int getRequestedRadius()`

Regresa un entero entre -32768 y 32767 representando el radio más reciente solicitando mediante un comando *drive* en mm.

3.3.1.82. `int getRequestedRVelocity()`

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada para la rueda derecha mediante un comando *drive* en mm/s.

3.3.1.83. `int getRequestedLVelocity()`

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada para la rueda izquierda mediante un comando *drive* en mm/s.

3.3.1.84. bool getStreamingState()

Regresa un valor lógico que indica si actualmente se transmite un flujo de sensores o no.

3.3.1.85. void getExternalSensors(int[NUMBER_OF_SENSORS] sensors)

sensors: Un arreglo de NUMBER_OF_SENSORS (14 en la implementación actual) enteros pasado por referencia para almacenar los resultados.

Instruye al microcontrolador conectado por USB realizar las lecturas y conversiones de los sensores externos, almacenar los resultados en *sensors* y enviarlos de regreso.

3.3.1.86. bool getExternalSensorsEnabledStatus()

Regresa una variable lógica indicando si el uso de los sensores externos está habilitado.

3.3.1.87. void setVerbosity(t_verbosity)

t_verbosity: El identificador del tipo de mensajes a mostrar.

Permite modificar la forma en que se muestran los mensajes de depuración durante la ejecución del programa. Los valores que puede tomar son:

VERBOSITY_NORMAL: Los mensajes se imprimen a la salida estándar.

VERBOSITY_OFF: No se imprimen mensajes.

3.3.1.88. int getExternalNthSensor(int n)

n: El número de sensor solicitado.

Obtiene el valor del enésimo sensor externo conectado al microcontrolador USB.

3.3.1.89. int toLittleEndian(unsigned char* source,int nbytes,int* destination,boolSigned sign)

source: Un arreglo de bytes que contiene los datos a convertir.

nbytes: Tamaño en bytes de los datos a convertir.

destination: Un apuntador a entero contenido la dirección de la variable en donde se almacenará el resultado final de la conversión.

sign: Una variable lógica que indica si los datos a convertir tienen o no signo.

Convierte el arreglo de bytes que recibe como entrada a una variable entera compatible con la arquitectura x86.

3.3.2. Privadas

3.3.2.1. void error(int error,void* info)

error: Un miembro de la enumeración errorCodes que identifica el tipo de error.

info: Información adicional que puede ser utilizada por el programador para la depuración del código.

Imprime de manera estandarizada los errores detectados en tiempo de ejecución.

3.3.2.2. void printRobotMessage(const char* message,...)

message: El mensaje a imprimir, acepta los mismos indicadores de formato que printf. Internamente se usa como primer parametro de vprintf.

... Una lista de parametros opcional que se pasara como segundo argumento a vprintf.

Es un *wrapper* alrededor de la función vprintf que provee una interfaz para imprimir mensajes durante la ejecución.

3.3.2.3. int readBumpsAndWheelDrops(unsigned char *data)

data: La lectura de sensores provenientes del robot.

Convierte la lectura del parachoques y sensores de caída de las ruedas a un formato compatible con la arquitectura x86, la almacena en las variables *bumpRight*, *bumpLeft* , *wheelDropRight*, *wheelDropLeft* y *wheelDropCaster*. Su valor de retorno es un entero cuyos 5 bits menos significativos representan, en ese orden, el estado del parachoques derecho, parachoques izquierdo y sensores de

caída de las ruedas derecha, izquierda y caster.

3.3.2.4. int readWall(unsigned char *data)

data: La lectura del sensor de paredes proveniente del robot.

Convierte la lectura del sensor de paredes a un formato compatible con la arquitectura x86 y la almacena en la variable *wall* que es también el valor de retorno de la función.

3.3.2.5. int readCliffLeft(unsigned char *data)

data: La lectura del sensor de bordes izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffLeft* que es también su valor de retorno.

3.3.2.6. int readCliffFrontLeft(unsigned char *data)

data: La lectura del sensor de bordes frontal izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes frontal izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffFrontLeft* que es también su valor de retorno.

3.3.2.7. int readCliffFrontRight(unsigned char *data)

data: La lectura del sensor de bordes frontal derecho proveniente del robot.

Convierte la lectura del sensor de bordes frontal derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffFrontRight* que es también su valor de retorno.

3.3.2.8. int readCliffRight(unsigned char *data)

data: La lectura del sensor de bordes derecho proveniente del robot.

Convierte la lectura del sensor de bordes derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffRight* que es también su valor de retorno.

3.3.2.9. int readVirtualWall(unsigned char *data)

data: La lectura del sensor de pared virtual proveniente del robot.

Convierte la lectura del sensor de pared virtual a un formato compatible con la arquitectura x86, la almacena en la variable *virtualWall* que es también su valor de retorno.

3.3.2.10. int readLSDriverAndWheelO(unsigned char *data)

data: La lectura del paquete de sensores *Low Side Driver and Wheel Overcurrents* proveniente del robot.

Convierte la lectura del paquete de sensores *Low Side Driver and Wheel Overcurrents* a un formato compatible con la arquitectura x86 y la almacena en las variables ld1, ld0, ld2, rightWheel y leftWheel correspondientes a los *low side drivers* 1, 0 y 2 y los sensores de sobrecorriente derecho e izquierdo respectivamente. Estos valores son empaquetados en un solo byte en ese orden (siendo ld0 el bit menos significativo) que se utiliza como valor de retorno.

3.3.2.11. int readInfraredByte(unsigned char *data)

data: La lectura del receptor infrarrojo proveniente del robot.

Convierte la lectura del receptor infrarrojo a un formato compatible con la arquitectura x86 y la almacena en la variable *infraredbyte* que es también su valor de retorno.

3.3.2.12. int readButtons(unsigned char *data)

data: La lectura del estado de los botones programables proveniente del robot. Convierte la lectura del estado de los botones del Create® a un formato compatible con la arquitectura x86 y la almacena en las variables *playbtn* y *advancebtn* que son empaquetadas como el bit 0 (menos significativo) y 2 respectivamente

de una variable entera que es el valor de retorno.

3.3.2.13. int readDistance(unsigned char *data)

data: El valor de la distancia recorrida proveniente del robot.

Convierte la lectura de distancia recorrida a un formato compatible con la arquitectura x86, la almacena en la variable *distance* que es también su valor de retorno.

3.3.2.14. int readAngle(unsigned char *data)

data: La lectura del ángulo rotado proveniente del robot.

Convierte la lectura del ángulo rotado desde la última vez que se ejecutó la instrucción a un formato compatible con la arquitectura x86 y la almacena en la variable *distance* que es también su valor de retorno.

3.3.2.15. int readChargingState(unsigned char *data)

data: La lectura del estado de carga proveniente del robot.

Convierte la lectura del estado de carga a un formato compatible con la arquitectura x86 y la almacena en la variable *chargingstate* que es también su valor de retorno.

3.3.2.16. int readVoltage(unsigned char *data)

data: La lectura del voltaje de la batería proveniente del robot.

Convierte la lectura del voltaje de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *voltage* que es también su valor de retorno.

3.3.2.17. int readCurrent(unsigned char *data)

data: La lectura de la corriente que consume o recibe el robot.

Convierte la lectura de corriente a un formato compatible con la arquitectura x86 y la almacena en la variable *current* que es también su valor de retorno.

3.3.2.18. int readBatteryTemperature(unsigned char *data)

data: La lectura del temperatura de la batería del robot.

Convierte la lectura de la temperatura de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *batterytemperature* que es también su valor de retorno.

3.3.2.19. int readBatteryCharge(unsigned char *data)

data: La lectura del la carga de la batería proveniente del robot.

Convierte la lectura de la carga de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *batterycapacity* que es también su valor de retorno.

3.3.2.20. int readWallSignal(unsigned char *data)

data: La lectura del sensor de paredes proveniente del robot.

Convierte la lectura del sensor de paredes a un formato compatible con la arquitectura x86 y la almacena en la variable *wallsignal* que es también su valor de retorno.

3.3.2.21. int readCliffLS(unsigned char *)data

data: La lectura del sensor de bordes izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffls* que es también su valor de retorno.

3.3.2.22. int readCliffFLS(unsigned char *data)

data: La lectura del sensor de bordes frontal izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes frontal izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *clifffls* que es también su valor de retorno.

3.3.2.23. int readCliffFRS(unsigned char *data)

data: La lectura del sensor de bordes frontal derecho proveniente del robot.

Convierte la lectura del sensor de bordes frontal derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *clifftrs* que es también su valor de retorno.

3.3.2.24. int readCliffRS(unsigned char *data)

data: La lectura del sensor de bordes derecho proveniente del robot.

Convierte la lectura del sensor de bordes derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffrs* que es también su valor de retorno.

3.3.2.25. int readDigitalInputs(unsigned char *data)

data: La lectura de las entradas digitales y el bit de cambio de velocidad de transmisión del robot.

Convierte la lectura de las entradas digitales y el bit de cambio de velocidad de transmisión a un formato compatible con la arquitectura x86 y la almacena en las variables *digitalinput0*, *digitalinput1*, *digitalinput2*, *digitalinput3* y *baudchanagerate* que se empaquetan en ese orden como un entero siendo *digitalinput0* el bit menos significativo, que se convierte en valor de retorno.

3.3.2.26. int readCargoAnalogSignal(unsigned char *data)

data: La lectura de la entrada analógica proveniente del robot.

Convierte la lectura de la entrada analógica del robot a un formato compatible con la arquitectura x86 y la almacena en la variable *cargoanalogsignal* que es también su valor de retorno.

3.3.2.27. int readChargingSources(unsigned char *data)

data: La lectura de las fuentes de carga disponibles proveniente del robot.

Convierte la lectura de las fuentes de carga disponibles a un formato compatible con la arquitectura x86 y la almacena en las variables *internalcharger* y *hombase* que se empaquetan como los bits 0 y 1 respectivamente de una variable entera que es también su valor de retorno.

3.3.2.28. int readOIMode(unsigned char *data)

data: La lectura del modo de operación proveniente del robot.

Convierte la lectura del modo de operación a un formato compatible con la arquitectura x86 y la almacena en la variable *oimode* que es también su valor de retorno.

3.3.2.29. int readSongNumber(unsigned char *data)

data: La lectura del número de canción proveniente del robot.

Convierte la lectura del número de canción a un formato compatible con la arquitectura x86 y la almacena en la variable *songnumber* que es también su valor de retorno.

3.3.2.30. int readSongPlaying(unsigned char *data)

data: La lectura del estado de ejecución de canción proveniente del robot.

Convierte la lectura del estado de ejecución de canción a un formato compatible con la arquitectura x86 y la almacena en la variable *songplaying* que es también su valor de retorno.

3.3.2.31. int readStreamPackets(unsigned char *data)

data: El número de sensores o paquetes de sensores solicitados mediante la instrucción *stream* proveniente del robot.

Convierte la lectura del número de sensores solicitados a un formato compatible con la arquitectura x86 y la almacena en la variable *strempackets* que es también su valor de retorno.

3.3.2.32. int readReqVelocity(unsigned char *data)

data: La lectura de la velocidad solicitada proveniente del robot.

Convierte la lectura de la última velocidad solicitada a un formato compatible con la arquitectura x86 y la almacena en la variable *reqvelocity* que es también su valor de retorno.

3.3.2.33. int readReqRadius(unsigned char *data)

data: La lectura del radio de giro solicitado al robot.

Convierte la lectura del último radio de giro solicitado a un formato compatible con la arquitectura x86 y la almacena en la variable *regradius* que es también su valor de retorno.

3.3.2.34. int readReqRVelocity(unsigned char *data)

data: La lectura de la última velocidad de la rueda derecha solicitada proveniente del robot.

Convierte la lectura de la velocidad de la rueda derecha a un formato compatible con la arquitectura x86 y la almacena en la variable *reqrvelocity* que es también su valor de retorno.

3.3.2.35. int readReqLVelocity(unsigned char *data)

data: La lectura de la última velocidad de la rueda izquierda solicitada proveniente del robot.

Convierte la lectura de la velocidad de la rueda izquierda a un formato compatible con la arquitectura x86 y la almacena en la variable *reqVelocity* que es también su valor de retorno.

3.3.2.36. int updateSensor(unsigned char packetID, int size)

packetID: El identificador del sensor o paquete de sensores que se actualizará.
size: El tamaño en bytes de los datos del sensor o paquete de sensores a actuarizar.

Actualiza el valor del sensor o paquete de sensores especificados por *packetID*.

3.3.2.37. void commonInitializationProcedures(string port,bool auto)

port: El nombre del enlace simbólico que representa el puerto serial que se comunicará con el robot.

auto: Indica si se debe o no buscar el enlace simbólico que representa al puerto serial de manera automática.

Efectúa todas las operaciones comunes a los distintos métodos constructores.

3.3.2.38. int16.toInt16(int integer)

integer: El valor a convertir.

Realiza una conversión de una variable entera a una estructura que representa un entero de 16 bits.

Capítulo 4

Code Samples