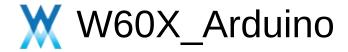


Modules

Here is a list of all modules:

		[detail level 12]
SPI_library	SPI library	
▼WiFi_library	WiFi library	
WiFi_class		
Client_class		
Server_class		
UDP_class		
IPAddress_class		
Wire_Library	Wire Library	
IO	Digital Analog and Advanced I/O	
Serial	Serial	
Stream	Stream	
Math	Math	

Generated by doxygen 1.8.14



SPI_library

Functions

SPI library. More...

Functions

- void SPIClass::begin (void)
 - Initialize the SPI instance. More...
- void SPIClass::end (void)

Deinitialize the SPI instance and stop it. More...

void SPIClass::beginTransaction (SPISettings settings)

This function should be used to configure the SPI instance in case you don't use the default parameters set by the **begin()** function. More...

void SPIClass::endTransaction (void)

settings associated to the SPI instance. More...

uint8 t SPIClass::transfer (uint8 t data)

Transfer one byte on the SPI bus. More...

uint16 t SPIClass::transfer16 (uint16 t data)

Transfer two bytes on the SPI bus in 16 bits format. More...

void SPIClass::transferWrite (void *_buf, size_t _count)

send several bytes. More...

void SPIClass::transferRead (void *_buf, size_t _count)

receive several bytes. More...

void SPIClass::transfer (void *_buf, size_t _count)

Transfer several bytes. Only one buffer used to send and receive data. More...

void SPIClass::transfer (void *_bufout, void *_bufin, size_t _count)

Transfer several bytes. One buffer contains the data to send and another one will contains the data received. More...

void SPIClass::setBitOrder (BitOrder)

Deprecated function. Configure the bit order: MSB first or LSB first. More...

void SPIClass::setDataMode (uint8_t _mode)

Deprecated function. Configure the data mode (clock polarity and clock phase) More...

void SPIClass::setFrequency (uint32_t freq)

Configure the spi frequency. More...

Detailed Description

SPI library.

Function Documentation

begin()

void SPIClass::begin (void)

Initialize the SPI instance.

Parameters

[in] none

Returns

none

Note

beginTransaction()

void SPIClass::beginTransaction (SPISettings settings)

This function should be used to configure the SPI instance in case you don't use the default parameters set by the **begin()** function.

Parameters

[in] settings SPI settings(clock speed, bit order, data mode).

Returns

none

Note

◆ end()

void SPIClass::end (void)

Deinitialize the SPI instance and stop it.

Parameters

[in] none

Returns

none

Note

endTransaction()

void SPIClass::endTransaction (void)

settings associated to the SPI instance.

Parameters

[in] none

Returns

none

Note

setBitOrder()

void SPIClass::setBitOrder (BitOrder _bitOrder)

Deprecated function. Configure the bit order: MSB first or LSB first.

Parameters

[in] _bitOrder MSBFIRST or LSBFIRST

Returns

none

Note

setDataMode()

```
void SPIClass::setDataMode ( uint8_t _mode )
```

Deprecated function. Configure the data mode (clock polarity and clock phase)

Parameters

Returns

none

Note

setFrequency()

void SPIClass::setFrequency (uint32_t freq)

Configure the spi frequency.

Parameters

[in] freq max 20MHz

Returns

none

Note

transfer() [1/3]

```
uint8_t SPIClass::transfer ( uint8_t data )
```

Transfer one byte on the SPI bus.

Parameters

[in] data byte to send.

Returns

byte received from the slave

Note

transfer() [2/3]

Transfer several bytes. Only one buffer used to send and receive data.

Parameters

[in]_buf pointer to the bytes to send. The bytes received are copy in this buffer.

[in] _count number of bytes to send/receive

Returns

none

Note

◆ transfer() [3/3]

```
void SPIClass::transfer ( void * _bufout,
	void * _bufin,
	size_t _count
)
```

Transfer several bytes. One buffer contains the data to send and another one will contains the data received.

Parameters

```
[in] _bufout pointer to the bytes to send.
[in] _bufin pointer to the bytes received.
[in] _count number of bytes to send/receive
```

Returns

none

Note

transfer16()

uint16_t SPIClass::transfer16 (uint16_t data)

Transfer two bytes on the SPI bus in 16 bits format.

Parameters

[in] data bytes to send.

Returns

bytes received from the slave in 16 bits format.

Note

transferRead()

receive several bytes.

Parameters

```
[in] _buf pointer to the bytes to received.
[in] _count number of bytes to received
```

Returns

none

Note

transferWrite()

send several bytes.

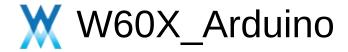
Parameters

```
[in] _buf pointer to the bytes to send.
[in] _count number of bytes to send
```

Returns

none

Note



WiFi_library

Modules

WiFi library. More...

Modules

WiFi_class

Client_class

Server_class

UDP_class

IPAddress_class

Detailed Description

WiFi library.

IPAddress class.

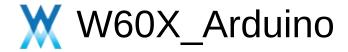
UDP class.

Server class.

Client class.

WiFi class.

Generated by doxygen 1.8.14



WiFi_class

WiFi_library

Functions

Functions

- int WiFiSTAClass::begin (const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)
 This function is used to start the wifi module as station mode. More...
- int WiFiSTAClass::begin (char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)
 This function is used to start the wifi module as station mode. More...
- int WiFiSTAClass::begin ()
 This function is used to start the wifi module as station mode. More...
- bool WiFiSTAClass::reconnect ()
 This function is used to reconect the AP. More...
- bool WiFiSTAClass::disconnect (bool wifioff=false)
 This function is used to disconnect the wifi. More...
- bool WiFiSTAClass::isConnected ()
 This function is used to get the wifi mode connect status.
 More...
 - bool WiFiSTAClass::setAutoConnect (bool autoConnect)
 This function is used to set auto connect flag. More...
 - bool WiFiSTAClass::getAutoConnect ()
 This function is used to get auto connect flag. More...
 - bool WiFiSTAClass::setAutoReconnect (bool autoReconnect)
 This function is used to set auto reconnect flag. More...

bool WiFiSTAClass::getAutoReconnect ()

This function is used to get auto reconnect flag. More...

uint8 t WiFiSTAClass::waitForConnectResult ()

This function is used to suspend until the WiFi is connected. More...

IPAddress WiFiSTAClass::localIP ()

This function is used to get the local ip address. More...

char * WiFiSTAClass::macAddress ()

This function is used to get local MAC address used by the module. More...

char * WiFiSTAClass::macAddressStr ()

This function is used to get local MAC address used by the module. More...

IPAddress WiFiSTAClass::subnetMask ()

This function is used to get subnet mask. More...

IPAddress WiFiSTAClass::getwayIP ()

This function is used to get gateway IP address. More...

IPAddress WiFiSTAClass::dnsIP (uint8_t dns_no=0)

This function is used to get DNS IP address. More...

char * WiFiSTAClass::hostname ()

bool WiFiSTAClass::hostname (char *aHostname)

bool WiFiSTAClass::hostname (const char *aHostname)

wl_status_t WiFiSTAClass::status ()

This function is used to get the status during Station mode. More...

char * WiFiSTAClass::statusStr ()

This function is used to get the status during Station mode. More...

char * WiFiSTAClass::SSID () const

This function is used to get the SSID used by the module. More...

char * WiFiSTAClass::psk () const

This function is used to get the psk used by the module. More...

uint8_t * WiFiSTAClass::BSSID ()

This function is used to get the BSSID which is connected by the module. More...

char * WiFiSTAClass::BSSIDstr ()

This function is used to get the BSSID which is connected by the module. More...

int32 t WiFiSTAClass::RSSI ()

This function is used to get the RSSI. More...

Detailed Description

Function Documentation

begin() [1/3]

This function is used to start the wifi module as station mode.

Parameters

```
[in] ssid Specify the SSID.
[in] passphrase Specify the passphrase.
[in] channel Specify the channel.
[in] bssid Specify the BSSID.
[in] connect Specify the connect.
```

Returns

If the paraments is invalid, negative is returned. Otherwise, wifi status is returned.

begin() [2/3]

This function is used to start the wifi module as station mode.

Parameters

```
[in] ssid Specify the SSID.
[in] passphrase Specify the passphrase.
[in] channel Specify the channel.
[in] bssid Specify the BSSID.
[in] connect Specify the connect.
```

Returns

If the paraments is invalid, negative is returned. Otherwise, wifi status is returned.

◆ begin() [3/3]

int WiFiSTAClass::begin (void)

This function is used to start the wifi module as station mode.

Parameters

[in] None.

Returns

If the paraments is invalid, negative is returned. Otherwise, wifi status is returned.

♦ BSSID()

```
uint8_t * WiFiSTAClass::BSSID ( )
```

This function is used to get the BSSID which is connected by the module.

Parameters

[in] None

Returns

The BSSID of the AP.

Note

Parameters

[in] none

Returns

The BSSID of the AP.

◆ BSSIDstr()

char * WiFiSTAClass::BSSIDstr()

This function is used to get the BSSID which is connected by the module.

Parameters

[in] None

Returns

The string of the BSSID of the AP.

disconnect()

bool WiFiSTAClass::disconnect (bool wifioff = false)

This function is used to disconnect the wifi.

Parameters

[in] wifioff Specify the parameter.

Returns

true is returned.

dnsIP()

```
IPAddress WiFiSTAClass::dnsIP ( uint8_t dns_no = 0 )
```

This function is used to get DNS IP address.

Parameters

[in] dns_no The index of the dns.

Returns

The DNS's IPv4 address.

Note

Parameters

[in] dns_no The index of the dns.

Returns

The DNS's IPv4 address

getAutoConnect()

bool WiFiSTAClass::getAutoConnect()

This function is used to get auto connect flag.

Parameters

[in] None

Returns

true - set the auto-connect flag, otherwise, return false.

getAutoReconnect()

bool WiFiSTAClass::getAutoReconnect()

This function is used to get auto reconnect flag.

Parameters

[in] None

Returns

true - set the auto-reconnect flag, otherwise, return false.

getwayIP()

IPAddress WiFiSTAClass::getwayIP()

This function is used to get gateway IP address.

Parameters

[in] None

Returns

The gateway's IPv4 address.

isConnected()

bool WiFiSTAClass::isConnected ()

This function is used to get the wifi mode connect status.

Parameters

[in] None

Returns

true - WiFi status is connected, otherwise, false.

◆ localIP()

IPAddress WiFiSTAClass::localIP()

This function is used to get the local ip address.

Parameters

[in] None

Returns

The local IPv4 address configure on the Module.

macAddress()

```
char * WiFiSTAClass::macAddress()
```

This function is used to get local MAC address used by the module.

Parameters

[in] None

Returns

The MAC address.

macAddressStr()

char * WiFiSTAClass::macAddressStr()

This function is used to get local MAC address used by the module.

Parameters

[in] None

Returns

The string of the MAC address.

◆ psk()

char * WiFiSTAClass::psk () const

This function is used to get the psk used by the module.

Parameters

[in] None

Returns

The string of the PSK.

reconnect()

bool WiFiSTAClass::reconnect()

This function is used to reconect the AP.

Parameters

[in] None

Returns

If reconnect successfully, return true, otherwise, return false.

♦ RSSI()

```
int32_t WiFiSTAClass::RSSI()
```

This function is used to get the RSSI.

Parameters

[in] None

Returns

the value of rssi in this connect.

setAutoConnect()

bool WiFiSTAClass::setAutoConnect (bool autoConnect)

This function is used to set auto connect flag.

Parameters

[in] autoReconnect Specify the auto-connect flag.

Returns

true - set successfully.

setAutoReconnect()

bool WiFiSTAClass::setAutoReconnect (bool autoReconnect)

This function is used to set auto reconnect flag.

Parameters

[in] autoReconnect Specify the auto-reconnect flag.

Returns

true - set successfully.

◆ SSID()

char * WiFiSTAClass::SSID () const

This function is used to get the SSID used by the module.

Parameters

[in] none

Returns

The string of the SSID.

Note

Parameters

[in] None

Returns

The string of the SSID.

status()

```
wl_status_t WiFiSTAClass::status ( )
```

This function is used to get the status during Station mode.

Parameters

[in] None

Returns

The status of WiFi Mode

statusStr()

```
char * WiFiSTAClass::statusStr()
```

This function is used to get the status during Station mode.

Parameters

[in] None

Returns

The string of the status of WiFi Mode.

subnetMask()

IPAddress WiFiSTAClass::subnetMask()

This function is used to get subnet mask.

Parameters

[in] None

Returns

The sub-net mask.

waitForConnectResult()

uint8_t WiFiSTAClass::waitForConnectResult()

This function is used to suspend until the WiFi is connected.

Parameters

[in] None

Returns

WiFi status.

Note

Generated by doxygen 1.8.14



Client_class

WiFi_library

Functions | Friends

Functions

WiFiClient::WiFiClient ()

This function is constructor, it's used to creates a client that can connect to to a specified internet IP address and port as defined in client.connect(). More...

WiFiClient::WiFiClient (const WiFiClient &)

WiFiClient & WiFiClient::operator= (const WiFiClient &)

virtual WiFiClient::~WiFiClient()

This function is deconstructor, it's used to release **WiFiClient** class. More...

uint8_t WiFiClient::status ()

return tcp status of WiFiClient. More...

virtual int WiFiClient::connect (IPAddress ip, uint16_t port)

This function is used to connect to the IP address and port specified in the constructor. More...

virtual int WiFiClient::connect (const char *host, uint16_t port)

This function is used to connect to the IP address and port specified in the constructor. More...

virtual int **WiFiClient::connect** (const String host, uint16_t port)

virtual size_t WiFiClient::write (uint8_t)

This function is used to write data to the server the client is connected to. More...

virtual size_t WiFiClient::write (const uint8_t *buf, size_t size)

This function is used to write data to the server the client is connected to. More...

virtual size_t WiFiClient::write_P (PGM_P buf, size_t size)

size_t WiFiClient::write (Stream &stream)

size_t **WiFiClient::write** (**Stream** &stream, size_t unitSize) __attribute__((deprecated))

virtual int WiFiClient::available ()

Returns the number of bytes available for reading (That is, the amount of data that has been written to the client by the server it is connected to). More...

virtual int WiFiClient::read ()

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int WiFiClient::read (uint8_t *buf, size_t size)

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int WiFiClient::peek ()

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual size t WiFiClient::peekBytes (uint8 t *buffer, size t length)

size_t WiFiClient::peekBytes (char *buffer, size_t length)

virtual void WiFiClient::flush ()

Discard any bytes that have been written to the client but not yet read. More...

virtual void WiFiClient::stop ()

This function is used to disconnect from the server. More...

virtual uint8_t WiFiClient::connected ()

Whether or not the client is connected. More...

virtual WiFiClient::operator bool ()

IPAddress WiFiClient::remotelP ()

This function is used to gets the IP address of the remote connection. More...

uint16 t WiFiClient::remotePort ()

This function is used to gets the port of the remote connection. More...

IPAddress WiFiClient::localIP ()

This function is used to gets the IP address of the local tcp connection. More...

uint16_t WiFiClient::localPort ()

This function is used to gets the port of the local tcp connection. More...

bool WiFiClient::getNoDelay ()

This function is used to get whether no delay of the tcp connection. More...

void WiFiClient::setNoDelay (bool nodelay)

This function is used to set no delay for the tcp connection. More...

static void WiFiClient::setLocalPortStart (uint16_t port)

This function is used to set local port number. More...

size_t WiFiClient::availableForWrite ()

This function is used to get the length that can be written. More...

static void WiFiClient::stopAll ()

This function is used to stop all **WiFiClient** session. More...

static void WiFiClient::stopAllExcept (WiFiClient *c)

This function is used to stop all **WiFiClient** session without exC. More...

void WiFiClient::keepAlive (uint16 t

idle_sec=TCP_DEFAULT_KEEPALIVE_IDLE_SEC, uint16_t intv_sec=TCP_DEFAULT_KEEPALIVE_INTERVAL_SEC, uint8_t count=TCP_DEFAULT_KEEPALIVE_COUNT) This function is used to set keep alive. More...

bool WiFiClient::isKeepAliveEnabled () const

This function is used to get whether enable keep alive. More...

uint16_t WiFiClient::getKeepAliveIdle () const

This function is used to get idle time interval. More...

uint16_t WiFiClient::getKeepAliveInterval () const

This function is used to get keep alive time interval. More...

uint8_t WiFiClient::getKeepAliveCount () const

This function is used to get keep alive count. More...

void WiFiClient::disableKeepAlive ()

This function is used to set disable keep alive. More...

Friends

class WiFiServer

Detailed Description

Function Documentation

available()

int WiFiClient::available (void)

virtual

Returns the number of bytes available for reading (That is, the amount of data that has been written to the client by the server it is connected to).

Parameters

[in] None
[out] None

Return values

The number of bytes available

Note

available() inherits from the Stream utility class.

availableForWrite()

```
size_t WiFiClient::availableForWrite()
```

This function is used to get the length that can be written.

Parameters

[in] None
[out] None

Return values

the length that can be written

connect() [1/2]

```
int WiFiClient::connect ( IPAddress ip, uint16_t port ) virtual
```

This function is used to connect to the IP address and port specified in the constructor.

Parameters

```
[in] ip the IP address that the client will connect to (array of 4 bytes)
[in] port the port that the client will connect to (int)
[out] None
```

Return values

1 the connection succeeds

0 the connection failed

Note

connect() [2/2]

This function is used to connect to the IP address and port specified in the constructor.

Parameters

Return values

1 the connection succeeds

0 the connection failed

Note

connected()

uint8_t WiFiClient::connected()

virtual

Whether or not the client is connected.

Parameters

[in] None
[out] None

Return values

1 the client is connected

0 the client is disconnected

Note

that a client is considered connected if the connection has been closed but there is still unread data.

disableKeepAlive()

void WiFiClient::disableKeepAlive ()

inline

This function is used to set disable keep alive.

Parameters

[in] None
[out] None

Returns

None

flush()

void WiFiClient::flush()

virtual

Discard any bytes that have been written to the client but not yet read.

Parameters

[in] None
[out] None

Returns

None

Note

flush() inherits from the Stream utility class.

getKeepAliveCount()

uint8_t WiFiClient::getKeepAliveCount () const

This function is used to get keep alive count.

Parameters

[in] None
[out] None

Return values

keep alive count

getKeepAliveIdle()

uint16_t WiFiClient::getKeepAliveIdle () const

This function is used to get idle time interval.

Parameters

[in] None
[out] None

Return values

idle time interval

getKeepAliveInterval()

uint16_t WiFiClient::getKeepAliveInterval() const

This function is used to get keep alive time interval.

Parameters

[in] None
[out] None

Return values

keep alive time interval

getNoDelay()

bool WiFiClient::getNoDelay()

This function is used to get whether no delay of the tcp connection.

Parameters

[in] None
[out] None

Return values

true no delay **false** delay

isKeepAliveEnabled()

bool WiFiClient::isKeepAliveEnabled () const

This function is used to get whether enable keep alive.

Parameters

[in] None
[out] None

Return values

true enable **false** disable

keepAlive()

This function is used to set keep alive.

Parameters

```
[in] idle_sec idle time interval
[in] intv_sec keep alive time interval
[in] count keep alive count
[out] None
```

Return values

None

◆ localIP()

```
IPAddress WiFiClient::localIP()
```

This function is used to gets the IP address of the local tcp connection.

Parameters

[in] None
[out] None

Return values

the IP address(4 bytes)

◆ localPort()

```
uint16_t WiFiClient::localPort ( )
```

This function is used to gets the port of the local tcp connection.

Parameters

[in] None
[out] None

Return values

the local port number

peek()

int WiFiClient::peek (void)

virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

Parameters

[in] None
[out] None

Return values

none is availableother the next byte or character

Note

This function inherited from the **Stream** class. See the **Stream** class main page for more information.

read() [1/2]

int WiFiClient::read (void)

virtual

Read the next byte received from the server the client is connected to (after the last call to read()).

Parameters

[in] None
[out] None

Return values

none is available.other The next character

Note

read() inherits from the Stream utility class

read() [2/2]

Read the next byte received from the server the client is connected to (after the last call to read()).

Parameters

```
[in] buf the byte to read
[in] size the size of the buf
[out] None
```

Return values

none is available.other The next byte

Note

read() inherits from the Stream utility class

remotelP()

IPAddress WiFiClient::remoteIP()

This function is used to gets the IP address of the remote connection.

Parameters

[in] None
[out] None

Return values

the IP address(4 bytes)

remotePort()

```
uint16_t WiFiClient::remotePort()
```

This function is used to gets the port of the remote connection.

Parameters

[in] None
[out] None

Return values

The port number

setLocalPortStart()

static void WiFiClient::setLocalPortStart (uint16_t port)

inline static

This function is used to set local port number.

Parameters

```
[in] port number
[out] None
```

Returns

None

setNoDelay()

void WiFiClient::setNoDelay (bool nodelay)

This function is used to set no delay for the tcp connection.

Parameters

[in] None
[out] None

Return values

the local port number

status()

```
uint8_t WiFiClient::status()
```

return tcp status of WiFiClient.

Parameters

[in] None
[out] None

Return values

tcp status

stop()

void WiFiClient::stop ()

virtual

This function is used to disconnect from the server.

Parameters

[in] None
[out] None

Returns

None

Note

◆ stopAll()

void WiFiClient::stopAll ()

static

This function is used to stop all WiFiClient session.

Parameters

[in] None
[out] None

Return values

None

stopAllExcept()

void WiFiClient::stopAllExcept (WiFiClient * except)

static

This function is used to stop all WiFiClient session without exC.

Parameters

[in] None
[out] None

Return values

None

WiFiClient()

WiFiClient::WiFiClient()

This function is constructor, it's used to creates a client that can connect to to a specified internet IP address and port as defined in client.connect().

Parameters

[in] None
[out] None

Returns

None

write() [1/2]

```
size_t WiFiClient::write ( uint8_t b )
```

virtual

This function is used to write data to the server the client is connected to.

Parameters

```
[in] the char to write
[out] None
```

Return values

the number of characters written. it is not necessary to read this value.

Note

write() [2/2]

This function is used to write data to the server the client is connected to.

Parameters

```
[in] buf the byte to write
[in] size the size of the buf
[out] None
```

Return values

the number of characters written. it is not necessary to read this value.

Note

~WiFiClient()

WiFiClient::~WiFiClient()

virtual

This function is deconstructor, it's used to release WiFiClient class.

Parameters

[in] None
[out] None

Returns

None

Note

Generated by doxygen 1.8.14



Server_class

WiFi_library

Functions

Functions

WiFiServer::WiFiServer (IPAddress addr, uint16_t port)This constructor is used to init **WiFiServer** object with the address and port specify by caller. More...

WiFiServer::WiFiServer (uint16_t port)

This constructor is used to init **WiFiServer** object with port specify by caller. More...

bool WiFiServer::hasClient ()

void WiFiServer::begin ()

This function is used to start the **WiFiServer**. More...

void WiFiServer::begin (uint16_t port)

This function is used to start the WiFiServer. More...

void WiFiServer::setNoDelay (bool nodelay)

This function is used to set no-delay flag. More...

bool WiFiServer::getNoDelay ()

This function is used to get no-delay flag. More...

virtual size t WiFiServer::write (uint8 t)

This function is used to send the message (one byte) to peer. More...

virtual size_t **WiFiServer::write** (const uint8_t *buf, size_t size)

This function is used to send the message to peer.

More...

uint8_t WiFiServer::status ()

void WiFiServer::close ()

This function is used to close the connection. More...

void WiFiServer::stop ()
This function is used to close the connection. More...

long WiFiServer::_accept (tcp_pcb *newpcb, long err)

static err_t WiFiServer::_s_accept (void *arg, tcp_pcb *newpcb, err_t err)

Detailed Description

Function Documentation

begin() [1/2]

void WiFiServer::begin (void)

virtual

This function is used to start the WiFiServer.

Parameters

[in] none
[out]

Implements **Server**.

begin() [2/2]

void WiFiServer::begin (uint16_t port)

This function is used to start the WiFiServer.

Parameters

[in] port Specify the port used by the object.

Returns

None

close()

void WiFiServer::close()

This function is used to close the connection.

Parameters

[in] None

Returns

None

getNoDelay()

bool WiFiServer::getNoDelay()

This function is used to get no-delay flag.

Parameters

[in] None

Returns

If the no-delay flag is true, return true, otherwise return false.

setNoDelay()

void WiFiServer::setNoDelay (bool nodelay)

This function is used to set no-delay flag.

Parameters

[in] nodelay Specify the flag of no-delay

Returns

None

stop()

void WiFiServer::stop()

This function is used to close the connection.

Parameters

[in] None

Returns

None

WiFiServer() [1/2]

```
WiFiServer::WiFiServer ( IPAddress addr, uint16_t port )
```

This constructor is used to init **WiFiServer** object with the address and port specify by caller.

Parameters

[in] addr Specify the IPv4 address.[in] port Specify the port used by the object.

Returns

None

♦ WiFiServer() [2/2]

WiFiServer::WiFiServer (uint16_t port)

This constructor is used to init **WiFiServer** object with port specify by caller.

Parameters

[in] port Specify the port used by the object.

Returns

None

write() [1/2]

size_t WiFiServer::write (uint8_t b)

virtual

This function is used to send the message (one byte) to peer.

Parameters

[in] **b** Specify the byte which will be sent to peer.

Returns

The length of the message sent to peer.

Note

Implements Print.

write() [2/2]

This function is used to send the message to peer.

Parameters

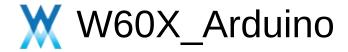
[in] buf Specify the buffer which will be sent to perr.[in] size Specify the length which will be sent.

Returns

The length of the message sent to peer.

Note

Reimplemented from Print.



UDP_class

WiFi_library

Functions

Functions

WiFiUDP::WiFiUDP()

This function is constructor, it's used to creates a named instance of the **WiFiUDP** class that can send and receive **UDP** messages. More...

WiFiUDP::WiFiUDP (const **WiFiUDP** &other)

WiFiUDP & WiFiUDP::operator= (const WiFiUDP &rhs)

WiFiUDP::~WiFiUDP()

This function is deconstructor, it's used to release **WiFiUDP** class. More...

WiFiUDP::operator bool () const

virtual uint8 t WiFiUDP::begin (uint16 t port)

This function is used to initializes the **WiFiUDP** library and network settings, Starts **UDP** socket, listening at local port. More...

virtual void WiFiUDP::stop ()

This function is used to disconnect from the server. Release any resource being used during the **UDP** session. More...

uint8_t WiFiUDP::beginMulticast (IPAddress interfaceAddr, IPAddress multicast, uint16_t port)
This function is used to join a multicast group and listen on the given port. More...

virtual int **WiFiUDP::beginPacket (IPAddress** ip, uint16_t port)

This function is used to starts a connection to write **UDP** data to the remote connection. More...

virtual int **WiFiUDP::beginPacket** (const char *host, uint16_t port)

This function is used to starts a connection to write **UDP** data to the remote connection. More...

virtual int WiFiUDP::beginPacketMulticast (IPAddress multicastAddress, uint16_t port, IPAddress interfaceAddress, int ttl=1)

This function is used to start building up a packet to send to the multicast address. More...

virtual int WiFiUDP::endPacket ()

This function is used to called after writing **UDP** data to the remote connection. It finishes off the packet and send it. More...

virtual size_t WiFiUDP::write (uint8_t)

This function is used to writes **UDP** data to the remote connection. More...

virtual size_t WiFiUDP::write (const uint8_t *buffer, size_t size)
This function is used to writes UDP data to the remote connection. More...

virtual int WiFiUDP::parsePacket ()

It starts processing the next available incoming packet, checks for the presence of a **UDP** packet, and reports the size. More...

virtual int WiFiUDP::available ()

Get the number of bytes (characters) available for reading from the buffer. This is is data that's already arrived. More...

virtual int WiFiUDP::read ()

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character

in the buffer. More...

virtual int WiFiUDP::read (unsigned char *buffer, size_t len)

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int WiFiUDP::read (char *buffer, size t len)

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int WiFiUDP::peek ()

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void WiFiUDP::flush ()

Discard any bytes that have been written to the client but not yet read. More...

virtual IPAddress WiFiUDP::remoteIP ()

This function is used to gets the IP address of the remote connection. More...

virtual uint16_t WiFiUDP::remotePort ()

This function is used to gets the port of the remote **UDP** connection. More...

IPAddress WiFiUDP::destinationIP ()

This function is used to distinguish multicast and ordinary packets. More...

uint16_t WiFiUDP::localPort ()

This function is used to gets the port of the local **UDP** connection. More...

static void WiFiUDP::stopAll ()

This function is used to stop all **WiFiUDP** session. More...

static void WiFiUDP::stopAllExcept (WiFiUDP *exC) This function is used to stop all WiFiUDP session

without exC. More...

Detailed Description

Function Documentation

available()

int WiFiUDP::available (void)

virtual

Get the number of bytes (characters) available for reading from the buffer. This is data that's already arrived.

Parameters

[in] None
[out] None

Return values

parsePacket hasn't been called yetother the number of bytes available in the current packet

Note

This function can only be successfully called after **parsePacket()**. **available()** inherits from the **Stream** utility class.

begin()

```
uint8_t WiFiUDP::begin ( uint16_t port )
```

virtual

This function is used to initializes the **WiFiUDP** library and network settings, Starts **UDP** socket, listening at local port.

Parameters

```
[in] the local port to listen on
[out] None
```

Return values

1 successful

0 there are no sockets available to use

Note

beginMulticast()

```
uint8_t WiFiUDP::beginMulticast ( IPAddress interfaceAddr, IPAddress multicast, uint16_t port )
```

This function is used to join a multicast group and listen on the given port.

Parameters

[in] interfaceAddress the local IP address of the interface

that should be used, use WiFi.localIP() or WiFi.softAPIP() depending on the

interface you need

[in] multicast

[in] port

[out] None

multicast group port number

Return values

1 successful

0 failed

beginPacket() [1/2]

```
int WiFiUDP::beginPacket ( IPAddress ip, uint16_t port ) virtual
```

This function is used to starts a connection to write **UDP** data to the remote connection.

Parameters

```
[in] ip the IP address of the remote connection (4 bytes)
[in] port the port of the remote connection (int)
[out] None
```

Return values

1 successful

0 there was a problem with the supplied IP address or port

Note

beginPacket() [2/2]

This function is used to starts a connection to write **UDP** data to the remote connection.

Parameters

Return values

1 successful

0 there was a problem with the supplied IP address or port

Note

beginPacketMulticast()

This function is used to start building up a packet to send to the multicast address.

Parameters

Return values

1 successful

0 there was a problem with the supplied IP address or port

destinationIP()

IPAddress WiFiUDP::destinationIP()

This function is used to distinguish multicast and ordinary packets.

Parameters

[in] None
[out] None

Return values

the destination address for incoming packets

endPacket()

int WiFiUDP::endPacket()

virtual

This function is used to called after writing **UDP** data to the remote connection. It finishes off the packet and send it.

Parameters

[in] None
[out] None

Return values

1 the packet was sent successfully

0 there was an error

Note

flush()

void WiFiUDP::flush()

virtual

Discard any bytes that have been written to the client but not yet read.

Parameters

[in] None
[out] None

Returns

None

Note

flush() inherits from the Stream utility class.

◆ localPort()

```
uint16_t WiFiUDP::localPort ( )
```

This function is used to gets the port of the local **UDP** connection.

Parameters

[in] None
[out] None

Return values

the local port for outgoing packets

parsePacket()

int WiFiUDP::parsePacket()

virtual

It starts processing the next available incoming packet, checks for the presence of a UDP packet, and reports the size.

Parameters

[in] None
[out] None

Return values

o no packets are availableother the size of the packet in bytes

Note

parsePacket() must be called before reading the buffer with read().

peek()

int WiFiUDP::peek (void)

virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

Parameters

[in] None
[out] None

Return values

none is availableother the next byte or character

Note

This function inherited from the **Stream** class. See the **Stream** class main page for more information.

read() [1/3]

int WiFiUDP::read (void)

virtual

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

[in] None
[out] None

Return values

no buffer is availableother the characters in the buffer (char)

Note

read() [2/3]

```
int WiFiUDP::read ( unsigned char * buffer, size_t len
) virtual
```

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

```
[in] buffer buffer to hold incoming packets (unsigned char*)
[in] len maximum size of the buffer (int)
[out] None
```

Return values

-1 no buffer is available other the size of the buffer

Note

◆ read() [3/3]

```
virtual int WiFiUDP::read ( char * buffer, size_t len
) inline virtual
```

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

```
[in] buffer buffer to hold incoming packets (char*)
[in] len maximum size of the buffer (int)
[out] None
```

Return values

-1 no buffer is available other the size of the buffer

Note

remoteIP()

IPAddress WiFiUDP::remoteIP()

virtual

This function is used to gets the IP address of the remote connection.

Parameters

[in] None
[out] None

Return values

the IP address of the host who sent the current incoming packet(4 bytes)

Note

This function must be called after parsePacket().

remotePort()

```
uint16 t WiFiUDP::remotePort()
```

virtual

This function is used to gets the port of the remote **UDP** connection.

Parameters

[in] None
[out] None

Return values

The port of the host who sent the current incoming packet

Note

This function must be called after parsePacket().

stop()

void WiFiUDP::stop()

virtual

This function is used to disconnect from the server. Release any resource being used during the **UDP** session.

Parameters

[in] None
[out] None

Returns

None

Note

◆ stopAll()

void WiFiUDP::stopAll ()

static

This function is used to stop all WiFiUDP session.

Parameters

[in] None
[out] None

Return values

None

stopAllExcept()

void WiFiUDP::stopAllExcept (WiFiUDP * exC)

static

This function is used to stop all WiFiUDP session without exC.

Parameters

[in] None
[out] None

Return values

None

◆ WiFiUDP()

WiFiUDP::WiFiUDP()

This function is constructor, it's used to creates a named instance of the **WiFiUDP** class that can send and receive **UDP** messages.

Parameters

[in] None
[out] None

Returns

None

write() [1/2]

size_t WiFiUDP::write (uint8_t byte)

virtual

This function is used to writes **UDP** data to the remote connection.

Parameters

[in] the outgoing byte
[out] None

Return values

single byte into the packet

Note

Must be wrapped between **beginPacket()** and **endPacket()**. **beginPacket()** initializes the packet of data, it is not sent until **endPacket()** is called.

write() [2/2]

This function is used to writes UDP data to the remote connection.

Parameters

```
[in] buffer the outgoing message
[in] size the size of the buffer
[out] None
```

Return values

bytes size from buffer into the packet

Note

Must be wrapped between **beginPacket()** and **endPacket()**. **beginPacket()** initializes the packet of data, it is not sent until **endPacket()** is called.

→ ~WiFiUDP()

WiFiUDP::~WiFiUDP()

This function is deconstructor, it's used to release WiFiUDP class.

Parameters

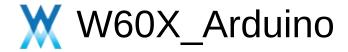
[in] None
[out] None

Returns

None

Note

Generated by doxygen 1.8.14



IPAddress_class

WiFi_library

Functions

Functions

	IPAddress::IPAddress () This constructor function is used to construct IPAddress object. More
	IPAddress::IPAddress (uint8_t first_oct, uint8_t sec_oct, uint8_t third_oct, uint8_t fourth_oct) This constructor function is used to construct IPAddress object. More
	IPAddress::IPAddress (uint32_t address) This constructor function is used to construct IPAddress object. More
	IPAddress::IPAddress (const uint8_t *address) This constructor function is used to construct IPAddress object. More
	IPAddress::operator uint32_t () const This operator overloading function is used to overloading 'uint32_t' operator. More
bool	IPAddress::operator== (const IPAddress &addr) const This operator overloading function is used to overloading '==' operator. More
bool	IPAddress::operator== (uint32_t addr) const This operator overloading function is used to overloading '==' operator. More
bool	IPAddress::operator== (const uint8_t *addr) const This operator overloading function is used to overloading '==' operator. More
uint8_t	<pre>IPAddress::operator[] (int index) const</pre>

This operator overloading function is used to overloading '[]' operator. More...

uint8_t & IPAddress::operator[] (int index)

This operator overloading function is used to overloading '[]' operator. More...

IPAddress & IPAddress::operator= (const uint8_t *address)

This operator overloading function is used to overloading '=' operator. More...

IPAddress & IPAddress::operator= (uint32_t address)

This operator overloading function is used to overloading '=' operator. More...

String **IPAddress::toString** () const

This function is used to transform from **IPAddress** object to String object. More...

bool **IPAddress::fromString** (const char *address)

This function is used to create the object from a string buffer. More...

bool IPAddress::fromString (const String &address)

This function is used to create the object from a String object. More...

virtual size t IPAddress::printTo (Print &p) const

This virtual function is used to called by print/println function. More...

Detailed Description

Function Documentation

fromString() [1/2]

bool IPAddress::fromString (const char * address)

This function is used to create the object from a string buffer.

Parameters

[in] address Specify the string buffer.

Returns

bool

fromString() [2/2]

bool IPAddress::fromString (const String & address)

This function is used to create the object from a String object.

Parameters

[in] address Specify the String object.

Returns

bool

◆ IPAddress() [1/4]

IPAddress::IPAddress()

This constructor function is used to construct **IPAddress** object.

Parameters

[in] None

Returns

None

◆ IPAddress() [2/4]

This constructor function is used to construct **IPAddress** object.

Parameters

Returns

None

◆ IPAddress() [3/4]

IPAddress::IPAddress (uint32_t address)

This constructor function is used to construct **IPAddress** object.

Parameters

[in] address Specify the address - uint32_t

Returns

None

◆ IPAddress() [4/4]

IPAddress::IPAddress (const uint8_t * address)

This constructor function is used to construct **IPAddress** object.

Parameters

[in] address Specify the address - uint8_t *

Returns

None

operator uint32_t()

IPAddress::operator uint32_t() const

inline

This operator overloading function is used to overloading 'uint32_t' operator.

Parameters

[in] None

Returns

The value of the **IPAddress** object in uint32_t type.

• operator=() [1/2]

IPAddress & IPAddress::operator= (const uint8_t * address)

This operator overloading function is used to overloading '=' operator.

Parameters

[in] address Sepcify the address - uint8_t *

Returns

the target IPAddress object.

• operator=() [2/2]

IPAddress & IPAddress::operator= (uint32_t address)

This operator overloading function is used to overloading '=' operator.

Parameters

[in] address Sepcify the address - uint32_t

Returns

the target IPAddress object.

• operator==() [1/3]

bool IPAddress::operator== (const IPAddress & addr) const



This operator overloading function is used to overloading '==' operator.

Parameters

[in] addr Sepcify the address - IPAddress

Returns

bool

• operator==() [2/3]

bool IPAddress::operator== (uint32_t addr) const

inline

This operator overloading function is used to overloading '==' operator.

Parameters

[in] addr Sepcify the address - uint32_t

Returns

bool

• operator==() [3/3]

bool IPAddress::operator== (const uint8_t * addr) const

This operator overloading function is used to overloading '==' operator.

Parameters

[in] addr Sepcify the address - uint8_t*

Returns

If they are equal, true is return, otherwise, false is return.

operator[]() [1/2]

uint8_t IPAddress::operator[] (int index) const

inline

This operator overloading function is used to overloading '[]' operator.

Parameters

[in] index Sepcify the address value of index.

Returns

the target value of **IPAddress** object.

operator[]() [2/2]

uint8_t& IPAddress::operator[] (int index)

inline

This operator overloading function is used to overloading '[]' operator.

Parameters

[in] index Sepcify the address value of index.

Returns

the target value of **IPAddress** object.

printTo()

size_t IPAddress::printTo (Print & p) const

virtual

This virtual function is used to called by print/println function.

Parameters

[in] p Specify the Print object.

Returns

bool

Note

The length of print successfully.

Implements Printable.

toString()

String IPAddress::toString () const

This function is used to transform from **IPAddress** object to String object.

Parameters

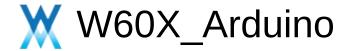
[in] None

Returns

the target String object.

Note

Generated by doxygen 1.8.14



Wire_Library

Functions

Wire Library. More...

Functions

```
void TwoWire::begin (int sda, int scl)
  void TwoWire::pins (int sda, int scl)
          attribute ((deprecated))
  void TwoWire::begin ()
  void TwoWire::begin (uint8 t)
  void TwoWire::begin (int)
  void TwoWire::setClock (uint32 t)
  void TwoWire::setClockStretchLimit (uint32 t)
  void TwoWire::beginTransmission (uint8 t)
  void TwoWire::beginTransmission (int)
uint8 t TwoWire::endTransmission (void)
uint8 t TwoWire::endTransmission (uint8 t)
size t TwoWire::requestFrom (uint8 t address, size t size,
       bool sendStop)
uint8 t TwoWire::status ()
uint8 t TwoWire::requestFrom (uint8 t, uint8 t)
uint8 t TwoWire::requestFrom (uint8 t, uint8 t, uint8 t)
uint8 t TwoWire::requestFrom (int, int)
```

uint8_t TwoWire::requestFrom (int, int, int)

virtual size t TwoWire::write (uint8 t)

This pure virtual function is used to define the operation that writes binary data. More...

virtual size_t TwoWire::write (const uint8_t *, size_t)

This function is used to write buffer to the interface defined by the object. More...

virtual int **TwoWire::available** (void)

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...

virtual int **TwoWire::read** (void)

read() reads characters from an incoming stream to the buffer. More...

virtual int **TwoWire::peek** (void)

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void TwoWire::flush (void)

void TwoWire::onReceive (void(*)(int))

void TwoWire::onRequest (void(*)(void))

size_t TwoWire::write (unsigned long n)

size_t TwoWire::write (long n)

size_t TwoWire::write (unsigned int n)

size t TwoWire::write (int n)

Detailed Description

Wire Library.

Function Documentation

available()

int TwoWire::available (void)

virtual

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

stream: an instance of a class that inherits from Stream

Note

peek()

int TwoWire::peek (void)

virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

None

Note

read()

int TwoWire::read (void)

virtual

read() reads characters from an incoming stream to the buffer.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

the first byte of incoming data available (or -1 if no data is available)

Note

write() [1/2]

size_t TwoWire::write (uint8_t)

virtual

This pure virtual function is used to define the operation that writes binary data.

Parameters

[in] val a value to send as a single byte

Returns

The length of write successfully (1 byte).

Note

Implements Print.

write() [2/2]

This function is used to write buffer to the interface defined by the object.

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully.

Note

Parameters

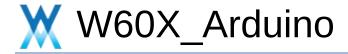
```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully (1 byte).

Note

Reimplemented from **Print**.



10

Functions

Digital Analog and Advanced I/O. More...

Functions

void tone (uint8 t pin, unsigned int frequency)

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. More...

void noTone (uint8 t pin)

Stops the generation of a square wave triggered by **tone()**. More...

void pinMode (uint32_t ulPin, uint32_t ulMode)

void digitalWrite (uint32 t ulPin, uint32 t ulVal)

int digitalRead (uint32_t ulPin)

int analogRead (unsigned char pin)

Reads the value from the specified analog pin. More...

Detailed Description

Digital Analog and Advanced I/O.

Parameters

ulPin The number of the pin whose mode you wish to set **ulMode** Either INPUT or OUTPUT

Function Documentation

analogRead()

int analogRead (unsigned char pin)

Reads the value from the specified analog pin.

Parameters

pin the number of the analog input pin to read from

Return values

average int (8192 -> 16384, look up for example to learn how to change it to voltage)

noTone()

```
void noTone ( uint8_t _pin )
```

Stops the generation of a square wave triggered by tone().

Parameters

[in] _pin gpio pin

Returns

none

Note

none

tone()

```
void tone ( uint8_t __pin,
unsigned int frequency
)
```

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin.

Parameters

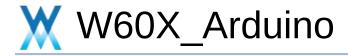
```
[in]_pin gpio pin
[in] frequency the frequency of the tone in hertz(1 to 156250)
```

Returns

none

Note

none



Serial

Functions

Serial. More...

Functions

HardwareSerial::HardwareSerial (int serial no)

This constructor is used to init hardware serial. More...

HardwareSerial::HardwareSerial (int serial_no, bool mul flag)

This constructor is used to init hardware serial. More...

HardwareSerial::HardwareSerial ()

This constructor is used to init hardware serial. More...

void HardwareSerial::begin ()

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. More...

void HardwareSerial::begin (unsigned long baud)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. More...

void **HardwareSerial::begin** (unsigned long baud, int modeChoose)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins

0 and 1 with a component that requires a particular baud rate. More...

virtual int HardwareSerial::read (void)

Reads incoming serial data. **read()** inherits from the **Stream** utility class. More...

virtual int HardwareSerial::available (void)

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). available() inherits from the **Stream** utility class. More...

virtual int HardwareSerial::peek ()

Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to **peek()** will return the same character, as will the next call to **read()**. **peek()** inherits from the **Stream** utility class. More...

Detailed Description

Serial.

Function Documentation

available()

int HardwareSerial::available (void)

virtual

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). **available()** inherits from the **Stream** utility class.

Parameters

[in] none

Returns

the number of bytes available to read

Note

begin() [1/3]

void HardwareSerial::begin (void)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

Parameters

[in] None

Returns

nothing

begin() [2/3]

void HardwareSerial::begin (unsigned long baud)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

Parameters

[in] baud speed: in bits per second (baud) - long

Returns

nothing

◆ begin() [3/3]

```
void HardwareSerial::begin ( unsigned long baud, int modeChoose )
```

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

Parameters

[in] baud speed: in bits per second (baud) - long [in] modeChoose Specify the mode.

Returns

nothing

◆ HardwareSerial() [1/3]

HardwareSerial::HardwareSerial (int serial_no)

This constructor is used to init hardware serial.

Parameters

[in] serial_no Specify serial_no

Returns

None

◆ HardwareSerial() [2/3]

```
HardwareSerial::HardwareSerial ( int serial_no, bool mul_flag )
```

This constructor is used to init hardware serial.

Parameters

```
[in] serial_no Specify serial_no
[in] mul_flag Specify mul_flag
```

Returns

None

◆ HardwareSerial() [3/3]

HardwareSerial::HardwareSerial()

inline

This constructor is used to init hardware serial.

Parameters

[in] none

Returns

None

peek()

int HardwareSerial::peek (void)

virtual

Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to **peek()** will return the same character, as will the next call to **read()**. **peek()** inherits from the **Stream** utility class.

Parameters

[in] None

Returns

the first byte of incoming serial data available (or -1 if no data is available) - int

Note

Implements **Stream**.

read()

int HardwareSerial::read (void)

virtual

Reads incoming serial data. **read()** inherits from the **Stream** utility class.

Parameters

[in] None

Returns

the first byte of incoming serial data available (or -1 if no data is available) - int

Note

Implements **Stream**.

Generated by doxygen 1.8.14



Stream

Functions

Stream. More...

Functions

virtual int Stream::available ()=0

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...

virtual int Stream::read ()=0

read() reads characters from an incoming stream to the buffer. More...

virtual int Stream::peek ()=0

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

void Stream::setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool Stream::find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool Stream::find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **Stream::find** (const char *target, size t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **Stream::find** (const uint8_t *target, size_t length) **find()** reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool Stream::find (char target) find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

- bool Stream::findUntil (const char *target, const char *terminator)
 findUntil() reads data from the stream until the target string of given length or terminator string is found. More...
- bool Stream::findUntil (const uint8_t *target, const char *terminator)
 findUntil() reads data from the stream until the target string of given length or terminator string is found. More...
- bool Stream::findUntil (const char *target, size_t targetLen, const char *terminate, size_t termLen) findUntil() reads data from the stream until the target string of given length or terminator string is found. More...
- bool Stream::findUntil (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) findUntil() reads data from the stream until the target string of given length or terminator string is found. More...
- long Stream::parseInt ()
 parseInt() returns the first valid (long) integer number
 from the serial buffer. Characters that are not integers (or
 the minus sign) are skipped. More...

float Stream::parseFloat ()

parseFloat() returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number. More...

- virtual size_t Stream::readBytes (char *buffer, size_t length)
 readBytes() read characters from a stream into a buffer.
 The function terminates if the determined length has been read, or it times out (see setTimeout()). More...
- virtual size_t Stream::readBytes (uint8_t *buffer, size_t length)
 readBytes() read characters from a stream into a buffer.
 The function terminates if the determined length has been read, or it times out (see setTimeout()). More...
 - size_t Stream::readBytesUntil (char terminator, char *buffer, size_t length)
 readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...
 - size_t Stream::readBytesUntil (char terminator, uint8_t *buffer, size_t length)
 readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...
 - String Stream::readString ()
 readString() reads characters from a stream into a
 string. The function terminates if it times out (see
 setTimeout()). More...
 - String Stream::readStringUntil (char terminator)
 readStringUntil() reads characters from a stream into a
 string. The function terminates if the terminator character

is detected or it times out (see **setTimeout()**). More...

Detailed Description

Stream.

Function Documentation

available()

virtual int Stream::available ()

pure virtual

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

stream: an instance of a class that inherits from Stream

Note

Implemented in WiFiUDP, WiFiClient, HardwareSerial, UDP, Client, and TwoWire.

• find() [1/5]

bool Stream::find (const char * target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] target: the string to search for (char)

Returns

boolean

• find() [2/5]

bool Stream::find (uint8_t * target)

inline

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] target: the string to search for (char)

Returns

boolean

◆ find() [3/5]

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
[in] length : Specify the length.
```

Returns

boolean

• find() [4/5]

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
[in] length : Specify the length.
```

Returns

boolean

find() [5/5]

bool Stream::find (char target)

inline

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] target: the string to search for (char)

Returns

boolean

• findUntil() [1/4]

```
bool Stream::findUntil ( const char * target,
const char * terminator
)
```

findUntil() reads data from the stream until the target string of given length or terminator string is found.

The function returns true if target string is found, false if timed out

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
[in] terminal : the terminal string in the search (char)
```

Returns

boolean

• findUntil() [2/4]

findUntil() reads data from the stream until the target string of given length or terminator string is found.

The function returns true if target string is found, false if timed out

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
[in] terminal : the terminal string in the search (char)
```

Returns

boolean

• findUntil() [3/4]

findUntil() reads data from the stream until the target string of given length or terminator string is found.

The function returns true if target string is found, false if timed out

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
```

[in] terminal: the terminal string in the search (char)

[in] terminate : Specify the terminate
[in] termLen : Specify the termLen

Returns

boolean

• findUntil() [4/4]

findUntil() reads data from the stream until the target string of given length or terminator string is found.

The function returns true if target string is found, false if timed out

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] target : the string to search for (char)
```

[in] terminal : the terminal string in the search (char)

[in] terminate : Specify the terminate
[in] termLen : Specify the termLen

Returns

boolean

parseFloat()

float Stream::parseFloat()

parseFloat() returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

float

parseInt()

long Stream::parseInt()

parseInt() returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped.

In particular:

- -- Initial characters that are not digits or a minus sign, are skipped;
- -- Parsing stops when no characters have been read for a configurable time-out value, or a non-digit is read;
- -- If no valid digits were read when the time-out
 (see Stream.setTimeout()) occurs, 0 is returned;

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

long

peek()

virtual int Stream::peek ()

pure virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

None

Note

Implemented in WiFiUDP, WiFiClient, UDP, HardwareSerial, Client, and TwoWire.

read()

virtual int Stream::read()

pure virtual

read() reads characters from an incoming stream to the buffer.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] None

Returns

the first byte of incoming data available (or -1 if no data is available)

Note

Implemented in WiFiUDP, WiFiClient, UDP, HardwareSerial, Client, and TwoWire.

readBytes() [1/2]

readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see **setTimeout()**).

readBytes() returns the number of bytes placed in the buffer. A 0 means no valid data was found.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] buffer the buffer to store the bytes in (char[] or byte[])
[in]
```

readBytes() [2/2]

readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see **setTimeout()**).

readBytes() returns the number of bytes placed in the buffer. A 0 means no valid data was found.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] buffer the buffer to store the bytes in (char[] or byte[])
[in]
```

readBytesUntil() [1/2]

```
size_t Stream::readBytesUntil ( char terminator, char * buffer, size_t length )
```

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**).

readBytesUntil() returns the number of bytes placed in the buffer. A 0 means no valid data was found.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

Returns

The number of bytes placed in the buffer

readBytesUntil() [2/2]

```
size_t Stream::readBytesUntil ( char terminator, uint8_t * buffer, size_t length ) inline
```

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**).

readBytesUntil() returns the number of bytes placed in the buffer. A 0 means no valid data was found.

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

Returns

The number of bytes placed in the buffer

readString()

String Stream::readString()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**).

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

[in] none

Returns

A string read from a stream

readStringUntil()

String Stream::readStringUntil (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see **setTimeout()**).

This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

```
[in] terminator: the character to search for (char)
[out]
```

setTimeout()

void Stream::setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information.

Parameters

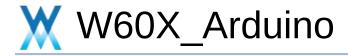
[in] timeout: timeout duration in milliseconds (long).

Returns

None

Note

Generated by doxygen 1.8.14



Math

Functions

Math. More...

Functions

unsigned int millis (void)

This function is used to get milliseconds since system startup. More...

uint32 t micros (void)

Returns the number of microseconds since the Arduino board began running the current program. More...

void **delay** (unsigned long ms)

This function is used to delay by milliseconds. More...

void **delayMicroseconds** (unsigned int us)

This function is to delay by micro seconds. More...

Detailed Description

Math.

Function Documentation

delay()

void delay (unsigned long ms)

This function is used to delay by milliseconds.

Parameters

[in] ms delay time, unit: ms

Returns

none

Note

none

delayMicroseconds()

void delayMicroseconds (unsigned int us)

This function is to delay by micro seconds.

Parameters

[in] us micro seconds

Returns

none

Note

none

micros()

uint32_t micros (void)

Returns the number of microseconds since the Arduino board began running the current program.

Returns

microseconds

Note

none

millis()

unsigned int millis (void)

This function is used to get milliseconds since system startup.

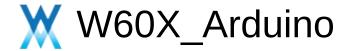
Returns

milliseconds

Note

none

Generated by doxygen 1.8.14



Data Structures

Here are the data structures with brief descriptions:

BufferDataSource	
BufferedStreamDataSource	
Client	
ClientContext	
CloudClass	
DataSource	
O DhcpClass	
O DNSClient	
O DNSServer	
HardwareSerial	
IPAddress	
Print	
Printable	
ProgmemStream	
G Server	
© SList	
© SPIClass	
© SPISettings	
G Stream	
TwoWire	
@ UDP	
UdpContext	
™ W600InnerFlashClass	
@ WiFiAPClass	

WiFiClass	
WiFiClient	
WiFiGenericClass	
WiFiOneshotClass	
WiFiScanClass	
WiFiServer	
WiFiSTAClass	
@ WiFiUDP	

Generated by doxygen 1.8.14

BufferDataSource Class Reference

Public Member Functions | Protected Attributes

Inherits **DataSource**.

Public Member Functions

BufferDataSource (const uint8_t *data, size_t size)

size_t available () override

const uint8_t * get_buffer (size_t size) override

void release_buffer (const uint8_t *buffer, size_t size)
 override

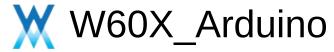
Protected Attributes

const uint8_t * _data

const size_t _size

size_t _**pos** = 0

Generated by doxygen 1.8.14



Public Member Functions | Protected Attributes

BufferedStreamDataSource< TStream > Class Template Reference

Inherits **DataSource**.

Public Member Functions

BufferedStreamDataSource (TStream &stream, size_t size)

size_t available () override

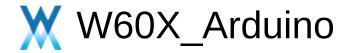
const uint8_t * get_buffer (size_t size) override

void release_buffer (const uint8_t *buffer, size_t size)
 override

Protected Attributes

TStream &	_stream
std::unique_ptr< uint8_t[]>	_buffer
size_t	_size
size_t	_pos = 0
size_t	_bufferSize = 0
size t	_streamPos = 0
	_

Generated by @@XYQ@M 1.8.14



Public Member Functions

Client Class Reference abstract

Inherits **Stream**.

Inherited by WiFiClient.

Public Member Functions

virtual int **connect** (**IPAddress** ip, uint16_t port)=0

This function is used to connect to the IP address and port specified in the constructor. More...

virtual int connect (const char *host, uint16 t port)=0

This function is used to connect to the IP address and port specified in the constructor. More...

virtual size_t write (uint8_t)=0

This function is used to write data to the server the client is connected to. More...

virtual size_t write (const uint8_t *buf, size_t size)=0

This function is used to write data to the server the client is connected to. More...

virtual int available ()=0

Returns the number of bytes available for reading (That is, the amount of data that has been written to the client by the server it is connected to). More...

virtual int read ()=0

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int **read** (uint8_t *buf, size_t size)=0

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int peek ()=0

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void **flush** ()=0

Discard any bytes that have been written to the client but not yet read. More...

virtual void stop ()=0

This function is used to disconnect from the server. More...

virtual uint8_t connected ()=0

Whether or not the client is connected. More...

virtual **operator bool** ()=0

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8 t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)
find() reads data from the stream until the target string
of given length is found The function returns true if target
string is found, false if timed out. More...

bool find (char target) find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **findUntil** (const char *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found.

More...

bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found.

More...

bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found.

More...

long parseInt ()
parseInt() returns the first valid (long) integer number
from the serial buffer. Characters that are not integers
(or the minus sign) are skipped. More...

float parseFloat ()

parseFloat() returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

- virtual size_t readBytes (uint8_t *buffer, size_t length)
 readBytes() read characters from a stream into a buffer.
 The function terminates if the determined length has been read, or it times out (see setTimeout()). More...
 - size_t readBytesUntil (char terminator, char *buffer, size_t length)
 readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...
 - size_t readBytesUntil (char terminator, uint8_t *buffer, size_t length)
 readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...
 - String readString() reads characters from a stream into a string. The function terminates if it times out (see setTimeout()). More...
 - String readStringUntil (char terminator) readStringUntil() reads characters from a stream into a string. The function terminates if the terminator

character is detected or it times out (see **setTimeout()**). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t print (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (int, int=DEC)

This function is used to print target to the interface

defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t **println** (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

Additional Inherited Members

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

Protected Member Functions inherited from Print

- Protected Member Functions inherited from Print void setWriteError (int err=1)
- Protected Attributes inherited from Stream unsigned long _timeout

unsigned long _startMillis

Member Function Documentation

available()

virtual int Client::available ()

pure virtual

Returns the number of bytes available for reading (That is, the amount of data that has been written to the client by the server it is connected to).

Parameters

[in] None
[out] None

Return values

The number of bytes available

Note

available() inherits from the Stream utility class.

Implements Stream.

connect() [1/2]

```
virtual int Client::connect ( IPAddress ip, uint16_t port ) pure virtual
```

This function is used to connect to the IP address and port specified in the constructor.

Parameters

```
[in] ip the IP address that the client will connect to (array of 4 bytes)
[in] port the port that the client will connect to (int)
[out] None
```

Return values

1 the connection succeeds

0 the connection failed

Note

connect() [2/2]

This function is used to connect to the IP address and port specified in the constructor.

Parameters

Return values

1 the connection succeeds0 the connection failed

Note

connected()

```
virtual uint8_t Client::connected()
```

pure virtual

Whether or not the client is connected.

Parameters

[in] None
[out] None

Return values

1 the client is connected

0 the client is disconnected

Note

that a client is considered connected if the connection has been closed but there is still unread data.

flush()

virtual void Client::flush ()

pure virtual

Discard any bytes that have been written to the client but not yet read.

Parameters

[in] None
[out] None

Returns

None

Note

flush() inherits from the Stream utility class.

peek()

virtual int Client::peek()

pure virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

Parameters

[in] None
[out] None

Return values

none is availableother the next byte or character

Note

This function inherited from the **Stream** class. See the **Stream** class main page for more information.

Implements **Stream**.

read() [1/2]

virtual int Client::read ()

pure virtual

Read the next byte received from the server the client is connected to (after the last call to read()).

Parameters

[in] None
[out] None

Return values

none is available.other The next character

Note

read() inherits from the Stream utility class

Implements **Stream**.

read() [2/2]

Read the next byte received from the server the client is connected to (after the last call to read()).

Parameters

```
[in] buf the byte to read
[in] size the size of the buf
[out] None
```

Return values

none is available.other The next byte

Note

read() inherits from the Stream utility class

stop()

virtual void Client::stop()

pure virtual

This function is used to disconnect from the server.

Parameters

[in] None
[out] None

Returns

None

Note

write() [1/2]

```
virtual size_t Client::write ( uint8_t )
```

pure virtual

This function is used to write data to the server the client is connected to.

Parameters

```
[in] the char to write
[out] None
```

Return values

the number of characters written. it is not necessary to read this value.

Note

Implements Print.

write() [2/2]

```
virtual size_t Client::write ( const uint8_t * buf, size_t size ) pure virtual
```

This function is used to write data to the server the client is connected to.

Parameters

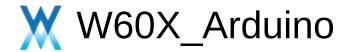
```
[in] buf the byte to write
[in] size the size of the buf
[out] None
```

Return values

the number of characters written. it is not necessary to read this value.

Note

Reimplemented from Print.



ClientContext Class Reference

Public Member Functions |
Protected Member Functions |
Static Protected Member Functions

Public Member Functions

```
ClientContext (tcp pcb *pcb, discard cb t discard cb
                void *discard cb arg)
          err t abort ()
          err t close ()
ClientContext * next () const
ClientContext * next (ClientContext *new next)
           void ref()
           void unref ()
             int connect (ip addr t *addr, uint16 t port)
         size t availableForWrite ()
           void setNoDelay (bool nodelay)
           bool getNoDelay ()
           void setTimeout (int timeout ms)
             int getTimeout ()
       uint32 t getRemoteAddress ()
       uint16_t getRemotePort ()
       uint32 t getLocalAddress ()
       uint16 t getLocalPort ()
```

```
size t getSize () const
   char read ()
  size t read (char *dst, size t size)
   char peek ()
  size t peekBytes (char *dst, size t size)
   void discard received ()
   void wait until sent ()
 uint8 t state () const
  size t write (const uint8 t *data, size t size)
  size t write (Stream &stream)
  size t write_P (PGM P buf, size t size)
   void keepAlive (uint16 t
        idle sec=TCP_DEFAULT_KEEPALIVE_IDLE_SEC,
        uint16 t
        intv sec=TCP DEFAULT KEEPALIVE INTERVAL SE
        uint8 t count=TCP DEFAULT KEEPALIVE COUNT)
   bool isKeepAliveEnabled () const
uint16 t getKeepAliveldle () const
uint16_t getKeepAliveInterval () const
 uint8 t getKeepAliveCount () const
```

Protected Member Functions

```
bool _is_timeout ()

void _notify_error ()

size_t _write_from_source (DataSource *ds)

bool _write_some ()

void _write_some_from_cb ()

err_t _acked (tcp_pcb *pcb, uint16_t len)

void _consume (size_t size)

err_t _recv (tcp_pcb *pcb, pbuf *pb, err_t err)

void _error (err_t err)

err_t _connected (struct tcp_pcb *pcb, err_t err)

err_t _poll (tcp_pcb *)
```

Static Protected Member Functions

```
static err_t _s_recv (void *arg, struct tcp_pcb *tpcb, struct pbuf *pb, err_t err)

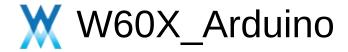
static void _s_error (void *arg, err_t err)

static err_t _s_poll (void *arg, struct tcp_pcb *tpcb)

static err_t _s_acked (void *arg, struct tcp_pcb *tpcb, uint16_t len)
```

static err t _s_connected (void *arg, struct tcp pcb *pcb, err t err)

Generated by doxygen 1.8.14



CloudClass Class Reference

Public Member Functions

CloudClass ()

This function is used to Init Giz Cloud Construct function to init some arguements or some configuration before use the object. More...

bool CloudInit ()
This function is used to Init Giz Cloud method. More...

Constructor & Destructor Documentation

◆ CloudClass()

CloudClass::CloudClass()

This function is used to Init Giz Cloud Construct function to init some arguements or some configuration before use the object.

Returns

None

Note

Member Function Documentation

CloudInit()

bool CloudClass::CloudInit()

This function is used to Init Giz Cloud method.

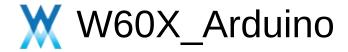
Returns

If the init process is successful, true is returned.

Note

This function must be called BEFORE using the object.

Generated by @@XYG@n 1.8.14



DataSource Class Reference abstract

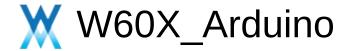
Inherited by **BufferDataSource**, and **BufferedStreamDataSource**< TStream >.

virtual size_t available ()=0

virtual const uint8_t * get_buffer (size_t size)=0

virtual void **release_buffer** (const uint8_t *buffer, size_t size)=0

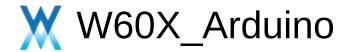
Generated by doxygen 1.8.14



DhcpClass Class Reference

Public Member Functions

```
uint32 t getLocalIP ()
uint32 t getSubnetMask ()
uint32_t getGatewaylp ()
uint32 t getDhcpServerIp ()
uint32 t getDnsServerIp ()
 char * getLocalIPStr ()
 char * getSubnetMaskStr ()
 char * getGatewaylpStr ()
 char * getDhcpServerlpStr ()
 char * getDnsServerlpStr ()
     int beginWithDHCP (uint8_t *, unsigned long timeout=60000,
        unsigned long responseTimeout=4000)
     int checkLease ()
```



DNSClient Class Reference

Public Member Functions | Protected Member Functions

void **begin** (unsigned int aDNSServer)

void begin (const char *aDNSServer)

int **getHostByName** (const char *aHostname, unsigned int &aResult) This function is used to resolve the hostname . More...

int **getHostByName** (const char *aHostname, char *&aResult) This function is used to resolve the hostname. More...

Protected Member Functions

uint16_t BuildRequest (const char *aName)

uint16_t **ProcessResponse** (uint16_t aTimeout, uint32_t &aAddress)

Member Function Documentation

getHostByName() [1/2]

```
int DNSClient::getHostByName ( const char * aHostname, unsigned int & aResult )
```

This function is used to resolve the hostname.

This function is used to resolve the hostname.

Parameters

```
[in] aHostname The hostname which user want to resolve.

[out] aResult The first IPv4 address which resolve via DNS protocol that display as a 32-bits value.
```

Returns

If the operation executes success, true is returned, otherwise, false is returned.

Note

getHostByName() [2/2]

```
int DNSClient::getHostByName ( const char * aHostname, char *& aResult )
```

This function is used to resolve the hostname.

Parameters

[in] **aHostname** The hostname which user want to resolve.

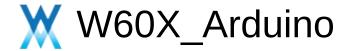
[out] **aResult** The first IPv4 address which resolve via DNS protocol that display as a Dotted Decimal Notation string.

Returns

If the operation executes success, true is returned, otherwise, false is returned.

Note

Generated by doxyden 1.8.14



DNSServer Class Reference

Public Member Functions

This function is used to start DNS Server. More...

bool **start** (const char *domainName)

This function is used to start DNS **Server** with some default arguements in port and resolvedIP. More...

void stop ()

This function is used to Stop DNS Server. More...

Member Function Documentation

start() [1/2]

This function is used to start DNS Server.

Parameters

```
[in] port Specify the server's dns port [in] domainName Sepcify the server's dns name [in] resolvedIP Unused arguement.
```

Returns

If the operation executes success, true is returned, otherwise, false is returned.

Note

start() [2/2]

bool DNSServer::start (const char * domainName)

This function is used to start DNS **Server** with some default arguements in port and resolvedIP.

Parameters

[in] domainName specify the server's dns name

Returns

If the operation executes success, true is returned, otherwise, false is returned.

Note

stop()

void DNSServer::stop ()

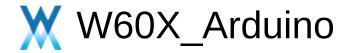
This function is used to Stop DNS Server.

Returns

None

Note

Generated by doxygen 1.8.14



HardwareSerial Class Reference

Public Member Functions

Inherits **Stream**.

HardwareSerial (int serial no)

This constructor is used to init hardware serial. More...

HardwareSerial (int serial no, bool mul flag)

This constructor is used to init hardware serial. More...

HardwareSerial ()

This constructor is used to init hardware serial. More...

void begin ()

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. More...

void **begin** (unsigned long baud)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. More...

void begin (unsigned long baud, int modeChoose)

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. More...

virtual int **read** (void)

Reads incoming serial data. **read()** inherits from the **Stream** utility class. More...

virtual int available (void)

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). **available()** inherits from the **Stream** utility class. More...

virtual int peek ()

Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to **peek()** will return the same character, as will the next call to **read()**. **peek()** inherits from the **Stream** utility class. More...

virtual size_t write (uint8_t c)

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the **print()** function instead. More...

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool **find** (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (char target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **findUntil** (const char *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool findUntil (const char *target, size_t targetLen, const char *terminate, size_t termLen)
findUntil() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

long parseInt ()

parseInt() returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped. More...

float parseFloat ()

parseFloat() returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

virtual size_t readBytes (uint8_t *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

size_t **readBytesUntil** (char terminator, char *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

size_t **readBytesUntil** (char terminator, uint8_t *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

String readString ()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**). More...

String readStringUntil (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see **setTimeout()**). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

virtual size t write (const uint8 t *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t print (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size t **print** (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size t println (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size t **println** (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t **println** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

Additional Inherited Members

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

Protected Member Functions inherited from Print

- Protected Member Functions inherited from Print void setWriteError (int err=1)
- Protected Attributes inherited from Stream unsigned long _timeout

unsigned long _startMillis

Member Function Documentation

write()

size t HardwareSerial::write (uint8 t c)

virtual

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the **print()** function instead.

Parameters

[in] c Specify the byte which will be sent to the console.

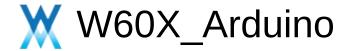
Returns

The length of sending to the console.

Note

Implements Print.

Generated by doxy/gen 1.8.14



IPAddress Class Reference **Public Member Functions**

Inherits **Printable**.

Public Member Functions

IPAddress ()

This constructor function is used to construct **IPAddress** object. More...

IPAddress (uint8_t first_oct, uint8_t sec_oct, uint8_t third_oct, uint8_t fourth_oct)

This constructor function is used to construct **IPAddress** object. More...

IPAddress (uint32_t address)

This constructor function is used to construct **IPAddress** object. More...

IPAddress (const uint8_t *address)

This constructor function is used to construct **IPAddress** object. More...

operator uint32_t () const

This operator overloading function is used to overloading 'uint32_t' operator. More...

bool operator== (const IPAddress &addr) const

This operator overloading function is used to overloading '==' operator. More...

bool **operator==** (uint32_t addr) const

This operator overloading function is used to overloading '==' operator. More...

bool operator== (const uint8_t *addr) const

This operator overloading function is used to overloading '==' operator. More...

uint8 t operator[] (int index) const

This operator overloading function is used to overloading '[]' operator. More...

uint8_t & operator[] (int index)

This operator overloading function is used to overloading '[]' operator. More...

IPAddress & **operator=** (const uint8 t *address)

This operator overloading function is used to overloading '=' operator. More...

IPAddress & operator= (uint32_t address)

This operator overloading function is used to overloading '=' operator. More...

String toString () const

This function is used to transform from **IPAddress** object to String object. More...

bool **fromString** (const char *address)

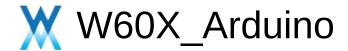
This function is used to create the object from a string buffer. More...

bool **fromString** (const String &address)

This function is used to create the object from a String object. More...

virtual size_t printTo (Print &p) const

This virtual function is used to called by print/println function. More...



Print Class Reference abstract

Public Member Functions |
Protected Member Functions

Inherited by Server, and Stream.

Public Member Functions

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

virtual size_t write (uint8_t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size t write (const uint8 t *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

Protected Member Functions

void setWriteError (int err=1)

Member Function Documentation

clearWriteError()

void Print::clearWriteError()

inline

This function is used to clear write error number.

Parameters

[in] None

Returns

None

getWriteError()

int Print::getWriteError()

inline

This function is used to get write error number.

Parameters

[in] None

Returns

The write error set by the object.

print() [1/10]

size_t Print::print(const String & s)

This function is used to print buffer to the interface defined by the object.

Parameters

[in] s Specify the String.

Returns

The length of print successfully.

print() [2/10]

size_t Print::print (const char str[])

This function is used to print buffer to the interface defined by the object.

Parameters

[in] s Specify the string buffer.

Returns

The length of print successfully.

print() [3/10]

size_t Print::print(char c)

This function is used to print buffer to the interface defined by the object.

Parameters

[in] c Specify the target - char.

Returns

The length of print successfully (1 byte).

print() [4/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] b Specify the target - char.[in] base Specify the base.
```

Returns

The length of print successfully.

print() [5/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] n Specify the target - int.[in] base Specify the base.
```

Returns

The length of print successfully.

print() [6/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] n Specify the target - unsigned int.[in] base Specify the base.
```

Returns

The length of print successfully.

print() [7/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] n Specify the target - long.[in] base Specify the base.
```

Returns

The length of print successfully.

print() [8/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] n Specify the target - unsigned long.[in] base Specify the base.
```

Returns

The length of print successfully.

print() [9/10]

This function is used to print target to the interface defined by the object.

Parameters

```
[in] n Specify the target - double.[in] digits Specify the digits.
```

Returns

The length of print successfully.

print() [10/10]

size_t Print::print (const Printable & x)

This function is used to print target to the interface defined by the object.

Parameters

[in] x Specify the target - Printable.

Returns

The length of print successfully.

println() [1/11]

```
size_t Print::println ( void )
```

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] None

Returns

The length of print successfully.

println() [2/11]

size_t Print::println (const String & s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] s Specify the String.

Returns

The length of print successfully.

println() [3/11]

size_t Print::println (const char c[])

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] c Specify the string buffer.

Returns

The length of print successfully.

println() [4/11]

size_t Print::println (char c)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] c Specify the target - char.

Returns

The length of print successfully.

println() [5/11]

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] n Specify the target - int.[in] base Specify the base.

Returns

The length of print successfully.

println() [6/11]

```
size_t Print::println ( int num,
int base = DEC
)
```

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] num Specify the target - int.[in] base Specify the base.

Returns

The length of print successfully.

println() [7/11]

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] num Specify the target - unsigned int.[in] base Specify the base.

Returns

The length of print successfully.

println() [8/11]

```
size_t Print::println ( long num,
int base = DEC
)
```

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

```
[in] num Specify the target - long.[in] base Specify the base.
```

Returns

The length of print successfully.

println() [9/11]

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

```
[in] num Specify the target - unsigned long.[in] base Specify the base.
```

Returns

The length of print successfully.

println() [10/11]

```
size_t Print::println ( double num,
int digits = BIN
)
```

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

```
[in] num Specify the target - double.[in] digits Specify the digits.
```

Returns

The length of print successfully.

println() [11/11]

size_t Print::println (const Printable & x)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' ').

Parameters

[in] x Specify the target - **Printable**.

Returns

The length of print successfully.

write() [1/4]

virtual size t Print::write (uint8 t)

pure virtual

This pure virtual function is used to define the operation that writes binary data.

Parameters

[in] val a value to send as a single byte

Returns

The length of write successfully (1 byte).

Note

Implemented in WiFiUDP, HardwareSerial, WiFiServer, WiFiClient, UDP, TwoWire, and Client.

write() [2/4]

This function is used to write buffer to the interface defined by the object.

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully.

Note

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully (1 byte).

Note

Reimplemented in WiFiUDP, WiFiClient, WiFiServer, UDP, Client, and TwoWire.

write() [3/4]

size_t Print::write (const char * str)

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] **str** Specify the buffer of string.

Returns

The length of write successfully.

Note

write() [4/4]

This function is used to write buffer to the interface defined by the object.

Parameters

[in] **buffer** Specify the buffer of string.

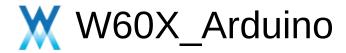
[in] size Specify the size.

Returns

The length of write successfully.

Note

Generated by doxyound 1.8.14



Printable Class Reference abstract

Inherited by **IPAddress**.

Member Function Documentation

printTo()

virtual size_t Printable::printTo (Print & p) const

pure virtual

This pure virtual function is used to called by print/println function.

Parameters

[in] p Specify the Print object.

Returns

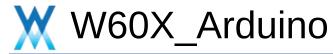
bool

Note

The length of print successfully.

Implemented in IPAddress.

Generated by doxygen 1.8.14



Public Member Functions | Protected Attributes

ProgmemStream Class Reference

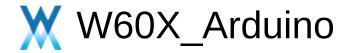
ProgmemStream (PGM_P buf, size_t size)

size_t readBytes (char *dst, size_t size)

Protected Attributes

PGM_P _buf
size_t _left

Generated by doxygen 1.8.14



Server Class Reference abstract

Inherits **Print**.

Inherited by WiFiServer.

virtual void begin ()=0

This pure-virtual function is used to start the object. More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

virtual size_t write (uint8_t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size t write (const uint8 t *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size t write (const char *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface

defined by the object with carriage ret (") and new line (' '). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

Additional Inherited Members

Protected Member Functions inherited from Print
void setWriteError (int err=1)

Member Function Documentation

begin()

virtual void Server::begin ()

pure virtual

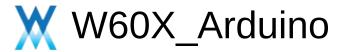
This pure-virtual function is used to start the object.

Parameters

[in] none
[out]

Implemented in WiFiServer.

Generated by doxygen 1.8.14



SList< T > Class Template Reference

Static Protected Member Functions | Protected Attributes | Static Protected Attributes

Static Protected Member Functions

static void _add (T *self)

static void _remove (T *self)

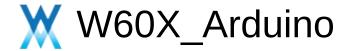
Protected Attributes

T * _next

Static Protected Attributes

static T * _s_first

Generated by doxygen 1.8.14



SPIClass Class Reference

Public Member Functions

void **begin** (void)

Initialize the SPI instance. More...

void end (void)

Deinitialize the SPI instance and stop it. More...

void **beginTransaction** (SPISettings settings)

This function should be used to configure the SPI instance in case you don't use the default parameters set by the **begin()** function. More...

void endTransaction (void)

settings associated to the SPI instance. More...

uint8_t transfer (uint8_t _data)

Transfer one byte on the SPI bus. More...

uint16_t transfer16 (uint16_t _data)

Transfer two bytes on the SPI bus in 16 bits format. More...

void transferWrite (void *_buf, size_t _count)

send several bytes. More...

void transferRead (void * buf, size t count)

receive several bytes. More...

void transfer (void *_buf, size_t _count)

Transfer several bytes. Only one buffer used to send and receive data. More...

void transfer (void *_bufout, void *_bufin, size_t _count)

Transfer several bytes. One buffer contains the data to send and another one will contains the data received. More...

void **setBitOrder** (BitOrder)

Deprecated function. Configure the bit order: MSB first or LSB first. More...

void setDataMode (uint8_t _mode)

Deprecated function. Configure the data mode (clock polarity and clock phase) More...

void setFrequency (uint32 t freq)

Configure the spi frequency. More...

Generated by doxygen 1.8.14



SPISettings Class Reference

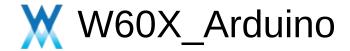
Public Member Functions | Friends

SPISettings (uint32_t clock, BitOrder bitOrder, uint8_t dataMode)

Friends

class **SPIClass**

Generated by @@XYQ@M 1.8.14



Stream Class Reference abstract

Public Member Functions |
Protected Member Functions |
Protected Attributes

Inherits **Print**.

Inherited by Client, HardwareSerial, TwoWire, and UDP.

virtual int available ()=0

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...

virtual int read ()=0

read() reads characters from an incoming stream to the buffer. More...

virtual int peek ()=0

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)
find() reads data from the stream until the target string of
given length is found The function returns true if target
string is found, false if timed out. More...

bool find (char target) find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

- bool **findUntil** (const char *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...
- bool **findUntil** (const uint8_t *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...
- bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...
- bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...
- long parseInt ()
 parseInt() returns the first valid (long) integer number
 from the serial buffer. Characters that are not integers (or
 the minus sign) are skipped. More...
- float parseFloat ()

 parseFloat() returns the first valid floating point number from the current position. Initial characters that are not

digits (or the minus sign) are skipped. **parseFloat()** is terminated by the first character that is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length)

readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see **setTimeout()**). More...

virtual size t readBytes (uint8 t *buffer, size t length)

readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see **setTimeout()**). More...

size_t readBytesUntil (char terminator, char *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

size_t **readBytesUntil** (char terminator, uint8_t *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

String readString ()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**). More...

String readStringUntil (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see **setTimeout()**). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

virtual size t write (uint8 t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size_t write (const uint8_t *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size t write (const char *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t print (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t print (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface

defined by the object. More...

size_t print (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (const Printable &)
This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

Protected Member Functions

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

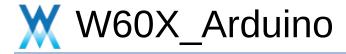
Protected Member Functions inherited from Print void setWriteError (int err=1)

Protected Attributes

unsigned long _timeout

unsigned long _startMillis

Generated by doxygen 1.8.14



TwoWire Class Reference

Public Member Functions

Inherits **Stream**.

Public Member Functions

```
void begin (int sda, int scl)
  void pins (int sda, int scl) attribute ((deprecated))
  void
        begin ()
  void begin (uint8 t)
  void begin (int)
  void setClock (uint32 t)
  void setClockStretchLimit (uint32 t)
  void beginTransmission (uint8 t)
  void beginTransmission (int)
uint8 t endTransmission (void)
uint8 t endTransmission (uint8 t)
size t requestFrom (uint8 t address, size t size, bool
        sendStop)
uint8 t status ()
uint8 t requestFrom (uint8 t, uint8 t)
uint8 t requestFrom (uint8 t, uint8 t, uint8 t)
uint8 t requestFrom (int, int)
uint8 t requestFrom (int, int, int)
```

virtual size_t write (uint8_t)

This pure virtual function is used to define the operation that writes binary data. More...

virtual size_t write (const uint8_t *, size_t)

This function is used to write buffer to the interface defined by the object. More...

virtual int available (void)

available() gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...

virtual int read (void)

read() reads characters from an incoming stream to the buffer. More...

virtual int **peek** (void)

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void flush (void)

void onReceive (void(*)(int))

void onRequest (void(*)(void))

size t write (unsigned long n)

size_t write (long n)

size_t write (unsigned int n)

size_t write (int n)

virtual size_t write (uint8_t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size_t write (const uint8_t *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target

string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)
find() reads data from the stream until the target string of
given length is found The function returns true if target
string is found, false if timed out. More...

bool find (char target)
find() reads data from the stream until the target string of
given length is found The function returns true if target
string is found, false if timed out. More...

bool **findUntil** (const char *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

long parseInt ()
parseInt() returns the first valid (long) integer number
from the serial buffer. Characters that are not integers (or
the minus sign) are skipped. More...

float parseFloat ()

parseFloat() returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has

The function terminates if the determined length has been read, or it times out (see **setTimeout()**). More...

virtual size_t readBytes (uint8_t *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

size_t **readBytesUntil** (char terminator, char *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

size_t **readBytesUntil** (char terminator, uint8_t *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

String readString ()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**). More...

String readStringUntil (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character

is detected or it times out (see **setTimeout()**). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size t print (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t print (const char [])

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size t **println** (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t **println** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

Additional Inherited Members

unsigned long _timeout

unsigned long _startMillis

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

Protected Member Functions inherited from Print
void setWriteError (int err=1)

Protected Attributes inherited from Stream

Member Function Documentation

write() [1/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] **str** Specify the buffer of string.

Returns

The length of write successfully.

Note

write() [2/4]

size t Print::write

This function is used to write buffer to the interface defined by the object.

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully.

Note

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully (1 byte).

Note

write() [3/4]

virtual size_t Print::write

This pure virtual function is used to define the operation that writes binary data.

Parameters

[in] val a value to send as a single byte

Returns

The length of write successfully (1 byte).

Note

write() [4/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] buffer Specify the buffer of string.

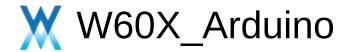
[in] size Specify the size.

Returns

The length of write successfully.

Note

Generated by doxygen 1.8.14



Public Member Functions

UDP Class Reference abstract

Inherits **Stream**.

Inherited by WiFiUDP.

Public Member Functions

virtual uint8_t begin (uint16_t)=0

This function is used to initializes the **UDP** library and network settings, Starts **UDP** socket, listening at local port. More...

virtual void stop ()=0

This function is used to disconnect from the server. Release any resource being used during the **UDP** session. More...

virtual int beginPacket (IPAddress ip, uint16_t port)=0

This function is used to starts a connection to write **UDP** data to the remote connection. More...

virtual int **beginPacket** (const char *host, uint16_t port)=0 This function is used to starts a connection to write

UDP data to the remote connection. More...

virtual int endPacket ()=0

This function is used to called after writing **UDP** data to the remote connection. It finishes off the packet and send it. More...

virtual size t write (uint8 t)=0

This function is used to writes **UDP** data to the remote connection. More...

virtual size_t write (const uint8_t *buffer, size_t size)=0

This function is used to writes **UDP** data to the remote connection. More...

virtual int parsePacket ()=0

It starts processing the next available incoming packet, checks for the presence of a **UDP** packet,

and reports the size. More...

virtual int available ()=0

Get the number of bytes (characters) available for reading from the buffer. This is is data that's already arrived. More...

virtual int read ()=0

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int read (unsigned char *buffer, size t len)=0

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int read (char *buffer, size_t len)=0

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int peek ()=0

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void flush ()=0

Discard any bytes that have been written to the client but not yet read. More...

virtual IPAddress remoteIP ()=0

This function is used to gets the IP address of the remote connection. More...

virtual uint16 t remotePort ()=0

This function is used to gets the port of the remote **UDP** connection. More...

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)
setTimeout() sets the maximum milliseconds to
wait for stream data, it defaults to 1000
milliseconds. This function is part of the Stream
class, and is called by any class that inherits from it
(Wire, Serial, etc). See the Stream class main page
for more information. More...

bool find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (uint8 t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out.

More...

bool find (const char *target, size t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (char target)

find() reads data from the stream until the target

string of given length is found The function returns true if target string is found, false if timed out.

More...

bool **findUntil** (const char *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found. More...

bool findUntil (const uint8_t *target, const char *terminator)
findUntil() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

long parseInt ()
parseInt() returns the first valid (long) integer
number from the serial buffer. Characters that are
not integers (or the minus sign) are skipped. More...

float parseFloat ()
parseFloat() returns the first valid floating point
number from the current position. Initial characters
that are not digits (or the minus sign) are skipped.
parseFloat() is terminated by the first character that
is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

virtual size_t readBytes (uint8_t *buffer, size_t length) readBytes() read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...

size_t readBytesUntil (char terminator, char *buffer, size_t length) readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...

size_t readBytesUntil (char terminator, uint8_t *buffer, size_t length)
readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...

String readString() reads characters from a stream into a string. The function terminates if it times out (see setTimeout()). More...

String readStringUntil (char terminator)
readStringUntil() reads characters from a stream
into a string. The function terminates if the
terminator character is detected or it times out (see

setTimeout()). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t print (const char [])

This function is used to print buffer to the interface defined by the object. More...

size t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size t println (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size_t println (const char [])

This function is used to print target to the interface

defined by the object with carriage ret (") and new line ('
'). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size t **println** (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size_t **println** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size_t println (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('
'). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new

line ('
'). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

Additional Inherited Members

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

Protected Member Functions inherited from Print

- Protected Member Functions inherited from Print void setWriteError (int err=1)
- Protected Attributes inherited from Stream unsigned long _timeout

unsigned long _startMillis

Member Function Documentation

available()

virtual int UDP::available ()

pure virtual

Get the number of bytes (characters) available for reading from the buffer. This is is data that's already arrived.

Parameters

[in] None
[out] None

Return values

parsePacket hasn't been called yetother the number of bytes available in the current packet

Note

This function can only be successfully called after **UDP.parsePacket()**. **available()** inherits from the **Stream** utility class.

Implements **Stream**.

begin()

```
virtual uint8_t UDP::begin ( uint16_t )
```

pure virtual

This function is used to initializes the **UDP** library and network settings, Starts **UDP** socket, listening at local port.

Parameters

```
[in] the local port to listen on
[out] None
```

Return values

1 successful

0 there are no sockets available to use

Note

beginPacket() [1/2]

```
virtual int UDP::beginPacket ( IPAddress ip, uint16_t port ) pure virtual
```

This function is used to starts a connection to write **UDP** data to the remote connection.

Parameters

```
[in] ip the IP address of the remote connection (4 bytes)
[in] port the port of the remote connection (int)
[out] None
```

Return values

1 successful

0 there was a problem with the supplied IP address or port

Note

beginPacket() [2/2]

```
virtual int UDP::beginPacket ( const char * host, uint16_t port ) pure virtual
```

This function is used to starts a connection to write **UDP** data to the remote connection.

Parameters

Return values

1 successful

0 there was a problem with the supplied IP address or port

Note

endPacket()

virtual int UDP::endPacket()

pure virtual

This function is used to called after writing **UDP** data to the remote connection. It finishes off the packet and send it.

Parameters

[in] None
[out] None

Return values

1 the packet was sent successfully

0 there was an error

Note

flush()

virtual void UDP::flush ()

pure virtual

Discard any bytes that have been written to the client but not yet read.

Parameters

[in] None
[out] None

Returns

None

Note

flush() inherits from the Stream utility class.

parsePacket()

virtual int UDP::parsePacket()

pure virtual

It starts processing the next available incoming packet, checks for the presence of a **UDP** packet, and reports the size.

Parameters

[in] None
[out] None

Return values

o no packets are availableother the size of the packet in bytes

Note

parsePacket() must be called before reading the buffer with UDP.read().

peek()

virtual int UDP::peek ()

pure virtual

Read a byte from the file without advancing to the next one. That is, successive calls to **peek()** will return the same value, as will the next call to **read()**.

Parameters

[in] None
[out] None

Return values

none is availableother the next byte or character

Note

This function inherited from the **Stream** class. See the **Stream** class main page for more information.

Implements **Stream**.

read() [1/3]

virtual int UDP::read ()

pure virtual

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

[in] None
[out] None

Return values

no buffer is availableother the characters in the buffer (char)

Note

Implements **Stream**.

read() [2/3]

```
virtual int UDP::read ( unsigned char * buffer, size_t len
) pure virtual
```

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

```
[in] buffer buffer to hold incoming packets (unsigned char*)
[in] len maximum size of the buffer (int)
[out] None
```

Return values

-1 no buffer is available other the size of the buffer

Note

◆ read() [3/3]

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

Parameters

```
[in] buffer buffer to hold incoming packets (char*)
[in] len maximum size of the buffer (int)
[out] None
```

Return values

-1 no buffer is available other the size of the buffer

Note

remoteIP()

virtual IPAddress UDP::remoteIP()

pure virtual

This function is used to gets the IP address of the remote connection.

Parameters

[in] None
[out] None

Return values

the IP address of the host who sent the current incoming packet(4 bytes)

Note

This function must be called after **UDP.parsePacket()**.

remotePort()

virtual uint16_t UDP::remotePort()

pure virtual

This function is used to gets the port of the remote **UDP** connection.

Parameters

[in] None
[out] None

Return values

The port of the host who sent the current incoming packet

Note

This function must be called after UDP.parsePacket().

stop()

virtual void UDP::stop ()

pure virtual

This function is used to disconnect from the server. Release any resource being used during the UDP session.

Parameters

[in] None
[out] None

Returns

None

Note

write() [1/2]

```
virtual size_t UDP::write ( uint8_t )
```

pure virtual

This function is used to writes **UDP** data to the remote connection.

Parameters

```
[in] the outgoing byte
[out] None
```

Return values

single byte into the packet

Note

Must be wrapped between **beginPacket()** and **endPacket()**. **beginPacket()** initializes the packet of data, it is not sent until **endPacket()** is called.

Implements Print.

write() [2/2]

```
virtual size_t UDP::write ( const uint8_t * buffer, size_t size ) pure virtual
```

This function is used to writes **UDP** data to the remote connection.

Parameters

```
[in] buffer the outgoing message
[in] size the size of the buffer
[out] None
```

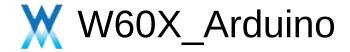
Return values

bytes size from buffer into the packet

Note

Must be wrapped between **beginPacket()** and **endPacket()**. **beginPacket()** initializes the packet of data, it is not sent until **endPacket()** is called.

Reimplemented from Print.



UdpContext Class Reference

Public Types | Public Member Functions

Public Types

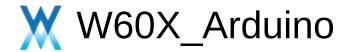
typedef std::function< void(void)> rxhandler_t

Public Member Functions

```
void ref()
   void unref ()
   bool connect (ip_addr_t addr, uint16_t port)
   bool listen (ip addr t addr, uint16 t port)
   void disconnect ()
   void setMulticastInterface (const ip addr t &addr)
   void setMulticastTTL (int ttl)
   void onRx (rxhandler t handler)
 size_t getSize () const
  size t tell () const
   void seek (const size t pos)
   bool isValidOffset (const size_t pos) const
uint32 t getRemoteAddress ()
uint16 t getRemotePort ()
uint32 t getDestAddress ()
uint16 t getLocalPort ()
   bool next ()
```

```
int read ()
size_t read (char *dst, size_t size)
int peek ()
void flush ()
size_t append (const char *data, size_t size)
bool send (ip_addr_t *addr=0, uint16_t port=0)
```

Generated by doxygen 1.8.14



W600InnerFlashClass Class Reference

Public Member Functions

Public Member Functions

int begin ()

This function is used to inialize inner flash driver. More...

bool flashEraseSector (uint32 t sector)

This function is used to erase flash by sectors. More...

bool flashWrite (uint32 t offset, uint8 t *data, size t size)

This function is used to write data to flash. More...

bool flashRead (uint32_t offset, uint8_t *data, size_t size)

This function is used to read data from flash. More...

Member Function Documentation

begin()

int W600InnerFlashClass::begin ()

This function is used to inialize inner flash driver.

Parameters

[in] None
[out] None

Returns

None

flashEraseSector()

bool W600InnerFlashClass::flashEraseSector (uint32_t sector)

This function is used to erase flash by sectors.

Parameters

```
[in] sector 240-251, other sector can not be erased for special usage
```

[out] None

Returns

None

Note

Only sectors from 240 to 251 can be used to user operation.

flashRead()

```
bool W600InnerFlashClass::flashRead ( uint32_t offset, uint8_t * data, size_t size )
```

This function is used to read data from flash.

Parameters

```
[in] offset range from 0x000000 to 0xFFFFF
        [in/out] data : data to be read from flash
[in] size : data length to be written to flash
[out] None
```

Returns

None

flashWrite()

```
bool W600InnerFlashClass::flashWrite ( uint32_t offset, uint8_t * data, size_t size )
```

This function is used to write data to flash.

Parameters

```
[in] offset range from 0xF0000 to 0xFBFFF
[in] data : data to be written to flash
[in] size : data length to be written to flash
[out] None
```

Returns

None

Note

Only area from 0xF0000 to 0xFBFFF can be written.

Parameters

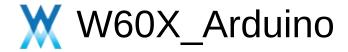
```
[in] offset :range from 0xF0000 to 0xFBFFF
[in] data : data to be written to flash
[in] size : data length to be written to flash
[out] None
```

Returns

None

Note

Only area from 0xF0000 to 0xFBFFF can be written.



Public Member Functions

WiFiAPClass Class Reference

Inherited by WiFiClass.

Public Member Functions

bool **softAPConfig** (uint32_t local_ip, uint32_t gateway, uint32_t subnet)
This function is used to config soft AP parameters. More...

bool **softAPConfig** (const char *local_ip, const char *gateway, const char *subnet)

This function is used to config soft AP parameters. More...

bool **softAPdisconnect** (bool wifioff=false)

This function is used to disconnect the ap function. More...

bool **softAPdestroy** ()
This function is used to destrory the AP function. More...

uint8_t softAPgetStationNum ()
This function is used to get the number of station. More...

char * softAPIP ()
This function is used to get AP's IP. More...

uint8_t * softAPmacAddress (uint8_t *mac)
This function is used to get AP's MAC address. More...

char * softAPmacAddress (void)
This function is used to get the AP's MAC address. More...

char * softAPSSID () const This function is used to get AP's SSID. More...

char * softAPPSK () const This function is used to get AP's PSK. More...

Member Function Documentation

softAP()

This function is used to start wifi module as AP mode.

Parameters

[in] ssidSpecify the SSID.[in] passphraseSpecify the passphrase.[in] channelSpecify the channel.[in] ssid_hiddenSpecify the ssid_hidden flag.[in] max_connectionSpecify the max_connection.

Returns

true - operate successfully, false - operate failure.

softAPConfig() [1/2]

This function is used to config soft AP parameters.

Parameters

```
[in] local_ip Specify the local_ip.[in] gateway Specify the gateway.[in] subnet Specify the subnet.
```

Returns

true - operate successfully, false - operate failure.

softAPConfig() [2/2]

This function is used to config soft AP parameters.

Parameters

```
[in] local_ip Specify the local_ip.[in] gateway Specify the gateway.[in] subnet Specify the subnet.
```

Returns

true - operate successfully, false - operate failure.

softAPdestroy()

bool WiFiAPClass::softAPdestroy ()

This function is used to destrory the AP function.

Parameters

[in] None

Returns

true - operate successfully, false - operate failure.

softAPdisconnect()

bool WiFiAPClass::softAPdisconnect (bool wifioff = false)

This function is used to disconnect the ap function.

Parameters

[in] wifioff Specify the wifioff.

Returns

true - operate successfully, false - operate failure.

softAPgetStationNum()

uint8_t WiFiAPClass::softAPgetStationNum ()

This function is used to get the number of station.

Parameters

[in] None

Returns

The number of station.

◆ softAPIP()

char * WiFiAPClass::softAPIP()

This function is used to get AP's IP.

Parameters

[in] None

Returns

The local AP's IPv4 address.

softAPmacAddress() [1/2]

uint8_t * WiFiAPClass::softAPmacAddress (uint8_t * mac)

This function is used to get AP's MAC address.

Parameters

[in] mac Specify the mac buffer.

Returns

The AP's MAC address.

softAPmacAddress() [2/2]

char * WiFiAPClass::softAPmacAddress (void)

This function is used to get the AP's MAC address.

Parameters

[in] None

Returns

The string of the AP's MAC address.

◆ softAPPSK()

char * WiFiAPClass::softAPPSK () const

This function is used to get AP's PSK.

Parameters

[in] None

Returns

The AP's PSK is returned.

◆ softAPSSID()

char * WiFiAPClass::softAPSSID() const

This function is used to get AP's SSID.

Parameters

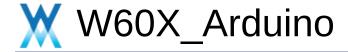
[in] None

Returns

The AP's SSID is returned.

Note

Generated by doxygen 1.8.14



Public Member Functions

WiFiClass Class Reference

Inherits WiFiGenericClass, WiFiSTAClass, WiFiScanClass, WiFiAPClass, and WiFiOneshotClass.

Public Member Functions

char * SSID () const

This function is used to get the SSID used by the module. More...

uint8 t * BSSID ()

This function is used to get the BSSID which is connected by the module. More...

char * BSSIDstr ()

This function is used to get the BSSID which is connected by the module. More...

int32_t RSSI()

This function is used to get the RSSI. More...

char * SSID (uint8 t networkItem)

This function is used to get ssid. More...

uint8_t * BSSID (uint8_t networkItem)

This function is used to get the bssid. More...

char * BSSIDstr (uint8_t networkItem)

This function is used to get the bssid. More...

int32 t RSSI (uint8 t networkItem)

This function is used to get RSSI. More...

int32_t channel (uint8_t networkItem)

This function is used to get the bssid. More...

▶ Public Member Functions inherited from WiFiGenericClass

WiFiMode_t getMode ()

This function is used to Get the WiFi's work mode...

More...

int **hostByName** (const char *aHostname, **IPAddress** &aResult)

This function is used to resolve hostname. More...

int hostByName (const char *aHostname, IPAddress &aResult, uint32 t timeout ms)

This function is used to resolve hostname. More...

Public Member Functions inherited from WiFiSTAClass

int **begin** (const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)

This function is used to start the wifi module as station mode. More...

int **begin** (char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)
This function is used to start the wifi module as station mode. More...

int begin ()

This function is used to start the wifi module as station mode. More...

bool reconnect ()

This function is used to reconect the AP. More...

bool **disconnect** (bool wifioff=false)

This function is used to disconnect the wifi. More...

bool isConnected ()

This function is used to get the wifi mode connect status. More...

bool **setAutoConnect** (bool autoConnect)

This function is used to set auto connect flag. More...

bool getAutoConnect ()

This function is used to get auto connect flag. More...

bool **setAutoReconnect** (bool autoReconnect)

This function is used to set auto reconnect flag. More...

bool getAutoReconnect ()

This function is used to get auto reconnect flag. More...

uint8 t waitForConnectResult ()

This function is used to suspend until the WiFi is connected. More...

IPAddress localIP ()

This function is used to get the local ip address. More...

char * macAddress ()

This function is used to get local MAC address used by the module. More...

char * macAddressStr ()

This function is used to get local MAC address used by the module. More...

IPAddress subnetMask ()

This function is used to get subnet mask. More...

IPAddress getwayIP ()

This function is used to get gateway IP address. More...

IPAddress dnsIP (uint8 t dns no=0)

This function is used to get DNS IP address. More...

char * hostname ()

bool **hostname** (char *aHostname)

bool hostname (const char *aHostname)

wl_status_t status()

This function is used to get the status during Station mode. More...

char * statusStr ()

This function is used to get the status during Station mode. More...

char * SSID () const

This function is used to get the SSID used by the module. More...

char * psk () const

This function is used to get the psk used by the module. More...

uint8 t* BSSID ()

This function is used to get the BSSID which is connected by the module. More...

char * BSSIDstr ()

This function is used to get the BSSID which is connected by the module. More...

int32_t RSSI()

This function is used to get the RSSI. More...

Public Member Functions inherited from WiFiScanClass

int8_t **scanNetworks** (bool async=false, bool show_hidden=false, uint8_t **channel**=0, uint8_t

*ssid=NULL)

This function is used to begin scan the WiFi network. More...

int8_t scanComplete ()

This function is used to get the scan status. More...

void scanDelete ()

This function is used to free the buffer of last scan result. More...

bool **getNetworkInfo** (uint8_t networkItem, char *&ssid, uint8_t &encryptionType, int32_t &RSSI, uint8_t *&BSSID, int32_t &channel, bool &isHidden)

This function is used to get the network info via last scan. More...

char * SSID (uint8_t networkItem)

This function is used to get ssid. More...

uint32 t encryptionType (uint8 t networkItem)

This function is used to get encryption type. More...

int32_t RSSI (uint8_t networkItem)

This function is used to get RSSI. More...

uint8_t * BSSID (uint8_t networkItem)

This function is used to get the bssid. More...

char * BSSIDstr (uint8_t networkItem)

This function is used to get the bssid. More...

int32 t channel (uint8 t networkItem)

This function is used to get the bssid. More...

bool **isHidden** (uint8_t networkItem)

This function is used to get the hidden flag. More...

▶ Public Member Functions inherited from WiFiAPClass

bool **softAP** (const char *ssid, const char *passphrase=NULL, int channel=1, int ssid_hidden=0, int max_connection=4)
This function is used to start wifi module as AP mode.
More...

bool **softAPConfig** (uint32_t local_ip, uint32_t gateway, uint32_t subnet)

This function is used to config soft AP parameters. More...

bool **softAPConfig** (const char *local_ip, const char *gateway, const char *subnet)

This function is used to config soft AP parameters. More...

bool **softAPdisconnect** (bool wifioff=false)
This function is used to disconnect the ap function.
More...

bool softAPdestroy ()
This function is used to destrory the AP function. More...

uint8_t softAPgetStationNum ()
This function is used to get the number of station. More...

char * softAPIP ()
This function is used to get AP's IP. More...

uint8_t * softAPmacAddress (uint8_t *mac)
This function is used to get AP's MAC address. More...

char * softAPmacAddress (void)
This function is used to get the AP's MAC address.
More...

char * **softAPSSID** () const

This function is used to get AP's SSID. More...

char * softAPPSK () const

This function is used to get AP's PSK. More...

▶ Public Member Functions inherited from WiFiOneshotClass

int oneshotStart ()

This function is used to start oneshot configuration network. More...

int oneshotStop ()

This function is used to stop oneshot configuration network. More...

int oneshotGetState ()

This function is used to get oneshot state. More...

int oneshotSetMode (ONESHOT MODE mode)

This function is used to get oneshot state. More...

int oneshotGetMode ()

This function is used to get oneshot mode. More...

Additional Inherited Members

Static Protected Member Functions inherited from WiFiScanClass static void _scanDone ()

static void * _getScanInfoByIndex (int i)

Member Function Documentation

◆ BSSID() [1/2]

uint8_t * WiFiSTAClass::BSSID

This function is used to get the BSSID which is connected by the module.

Parameters

[in] None

Returns

The BSSID of the AP.

Note

Parameters

[in] none

Returns

The BSSID of the AP.

◆ BSSID() [2/2]

uint8_t * WiFiScanClass::BSSID

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The bssid

Note

Parameters

[in] **networkItem** Specify the index of scan result.

Returns

The bssid.

♦ BSSIDstr() [1/2]

char * WiFiSTAClass::BSSIDstr

This function is used to get the BSSID which is connected by the module.

Parameters

[in] None

Returns

The string of the BSSID of the AP.

♦ BSSIDstr() [2/2]

char * WiFiScanClass::BSSIDstr

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The string of the bssid.

channel()

int32_t WiFiScanClass::channel

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The channel of the target.

◆ RSSI() [1/2]

int32_t WiFiSTAClass::RSSI

This function is used to get the RSSI.

Parameters

[in] None

Returns

the value of rssi in this connect.

◆ RSSI() [2/2]

int32_t WiFiScanClass::RSSI

This function is used to get RSSI.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The RSSI of the target

◆ SSID() [1/2]

char * WiFiSTAClass::SSID

This function is used to get the SSID used by the module.

Parameters

[in] none

Returns

The string of the SSID.

Note

Parameters

[in] None

Returns

The string of the SSID.

◆ SSID() [2/2]

char * WiFiScanClass::SSID

This function is used to get ssid.

Parameters

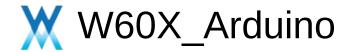
[in] **networkItem** Specify the index of scan result.

Returns

The SSID of the target

Note

Generated by doxygen 1.8.14



WiFiClient Class Reference

Public Member Functions | Static Public Member Functions | Protected Member Functions | Static Protected Member Functions |

Protected Attributes | Static Protected Attributes

Inherits Client, and SList< WiFiClient >.

Friends

Public Member Functions

WiFiClient ()

This function is constructor, it's used to creates a client that can connect to to a specified internet IP address and port as defined in client.connect(). More...

WiFiClient (const WiFiClient &)

WiFiClient & **operator=** (const **WiFiClient** &)

virtual ~WiFiClient ()

This function is deconstructor, it's used to release **WiFiClient** class. More...

uint8_t status ()

return top status of WiFiClient. More...

virtual int **connect** (**IPAddress** ip, uint16_t port)

This function is used to connect to the IP address and port specified in the constructor. More...

virtual int connect (const char *host, uint16_t port)

This function is used to connect to the IP address and port specified in the constructor. More...

virtual int **connect** (const String host, uint16 t port)

virtual size_t write (uint8_t)

This function is used to write data to the server the client is connected to. More...

virtual size_t write (const uint8_t *buf, size_t size)

This function is used to write data to the server the client is connected to. More...

virtual size_t write_P (PGM_P buf, size_t size)

size_t write (Stream &stream)

size_t write (Stream &stream, size_t unitSize) attribute ((deprecated))

virtual int available ()

Returns the number of bytes available for reading (That is, the amount of data that has been written to the client by the server it is connected to). More...

virtual int read ()

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int read (uint8_t *buf, size_t size)

Read the next byte received from the server the client is connected to (after the last call to **read()**). More...

virtual int peek ()

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual size t peekBytes (uint8 t *buffer, size t length)

size_t peekBytes (char *buffer, size_t length)

virtual void flush ()

Discard any bytes that have been written to the client but not yet read. More...

virtual void stop ()

This function is used to disconnect from the server. More...

virtual uint8 t connected ()

Whether or not the client is connected. More...

virtual operator bool ()

IPAddress remotelP ()

This function is used to gets the IP address of the remote connection. More...

uint16_t remotePort ()

This function is used to gets the port of the remote connection. More...

IPAddress localIP ()

This function is used to gets the IP address of the local tcp connection. More...

uint16 t localPort ()

This function is used to gets the port of the local tcp connection. More...

bool getNoDelay ()

This function is used to get whether no delay of the tcp connection. More...

void setNoDelay (bool nodelay)

This function is used to set no delay for the tcp connection. More...

size t availableForWrite ()

This function is used to get the length that can be written. More...

void keepAlive (uint16_t idle_sec=TCP_DEFAULT_KEEPALIVE_IDLE_SEC,

uint16 t

intv_sec=TCP_DEFAULT_KEEPALIVE_INTERVAL_SEC, uint8_t count=TCP_DEFAULT_KEEPALIVE_COUNT)

This function is used to set keep alive. More...

bool isKeepAliveEnabled () const

This function is used to get whether enable keep alive. More...

uint16 t getKeepAliveldle () const

This function is used to get idle time interval. More...

uint16 t getKeepAliveInterval () const

This function is used to get keep alive time interval. More...

uint8_t getKeepAliveCount () const

This function is used to get keep alive count. More...

void disableKeepAlive ()

This function is used to set disable keep alive. More...

virtual size_t write (uint8_t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size t write (const uint8 t *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool **find** (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8 t *target, size t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (char target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **findUntil** (const char *target, const char *terminator) **findUntil()** reads data from the stream until the target string of given length or terminator string is found.

More...

- bool **findUntil** (const uint8_t *target, const char *terminator) **findUntil**() reads data from the stream until the target string of given length or terminator string is found.

 More...
- bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found.

 More...
- bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found.
 More...
- long parseInt ()
 parseInt() returns the first valid (long) integer number
 from the serial buffer. Characters that are not integers
 (or the minus sign) are skipped. More...
- float parseFloat ()
 parseFloat() returns the first valid floating point number
 from the current position. Initial characters that are not
 digits (or the minus sign) are skipped. parseFloat() is
 terminated by the first character that is not a floating
 point number. More...
- virtual size_t readBytes (char *buffer, size_t length)
 readBytes() read characters from a stream into a buffer.
 The function terminates if the determined length has been read, or it times out (see setTimeout()). More...
- virtual size_t readBytes (uint8_t *buffer, size_t length) readBytes() read characters from a stream into a buffer.

The function terminates if the determined length has been read, or it times out (see **setTimeout()**). More...

size_t readBytesUntil (char terminator, char *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

size_t **readBytesUntil** (char terminator, uint8_t *buffer, size_t length)

readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

String readString ()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**). More...

String readStringUntil (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see **setTimeout()**). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size t **print** (const char □)

This function is used to print buffer to the interface defined by the object. More...

size t print (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size t **println** (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size t println (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

Static Public Member Functions

static void setLocalPortStart (uint16_t port)

This function is used to set local port number. More...

static void stopAll ()

This function is used to stop all **WiFiClient** session. More...

static void stopAllExcept (WiFiClient *c)

This function is used to stop all **WiFiClient** session without exC. More...

Protected Member Functions

WiFiClient (ClientContext *client)

int8_t _connected (void *tpcb, int8_t err)

void _err (int8 t err)

Protected Member Functions inherited from Stream

int timedRead ()

int timedPeek ()

int peekNextDigit ()

long parseInt (char skipChar)

float parseFloat (char skipChar)

Protected Member Functions inherited from Print void setWriteError (int err=1)

Static Protected Member Functions

static int8_t _s_connected (void *arg, void *tpcb, int8_t err)

static void _s_err (void *arg, int8_t err)

Static Protected Member Functions inherited from SList
WiFiClient >

static void _add (WiFiClient *self)

static void _remove (WiFiClient *self)

Protected Attributes

ClientContext * _client

Protected Attributes inherited from Stream unsigned long _timeout

unsigned long _startMillis

Protected Attributes inherited from SList< WiFiClient >
WiFiClient * _next

Static Protected Attributes

static uint16_t _localPort = 0

Static Protected Attributes inherited from SList< WiFiClient > static WiFiClient * _s_first

Friends

class WiFiServer

Member Function Documentation

write() [1/4]

size_t Print::write

This function is used to write buffer to the interface defined by the object.

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully.

Note

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully (1 byte).

write() [2/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] **str** Specify the buffer of string.

Returns

The length of write successfully.

write() [3/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] buffer Specify the buffer of string.

[in] **size** Specify the size.

Returns

The length of write successfully.

write() [4/4]

virtual size_t Print::write

This pure virtual function is used to define the operation that writes binary data.

Parameters

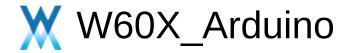
[in] val a value to send as a single byte

Returns

The length of write successfully (1 byte).

Note

Generated by doxygen 1.8.14



WiFiGenericClass Class Reference **Public Member Functions**

Inherited by WiFiClass.

Public Member Functions

WiFiMode_t getMode ()

This function is used to Get the WiFi's work mode...

int **hostByName** (const char *aHostname, **IPAddress** &aResult)

This function is used to resolve hostname. More...

int **hostByName** (const char *aHostname, **IPAddress** &aResult, uint32_t timeout_ms)

This function is used to resolve hostname. More...

Member Function Documentation

getMode()

WiFiMode_t WiFiGenericClass::getMode()

This function is used to Get the WiFi's work mode..

Returns

enum of WiFiMode: WIFI_OFF, WIFI_STA or WIFI_AP

hostByName() [1/2]

```
int WiFiGenericClass::hostByName (const char * aHostname,

IPAddress & aResult
)
```

This function is used to resolve hostname.

Parameters

```
[in] aHostname The hostname which you want to resolve.[out] aResult The result of resove the hostname.
```

Returns

1 - success, 0 - failure.

hostByName() [2/2]

```
int WiFiGenericClass::hostByName ( const char * aHostname, IPAddress & aResult, uint32_t timeout_ms )
```

This function is used to resolve hostname.

Parameters

```
[in] aHostname The hostname which you want to resolve.
```

[in] **timeout ms** The timeout when resolve.

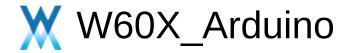
[out] **aResult** The result of resove the hostname.

Returns

1 - success, 0 - failure.

Note

Generated by doxygen 1.8.14



WiFiOneshotClass Class Reference

Public Member Functions

Inherited by WiFiClass.

Public Member Functions

int oneshotStart ()

This function is used to start oneshot configuration network. More...

int oneshotStop ()

This function is used to stop oneshot configuration network. More...

int oneshotGetState ()

This function is used to get oneshot state. More...

int oneshotSetMode (ONESHOT MODE mode)

This function is used to get oneshot state. More...

int oneshotGetMode ()

This function is used to get oneshot mode. More...

Member Function Documentation

oneshotGetMode()

int WiFiOneshotClass::oneshotGetMode ()

This function is used to get oneshot mode.

Parameters

[in] None
[out] None

Return values

0 UDP MODE 1: AP+SOCKET MODE 2: AP+WEBSEVER MODE

oneshotGetState()

int WiFiOneshotClass::oneshotGetState ()

This function is used to get oneshot state.

Parameters

[in] None
[out] None

Return values

0 is in oneshot state 1: is not in oneshot state

oneshotSetMode()

This function is used to get oneshot state.

Parameters

Returns

None

oneshotStart()

int WiFiOneshotClass::oneshotStart()

This function is used to start oneshot configuration network.

Parameters

[in] None
[out] None

Returns

None

oneshotStop()

int WiFiOneshotClass::oneshotStop()

This function is used to stop oneshot configuration network.

Parameters

[in] None
[out] None

Returns

None

Note

Generated by doxygen 1.8.14

WiFiScanClass Class Reference

Public Member Functions | Static Protected Member Functions

Inherited by WiFiClass.

Public Member Functions

int8_t **scanNetworks** (bool async=false, bool show_hidden=false, uint8 t **channel**=0, uint8 t *ssid=NULL)

This function is used to begin scan the WiFi network. More...

int8 t scanComplete ()

This function is used to get the scan status. More...

void scanDelete ()

This function is used to free the buffer of last scan result. More...

bool **getNetworkInfo** (uint8_t networkItem, char *&ssid, uint8_t &encryptionType, int32_t &RSSI, uint8_t *&BSSID, int32_t &channel, bool &isHidden)

This function is used to get the network info via last scan. More...

char * SSID (uint8_t networkItem)

This function is used to get ssid. More...

uint32_t encryptionType (uint8_t networkItem)

This function is used to get encryption type. More...

int32 t RSSI (uint8 t networkItem)

This function is used to get RSSI. More...

uint8 t * **BSSID** (uint8 t networkItem)

This function is used to get the bssid. More...

char * BSSIDstr (uint8_t networkItem)

This function is used to get the bssid. More...

int32_t channel (uint8_t networkItem)

This function is used to get the bssid. More...

bool isHidden (uint8_t networkItem)
This function is used to get the hidden flag. More...

Static Protected Member Functions

static void _scanDone ()

static void * _getScanInfoByIndex (int i)

Member Function Documentation

◆ BSSID()

uint8_t * WiFiScanClass::BSSID (uint8_t networkItem)

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The bssid

Note

Parameters

[in] networkItem Specify the index of scan result.

Returns

The bssid.

◆ BSSIDstr()

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The string of the bssid.

channel()

int32_t WiFiScanClass::channel (uint8_t networkItem)

This function is used to get the bssid.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The channel of the target.

• encryptionType()

uint32_t WiFiScanClass::encryptionType (uint8_t networkItem)

This function is used to get encryption type.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The encryptionType of the target

getNetworkInfo()

```
bool WiFiScanClass::getNetworkInfo ( uint8_t networkItem, char *& ssid, uint8_t & encryptionType, int32_t & RSSI, uint8_t *& BSSID, int32_t & channel, bool & isHidden )
```

This function is used to get the network info via last scan.

Parameters

[in] **networkItem** Specify the index of scan result.

[out] **ssid** The ssid in the scan result, indexed by

networkItem.

[out] encryptionType The encryptionType in the scan result,

indexed by networkItem.

[out] RSSI in the scan result, indexed by

networkItem.

[out] **BSSID** The BSSID in the scan result, indexed by

networkItem.

[out] channel The channel in the scan result, indexed

by networkItem.

[out] isHidden The isHidden flag in the scan result,

indexed by networkItem.

Returns

None

◆ isHidden()

bool WiFiScanClass::isHidden (uint8_t networkItem)

This function is used to get the hidden flag.

Parameters

[in] **networkItem** Specify the index of scan result.

Returns

false

◆ RSSI()

int32_t WiFiScanClass::RSSI (uint8_t _networkItem)

This function is used to get RSSI.

Parameters

[in] networkItem Specify the index of scan result.

Returns

The RSSI of the target

scanComplete()

int8_t WiFiScanClass::scanComplete ()

This function is used to get the scan status.

Parameters

[in] None

Returns

If during scanning, WM_WIFI_SCANNING_BUSY is returned. If finish scanning, the number of network is returned, otherwise, -1 is returned.

scanDelete()

void WiFiScanClass::scanDelete ()

This function is used to free the buffer of last scan result.

Parameters

[in] None

Returns

None

scanNetworks()

This function is used to begin scan the WiFi network.

Parameters

```
[in] async Specify the async.[in] show_hidden Specify the show_hidden.[in] channel Specify the channel.[in] ssid Specify the ssid.
```

Returns

If not SUCCESS, user needs to call this function again to trigger scan

♦ SSID()

char * WiFiScanClass::SSID (uint8_t _networkItem)

This function is used to get ssid.

Parameters

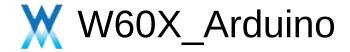
[in] **networkItem** Specify the index of scan result.

Returns

The SSID of the target

Note

Generated by doxygen 1.8.14



WiFiServer Class Reference

Public Member Functions |
Static Public Member Functions |
Protected Attributes

Inherits **Server**.

Public Member Functions

WiFiServer (IPAddress addr, uint16 t port)

This constructor is used to init **WiFiServer** object with the address and port specify by caller. More...

WiFiServer (uint16 t port)

This constructor is used to init **WiFiServer** object with port specify by caller. More...

bool hasClient ()

void begin ()

This function is used to start the WiFiServer. More...

void begin (uint16_t port)

This function is used to start the WiFiServer. More...

void setNoDelay (bool nodelay)

This function is used to set no-delay flag. More...

bool getNoDelay ()

This function is used to get no-delay flag. More...

virtual size t write (uint8 t)

This function is used to send the message (one byte) to peer. More...

virtual size_t write (const uint8_t *buf, size_t size)

This function is used to send the message to peer. More...

uint8_t status ()

void close ()

This function is used to close the connection. More...

void stop ()

This function is used to close the connection. More...

long _accept (tcp pcb *newpcb, long err)

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size t write (const char *buffer, size t size)

This function is used to write buffer to the interface defined by the object. More...

size t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (const char [])

This function is used to print buffer to the interface defined by the object. More...

size t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t **print** (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size_t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t **println** (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size t println (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

size_t **println** (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('). More...

size_t **println** (const **Printable** &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line (' '). More...

Static Public Member Functions

static err_t _**s_accept** (void *arg, tcp_pcb *newpcb, err_t err)

Protected Attributes

uint16_t _**port**

IPAddress _addr

tcp_pcb * _pcb

bool _noDelay = false

Additional Inherited Members

Protected Member Functions inherited from Print
void setWriteError (int err=1)

Generated by doxygen 1.8.14



Public Member Functions

WiFiSTAClass Class Reference

Inherited by WiFiClass.

Public Member Functions

int begin (const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)

This function is used to start the wifi module as station mode. More...

int **begin** (char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)

This function is used to start the wifi module as station mode. More...

int begin ()

This function is used to start the wifi module as station mode. More...

bool reconnect ()

This function is used to reconect the AP. More...

bool disconnect (bool wifioff=false)

This function is used to disconnect the wifi. More...

bool isConnected ()

This function is used to get the wifi mode connect status. More...

bool setAutoConnect (bool autoConnect)

This function is used to set auto connect flag. More...

bool getAutoConnect ()

This function is used to get auto connect flag. More...

bool **setAutoReconnect** (bool autoReconnect)

This function is used to set auto reconnect flag. More...

bool getAutoReconnect ()

This function is used to get auto reconnect flag. More...

uint8_t waitForConnectResult ()

This function is used to suspend until the WiFi is connected. More...

IPAddress localIP ()

This function is used to get the local ip address. More...

char * macAddress ()

This function is used to get local MAC address used by the module. More...

char * macAddressStr ()

This function is used to get local MAC address used by the module. More...

IPAddress subnetMask ()

This function is used to get subnet mask. More...

IPAddress getwayIP ()

This function is used to get gateway IP address. More...

IPAddress dnsIP (uint8 t dns no=0)

This function is used to get DNS IP address. More...

char * hostname ()

bool **hostname** (char *aHostname)

bool **hostname** (const char *aHostname)

wl status t status ()

This function is used to get the status during Station mode. More...

char * statusStr ()

This function is used to get the status during Station mode. More...

char * SSID () const

This function is used to get the SSID used by the module. More...

char * psk () const

This function is used to get the psk used by the module. More...

uint8_t * BSSID ()

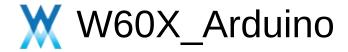
This function is used to get the BSSID which is connected by the module. More...

char * BSSIDstr ()

This function is used to get the BSSID which is connected by the module. More...

int32_t RSSI ()

This function is used to get the RSSI. More...



WiFiUDP Class Reference

Public Member Functions |
Static Public Member Functions

Inherits UDP, and SList< WiFiUDP >.

Public Member Functions

WiFiUDP ()

This function is constructor, it's used to creates a named instance of the **WiFiUDP** class that can send and receive **UDP** messages. More...

WiFiUDP (const **WiFiUDP** &other)

WiFiUDP & **operator=** (const **WiFiUDP** &rhs)

~WiFiUDP ()

This function is deconstructor, it's used to release **WiFiUDP** class. More...

operator bool () const

virtual uint8 t **begin** (uint16 t port)

This function is used to initializes the **WiFiUDP** library and network settings, Starts **UDP** socket, listening at local port. More...

virtual void stop ()

This function is used to disconnect from the server. Release any resource being used during the **UDP** session. More...

uint8_t beginMulticast (IPAddress interfaceAddr, IPAddress multicast, uint16_t port)

This function is used to join a multicast group and listen on the given port. More...

virtual int **beginPacket** (**IPAddress** ip, uint16_t port)

This function is used to starts a connection to write **UDP** data to the remote connection. More...

virtual int **beginPacket** (const char *host, uint16_t port)

This function is used to starts a connection to write **UDP** data to the remote connection. More...

virtual int beginPacketMulticast (IPAddress

multicastAddress, uint16_t port, **IPAddress** interfaceAddress, int ttl=1)

This function is used to start building up a packet to send to the multicast address. More...

virtual int endPacket ()

This function is used to called after writing **UDP** data to the remote connection. It finishes off the packet and send it. More...

virtual size t write (uint8 t)

This function is used to writes **UDP** data to the remote connection. More...

virtual size_t write (const uint8_t *buffer, size_t size)

This function is used to writes **UDP** data to the remote connection. More...

virtual int parsePacket ()

It starts processing the next available incoming packet, checks for the presence of a **UDP** packet, and reports the size. More...

virtual int available ()

Get the number of bytes (characters) available for reading from the buffer. This is is data that's already arrived. More...

virtual int read ()

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int **read** (unsigned char *buffer, size_t len)

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int read (char *buffer, size_t len)

Reads **UDP** data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...

virtual int peek ()

Read a byte from the file without advancing to the next one.

That is, successive calls to **peek()** will return the same value, as will the next call to **read()**. More...

virtual void flush ()

Discard any bytes that have been written to the client but not yet read. More...

virtual IPAddress remoteIP ()

This function is used to gets the IP address of the remote connection. More...

virtual uint16 t remotePort ()

This function is used to gets the port of the remote **UDP** connection. More...

IPAddress destinationIP ()

This function is used to distinguish multicast and ordinary packets. More...

uint16 t localPort ()

This function is used to gets the port of the local **UDP** connection. More...

virtual size_t write (uint8_t)=0

This pure virtual function is used to define the operation that writes binary data. More...

virtual size_t write (const uint8_t *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

Public Member Functions inherited from Stream

void setTimeout (unsigned long timeout)

setTimeout() sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the **Stream** class, and is called by any class that inherits from it (Wire, Serial, etc). See the **Stream** class main page for more information. More...

bool find (const char *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (uint8_t *target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool **find** (const char *target, size_t length)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool find (const uint8_t *target, size_t length)
find() reads data from the stream until the target
string of given length is found The function returns
true if target string is found, false if timed out.
More...

bool **find** (char target)

find() reads data from the stream until the target string of given length is found The function returns true if target string is found, false if timed out. More...

bool findUntil (const char *target, const char *terminator) findUntil() reads data from the stream until the target string of given length or terminator string is found. More...

findUntil() reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const char *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is found. More...

bool **findUntil** (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen) **findUntil()** reads data from the stream until the target string of given length or terminator string is

found. More...

long parseInt ()

parseInt() returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped. More...

float parseFloat ()

parseFloat() returns the first valid floating point
number from the current position. Initial characters
that are not digits (or the minus sign) are skipped.
parseFloat() is terminated by the first character that
is not a floating point number. More...

virtual size_t readBytes (char *buffer, size_t length)
readBytes() read characters from a stream into a
buffer. The function terminates if the determined
length has been read, or it times out (see
setTimeout()). More...

virtual size_t readBytes (uint8_t *buffer, size_t length)
readBytes() read characters from a stream into a
buffer. The function terminates if the determined
length has been read, or it times out (see
setTimeout()). More...

size_t readBytesUntil (char terminator, char *buffer, size_t length)
readBytesUntil() reads characters from a stream into a buffer. The function terminates if the terminator character is detected, the determined length has been read, or it times out (see setTimeout()). More...

size_t readBytesUntil (char terminator, uint8_t *buffer, size_t length)
readBytesUntil() reads characters from a stream into a buffer. The function terminates if the

terminator character is detected, the determined length has been read, or it times out (see **setTimeout()**). More...

String readString ()

readString() reads characters from a stream into a string. The function terminates if it times out (see **setTimeout()**). More...

String **readStringUntil** (char terminator)

readStringUntil() reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see setTimeout()). More...

Public Member Functions inherited from Print

int getWriteError ()

This function is used to get write error number. More...

void clearWriteError ()

This function is used to clear write error number. More...

size_t write (const char *str)

This function is used to write buffer to the interface defined by the object. More...

size_t write (const char *buffer, size_t size)

This function is used to write buffer to the interface defined by the object. More...

size_t **print** (const String &)

This function is used to print buffer to the interface defined by the object. More...

size_t print (const char [])

This function is used to print buffer to the interface defined by the object. More...

size t **print** (char)

This function is used to print buffer to the interface defined by the object. More...

size_t print (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t **print** (int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t **print** (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object. More...

size t print (long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object. More...

size_t print (double, int=BIN)

This function is used to print target to the interface defined by the object. More...

size t print (const Printable &)

This function is used to print target to the interface defined by the object. More...

size_t **println** (void)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size t println (const String &s)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (const char [])

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t **println** (char)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (unsigned char, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t **println** (int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (unsigned int, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new

line ('
'). More...

size_t println (long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (unsigned long, int=DEC)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size_t println (double, int=BIN)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

size t println (const Printable &)

This function is used to print target to the interface defined by the object with carriage ret (") and new line ('

'). More...

Static Public Member Functions

static void stopAll ()
This function is used to stop all WiFiUDP session. More...

static void stopAllExcept (WiFiUDP *exC)
This function is used to stop all WiFiUDP session without exC. More...

Additional Inherited Members

Protected Member Functions inherited from Stream int timedRead () int timedPeek () int peekNextDigit () long parseInt (char skipChar) float parseFloat (char skipChar) Protected Member Functions inherited from Print void setWriteError (int err=1) Static Protected Member Functions inherited from SList<</p> WiFiUDP > static void _add (WiFiUDP *self) static void **remove (WiFiUDP** *self) Protected Attributes inherited from Stream unsigned long _timeout unsigned long _startMillis Protected Attributes inherited from SList< WiFiUDP > WiFiUDP * next Static Protected Attributes inherited from SList< WiFiUDP > static WiFiUDP * s first

Member Function Documentation

write() [1/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] **str** Specify the buffer of string.

Returns

The length of write successfully.

Note

write() [2/4]

virtual size_t Print::write

This pure virtual function is used to define the operation that writes binary data.

Parameters

[in] val a value to send as a single byte

Returns

The length of write successfully (1 byte).

Note

write() [3/4]

size_t Print::write

This function is used to write buffer to the interface defined by the object.

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully.

Note

Parameters

```
[in] buffer Specify the buffer.[in] size Specify the size.
```

Returns

The length of write successfully (1 byte).

Note

write() [4/4]

size_t Print::write

inline

This function is used to write buffer to the interface defined by the object.

Parameters

[in] buffer Specify the buffer of string.

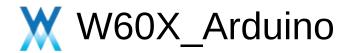
[in] size Specify the size.

Returns

The length of write successfully.

Note

Generated by doxygen 1.8.14



Data Structure Index

b|c|d|h|i|p|s|t|u|w

b DuffeyDete Course

BufferDataSource BufferedStreamDataSource

С

Client ClientContext CloudClass

d

DataSource

DhcpClass DNSClient DNSServer

h

HardwareSerial



IPAddress



Print

Printable ProgmemStream

S

Server SList SPIClass SPISettings

Œ

Stream

TwoWire

b | c | d | h | i | p | s | t | u | w

Generated by doxygen 1.8.14



Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

	[detail level 1234]
ClientContext	
CloudClass	
▼	
BufferDataSource	
BufferedStreamDataSource< TStream >	
O DhcpClass	
O DNSClient	
O DNSServer	
▼	
▼	
WiFiServer	
▼	
▼	
© WiFiClient	
HardwareSerial	
TwoWire	
▼ (a) UDP	
© WiFiUDP	
▼ 	
IPAddress	
ProgmemStream	
G SList< T >	
▼ SList< WiFiClient >	

© WiFiClient	
▼ SList< WiFiUDP >	
© WiFiUDP	
© SPIClass	
SPISettings	
UdpContext	
™ W600InnerFlashClass	
▼ (i) WiFiAPClass	
WiFiClass	
▼ 	
WiFiClass	
▼ 	
@ WiFiClass	
▼ 	
@ WiFiClass	
▼ 	
@ WiFiClass	

Generated by doxygen 1.8.14

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- a -

- available(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- availableForWrite(): WiFiClient

Generated by doxygen 1.8.14

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- b -

- begin(): HardwareSerial, Server, SPIClass, UDP,
 W600InnerFlashClass, WiFiServer, WiFiSTAClass, WiFiUDP
- beginMulticast(): WiFiUDP
- beginPacket(): UDP, WiFiUDP
- beginPacketMulticast(): WiFiUDP
- beginTransaction(): SPIClass
- BSSID(): WiFiScanClass, WiFiSTAClass
- BSSIDstr(): WiFiScanClass, WiFiSTAClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- C -

channel(): WiFiScanClassclearWriteError(): Print

• close() : WiFiServer

• CloudClass(): CloudClass

• CloudInit(): CloudClass

connect(): Client, WiFiClientconnected(): Client, WiFiClient

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- d -

• destinationIP(): WiFiUDP

disableKeepAlive(): WiFiClientdisconnect(): WiFiSTAClass

• dnsIP(): WiFiSTAClass

Generated by doxygen 1.8.14

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- e -

• encryptionType(): WiFiScanClass

• end(): SPIClass

endPacket(): UDP , WiFiUDP
 endTransaction(): SPIClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- f -

- find(): Stream
- findUntil(): Stream
- flashEraseSector(): W600InnerFlashClass
- flashRead(): W600InnerFlashClass
- flashWrite() : W600InnerFlashClass
- flush(): Client, UDP, WiFiClient, WiFiUDP
- fromString(): IPAddress

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- g -

- getAutoConnect(): WiFiSTAClass
- getAutoReconnect(): WiFiSTAClass
- getHostByName(): DNSClient
- getKeepAliveCount(): WiFiClient
- getKeepAliveIdle(): WiFiClient
- getKeepAliveInterval(): WiFiClient
- getMode(): WiFiGenericClass
- getNetworkInfo(): WiFiScanClass
- getNoDelay(): WiFiClient, WiFiServer
- getwayIP(): WiFiSTAClass
- getWriteError(): Print

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- h -
 - HardwareSerial(): HardwareSerialhostByName(): WiFiGenericClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- i -

• IPAddress(): IPAddress

isConnected(): WiFiSTAClassisHidden(): WiFiScanClass

• isKeepAliveEnabled(): WiFiClient

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- k -

• keepAlive(): WiFiClient

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- | -

- localIP(): WiFiClient, WiFiSTAClass
- localPort(): WiFiClient, WiFiUDP

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- m -

macAddress(): WiFiSTAClassmacAddressStr(): WiFiSTAClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- 0 -

oneshotGetMode(): WiFiOneshotClass
 oneshotGetState(): WiFiOneshotClass
 oneshotSetMode(): WiFiOneshotClass

oneshotStart(): WiFiOneshotClass
 oneshotStop(): WiFiOneshotClass
 operator uint32_t(): IPAddress

operator=(): IPAddress
 operator==(): IPAddress
 operator[](): IPAddress

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- p -

- parseFloat(): Stream
- parseInt(): Stream
- parsePacket(): UDP , WiFiUDP
- peek(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- print(): Printprintln(): Print
- printTo(): IPAddress, Printable
- psk(): WiFiSTAClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- r -

- read(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- readBytes(): Stream
- readBytesUntil(): Stream
- readString(): Stream
- readStringUntil(): Stream
- reconnect(): WiFiSTAClass
- remoteIP(): UDP, WiFiClient, WiFiUDP
- remotePort(): UDP, WiFiClient, WiFiUDP
- RSSI(): WiFiScanClass, WiFiSTAClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- S -

- scanComplete(): WiFiScanClass
- scanDelete(): WiFiScanClass
- scanNetworks(): WiFiScanClass
- setAutoConnect(): WiFiSTAClass
- setAutoReconnect(): WiFiSTAClass
- setBitOrder(): SPIClass
- setDataMode(): SPIClass
- setFrequency(): SPIClass
- setLocalPortStart(): WiFiClient
- setNoDelay(): WiFiClient, WiFiServer
- setTimeout() : Stream
- softAP(): WiFiAPClass
- softAPConfig(): WiFiAPClass
- softAPdestroy(): WiFiAPClass
- softAPdisconnect(): WiFiAPClass
- softAPgetStationNum(): WiFiAPClass
- softAPIP(): WiFiAPClass
- softAPmacAddress(): WiFiAPClass
- softAPPSK(): WiFiAPClass
- softAPSSID(): WiFiAPClass
- SSID(): WiFiScanClass, WiFiSTAClass
- start(): DNSServer
- status(): WiFiClient, WiFiSTAClass
- statusStr(): WiFiSTAClass
- stop(): Client, DNSServer, UDP, WiFiClient, WiFiServer, WiFiUDP
- stopAll(): WiFiClient, WiFiUDP
- stopAllExcept(): WiFiClient, WiFiUDP
- subnetMask(): WiFiSTAClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- t -

toString(): IPAddress
 transfer(): SPIClass
 transfer16(): SPIClass
 transferRead(): SPIClass
 transferWrite(): SPIClass

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- W -

waitForConnectResult(): WiFiSTAClass

WiFiClient(): WiFiClientWiFiServer(): WiFiServerWiFiUDP(): WiFiUDP

 write(): Client, HardwareSerial, Print, TwoWire, UDP, WiFiClient, WiFiServer, WiFiUDP

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

_ ~ _

~WiFiClient(): WiFiClient~WiFiUDP(): WiFiUDP

- a -

- available(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- availableForWrite(): WiFiClient

- b -

- begin(): HardwareSerial, Server, SPIClass, UDP,
 W600InnerFlashClass, WiFiServer, WiFiSTAClass, WiFiUDP
- beginMulticast(): WiFiUDP
- beginPacket(): UDP , WiFiUDP
- beginPacketMulticast(): WiFiUDP
- beginTransaction(): SPIClass
- BSSID(): WiFiScanClass, WiFiSTAClass
- BSSIDstr(): WiFiScanClass, WiFiSTAClass

- C -

channel(): WiFiScanClassclearWriteError(): Print

• close(): WiFiServer

CloudClass(): CloudClassCloudInit(): CloudClass

connect() : Client , WiFiClient
 connected() : Client , WiFiClient

- d -

• destinationIP(): WiFiUDP

disableKeepAlive(): WiFiClient
 disconnect(): WiFiSTAClass

• dnsIP(): WiFiSTAClass

- e -

• encryptionType(): WiFiScanClass

• end(): SPIClass

endPacket(): UDP , WiFiUDP
 endTransaction(): SPIClass

- f -

• find(): Stream

• findUntil(): Stream

• flashEraseSector(): W600InnerFlashClass

• flashRead(): W600InnerFlashClass

• flashWrite() : W600InnerFlashClass

• flush(): Client, UDP, WiFiClient, WiFiUDP

• fromŠtring(): IPAddress

- g -

- getAutoConnect(): WiFiSTAClassgetAutoReconnect(): WiFiSTAClass
- getHostByName(): DNSClient
- getKeepAliveCount(): WiFiClient
- getKeepAliveIdle(): WiFiClient
- getKeepAliveInterval(): WiFiClient
- getMode(): WiFiGenericClass
- getNetworkInfo(): WiFiScanClass
- getNoDelay(): WiFiClient, WiFiServer
- getwayIP(): WiFiSTAClass
- getWriteError(): Print

- h -

HardwareSerial(): HardwareSerialhostByName(): WiFiGenericClass

- i -

• IPAddress(): IPAddress

• isConnected(): WiFiSTAClass

isHidden(): WiFiScanClass
 isKeepAliveEnabled(): WiFiClient

- k -

• keepAlive(): WiFiClient

Generated by @@XYG@n 1.8.14

- | -

• localIP(): WiFiClient, WiFiSTAClass

• localPort(): WiFiClient, WiFiUDP

- m -

macAddress(): WiFiSTAClassmacAddressStr(): WiFiSTAClass

- 0 -

oneshotGetMode(): WiFiOneshotClass
 oneshotGetState(): WiFiOneshotClass
 oneshotSetMode(): WiFiOneshotClass
 oneshotStart(): WiFiOneshotClass
 oneshotStop(): WiFiOneshotClass
 operator uint32_t(): IPAddress

operator=(): IPAddress
 operator==(): IPAddress
 operator[](): IPAddress

- p -

parseFloat(): StreamparseInt(): Stream

parsePacket(): UDP , WiFiUDP

• peek(): Client, HardwareSerial, Stream, TwoWire, UDP,

WiFiClient, WiFiUDP

print(): Printprintln(): Print

• printTo(): IPAddress, Printable

• psk(): WiFiSTAClass

- r -

- read(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- readBytes(): Stream
- readBytesUntil(): Stream
- readString(): Stream
- readStringUntil(): Stream
- reconnect(): WiFiSTAClass
- remoteIP() : UDP , WiFiClient , WiFiUDP
- remotePort(): UDP , WiFiClient , WiFiUDP
- RSSI(): WiFiScanClass, WiFiSTAClass

- S -

- scanComplete(): WiFiScanClass
- scanDelete(): WiFiScanClass
- scanNetworks(): WiFiScanClass
- setAutoConnect(): WiFiSTAClass
- setAutoReconnect(): WiFiSTAClass
- setBitOrder(): SPIClass
- setDataMode(): SPIClass
- setFrequency(): SPIClass
- setLocalPortStart(): WiFiClient
- setNoDelay(): WiFiClient, WiFiServer
- setTimeout(): Stream
- softAP(): WiFiAPClass
- softAPConfig(): WiFiAPClass
- softAPdestroy(): WiFiAPClass
- softAPdisconnect(): WiFiAPClass
- softAPgetStationNum(): WiFiAPClass
- softAPIP(): WiFiAPClass
- softAPmacAddress(): WiFiAPClass
- softAPPSK(): WiFiAPClass
- softAPSSID(): WiFiAPClass
- SSID(): WiFiScanClass , WiFiSTAClass
- start(): DNSServer
- status(): WiFiClient, WiFiSTAClass
- statusStr(): WiFiSTAClass
- stop(): Client, DNSServer, UDP, WiFiClient, WiFiServer, WiFiUDP
- stopAll(): WiFiClient, WiFiUDP
- stopAllExcept(): WiFiClient, WiFiUDP
- subnetMask() : WiFiSTAClass

- t -

toString(): IPAddress
 transfer(): SPIClass
 transfer16(): SPIClass
 transferRead(): SPIClass
 transferWrite(): SPIClass

- W -

• waitForConnectResult(): WiFiSTAClass

WiFiClient(): WiFiClientWiFiServer(): WiFiServer

• WiFiUDP() : WiFiUDP

 write(): Client, HardwareSerial, Print, TwoWire, UDP, WiFiClient, WiFiServer, WiFiUDP

_ ~ _

~WiFiClient(): WiFiClient~WiFiUDP(): WiFiUDP