Arduino API

A reference to the Arduino Programming Language.

杜孟桓

Last revision2024/10/14

Compact version of the Arduino Language Reference. This document is a **TLDR;** of the Arduino API.

Please note that as of 2024/01/15, this article is still a work in progress.

Functions

Digital I/O

Method & Parameters	Description	Returns
`int digitalRead(int pin)`	Reads the state of a digital pin.	`int`
`void digitalWrite(int pin, int state)`	Writes a state to a digital pin.	Nothing
`void pinMode(int pin, int mode)` *	Define the mode of a pin.	Nothing

*Available modes are:

- INPUT(0)
- OUTPUT(1)
- INPUT_PULLUP (2)
- INPUT_PULLDOWN (3)
- OUTPUT_OPENDRAIN(4)

Analog I/O

Method & Parameters	Description	Returns
`int analogRead(int pin)`	Reads the value of an analog pin in a 10-bit resolution (0-1023).*	`int`
`void analogReadResolution(int resolution)`	Sets ADC read resolution in bits.	Nothing
`void analogReference(int reference)`	Changes the voltage reference for a board.**	Nothing
`void analogWrite(int pin, int value)`	Writes a value to a PWM supported pin in a 8-bit resolution (0-255).**	Nothing
`void analogWriteResolution(int resolution)`	Sets write resolution for a board.	Nothing

- *The value range changes based on the resolution. 0-1023 is 10-bit resolution, 0-4096 is 12-bit and so on.
- **Each board/architecture has a set of different reference voltages available.
- ***The value range changes based on the resolution. 0-255 is default (8-bit).

Advanced I/O

Method & Parameters	Description	Returns
`void tone(int pin, int frequency, long duration)`	Generates a square wave on specified pin, with 50% duty cycle.	Nothing
`void noTone(int pin)`	Stops generation of square wave on the specified pin.	Nothing
`long pulseIn(int pin, int state, long timeout)`	Reads a pulse (either HIGH or LOW) on a pin and returns the length of the pulse (in microseconds)	`long`
`long pulseInLong(int pin, int state, long timeout)`	Returns the length of the pulse (in microseconds)	`long`
`int shiftIn(int pin, int clockPin, int bitOrder)` *	Shifts in a byte of data one bit at a time, and returns the value of the bit read.	`byte`
`void shiftOut(int pin, int clockPin, int bitOrder, byte value)` **	Shifts out a byte of data one bit at a time.	Nothing

- *The bitOrder parameter is either MSBFIRST (1) or LSBFIRST (0) (most / least significant bits).
- **The pin used for shiftOut() needs to be configured as an OUTPUT, using pinMode()

Time

Method & Parameters	Description	Returns
`void delay(long milliseconds)`	Freezes program execution for specified number of **milliseconds**.	Nothing
`void delayMicroseconds(int microseconds)`	Freezes program execution for specified number of **microseconds**.	Nothing
`long millis()`	Returns **milliseconds** passed since program start.	`long`
`long micros()`	Returns **microseconds** passed since program start.	`long`

Math

Method & Parameters	Description	Returns
`int abs(int value)`	Calculates the absolute value of a number.	`int`
`int constrain(int value, int min, int max)`	Constrains a number to be within a range.	`int`
`long map(long val, long min, long max, long newMin, long newMax)`	Re-maps a number from one range to another.	`long`
`int max(int val1, int val2)`	Returns the greater of two values.	`int`
`int min(int val1, int val2)`	Returns the smaller of two values.	`int`
`double pow(double base, double exponent)`	Raises a base to the power of an exponent.	`double`
`int sq(int value)`	Calculates the square of a number.	`int`
`double sqrt(double value)`	Calculates the square root of a number.	`double`

Trigonometry

Method & Parameters	Description	Returns
`cos(double angle)`	Calculates the cosine of an angle in radians.	`double`
`sin(double angle)`	Calculates the sine of an angle in radians.	`double`
`tan(double angle)`	Calculates the tangent of an angle in radians.	`double`

Characters

Method & Parameters	Description	Returns
`boolean isAlpha(char c)`	Checks if the character is an alphabetic character.	`boolean`
`boolean isAlphaNumeric(char c)`	Checks if the character is an alphanumeric character.	`boolean`
`boolean isAscii(char c)`	Checks if the character is a 7-bit ASCII character.	`boolean`
`boolean isControl(char c)`	Checks if the character is a control character.	`boolean`
`boolean isDigit(char c)`	Checks if the character is a digit (0-9).	`boolean`
`boolean isGraph(char c)`	Checks if the character is a printable character, excluding space.	`boolean`
`boolean isHexadecimalDigit(char c)`	Checks if the character is a hexadecimal digit (0-9, A-F, a-f).	`boolean`
`boolean isLowerCase(char c)`	Checks if the character is a lowercase alphabetic character.	`boolean`
`boolean isPrintable(char c)`	Checks if the character is a printable character, including space.	`boolean`
`boolean isPunct(char c)`	Checks if the character is a punctuation character.	`boolean`
`boolean isSpace(char c)`	Checks if the character is a whitespace character.	`boolean`
`boolean isUpperCase(char c)`	Checks if the character is an uppercase alphabetic character.	`boolean`
`boolean isWhitespace(char c)`	Checks if the character is a whitespace character according to `isSpaceChar()` method.	`boolean`

Random Numbers

Method & Parameters	Description	Returns
`int random()`	Generates a pseudo-random number between 0 and `RAND_MAX`.	`int`
`void randomSeed(unsigned long seed)`	Seeds the random number generator.	Nothing

Bits and Bytes

Method & Parameters	Description	Returns
`boolean bit(int value, int bitNumber)`	Gets the value of a specific bit.	`boolean`
`void bitClear(int &value, int bit)`	Clears a specific bit.	Nothing
`boolean bitRead(int value, int bitNumber)`	Reads the value of a specific bit.	`boolean`
`void bitSet(int &value, int bit)`	Sets a specific bit.	Nothing
`void bitWrite(int &value, int bit, int bitValue)`	Writes a value to a specific bit.	Nothing
`byte highByte(int value)`	Returns the high byte of an `int`.	`byte`
`byte lowByte(int value)`	Returns the low byte of an `int`.	`byte`

External Interrupts

Method & Parameters	Description	Returns
`void attachInterrupt(int pin, void (*function)(void), int mode)`	Attaches an interrupt to a specific pin.	Nothing
`void detachInterrupt(int pin)`	Detaches an interrupt from a specific pin.	Nothing

Interrupts

Method & Parameters	Description	Returns
`void interrupts()`	Enables interrupts globally.	Nothing
`void noInterrupts()`	Disables interrupts globally.	Nothing

Stream

Method & Parameters	Description	Returns
`int available()`	Returns the number of bytes available in the serial buffer.	`int`
`int read()`	Reads the next byte from the serial buffer.	`int`
`void flush()`	Waits for the transmission of outgoing serial data to complete.	Nothing
`int find(char *target)`	Searches for a target string in the serial buffer.	`int`
`int findUntil(char *target, char *terminate)`	Searches for a target string until a specified termination string is found.	`int`
`int peek()`	Returns the next byte in the serial buffer without removing it.	`int`
`int readBytes(char *buffer, int length)`	Reads characters from the serial buffer into a buffer.	`int`
`int readBytesUntil(char terminator, char *buffer, int length)`	Reads characters from the serial buffer into a buffer until a terminator is found.	`int`
`String readString()`	Reads characters from the serial buffer into a String until a newline character is found.	`String`
`String readStringUntil(char terminator)`	Reads characters from the serial buffer into a String until a specified terminator is found.	`String`
`int parseInt()`	Reads characters from the serial buffer and converts them to an integer.	`int`
`float parseFloat()`	Reads characters from the serial buffer and converts them to a float.	`float`
`void setTimeout(unsigned long timeout)`	Sets the maximum duration for `find()`, `findUntil()`, `parseInt()`, and `parseFloat()`.	Nothing

Serial

Method & Parameters	Description	Returns
`if(Serial)`	Checks if the Serial object is available.	`boolean`
`int available()`	Returns the number of bytes available for reading.	`int`
`int availableForWrite()`	Returns the number of bytes available for writing.	`int`
`void begin(unsigned long baudrate)`	Initializes the Serial communication with the specified baud rate.	`void`
`void end()`	Ends the Serial communication.	`void`
`int find(char *target)`	Searches for a target string in the serial buffer.	`int`
`int findUntil(char *target, char *terminate)`	Searches for a target string until a specified termination string is found.	`int`
`void flush()`	Waits for the transmission of outgoing serial data to complete.	`void`
`float parseFloat()`	Reads characters from the serial buffer and converts them to a float.	`float`
`int parseInt()`	Reads characters from the serial buffer and converts them to an integer.	`int`
`int peek()`	Returns the next byte in the serial buffer without removing it.	`int`
`size_t print()`	Prints data to the serial port.	`size_t`
`size_t println()`	Prints data to the serial port followed by a newline character.	`size_t`
`int read()`	Reads the next byte from the serial buffer.	`int`
`int readBytes(char *buffer, size_t length)`	Reads characters from the serial buffer into a buffer.	`int`
`int readBytesUntil(char terminator, char *buffer, size_t length)`	Reads characters from the serial buffer into a buffer until a terminator is found.	`int`
`String readString()`	Reads characters from the serial buffer into a String until a newline character is found.	`String`
`String readStringUntil(char terminator)`	Reads characters from the serial buffer into a String until a specified terminator is found.	`String`
`void setTimeout(unsigned long timeout)`	Sets the maximum duration for `find()`, `findUntil()`, `parseInt()`, and `parseFloat()`.	`void`
`size_t write(uint8_t)`	Writes a byte to the serial port.	`size_t`
`void serialEvent()`	Called when data is available in the serial buffer.	`void`

SPI

Method & Parameters	Description	Returns
`SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)`	Creates an SPISettings object with the specified clock, bit order, and data mode.	`SPISettings`
`void begin()`	Initializes the SPI library.	`void`
`void beginTransaction(SPISettings settings)`	Begins an SPI transaction with the specified settings.	`void`
`void endTransaction()`	Ends the current SPI transaction.	`void`
`void end()`	Ends the SPI library.	`void`
`void setBitOrder(uint8_t bitOrder)`	Sets the bit order (MSBFIRST or LSBFIRST) for SPI communication.	`void`
`void setClockDivider(uint8_t divider)`	Sets the clock divider for SPI communication.	`void`
`void setDataMode(uint8_t dataMode)`	Sets the data mode for SPI communication.	`void`
`byte transfer(byte value)`	Transfers a byte over SPI.	`byte`
`void usingInterrupt(int interruptNumber)`	Specifies which interrupt to use for SPI transactions.	`void`

I2C (Wire)

Method & Parameters	Description	Returns
`void begin()`	Initializes the Wire library.	`void`
`void end()`	Ends the Wire library.	`void`
`int requestFrom(int address, int quantity)`	Requests data from a slave device with the specified address and quantity of bytes.	`int`
`void beginTransmission(int address)`	Begins a transmission to the slave device with the specified address.	`void`
`int endTransmission()`	Ends the transmission and returns the status.	`int`
`size_t write(uint8_t data)`	Writes a byte to the I2C bus.	`size_t`
`int available()`	Returns the number of bytes available for reading.	`int`
`int read()`	Reads a byte from the I2C bus.	`int`
`void setClock(uint32_t frequency)`	Sets the I2C clock frequency.	`void`
`void onReceive(void (*function)(int))`	Sets a function to be called when data is received by the slave.	`void`
`void onRequest(void (*function)(void))`	Sets a function to be called when the master requests data from the slave.	`void`
`void setWireTimeout(uint32_t timeout)`	Sets the timeout for I2C operations.	`void`
`void clearWireTimeoutFlag()`	Clears the timeout flag.	`void`
`bool getWireTimeoutFlag()`	Returns the timeout flag status.	`bool`

Variables

Enums

Enum Type	Enumeration	Description
`PinStatus`	`HIGH` / `LOW`	Logical HIGH and LOW values (`1` and `0`).
`PinMode`	`INPUT` / `OUTPUT` / `INPUT_PULLUP` / `INPUT_PULLDOWN` / `OUTPUT_OPENDRAIN`	Constants for specifying pin modes (`0` , `1` , `2` , `3` , `4`).
	`LED_BUILTIN`	Constant representing the built-in LED pin.*
	`true` / `false`	Boolean constants for true and false (`1` and `0`).

Conversion

Method & Parameter	Description
`(unsigned int)`	Type casting to unsigned int.
`(unsigned long)`	Type casting to unsigned long.
`byte()`	Type casting to byte.
`char()`	Type casting to char.
`float()`	Type casting to float.
`int()`	Type casting to int.
`long()`	Type casting to long.
`word()`	Type casting to word.

Data Types

Method & Parameter	Description
`array`	Collection of variables of the same type.
`bool`	Boolean data type.
`boolean`	Boolean data type (synonym for bool).
`byte`	8-bit unsigned data type.
`char`	8-bit character data type.
`double`	Double-precision floating-point data type.
`float`	Single-precision floating-point data type.
`int`	Integer data type.
`long`	Long integer data type.
`short`	Short integer data type.
`size_t`	Unsigned integer data type.
`string`	Sequence of characters (not a primitive type).
`String()`	String class in Arduino.
`unsigned char`	Unsigned 8-bit character data type.
`unsigned int`	Unsigned integer data type.
`unsigned long`	Unsigned long integer data type.
`void`	Represents the absence of a type.
`word`	16-bit unsigned data type.

Variable Scope & Qualifiers

Method & Parameter	Description
`const`	Qualifier to define constants.
`scope`	Not a specific keyword; refers to variable scope.
`static`	Qualifier to declare static variables.
`volatile`	Qualifier to declare volatile variables.

Utilities

Method & Parameter	Description
`PROGMEM`	Qualifier to store data in program memory.
`sizeof()`	Operator to determine the size of a data type or variable.

Structure

Sketch

Method & Parameter	Description
`void loop()`	Main function for continuous code execution.
`void setup()`	Initialization function, called once at startup.

Control Structure

Method & Parameter	Description
`break`	Exits a loop or switch statement.
`continue`	Skips the rest of a loop iteration.
`do...while`	Executes a block of code repeatedly while a specified condition is true.
`else`	Part of the if-else statement.
`for`	Creates a loop with a specified initialization, condition, and increment.
`goto`	Transfers control to a labeled statement.
`if`	Conditional statement for decision-making.
`return`	Exits a function and optionally returns a value.
`switch...case`	Multi-way branch statement.
`while`	Creates a loop with a specified condition.

Further Syntax

Method & Parameter	Description
`#define (define)`	Macro definition for code substitution.
`#include (include)`	Includes a file in the source code.
`/* */ (block comment)`	Block comment for multiple lines.
`// (single line comment)`	Single line comment.
`; (semicolon)`	Statement terminator.
`{} (curly braces)`	Block of code, often used with control structures.

Arithmetic Operators

Method & Parameter	Description
`% (remainder)`	Modulo operator for finding the remainder of a division.
`* (multiplication)`	Multiplication operator.
`+ (addition)`	Addition operator.
`- (subtraction)`	Subtraction operator.
`/ (division)`	Division operator.
`= (assignment operator)`	Assignment operator.

Comparison Operators

Method & Parameter	Description
`!= (not equal to)`	Checks if two values are not equal.
`< (less than)`	Checks if the left value is less than the right value.
`<= (less than or equal to)`	Checks if the left value is less than or equal to the right value.
`== (equal to)`	Checks if two values are equal.
`> (greater than)`	Checks if the left value is greater than the right value.
`>= (greater than or equal to)`	Checks if the left value is greater than or equal to the right value.

Boolean Operators

Method & Parameter	Description
`& (bitwise and)`	Performs bitwise AND operation.
`<< (bitshift left)`	Shifts bits to the left.
`>> (bitshift right)`	Shifts bits to the right.
`^ (bitwise xor)`	Performs bitwise XOR (exclusive OR) operation.
`| (bitwise or)`	Performs bitwise OR operation.
`~ (bitwise not)`	Inverts all bits.

Pointer Access Operators

Method & Parameter	Description
`& (reference operator)`	Returns the memory address of a variable.
`* (dereference operator)`	Accesses the value pointed to by a pointer.

Bitwise Operators

Method & Parameter	Description
`& (bitwise and)`	Performs bitwise AND operation.
`<< (bitshift left)`	Shifts bits to the left.
`>> (bitshift right)`	Shifts bits to the right.
`^ (bitwise xor)`	Performs bitwise XOR (exclusive OR) operat
`| (bitwise or)`	Performs bitwise OR operation.
`~ (bitwise not)`	Inverts all bits.

Compound Operators

Method & Parameter	Description
`%= (compound remainder)`	Performs a modulo operation and assigns the result to the left operand.
`&= (compound bitwise and)`	Performs a bitwise AND operation and assigns the result to the left operand.
`*= (compound multiplication)`	Multiplies the left operand by the right operand and assigns the result to the left operand.
`++ (increment)`	Increments the value of the operand by 1.
`+= (compound addition)`	Adds the right operand to the left operand and assigns the result to the left operand.
`-- (decrement)`	Decrements the value of the operand by 1.
`-= (compound subtraction)`	Subtracts the right operand from the left operand and assigns the result to the left operand.
`/= (compound division)`	Divides the left operand by the right operand and assigns the result to the left operand.
`^= (compound bitwise xor)`	Performs a bitwise XOR operation and assigns the result to the left operand.
`|= (compound bitwise or)`	Performs a bitwise OR operation and assigns the result to the left operand.