## Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_library</td>
<td>SPI library</td>
</tr>
<tr>
<td>WiFi_library</td>
<td>WiFi library</td>
</tr>
<tr>
<td>WiFi_class</td>
<td></td>
</tr>
<tr>
<td>Client_class</td>
<td></td>
</tr>
<tr>
<td>Server_class</td>
<td></td>
</tr>
<tr>
<td>UDP_class</td>
<td></td>
</tr>
<tr>
<td>IPAddress_class</td>
<td></td>
</tr>
<tr>
<td>Wire_Library</td>
<td>Wire Library</td>
</tr>
<tr>
<td>IO</td>
<td>Digital Analog and Advanced I/O</td>
</tr>
<tr>
<td>Serial</td>
<td>Serial</td>
</tr>
<tr>
<td>Stream</td>
<td>Stream</td>
</tr>
<tr>
<td>Math</td>
<td>Math</td>
</tr>
</tbody>
</table>

Generated by doxygen 1.8.14
<table>
<thead>
<tr>
<th>SPI_library</th>
<th>Functions</th>
</tr>
</thead>
</table>

SPI library. More...
## Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>SPIClass::begin (void)</code></td>
<td>Initialize the SPI instance.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::end (void)</code></td>
<td>Deinitialize the SPI instance and stop it.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::beginTransaction (SPISettings settings)</code></td>
<td>This function should be used to configure the SPI instance in case you don’t use the default parameters set by the <code>begin()</code> function.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::endTransaction (void)</code></td>
<td>settings associated to the SPI instance.</td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td><code>SPIClass::transfer (uint8_t _data)</code></td>
<td>Transfer one byte on the SPI bus.</td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td><code>SPIClass::transfer16 (uint16_t _data)</code></td>
<td>Transfer two bytes on the SPI bus in 16 bits format.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::transferWrite (void *buf, size_t _count)</code></td>
<td>send several bytes.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::transferRead (void *buf, size_t _count)</code></td>
<td>receive several bytes.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>SPIClass::transfer (void *bufout, void *bufin, size_t _count)</code></td>
<td>Transfer several bytes. One buffer contains the data to send and another one will contains the data received.</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void <strong>SPIClass::setBitOrder</strong> (BitOrder)</td>
<td>Deprecated function. Configure the bit order: MSB first or LSB first.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void <strong>SPIClass::setDataMode</strong> (uint8_t _mode)</td>
<td>Deprecated function. Configure the data mode (clock polarity and clock phase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void <strong>SPIClass::setFrequency</strong> (uint32_t freq)</td>
<td>Configure the spi frequency.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

SPI library.
Function Documentation
**begin()**

```c
void SPIClass::begin ( void )
```

Initialize the SPI instance.

**Parameters**

- `[in] none`

**Returns**

- none

**Note**

- none
**beginTransaction()**

```c
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**

```
none
```
◆ end()

```cpp
void SPIClass::end ( void )
```

Deinitialize the SPI instance and stop it.

**Parameters**

[in] none

**Returns**

none

**Note**

none
endTransaction()

```cpp
void SPI::endTransaction ( void )
```

settings associated to the SPI instance.

**Parameters**

[in] none

**Returns**

none

**Note**

none
setBitOrder()

```cpp
void SPI::setBitOrder ( BitOrder _bitOrder )
```

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

**Parameters**

[in] `_bitOrder` MSBFIRST or LSBFIRST

**Returns**

none

**Note**

none
◆ setDataMode()

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

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

**Parameters**

[in] `_mode` SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3

**Returns**

none

**Note**

none
`setFrequency()`

```c
void SPI::setFrequency ( uint32_t freq )
```

Configure the spi frequency.

**Parameters**

- `[in] freq` max 20MHz

**Returns**

- none

**Note**

- none
Transfer one byte on the SPI bus.

**Parameters**

- `[in]` **data** byte to send.

**Returns**

byte received from the slave

**Note**

none
void SPI::transfer ( void * _buf,
                      size_t _count )

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
   none
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**

- `none`
◆ transfer16()

```cpp
uint16_t SPI::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**
- none
◆ transferRead()

```c
void SPI::transferRead ( void * _buf,
                         size_t _count
)
```

receive several bytes.

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

**Returns**
- none

**Note**
- none
void SPIClass::transferWrite ( void * _buf,
                             size_t _count
)

send several bytes.

Parameters

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

Returns

    none

Note

    none
WiFi library. More...
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi_class</td>
</tr>
<tr>
<td>Client_class</td>
</tr>
<tr>
<td>Server_class</td>
</tr>
<tr>
<td>UDP_class</td>
</tr>
<tr>
<td>IPAddress_class</td>
</tr>
</tbody>
</table>
Detailed Description

WiFi library.

IPAddress class.

UDP class.

Server class.

Client class.

WiFi class.

Generated by doxygen 1.8.14
<table>
<thead>
<tr>
<th>WiFi_class</th>
<th>WiFi_library</th>
</tr>
</thead>
</table>

Functions
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <strong>WiFiSTAClass::begin</strong> (const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)</td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td>int <strong>WiFiSTAClass::begin</strong> (char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)</td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td>int <strong>WiFiSTAClass::begin</strong> ()</td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::reconnect</strong> ()</td>
<td>This function is used to reconnect the AP. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::disconnect</strong> (bool wifioff=false)</td>
<td>This function is used to disconnect the wifi. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::isConnected</strong> ()</td>
<td>This function is used to get the wifi mode connect status. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::setAutoConnect</strong> (bool autoConnect)</td>
<td>This function is used to set auto connect flag. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::getAutoConnect</strong> ()</td>
<td>This function is used to get auto connect flag. More...</td>
</tr>
<tr>
<td>bool <strong>WiFiSTAClass::setAutoReconnect</strong> (bool autoReconnect)</td>
<td>This function is used to set auto reconnect flag. More...</td>
</tr>
<tr>
<td>Function Type</td>
<td>Function Name</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>bool</td>
<td>WiFiSTAClass::getAutoReconnect ()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>WiFiSTAClass::waitForConnectResult ()</td>
</tr>
<tr>
<td>IPAddress</td>
<td>WiFiSTAClass::localIP ()</td>
</tr>
<tr>
<td>char *</td>
<td>WiFiSTAClass::macAddress ()</td>
</tr>
<tr>
<td>char *</td>
<td>WiFiSTAClass::macAddressStr ()</td>
</tr>
<tr>
<td>IPAddress</td>
<td>WiFiSTAClass::subnetMask ()</td>
</tr>
<tr>
<td>IPAddress</td>
<td>WiFiSTAClass::gatewayIP ()</td>
</tr>
<tr>
<td>IPAddress</td>
<td>WiFiSTAClass::dnsIP (uint8_t dns_no=0)</td>
</tr>
<tr>
<td>char *</td>
<td>WiFiSTAClass::hostname ()</td>
</tr>
<tr>
<td>bool</td>
<td>WiFiSTAClass::hostname (char *aHostname)</td>
</tr>
<tr>
<td>bool</td>
<td>WiFiSTAClass::hostname (const char *aHostname)</td>
</tr>
<tr>
<td>wl_status_t</td>
<td>WiFiSTAClass::status ()</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>char * WiFiSTAClass::statusStr ()</code></td>
<td>This function is used to get the status during Station mode. More...</td>
</tr>
<tr>
<td><code>char * WiFiSTAClass::SSID () const</code></td>
<td>This function is used to get the SSID used by the module. More...</td>
</tr>
<tr>
<td><code>char * WiFiSTAClass::psk () const</code></td>
<td>This function is used to get the psk used by the module. More...</td>
</tr>
<tr>
<td><code>uint8_t * WiFiSTAClass::BSSID ()</code></td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td><code>char * WiFiSTAClass::BSSIDstr ()</code></td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td><code>int32_t WiFiSTAClass::RSSI ()</code></td>
<td>This function is used to get the RSSI. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation
```cpp
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.

**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 parameters is invalid, negative is returned. Otherwise, wifi status is returned.

**Note**
**begin() [2/3]**

```c
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.

**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 parameters is invalid, negative is returned. Otherwise, wifi status is returned.

**Note**
begin() [3/3]

```cpp
int WiFiSTAClass::begin (void)
```

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

**Parameters**

[in] None.

**Returns**

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

**Note**
BSSID()

uint8_t * WiFiSTA::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.

Note
◆ 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.

| Note |
**disconnect()**

```cpp
bool WiFiSTAClass::disconnect ( bool wifioff = false )
```

This function is used to disconnect the wifi.

**Parameters**

[in] `wifioff` Specify the parameter.

**Returns**

true is returned.

| Note |
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

Note
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.

Note
◆ **getAutoReconnect()**

```cpp
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.

**Note**
◆ getwayIP()

**IPAddress WiFiSTAClass::getwayIP ( )**

This function is used to get gateway IP address.

**Parameters**

[**in**] **None**

**Returns**

The gateway's IPv4 address.

**Note**
isConnected()

```cpp
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.

**Note**
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.

Note
macAddress()

char * WiFiSTAClass::macAddress ( )

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

Parameters

[in] None

Returns

The MAC address.

Note
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.

**Note**
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.

**Note**
# reconnect()

```cpp
bool WiFiSTAClass::reconnect()
```

This function is used to reconnect the AP.

**Parameters**

- `[in]` *None*

**Returns**

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

**Note**
◆ RSSI()

int32_t WiFiSTA::RSSI()

This function is used to get the RSSI.

**Parameters**

[in] None

**Returns**

the value of rssi in this connect.

| Note |
**setAutoConnect()**

```cpp
bool WiFiSTA::setAutoConnect ( bool autoConnect )
```

This function is used to set auto connect flag.

**Parameters**

[in] `autoReconnect` Specify the auto-connect flag.

**Returns**

true - set successfully.

**Note**
setAutoReconnect()

```cpp
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.

**Note**
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.

**Note**
function status()


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

Parameters

[in] None

Returns

The status of WiFi Mode

Note
**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.

**Note**
**subnetMask()**

**IPAddress WiFiSTAClass::subnetMask ()**

This function is used to get subnet mask.

**Parameters**

[in] None

**Returns**

The sub-net mask.

**Note**
waitForConnectResult()

```c
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
<table>
<thead>
<tr>
<th><strong>Client_class</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WiFi_library</strong></td>
</tr>
</tbody>
</table>
### 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...

**virtual 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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual size_t WiFiClient::write_P</code></td>
<td>(PGM_P buf, size_t size)</td>
</tr>
<tr>
<td><code>size_t WiFiClient::write</code></td>
<td>(Stream &amp;stream)</td>
</tr>
<tr>
<td><code>size_t WiFiClient::write</code></td>
<td>(Stream &amp;stream, size_t unitSize) <strong>attribute</strong>((deprecated))</td>
</tr>
<tr>
<td><code>virtual int WiFiClient::available()</code></td>
<td>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...</td>
</tr>
<tr>
<td><code>virtual int WiFiClient::read()</code></td>
<td>Read the next byte received from the server the client is connected to (after the last call to <code>read()</code>). More...</td>
</tr>
<tr>
<td><code>virtual int WiFiClient::read(uint8_t *buf, size_t size)</code></td>
<td>Read the next byte received from the server the client is connected to (after the last call to <code>read()</code>). More...</td>
</tr>
<tr>
<td><code>virtual int WiFiClient::peek()</code></td>
<td>Read a byte from the file without advancing to the next one. That is, successive calls to <code>peek()</code> will return the same value, as will the next call to <code>read()</code>. More...</td>
</tr>
<tr>
<td><code>virtual size_t WiFiClient::peekBytes(uint8_t *buffer, size_t length)</code></td>
<td></td>
</tr>
<tr>
<td><code>size_t WiFiClient::peekBytes(char *buffer, size_t length)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual void WiFiClient::flush()</code></td>
<td>Discard any bytes that have been written to the client but not yet read. More...</td>
</tr>
<tr>
<td><code>virtual void WiFiClient::stop()</code></td>
<td>This function is used to disconnect from the server. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>virtual uint8_t WiFiClient::connected ()</code></td>
<td>Whether or not the client is connected. More...</td>
</tr>
<tr>
<td><code>virtual WiFiClient::operator bool ()</code></td>
<td></td>
</tr>
<tr>
<td><code>IPAddress WiFiClient::remoteIP ()</code></td>
<td>This function is used to get the IP address of the remote connection. More...</td>
</tr>
<tr>
<td><code>uint16_t WiFiClient::remotePort ()</code></td>
<td>This function is used to get the port of the remote connection. More...</td>
</tr>
<tr>
<td><code>IPAddress WiFiClient::localIP ()</code></td>
<td>This function is used to get the IP address of the local tcp connection. More...</td>
</tr>
<tr>
<td><code>uint16_t WiFiClient::localPort ()</code></td>
<td>This function is used to get the port of the local tcp connection. More...</td>
</tr>
<tr>
<td><code>bool WiFiClient::getNoDelay ()</code></td>
<td>This function is used to get whether no delay of the tcp connection. More...</td>
</tr>
<tr>
<td><code>void WiFiClient::setNoDelay (bool nodelay)</code></td>
<td>This function is used to set no delay for the tcp connection. More...</td>
</tr>
<tr>
<td><code>static void WiFiClient::setLocalPortStart (uint16_t port)</code></td>
<td>This function is used to set local port number. More...</td>
</tr>
<tr>
<td><code>size_t WiFiClient::availableForWrite ()</code></td>
<td>This function is used to get the length that can be written. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>static void WiFiClient::stopAll ()</code></td>
<td>This function is used to stop all WiFiClient session. More...</td>
</tr>
<tr>
<td><code>static void WiFiClient::stopAllExcept (WiFiClient *c)</code></td>
<td>This function is used to stop all WiFiClient session without exC. More...</td>
</tr>
<tr>
<td><code>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)</code></td>
<td>This function is used to set keep alive. More...</td>
</tr>
<tr>
<td><code>bool WiFiClient::isKeepAliveEnabled () const</code></td>
<td>This function is used to get whether enable keep alive. More...</td>
</tr>
<tr>
<td><code>uint16_t WiFiClient::getKeepAliveIdle () const</code></td>
<td>This function is used to get idle time interval. More...</td>
</tr>
<tr>
<td><code>uint16_t WiFiClient::getKeepAliveInterval () const</code></td>
<td>This function is used to get keep alive time interval. More...</td>
</tr>
<tr>
<td><code>uint8_t WiFiClient::getKeepAliveCount () const</code></td>
<td>This function is used to get keep alive count. More...</td>
</tr>
<tr>
<td><code>void WiFiClient::disableKeepAlive ()</code></td>
<td>This function is used to set disable keep alive. More...</td>
</tr>
</tbody>
</table>
## Friends

class WiFiServer
Detailed Description

Function Documentation
available()

```cpp
int WiFiClient::available ( void )
```

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 Client.
**availableForWrite()**

```cpp
template
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

**Note**
connect() [1/2]

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

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**

Implements `Client`. 
◆ connect() [2/2]

```cpp
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.

**Parameters**
- `[in] host` the domain name the client will connect to (string, ex.:"arduino.cc")
- `[in] port` the port that the client will connect to (int)
- `[out] None`

**Return values**
- 1 the connection succeeds
- 0 the connection failed

**Note**
Implements **Client**.
**connected()**

```c
uint8_t WiFiClient::connected()
```

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.

Implements *Client*. 
**disableKeepAlive()**

```c
void WiFiClient::disableKeepAlive ( )
```

This function is used to set disable keep alive.

**Parameters**

- `[in] None`
- `[out] None`

**Returns**

None

**Note**
◆ flush()

```cpp
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.

Implements **Client**.
## getKeepAliveCount()

```cpp
uint8_t WiFiClient::getKeepAliveCount() const
```

This function is used to get keep alive count.

**Parameters**
- `[in] None`
- `[out] None`

**Return values**
- `keep alive count`

**Note**
getKeepAliveIdle()

```cpp
uint16_t WiFiClient::getKeepAliveIdle() const
```

This function is used to get idle time interval.

**Parameters**

- `[in]` None
- `[out]` None

**Return values**

- `idle` time interval

**Note**
**getKeepAliveInterval()**

```cpp
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

**Note**
**getNoDelay()**

```cpp
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

**Note**
◆ isKeepAliveEnabled()

```cpp
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

**Note**
◆ keepAlive()

```c
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.

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

**Return values**
- **None**

**Note**
**localIP()**

**IPAddress WiFiClient::localIP ( )**

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

**Parameters**

[in] None

[out] None

**Return values**

the IP address (4 bytes)

**Note**
localPort()

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

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

**Parameters**
- `[in] None`
- `[out] None`

**Return values**
- `the local port number`

**Note**
peek()

```c
int WiFiClient::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()`.

**Parameters**

- **[in]** None
- **[out]** None

**Return values**

- `-1` none is available
- **other** the next byte or character

**Note**

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

Implements `Client`.
◆ read() [1/2]

```cpp
int WiFiClient::read ( void )
```

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**

-1 none is available.
other The next character

**Note**

`read()` inherits from the **Stream** utility class

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

**Parameters**

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

**Return values**

- `-1` none is available.
- `other` The next byte

**Note**

`read()` inherits from the `Stream` utility class

Implements `Client`.
Function: `remoteIP()`

Function: WiFiClient::remoteIP()

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

**Parameters**
- `[in]` None
- `[out]` None

**Return values**
- `the` IP address (4 bytes)

**Note**
remotePort()

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

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

**Parameters**

- [in] None
- [out] None

**Return values**

- The port number

**Note**
setLocalPortStart()

```cpp
static void WiFiClient::setLocalPortStart ( uint16_t port )
```

This function is used to set local port number.

**Parameters**
- [in] `port` number
- [out] `None`

**Returns**
- `None`

**Note**
◆ setNoDelay()

```cpp
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

| Note |
status()

uint8_t WiFiClient::status()

return tcp status of WiFiClient.

Parameters
[in] None
[out] None

Return values
tcp status

Note
stop()

```c
void WiFiClient::stop()
```

This function is used to disconnect from the server.

**Parameters**

[in] None

[out] None

**Returns**

None

**Note**

Implements **Client**.
**stopAll()**

```cpp
void WiFiClient::stopAll()
```

This function is used to stop all `WiFiClient` session.

**Parameters**

- `[in]` None
- `[out]` None

**Return values**

None

**Note**
◆ stopAllExcept()

```cpp
void WiFiClient::stopAllExcept (WiFiClient * except)
```

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

**Parameters**
- `[in]` None
- `[out]` None

**Return values**
- None

**Note**
**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

| Note |
write() [1/2]

size_t WiFiClient::write ( uint8_t b )  

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 Client.
write() [2/2]

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.

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

Implementes Client.
◆ ~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
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server_class</strong></td>
</tr>
<tr>
<td><strong>WiFi_library</strong></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WiFiServer::WiFiServer (IPAddress addr, uint16_t port)</code></td>
<td>This constructor is used to init <code>WiFiServer</code> object with the address and port specify by caller. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>WiFiServer::WiFiServer (uint16_t port)</code></td>
<td>This constructor is used to init <code>WiFiServer</code> object with port specify by caller. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool WiFiServer::hasClient ()</code></td>
<td></td>
</tr>
<tr>
<td><code>void WiFiServer::begin ()</code></td>
<td>This function is used to start the <code>WiFiServer</code>. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WiFiServer::begin (uint16_t port)</code></td>
<td>This function is used to start the <code>WiFiServer</code>. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WiFiServer::setNoDelay (bool nodelay)</code></td>
<td>This function is used to set no-delay flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool WiFiServer::getNoDelay ()</code></td>
<td>This function is used to get no-delay flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual size_t WiFiServer::write (uint8_t)</code></td>
<td>This function is used to send the message (one byte) to peer. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual size_t WiFiServer::write (const uint8_t *buf, size_t size)</code></td>
<td>This function is used to send the message to peer. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint8_t WiFiServer::status ()</code></td>
<td></td>
</tr>
<tr>
<td><code>void WiFiServer::close ()</code></td>
<td>This function is used to close the connection. <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>void WiFiServer::stop ()</td>
<td>This function is used to close the connection. More...</td>
</tr>
<tr>
<td>long WiFiServer::_accept (tcp_pcb *newpcb, long err)</td>
<td></td>
</tr>
<tr>
<td>static err_t WiFiServer::_s_accept (void *arg, tcp_pcb *newpcb, err_t err)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation
begin() [1/2]

```c
void WiFiServer::begin ( void ) virtual
```

This function is used to start the `WiFiServer`.

**Parameters**

- **[in]** *none*
- **[out]**

Implements `Server`. 
begin() [2/2]

```cpp
void WiFiServer::begin(uint16_t port)
```

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

**Parameters**

- `port` Specify the port used by the object.

**Returns**

None

**Note**
◆ close()

```cpp
void WiFiServer::close()
```

This function is used to close the connection.

**Parameters**

[in] None

**Returns**

None

**Note**
**getNoDelay()**

```cpp
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.

**Note**
setNoDelay()

```cpp
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

**Note**
void WiFiServer::stop ()

This function is used to close the connection.

**Parameters**

[in] None

**Returns**

None

**Note**
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

**Note**
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

**Note**
◆ write() [1/2]

size_t WiFiServer::write ( uint8_t b )  

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()

```cpp
size_t WiFiServer::write ( const uint8_t * buffer,
                          size_t size )
```

This function is used to send the message to peer.

**Parameters**

- **buf** [in] Specify the buffer which will be sent to peer.
- **size** [in] Specify the length which will be sent.

**Returns**

The length of the message sent to peer.

**Note**

Reimplemented from [Print](#).

---

Generated by **doxygen** 1.8.14
<table>
<thead>
<tr>
<th>W60X_Arduino</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP_class</strong></td>
</tr>
<tr>
<td><strong>WiFi_library</strong></td>
</tr>
</tbody>
</table>
**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFiUDP::WiFiUDP ()</td>
<td>This function is constructor, it's used to creates a named instance of the WiFiUDP class that can send and receive UDP messages.</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::WiFiUDP (const WiFiUDP &amp;other)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFiUDP &amp; WiFiUDP::operator= (const WiFiUDP &amp;rhs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::~WiFiUDP ()</td>
<td>This function is deconstructor, it's used to release WiFiUDP class.</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::operator bool () const</td>
<td></td>
<td></td>
</tr>
<tr>
<td>virtual uint8_t WiFiUDP::begin (uint16_t port)</td>
<td>This function is used to initializes the WiFiUDP library and network settings, Starts UDP socket, listening at local port.</td>
<td></td>
</tr>
<tr>
<td>virtual void WiFiUDP::stop ()</td>
<td>This function is used to disconnect from the server. Release any resource being used during the UDP session.</td>
<td></td>
</tr>
<tr>
<td>uint8_t WiFiUDP::beginMulticast (IPAddress interfaceAddr, IPAddress multicast, uint16_t port)</td>
<td>This function is used to join a multicast group and listen on the given port.</td>
<td></td>
</tr>
<tr>
<td>virtual int WiFiUDP::beginPacket (IPAddress ip, uint16_t port)</td>
<td>This function is used to starts a connection to write UDP data to the remote connection.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::beginPacket</td>
<td>This function is used to start a connection to write UDP data to the remote connection. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::beginPacketMulticast</td>
<td>This function is used to start building up a packet to send to the multicast address. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::endPacket</td>
<td>This function is used to called after writing UDP data to the remote connection. It finishes off the packet and send it. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::write</td>
<td>This function is used to writes UDP data to the remote connection. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::write</td>
<td>This function is used to writes UDP data to the remote connection. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::parsePacket</td>
<td>It starts processing the next available incoming packet, checks for the presence of a UDP packet, and reports the size. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::available</td>
<td>Get the number of bytes (characters) available for reading from the buffer. This is is data that's already arrived. More...</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP::read</td>
<td>Reads UDP data from the specified buffer. If no arguments are given, it will return the next character</td>
<td></td>
</tr>
</tbody>
</table>
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...
<table>
<thead>
<tr>
<th>static void WiFiUDP::stopAll ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to stop all WiFiUDP session. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void WiFiUDP::stopAllExcept (WiFiUDP *exC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to stop all WiFiUDP session without exC. More...</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation
◆ available()  

```cpp
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**

- 0 parsePacket hasn't been called yet
- **other** 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.

Implements **UDP**.
begin()

```cpp
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.

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

**Return values**
- 1 successful
- 0 there are no sockets available to use

**Note**
- Implements UDP.
beginMulticast()

```cpp
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

- **interfaceAddress** [in] the local IP address of the interface that should be used, use WiFi.localIP() or WiFi.softAPIP() depending on the interface you need
- **multicast** [in] multicast group
- **port** [in] port number
- **None** [out] None

### Return values

- 1 successful
- 0 failed

### Note
◆ beginPacket( ) [1/2]

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

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

**Parameters**
- **host** the address of the remote host. It accepts a character string or an IPAddress
- **port** the port of the remote connection (int)
- **None**

**Return values**
- 1 successful
- 0 there was a problem with the supplied IP address or port

**Note**
Implements UDP.
◆ beginPacketMulticast()

```
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.

**Parameters**

- **[in]** `multicastAddress` multicast address to send to
- **[in]** `port` port number
- **[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]** `ttl` multicast packet TTL (default is 1)
- **[out]** `None`

**Return values**

- **1** successful
- **0** there was a problem with the supplied IP address or port

**Note**
◆ destinationIP()

**IPAddress WiFiUDP::destinationIP ( )**

This function is used to distinguish multicast and ordinary packets.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>None</td>
</tr>
<tr>
<td>[out]</td>
<td>None</td>
</tr>
</tbody>
</table>

**Return values**

the destination address for incoming packets

**Note**
`endPacket()`

```cpp
int WiFiUDP::endPacket ( )
```

This function is used to call 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**
Implements UDP.
flush()

```cpp
virtual void WiFiUDP::flush ()
```

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.

Implements UDP.
**localPort()**

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

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

**Parameters**

[in] None

[out] None

**Return values**

the local port for outgoing packets

| Note |
**parsePacket()**

```c
int WiFiUDP::parsePacket ()
```

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**

- **0**  no packets are available
- **other** the size of the packet in bytes

**Note**

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

Implements **UDP**.
◆ peek()

```cpp
int WiFiUDP::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()`.

**Parameters**

- **[in]** None
- **[out]** None

**Return values**

- `-1` none is available
- `other` the next byte or character

**Note**

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

Implements `UDP`. 

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**
- `-1` no buffer is available
- `other` the characters in the buffer (char)

**Note**

Implements **UDP**.
**read()** [2/3]

```cpp
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.

**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**

Implements **UDP**.
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.

**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**

Implements **UDP**.
remoteIP()

**IPAddress WiFiUDP::remoteIP()**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>None</td>
</tr>
<tr>
<td>[out]</td>
<td>None</td>
</tr>
</tbody>
</table>

**Return values**

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

**Note**

This function must be called after `parsePacket()`.

Implements **UDP**.
remotePort()

```cpp
uint16_t WiFiUDP::remotePort ()
```

This function is used to get 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()`.

Implements UDP.
stop()

void WiFiUDP::stop ( )

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**

Implements UDP.
stopAll()

```c
void WiFiUDP::stopAll()
```

This function is used to stop all WiFiUDP session.

**Parameters**
- `[in] None`
- `[out] None`

**Return values**
- `None`

**Note**
◆ stopAllExcept()

```cpp
void WiFiUDP::stopAllExcept (WiFiUDP * exC )
```

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

**Parameters**

- *[in]* None
- *[out]* None

**Return values**

None

**Note**
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

**Note**
size_t WiFiUDP::write ( uint8_t byte )

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 UDP.
This function is used to write 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.

Implements UDP.
◆ ~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
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPAddress_class</td>
</tr>
<tr>
<td>WiFi_library</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPAddress::IPAddress ()</strong>&lt;br&gt;This constructor function is used to construct IPAddress object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>IPAddress::IPAddress (uint8_t first_oct, uint8_t sec_oct, uint8_t third_oct, uint8_t fourth_oct)</strong>&lt;br&gt;This constructor function is used to construct IPAddress object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>IPAddress::IPAddress (uint32_t address)</strong>&lt;br&gt;This constructor function is used to construct IPAddress object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>IPAddress::IPAddress (const uint8_t *address)</strong>&lt;br&gt;This constructor function is used to construct IPAddress object.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>IPAddress::operator uint32_t () const</strong>&lt;br&gt;This operator overloading function is used to overloading 'uint32_t' operator.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>bool IPAddress::operator== (const IPAddress &amp;addr) const</strong>&lt;br&gt;This operator overloading function is used to overloading '==' operator.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>bool IPAddress::operator== (uint32_t addr) const</strong>&lt;br&gt;This operator overloading function is used to overloading '==' operator.</td>
<td>More...</td>
</tr>
<tr>
<td>*<em>bool IPAddress::operator== (const uint8_t <em>addr) const</em></em>&lt;br&gt;This operator overloading function is used to overloading '==' operator.</td>
<td>More...</td>
</tr>
<tr>
<td><strong>uint8_t IPAddress::operator[] (int index) const</strong></td>
<td></td>
</tr>
</tbody>
</table>
This operator overloading function is used to overloading '[]' operator. **More...**

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t &amp;</td>
<td><strong>IPAddress::operator[]</strong> (int index)</td>
<td>This operator overloading function is used to overloading '[]' operator. <strong>More...</strong></td>
</tr>
<tr>
<td>IPAddress &amp;</td>
<td><strong>IPAddress::operator=</strong> (const uint8_t *address)</td>
<td>This operator overloading function is used to overloading '=' operator. <strong>More...</strong></td>
</tr>
<tr>
<td>IPAddress &amp;</td>
<td><strong>IPAddress::operator=</strong> (uint32_t address)</td>
<td>This operator overloading function is used to overloading '=' operator. <strong>More...</strong></td>
</tr>
<tr>
<td>String</td>
<td><strong>IPAddress::toString</strong> () const</td>
<td>This function is used to transform from <strong>IPAddress</strong> object to String object. <strong>More...</strong></td>
</tr>
<tr>
<td>bool</td>
<td><strong>IPAddress::fromString</strong> (const char *address)</td>
<td>This function is used to create the object from a string buffer. <strong>More...</strong></td>
</tr>
<tr>
<td>bool</td>
<td><strong>IPAddress::fromString</strong> (const String &amp;address)</td>
<td>This function is used to create the object from a String object. <strong>More...</strong></td>
</tr>
<tr>
<td>virtual size_t</td>
<td><strong>IPAddress::printTo</strong> (Print &amp;p) const</td>
<td>This virtual function is used to called by print/println function. <strong>More...</strong></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation
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

**Note**
**fromString() [2/2]**

```cpp
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

**Note**
**IPAddress()** [1/4]

IPAddress::IPAddress ( )

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

**Parameters**

[**in**] None

**Returns**

None

**Note**
◆ IPAddress() [2/4]

```cpp
IPAddress::IPAddress ( uint8_t  first_octet,
                     uint8_t  second_octet,
                     uint8_t  third_octet,
                     uint8_t  fourth_octet
                   )
```

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

**Parameters**

- `[in] first_oct` Specify the first_oct - uint8_t
- `[in] sec_oct` Specify the sec_oct - uint8_t
- `[in] third_oct` Specify the third_oct - uint8_t
- `[in] fourth_oct` Specify the fourth_oct - uint8_t

**Returns**

None

**Note**
**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

**Note**
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

**Note**
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.

| Note |
operator=() [1/2]

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

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

Parameters
   [in] address Specify the address - uint8_t *

Returns
   the target IPAddress object.

Note
◆ operator=() [2/2]

`IPAddress & IPAddress::operator= ( uint32_t address )`

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

**Parameters**

[in] `address` Specify the address - `uint32_t`

**Returns**

the target `IPAddress` object.

**Note**
**operator==() [1/3]**

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

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

**Parameters**

[in] `addr` Specify the address - `IPAddress`

**Returns**

`bool`

**Note**
operator==( )  [2/3]

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

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

**Parameters**

[in] `addr` Specify the address - `uint32_t`

**Returns**

`bool`

**Note**


◆ operator==( ) [3/3]

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

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

**Parameters**

[in] `addr` Specify the address - `uint8_t*`

**Returns**

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

**Note**
operator[]( ) [1/2]

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

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

**Parameters**

- `[in] index` Specify the address value of index.

**Returns**

the target value of `IPAddress` object.

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

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

**Parameters**

[in] **index** Specify the address value of index.

**Returns**

the target value of **IPAddress** object.

| Note |
◆ printTo()

```cpp
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**
<table>
<thead>
<tr>
<th>W60X_Arduino</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wire_Library</strong></td>
<td></td>
</tr>
</tbody>
</table>

Wire Library. [More...](#)
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void TwoWire::begin</code></td>
<td>(int sda, int scl)</td>
</tr>
<tr>
<td><code>void TwoWire::pins</code></td>
<td>(int sda, int scl)</td>
</tr>
<tr>
<td></td>
<td><strong>attribute</strong>((deprecated))</td>
</tr>
<tr>
<td><code>void TwoWire::begin</code></td>
<td>()</td>
</tr>
<tr>
<td><code>void TwoWire::begin</code></td>
<td>(uint8_t)</td>
</tr>
<tr>
<td><code>void TwoWire::begin</code></td>
<td>(int)</td>
</tr>
<tr>
<td><code>void TwoWire::setClock</code></td>
<td>(uint32_t)</td>
</tr>
<tr>
<td><code>void TwoWire::setClockStretchLimit</code></td>
<td>(uint32_t)</td>
</tr>
<tr>
<td><code>void TwoWire::beginTransmission</code></td>
<td>(uint8_t)</td>
</tr>
<tr>
<td><code>void TwoWire::beginTransmission</code></td>
<td>(int)</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::endTransmission</code></td>
<td>(void)</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::endTransmission</code></td>
<td>(uint8_t)</td>
</tr>
<tr>
<td><code>size_t TwoWire::requestFrom</code></td>
<td>(uint8_t address, size_t size, bool sendStop)</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::status</code></td>
<td>()</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::requestFrom</code></td>
<td>(uint8_t, uint8_t)</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::requestFrom</code></td>
<td>(uint8_t, uint8_t, uint8_t)</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::requestFrom</code></td>
<td>(int, int)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>uint8_t TwoWire::requestFrom (int, int, int)</code></td>
<td>This pure virtual function is used to define the operation that writes binary data. More...</td>
</tr>
<tr>
<td><code>virtual size_t TwoWire::write (uint8_t)</code></td>
<td>This function is used to write buffer to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>virtual size_t TwoWire::write (const uint8_t *, size_t)</code></td>
<td>This function is used to write buffer to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>virtual int TwoWire::available (void)</code></td>
<td><code>available()</code> gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...</td>
</tr>
<tr>
<td><code>virtual int TwoWire::read (void)</code></td>
<td><code>read()</code> reads characters from an incoming stream to the buffer. More...</td>
</tr>
<tr>
<td><code>virtual int TwoWire::peek (void)</code></td>
<td>Read a byte from the file without advancing to the next one. That is, successive calls to <code>peek()</code> will return the same value, as will the next call to <code>read()</code>. More...</td>
</tr>
<tr>
<td><code>virtual void TwoWire::flush (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>void TwoWire::onReceive (void(*)(int))</code></td>
<td></td>
</tr>
<tr>
<td><code>void TwoWire::onRequest (void(*)(void))</code></td>
<td></td>
</tr>
<tr>
<td><code>size_t TwoWire::write (unsigned long n)</code></td>
<td></td>
</tr>
<tr>
<td><code>size_t TwoWire::write (long n)</code></td>
<td></td>
</tr>
<tr>
<td><code>size_t TwoWire::write (unsigned int n)</code></td>
<td></td>
</tr>
<tr>
<td><code>size_t TwoWire::write (int n)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Wire Library.
Function Documentation
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 |

Implements Stream.
**peek()**

```c
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()`.

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**

Implements **Stream**.
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**

Implements Stream.
◆ write() [1/2]

size_t TwoWire::write ( uint8_t val )

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.
size_t TwoWire::write (const uint8_t *buffer, size_t size)

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**.
Digital Analog and Advanced I/O. More...
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void tone(uint8_t _pin, unsigned int frequency)</code></td>
<td>Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void noTone(uint8_t _pin)</code></td>
<td>Stops the generation of a square wave triggered by <code>tone()</code>. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void pinMode(uint32_t ulPin, uint32_t ulMode)</code></td>
<td></td>
</tr>
<tr>
<td><code>void digitalWrite(uint32_t ulPin, uint32_t ulVal)</code></td>
<td></td>
</tr>
<tr>
<td><code>int digitalRead(uint32_t ulPin)</code></td>
<td></td>
</tr>
<tr>
<td><code>int analogRead(unsigned char pin)</code></td>
<td>Reads the value from the specified analog pin. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
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()

```c
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()**

```c
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()

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HardwareSerial::HardwareSerial</strong> (int serial_no)</td>
<td>This constructor is used to init hardware serial. More...</td>
</tr>
<tr>
<td><strong>HardwareSerial::HardwareSerial</strong> (int serial_no, bool mul_flag)</td>
<td>This constructor is used to init hardware serial. More...</td>
</tr>
<tr>
<td><strong>HardwareSerial::HardwareSerial</strong> ()</td>
<td>This constructor is used to init hardware serial. More...</td>
</tr>
<tr>
<td><strong>void HardwareSerial::begin</strong> ()</td>
<td>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...</td>
</tr>
<tr>
<td><strong>void HardwareSerial::begin</strong> (unsigned long baud)</td>
<td>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...</td>
</tr>
<tr>
<td><strong>void HardwareSerial::begin</strong> (unsigned long baud, int modeChoose)</td>
<td>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</td>
</tr>
</tbody>
</table>
0 and 1 with a component that requires a particular baud rate. More...

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual int HardwareSerial::read (void)</code>&lt;br&gt;Reads incoming serial data. <code>&lt;code&gt;read()&lt;/code&gt;</code> inherits from the <code>&lt;code&gt;Stream&lt;/code&gt;</code> utility class. More...</td>
<td></td>
</tr>
<tr>
<td><code>virtual int HardwareSerial::available (void)</code>&lt;br&gt;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). <code>&lt;code&gt;available()&lt;/code&gt;</code> inherits from the <code>&lt;code&gt;Stream&lt;/code&gt;</code> utility class. More...</td>
<td></td>
</tr>
<tr>
<td><code>virtual int HardwareSerial::peek ()</code>&lt;br&gt;Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to <code>&lt;code&gt;peek()&lt;/code&gt;</code> will return the same character, as will the next call to <code>&lt;code&gt;read()&lt;/code&gt;. </code>&lt;code&gt;peek()&lt;/code&gt;<code>inherits from the</code>&lt;code&gt;Stream&lt;/code&gt;` utility class. More...</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Serial.
Function Documentation
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**

Implements `Stream`. 

```cpp
int HardwareSerial::available ( void )
```
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

|Note|
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

**Note**


begin() [3/3]

```cpp
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

**Note**
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

Note
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

Note
This constructor is used to init hardware serial.

**Parameters**

[\texttt{in}] \texttt{none}

**Returns**

None

**Note**
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**.
| Stream | More... |
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual int Stream::available ()=0</code></td>
<td><code>available()</code> gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...</td>
</tr>
<tr>
<td><code>virtual int Stream::read ()=0</code></td>
<td><code>read()</code> reads characters from an incoming stream to the buffer. More...</td>
</tr>
<tr>
<td><code>virtual int Stream::peek ()=0</code></td>
<td>Read a byte from the file without advancing to the next one. That is, successive calls to <code>peek()</code> will return the same value, as will the next call to <code>read()</code>. More...</td>
</tr>
<tr>
<td><code>void Stream::setTimeout (unsigned long timeout)</code></td>
<td><code>setTimeout()</code> sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the <code>Stream</code> class, and is called by any class that inherits from it (Wire, Serial, etc). See the <code>Stream</code> class main page for more information. More...</td>
</tr>
<tr>
<td><code>bool Stream::find (const char *target)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool Stream::find (uint8_t *target)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool Stream::find (const char *target, size_t length)</code></td>
<td><code>find()</code> 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...</td>
</tr>
</tbody>
</table>
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...

<table>
<thead>
<tr>
<th><strong>virtual size_t</strong></th>
<th>*<em>Stream::readBytes (char <em>buffer, size_t length)</em></em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytes()</strong></td>
<td>read characters from a stream into a buffer.</td>
</tr>
<tr>
<td></td>
<td>The function terminates if the determined length has</td>
</tr>
<tr>
<td></td>
<td>been read, or it times out (see <strong>setTimeout()</strong>).</td>
</tr>
<tr>
<td>More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>virtual size_t</strong></th>
<th>*<em>Stream::readBytes (uint8_t <em>buffer, size_t length)</em></em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytes()</strong></td>
<td>read characters from a stream into a buffer.</td>
</tr>
<tr>
<td></td>
<td>The function terminates if the determined length has</td>
</tr>
<tr>
<td></td>
<td>been read, or it times out (see <strong>setTimeout()</strong>).</td>
</tr>
<tr>
<td>More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>size_t</strong></th>
<th>*<em>Stream::readBytesUntil (char terminator, char <em>buffer, size_t length)</em></em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytesUntil()</strong></td>
<td>reads characters from a stream into a buffer.</td>
</tr>
<tr>
<td></td>
<td>The function terminates if the terminator character is detected, the</td>
</tr>
<tr>
<td></td>
<td>determined length has been read, or it times out (see <strong>setTimeout()</strong>).</td>
</tr>
<tr>
<td>More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>size_t</strong></th>
<th>*<em>Stream::readBytesUntil (char terminator, uint8_t <em>buffer, size_t length)</em></em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytesUntil()</strong></td>
<td>reads characters from a stream into a buffer.</td>
</tr>
<tr>
<td></td>
<td>The function terminates if the terminator character is detected, the</td>
</tr>
<tr>
<td></td>
<td>determined length has been read, or it times out (see <strong>setTimeout()</strong>).</td>
</tr>
<tr>
<td>More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>String</strong></th>
<th><strong>Stream::readString ()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readString()</strong></td>
<td>reads characters from a stream into a string. The function terminates if</td>
</tr>
<tr>
<td></td>
<td>it times out (see <strong>setTimeout()</strong>).</td>
</tr>
<tr>
<td>More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>String</strong></th>
<th><strong>Stream::readStringUntil (char terminator)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readStringUntil()</strong></td>
<td>reads characters from a stream into a string.</td>
</tr>
<tr>
<td></td>
<td>The function terminates if the terminator character</td>
</tr>
</tbody>
</table>
is detected or it times out (see `setTimeout()`). More...
Detailed Description

Stream.
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]

```cpp
bool Stream::find ( const char * target )
```

The `find()` function 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`

**Note**
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

**Note**
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

**Note**
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

**Note**
**find()** [5/5]

```cpp
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.

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

**Note**
◆ 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

| Note |
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

| Note |
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

**Note**
**findUntil()**

```cpp
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.

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

**Note**
## parseFloat()

```c
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`

### Note
**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

**Note**
精神文明

**peek()**

```
virtual int Stream::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()`.

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() 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]

```c
size_t Stream::readBytes ( char * buffer,
                          size_t length )
```

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

`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[])
**readBytes()** [2/2]

```cpp
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()`).

`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[])
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**

*character* : the character to search for (char)
*buffer* the buffer to store the bytes in (char[] or byte[])
*length* : the number of bytes to read (int)

**Returns**
The number of bytes placed in the buffer

**Note**
**readBytesUntil() [2/2]**

```cpp
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()`).

`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**
- `[in] character`: the character to search for (char)
- `[in] buffer`: the buffer to store the bytes in (char[] or byte[])
- `[in] length`: the number of bytes to read (int)

**Returns**
- The number of bytes placed in the buffer

**Note**
**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

| Note |
**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()`

```c++
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**


**Returns**

None

---

**Note**

Generated by [doxygen](https://www.doxygen.org/) 1.8.14
<table>
<thead>
<tr>
<th>Math</th>
</tr>
</thead>
</table>

Math. More...
**Functions**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned int</code></td>
<td><code>millis (void)</code></td>
<td>This function is used to get milliseconds since system startup.</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td><code>micros (void)</code></td>
<td>Returns the number of microseconds since the Arduino board began running the current program.</td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>delay (unsigned long ms)</code></td>
<td>This function is used to delay by milliseconds.</td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>delayMicroseconds (unsigned int us)</code></td>
<td>This function is to delay by microseconds.</td>
</tr>
</tbody>
</table>
Detailed Description

Math.
Function Documentation
**delay()**

```c
void delay ( unsigned long ms )
```

This function is used to delay by milliseconds.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] <strong>ms</strong></td>
<td>delay time, unit: ms</td>
</tr>
</tbody>
</table>

**Returns**

- none

**Note**

- none
**delayMicroseconds()**

void delayMicroseconds ( unsigned int \texttt{us} )

This function is to delay by micro seconds.

**Parameters**

\[ \texttt{[in]} \ \texttt{us} \ \text{micro seconds} \]

**Returns**

none

**Note**

none
**micros()**

```c
uint32_t micros ( void )
```

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

**Returns**

- microseconds

**Note**

- none
unsigned int millis ( void )

This function is used to get milliseconds since system startup.

Returns
milliseconds

Note
none
## Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferDataSource</td>
</tr>
<tr>
<td>BufferedStreamDataSource</td>
</tr>
<tr>
<td>Client</td>
</tr>
<tr>
<td>ClientContext</td>
</tr>
<tr>
<td>CloudClass</td>
</tr>
<tr>
<td>DataSource</td>
</tr>
<tr>
<td>DhcpClass</td>
</tr>
<tr>
<td>DNSClient</td>
</tr>
<tr>
<td>DNSServer</td>
</tr>
<tr>
<td>HardwareSerial</td>
</tr>
<tr>
<td>IPAddress</td>
</tr>
<tr>
<td>Print</td>
</tr>
<tr>
<td>Printable</td>
</tr>
<tr>
<td>ProgmemStream</td>
</tr>
<tr>
<td>Server</td>
</tr>
<tr>
<td>SList</td>
</tr>
<tr>
<td>SPIClass</td>
</tr>
<tr>
<td>SPISettings</td>
</tr>
<tr>
<td>Stream</td>
</tr>
<tr>
<td>TwoWire</td>
</tr>
<tr>
<td>UDP</td>
</tr>
<tr>
<td>UdpContext</td>
</tr>
<tr>
<td>W600InnerFlashClass</td>
</tr>
<tr>
<td>WiFiAPClass</td>
</tr>
<tr>
<td>Class</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>WiFiClass</td>
</tr>
<tr>
<td>WiFiClient</td>
</tr>
<tr>
<td>WiFiGenericClass</td>
</tr>
<tr>
<td>WiFiOneshotClass</td>
</tr>
<tr>
<td>WiFiScanClass</td>
</tr>
<tr>
<td>WiFiServer</td>
</tr>
<tr>
<td>WiFiSTAclass</td>
</tr>
<tr>
<td>WiFiUDP</td>
</tr>
</tbody>
</table>

Generated by doxygen 1.8.14
W60X_Arduino

BufferDataSource
Class Reference

Inherits DataSource.
### Public Member Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BufferDataSource</strong> (const uint8_t *data, size_t size)</td>
<td></td>
</tr>
<tr>
<td>size_t available () override</td>
<td></td>
</tr>
<tr>
<td>const uint8_t * get_buffer (size_t size) override</td>
<td></td>
</tr>
<tr>
<td>void release_buffer (const uint8_t *buffer, size_t size) override</td>
<td></td>
</tr>
</tbody>
</table>
### Protected Attributes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>const uint8_t * _data</strong></td>
</tr>
<tr>
<td></td>
<td><strong>const size_t _size</strong></td>
</tr>
<tr>
<td></td>
<td><strong>size_t _pos = 0</strong></td>
</tr>
</tbody>
</table>

Generated by [doxygen](http://doxygen.github.io/1.8.14) 1.8.14
Inherits `DataSource`. 
Public Member Functions

**BufferedStreamDataSource** (TStream &stream, size_t size)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Override</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t available ()</td>
<td>override</td>
<td></td>
</tr>
<tr>
<td>const uint8_t * get_buffer(size_t size)</td>
<td>override</td>
<td></td>
</tr>
<tr>
<td>void release_buffer(const uint8_t *buffer, size_t size)</td>
<td>override</td>
<td></td>
</tr>
</tbody>
</table>
### Protected Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TStream &amp;</td>
<td><code>_stream</code></td>
</tr>
<tr>
<td>std::unique_ptr&lt;uint8_t[]&gt;</td>
<td><code>_buffer</code></td>
</tr>
<tr>
<td>size_t</td>
<td><code>_size</code></td>
</tr>
<tr>
<td>size_t</td>
<td><code>_pos</code> = 0</td>
</tr>
<tr>
<td>size_t</td>
<td><code>_bufferSize</code> = 0</td>
</tr>
<tr>
<td>size_t</td>
<td><code>_streamPos</code> = 0</td>
</tr>
</tbody>
</table>
Inherits **Stream**.

Inherited by **WiFiClient**.
<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual int</td>
<td><code>connect</code></td>
<td><code>IPAddress ip, uint16_t port</code>)=0`</td>
<td>This function is used to connect to the IP address and port specified in the constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual int</td>
<td><code>connect</code></td>
<td><code>const char *host, uint16_t port)=0</code></td>
<td>This function is used to connect to the IP address and port specified in the constructor. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual size_t</td>
<td><code>write</code></td>
<td><code>(uint8_t)=0</code></td>
<td>This function is used to write data to the server the client is connected to. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual size_t</td>
<td><code>write</code></td>
<td><code>(const uint8_t *buf, size_t size)=0</code></td>
<td>This function is used to write data to the server the client is connected to. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual int</td>
<td><code>available</code></td>
<td><code>()=0</code></td>
<td>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). <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual int</td>
<td><code>read</code></td>
<td><code>()=0</code></td>
<td>Read the next byte received from the server the client is connected to (after the last call to <code>read()</code>). <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual int</td>
<td><code>read</code></td>
<td><code>(uint8_t *buf, size_t size)=0</code></td>
<td>Read the next byte received from the server the client is connected to (after the last call to <code>read()</code>). <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual int</td>
<td><code>peek</code></td>
<td><code>()=0</code></td>
<td>Read a byte from the file without advancing to the next one. That is, successive calls to <code>peek()</code> will return the same value, as will the next call to <code>read()</code>. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
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...**
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool find (const uint8_t *target, size_t length)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool find (char target)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool findUntil (const char *target, const char *terminator)</code></td>
<td><code>findUntil()</code> reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
<tr>
<td><code>bool findUntil (const uint8_t *target, const char *terminator)</code></td>
<td><code>findUntil()</code> reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
<tr>
<td><code>bool findUntil (const char *target, size_t targetLen, const char *terminate, size_t termLen)</code></td>
<td><code>findUntil()</code> reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
<tr>
<td><code>bool findUntil (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen)</code></td>
<td><code>findUntil()</code> reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
<tr>
<td><code>long parseInt ()</code></td>
<td><code>parseInt()</code> returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped. More...</td>
</tr>
<tr>
<td><code>float parseFloat ()</code></td>
<td></td>
</tr>
</tbody>
</table>
**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...

<table>
<thead>
<tr>
<th>virtual size_t</th>
<th>readBytes (char *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytes()</strong></td>
<td>reads characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual size_t</th>
<th>readBytes (uint8_t *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytes()</strong></td>
<td>reads characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see setTimeout()). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>readBytesUntil (char terminator, char *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytesUntil()</strong></td>
<td>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...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>readBytesUntil (char terminator, uint8_t *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readBytesUntil()</strong></td>
<td>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...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>readString ()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readString()</strong></td>
<td>reads characters from a stream into a string. The function terminates if it times out (see setTimeout()). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>readStringUntil (char terminator)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readStringUntil()</strong></td>
<td>reads characters from a stream into a string. The function terminates if the terminator...</td>
</tr>
</tbody>
</table>
character is detected or it times out (see `setTimeout()`). More...

**Public Member Functions inherited from Print**

```cpp
int getWriteError ()
This function is used to get write error number. More...
```

```cpp
void clearWriteError ()
This function is used to clear write error number. More...
```

```cpp
size_t write (const char *str)
This function is used to write buffer to the interface defined by the object. More...
```

```cpp
size_t write (const char *buffer, size_t size)
This function is used to write buffer to the interface defined by the object. More...
```

```cpp
size_t print (const String &)
This function is used to print buffer to the interface defined by the object. More...
```

```cpp
size_t print (const char [])
This function is used to print buffer to the interface defined by the object. More...
```

```cpp
size_t print (char)
This function is used to print buffer to the interface defined by the object. More...
```

```cpp
size_t print (unsigned char, int=DEC)
This function is used to print target to the interface defined by the object. More...
```

```cpp
size_t print (int, int=DEC)
This function is used to print target to the interface defined by the object. More...
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t print</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t print</td>
<td>(unsigned int, int=DEC) This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t print</td>
<td>(long, int=DEC) This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t print</td>
<td>(unsigned long, int=DEC) This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t print</td>
<td>(double, int=BIN) This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t print</td>
<td>(const Printable &amp;) This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>size_t println</td>
<td>(void) This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
<tr>
<td>size_t println</td>
<td>(const String &amp;s) This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
<tr>
<td>size_t println</td>
<td>(const char []) This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>
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** [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)`

- **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.

Implemented in WiFiClient.
connect() [1/2]

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

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
Implemented in WiFiClient.
**connect() [2/2]**

```cpp
virtual int Client::connect ( const char * host, uint16_t port )
```

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

**Parameters**

- **[in]** `host` the domain name the client will connect to (string, ex.:"arduino.cc")
- **[in]** `port` the port that the client will connect to (int)
- **[out]** `None`

**Return values**

- **1** the connection succeeds
- **0** the connection failed

**Note**

Implemented in **WiFiClient**.
◆ connected()

virtual uint8_t Client::connected ( )

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.

Implemented in **WiFiClient**.
◆ **flush()**

```cpp
virtual void Client::flush()
```

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.

Implemented in `WiFiClient`. 
**peek()**

```c
virtual int Client::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()`.

**Parameters**

- **[in]** None
- **[out]** None

**Return values**

- -1: none is available
- **other**: the next byte or character

**Note**

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

Implements Stream.

Implemented in WiFiClient.
Virtual int Client::read() 

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

- `-1` none is available.
- **other** The next character

Note

`read()` inherits from the `Stream` utility class

Implements `Stream`.

Implemented in `WiFiClient`. 
virtual int Client::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()).

**Parameters**
- [in] buf the byte to read
- [in] size the size of the buf
- [out] None

**Return values**
-1 none is available.
other The next byte

**Note**
read() inherits from the Stream utility class

Implemented in WiFiClient.
◆ stop()

virtual void Client::stop ()

This function is used to disconnect from the server.

**Parameters**

[**in**] None

[**out**] None

**Returns**

None

**Note**

Implemented in WiFiClient.
### `write()` [1/2]

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

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**.

Implemented in **WiFiClient**.
virtual size_t Client::write ( const uint8_t * buf, size_t size )

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.

Implemented in WiFiClient.
ClientContext Class Reference
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClientContext (tcp pcb *pcb, discard cb_t discard cb void *discard cb_arg)</code></td>
</tr>
<tr>
<td><code>err_t abort ()</code></td>
</tr>
<tr>
<td><code>err_t close ()</code></td>
</tr>
<tr>
<td><code>ClientContext * next () const</code></td>
</tr>
<tr>
<td><code>ClientContext * next (ClientContext *new next)</code></td>
</tr>
<tr>
<td><code>void ref ()</code></td>
</tr>
<tr>
<td><code>void unref ()</code></td>
</tr>
<tr>
<td><code>int connect (ip addr_t *addr, uint16_t port)</code></td>
</tr>
<tr>
<td><code>size_t availableForWrite ()</code></td>
</tr>
<tr>
<td><code>void setNoDelay (bool nodelay)</code></td>
</tr>
<tr>
<td><code>bool getNoDelay ()</code></td>
</tr>
<tr>
<td><code>void setTimeout (int timeout_ms)</code></td>
</tr>
<tr>
<td><code>int getTimeout ()</code></td>
</tr>
<tr>
<td><code>uint32_t getRemoteAddress ()</code></td>
</tr>
<tr>
<td><code>uint16_t getRemotePort ()</code></td>
</tr>
<tr>
<td><code>uint32_t getLocalAddress ()</code></td>
</tr>
<tr>
<td><code>uint16_t getLocalPort ()</code></td>
</tr>
</tbody>
</table>
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_SEC,
                uint8_t count=TCP_DEFAULT_KEEPALIVE_COUNT)

bool isKeepAliveEnabled () const

uint16_t getKeepAliveIdle () const

uint16_t getKeepAliveInterval () const

uint8_t getKeepAliveCount () const
## Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool _is_timeout ()</td>
<td>A function to check if a timeout has occurred.</td>
</tr>
<tr>
<td>void _notify_error ()</td>
<td>A function to notify any potential errors.</td>
</tr>
<tr>
<td>size_t _write_from_source (DataSource *ds)</td>
<td>A function to write data from a source.</td>
</tr>
<tr>
<td>bool _write_some ()</td>
<td>A function to write some data.</td>
</tr>
<tr>
<td>void _write_some_from_cb ()</td>
<td>A function to write some data from a callback.</td>
</tr>
<tr>
<td>err_t _acked (tcp_pcb *pcb, uint16_t len)</td>
<td>A function to acknowledge received data.</td>
</tr>
<tr>
<td>void _consume (size_t size)</td>
<td>A function to consume a certain amount of data.</td>
</tr>
<tr>
<td>err_t _recv (tcp_pcb *pcb, pbuf *pb, err_t err)</td>
<td>A function to receive data.</td>
</tr>
<tr>
<td>void _error (err_t err)</td>
<td>A function to handle errors.</td>
</tr>
<tr>
<td>err_t _connected (struct tcp_pcb *pcb, err_t err)</td>
<td>A function to handle connection status.</td>
</tr>
<tr>
<td>err_t _poll (tcp_pcb *)</td>
<td>A function to poll for data availability.</td>
</tr>
</tbody>
</table>
## Static Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static err_t _s_recv</td>
<td>(void *arg, struct tcp_pcb *tpcb, struct pbuf *pb, err_t err)</td>
</tr>
<tr>
<td>static void _s_error</td>
<td>(void *arg, err_t err)</td>
</tr>
<tr>
<td>static err_t _s_poll</td>
<td>(void *arg, struct tcp_pcb *tpcb)</td>
</tr>
<tr>
<td>static err_t _s_acked</td>
<td>(void *arg, struct tcp_pcb *tpcb, uint16_t len)</td>
</tr>
<tr>
<td>static err_t _s_connected</td>
<td>(void *arg, struct tcp_pcb *pcb, err_t err)</td>
</tr>
</tbody>
</table>

Generated by [doxygen](https://www.doxygen.org/) 1.8.14
CloudClass Class Reference
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CloudClass ()</strong></td>
<td>This function is used to Init Giz Cloud Construct function to init some arguments or some configuration before use the object. More...</td>
</tr>
<tr>
<td><strong>bool CloudInit ()</strong></td>
<td>This function is used to Init Giz Cloud method. More...</td>
</tr>
</tbody>
</table>
Constructor & Destructor Documentation
CloudClass()

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

Returns
None

Note
Member Function Documentation
◆ CloudInit()

```cpp
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.
Inherited by `BufferDataSource`, and `BufferedStreamDataSource<TStream>`.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t available()</td>
<td>virtual size_t available ()=0</td>
</tr>
<tr>
<td>const uint8_t * get_buffer(size_t size)</td>
<td>virtual const uint8_t * get_buffer(size_t size)=0</td>
</tr>
<tr>
<td>release_buffer(const uint8_t *buffer, size_t size)</td>
<td>virtual void release_buffer(const uint8_t *buffer, size_t size)=0</td>
</tr>
</tbody>
</table>
## W60X_Arduino

### DhcpClass Class Reference

| Public Member Functions |
|-------------------------|---|
| **W60X_Arduino**       |   |
Public Member Functions

```c
uint32_t  getLocalIP ()

uint32_t  getSubnetMask ()

uint32_t  getGatewayIp ()

uint32_t  getDhcpServerIp ()

uint32_t  getDnsServerIp ()

char *   getLocalIPStr ()

char *   getSubnetMaskStr ()

char *   getGatewayIpStr ()

char *   getDhcpServerIpStr ()

char *   getDnsServerIpStr ()

int      beginWithDHCP (uint8_t *, unsigned long timeout=60000, unsigned long responseTimeout=4000)

int      checkLease ()
```
<table>
<thead>
<tr>
<th>W60X_Arduino</th>
<th>DNSClient Class Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Member Functions</td>
<td></td>
</tr>
<tr>
<td>Protected Member Functions</td>
<td></td>
</tr>
<tr>
<td>Public Member Functions</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>void</strong> <code>begin</code> <em>(unsigned int aDNSServer)</em></td>
<td></td>
</tr>
<tr>
<td><strong>void</strong> <code>begin</code> *(const char <em>aDNSServer)</em></td>
<td></td>
</tr>
</tbody>
</table>
| **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()**  

```cpp
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**
getHostByNames() [2/2]

int DNSClient::getHostByNames ( 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 doxygen 1.8.14
<table>
<thead>
<tr>
<th>Public Member Functions</th>
</tr>
</thead>
</table>

**W60X_Arduino**

**DNSServer Class Reference**
### Public Member Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td><code>start</code> (const uint16_t &amp;port, const char *domainName, const char *resolvedIP)</td>
<td>This function is used to start DNS Server. More...</td>
</tr>
<tr>
<td>bool</td>
<td><code>start</code> (const char *domainName)</td>
<td>This function is used to start DNS Server with some default arguments in port and resolvedIP. More...</td>
</tr>
<tr>
<td>void</td>
<td><code>stop</code> ()</td>
<td>This function is used to Stop DNS Server. More...</td>
</tr>
</tbody>
</table>
Member Function Documentation
This function is used to start DNS Server.

Parameters
- **[in]** `port` Specify the server's dns port
- **[in]** `domainName` Specify the server's dns name
- **[in]** `resolvedIP` Unused argument.

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

Note
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**
void DNSServer::stop ( )

This function is used to Stop DNS Server.

Returns
None

Note
Inherits Stream.
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HardwareSerial</strong> (int serial_no)</td>
<td>This constructor is used to init hardware serial. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>HardwareSerial</strong> (int serial_no, bool mul_flag)</td>
<td>This constructor is used to init hardware serial. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>HardwareSerial</strong> ()</td>
<td>This constructor is used to init hardware serial. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

```c
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...](#)

```c
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...](#)

```c
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...](#)
### read (void)
Reads incoming serial data. `read()` inherits from the **Stream** utility class. More...

### 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...

### 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...

### 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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long <code>parseInt()</code></td>
<td><code>parseInt()</code> returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped. <a href="#">More...</a></td>
</tr>
<tr>
<td>float <code>parseFloat()</code></td>
<td><code>parseFloat()</code> returns the first valid floating point number from the current position. Initial characters that are not digits (or the minus sign) are skipped. <code>parseFloat()</code> is terminated by the first character that is not a floating point number. <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual size_t <code>readBytes(char *buffer, size_t length)</code></td>
<td><code>readBytes()</code> read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see <code>setTimeout()</code>). <a href="#">More...</a></td>
</tr>
<tr>
<td>virtual size_t <code>readBytes(uint8_t *buffer, size_t length)</code></td>
<td><code>readBytes()</code> read characters from a stream into a buffer. The function terminates if the determined length has been read, or it times out (see <code>setTimeout()</code>). <a href="#">More...</a></td>
</tr>
<tr>
<td>size_t <code>readBytesUntil(char terminator, char *buffer, size_t length)</code></td>
<td><code>readBytesUntil()</code> 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 <code>setTimeout()</code>). <a href="#">More...</a></td>
</tr>
<tr>
<td>size_t <code>readBytesUntil(char terminator, uint8_t *buffer, size_t length)</code></td>
<td><code>readBytesUntil()</code> 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 <code>setTimeout()</code>). <a href="#">More...</a></td>
</tr>
</tbody>
</table>
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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_t print (char)</code></td>
<td>This function is used to print buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned char, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (double, int=BIN)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (const Printable &amp;)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
| `size_t println (void)` | This function is used to print target to the interface defined by the object with carriage ret ("") and new line (‘
’). |
<table>
<thead>
<tr>
<th>size_t</th>
<th>println (const String &amp;s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (const char [])</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (char)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (unsigned char, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (int, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (unsigned int, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (long, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\n'). More...</td>
</tr>
<tr>
<td>size_t</td>
<td>println (unsigned long, int=DEC)</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;\n&quot;) and new line (\n'). More...</td>
</tr>
<tr>
<td>size_t</td>
<td>println (double, int=BIN)</td>
</tr>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;\n&quot;) and new line (\n'). More...</td>
</tr>
<tr>
<td>size_t</td>
<td>println (const Printable &amp;)</td>
</tr>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;\n&quot;) and new line (\n'). More...</td>
</tr>
</tbody>
</table>
### 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)`

- **Protected Attributes inherited from Stream**
  - `unsigned long _timeout`
  - `unsigned long _startMillis`
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.
## W60X_Arduino

<table>
<thead>
<tr>
<th>IPAddress Class Reference</th>
</tr>
</thead>
</table>

Inherits [Printable](#).
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPAddress ()</strong></td>
<td>This constructor function is used to construct <code>IPAddress</code> object. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>IPAddress (uint8_t first_oct, uint8_t sec_oct, uint8_t third_oct, uint8_t fourth_oct)</strong></td>
<td>This constructor function is used to construct <code>IPAddress</code> object. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>IPAddress (uint32_t address)</strong></td>
<td>This constructor function is used to construct <code>IPAddress</code> object. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>IPAddress (const uint8_t <em>address)</em></em></td>
<td>This constructor function is used to construct <code>IPAddress</code> object. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>operator uint32_t () const</strong></td>
<td>This operator overloading function is used to overloading 'uint32_t' operator. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool operator== (const IPAddress &amp;addr) const</strong></td>
<td>This operator overloading function is used to overloading '==' operator. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>bool operator== (uint32_t addr) const</strong></td>
<td>This operator overloading function is used to overloading '==' operator. <a href="#">More...</a></td>
</tr>
<tr>
<td>*<em>bool operator== (const uint8_t <em>addr) const</em></em></td>
<td>This operator overloading function is used to overloading '==' operator. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>uint8_t operator[] (int index) const</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Function</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>uint8_t &amp;</td>
<td>operator[](int index)</td>
</tr>
<tr>
<td>IPAddress &amp;</td>
<td>operator=(const uint8_t *address)</td>
</tr>
<tr>
<td>IPAddress &amp;</td>
<td>operator=(uint32_t address)</td>
</tr>
<tr>
<td>String</td>
<td>toString() const</td>
</tr>
<tr>
<td>bool</td>
<td>fromString(const char *address)</td>
</tr>
<tr>
<td>bool</td>
<td>fromString(const String &amp;address)</td>
</tr>
<tr>
<td>virtual size_t</td>
<td>printTo(Print &amp;p) const</td>
</tr>
</tbody>
</table>
Inherited by Server, and Stream.
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getWriteError()</code></td>
<td>This function is used to get write error number.</td>
<td></td>
</tr>
<tr>
<td><code>void clearWriteError()</code></td>
<td>This function is used to clear write error number.</td>
<td></td>
</tr>
<tr>
<td><code>virtual size_t write (uint8_t)=0</code></td>
<td>This pure virtual function is used to define the operation that writes binary data.</td>
<td></td>
</tr>
<tr>
<td><code>virtual size_t write (const uint8_t *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td><code>size_t write (const char *str)</code></td>
<td>This function is used to write buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td><code>size_t write (const char *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td><code>size_t print (const String &amp;)</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td><code>size_t print (const char [])</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td><code>size_t print (char)</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>size_t   print(unsigned char, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(unsigned int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(unsigned long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(double, int=BIN)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   print(const Printable &amp;)</td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   println(void)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (''). <a href="#">More...</a></td>
<td></td>
</tr>
<tr>
<td>size_t   println(const String &amp;s)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (''). <a href="#">More...</a></td>
<td></td>
</tr>
</tbody>
</table>
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)
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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setWriteError(int err=1)</code></td>
<td>Protected member function to set a write error</td>
</tr>
</tbody>
</table>

**Protected Member Functions**
clearWriteError()

```c
void Print::clearWriteError ( )
```

This function is used to clear write error number.

**Parameters**

[in] None

**Returns**

None

**Note**
◆ getWriteError()

int Print::getWriteError ( )

This function is used to get write error number.

**Parameters**

[in] None

**Returns**

The write error set by the object.

**Note**


**print() [1/10]**

```cpp
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.

| Note |
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.

| Note |
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).

| Note |
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.

**Note**
◆ print() [5/10]

```cpp
size_t Print::print ( int n,
                     int base = DEC
                 )
```

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.

**Note**
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.

**Note**
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.

**Note**
print() [8/10]

```c
size_t Print::print ( unsigned long n,
                      int base = DEC )
```

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.

**Note**
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.

**Note**
◆ 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.

**Note**
**println() [1/11]**

```cpp
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.

**Note**
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 ("\r") and new line ("\n").

**Parameters**

[in] s Specify the String.

**Returns**

The length of print successfully.

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

This function is used to print target to the interface defined by the object with carriage return ("\r") and new line ("\n").

**Parameters**

[in] c Specify the string buffer.

**Returns**

The length of print successfully.

**Note**
`println()` [4/11]

```cpp
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.

**Note**
println() [5/11]

```cpp
size_t Print::println ( unsigned char b,
                      int base = DEC )
```

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

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

**Returns**
The length of print successfully.

**Note**
println() [6/11]

```c
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.

**Note**
println() [7/11]

```cpp
size_t Print::println ( unsigned 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 - unsigned int.

[in] **base** Specify the base.

**Returns**

The length of print successfully.

**Note**
println() [8/11]

```cpp
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 ('\n') and new line ('\n').

**Parameters**
- [in] **num** Specify the target - long.
- [in] **base** Specify the base.

**Returns**
The length of print successfully.

**Note**
println() [9/11]

```cpp
size_t Print::println (unsigned long num,
                       int         base = DEC  // DEC for decimal
                    )
```

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

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

**Returns**
- The length of print successfully.

**Note**
println() [10/11]

```cpp
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.

**Note**
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.

Note
◆ **write() [1/4]**

```cpp
virtual size_t Print::write ( uint8_t )
```

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).

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implemented in WiFiUDP, HardwareSerial, WiFiServer, WiFiClient, UDP, TwoWire, and Client.</td>
</tr>
</tbody>
</table>
**write() [2/4]**

```cpp
size_t Print::write ( const uint8_t * buffer, 
                     size_t size )
```

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]

```cpp
size_t Print::write ( const char * str )
```

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]

```c
size_t Print::write ( const char * buffer, size_t size )
```

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](https://www.doxygen.org/) 1.8.14
Inherited by IPAddress.
### Public Member Functions

<table>
<thead>
<tr>
<th>virtual size_t</th>
<th><code>printTo (Print &amp;p)</code> const =0</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pure virtual function is used to called by print/println function.</td>
<td>More...</td>
</tr>
</tbody>
</table>
◆ printTo()

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

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`.
## ProgmemStream Class Reference

<table>
<thead>
<tr>
<th>Public Member Functions</th>
<th>Protected Attributes</th>
</tr>
</thead>
</table>

---

**W60X_Arduino**

[74x679 to 109x714]
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgmemStream</td>
<td>(PGM_P buf, size_t size)</td>
</tr>
<tr>
<td>size_t readBytes</td>
<td>(char *dst, size_t size)</td>
</tr>
</tbody>
</table>
### Protected Attributes

<table>
<thead>
<tr>
<th>PGM_P</th>
<th>_buf</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>_left</td>
</tr>
</tbody>
</table>

Generated by [doxygen] 1.8.14
Inherits **Print**.

Inherited by **WiFiServer**.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void <code>begin()</code> = 0</td>
<td>This pure-virtual function is used to start the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

#### Public Member Functions inherited from Print

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>getWriteError()</code></td>
<td>This function is used to get write error number. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <code>clearWriteError()</code></td>
<td>This function is used to clear write error number. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual size_t <code>write(uint8_t)</code> = 0</td>
<td>This pure virtual function is used to define the operation that writes binary data. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual size_t <code>write(const uint8_t *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t <code>write(const char *str)</code></td>
<td>This function is used to write buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t <code>write(const char *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t <code>print(const String &amp;)</code></td>
<td>This function is used to print buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t <code>print(const char [])</code></td>
<td>This function is used to print buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>size_t print (char)</code></td>
<td>This function is used to print buffer to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned char, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (unsigned long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (double, int=BIN)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t print (const Printable &amp;)</code></td>
<td>This function is used to print target to the interface defined by the object. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>size_t println (void)</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;\n&quot;) and new line (&quot;\n&quot;). <a href="#">More...</a></td>
</tr>
<tr>
<td>Function</td>
<td>Signature</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>size_t println</td>
<td>(const String &amp;s)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(const char [])</td>
</tr>
<tr>
<td>size_t println</td>
<td>(char)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(unsigned char, int=DEC)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(int, int=DEC)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(unsigned int, int=DEC)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(long, int=DEC)</td>
</tr>
<tr>
<td>size_t println</td>
<td>(unsigned long, int=DEC)</td>
</tr>
</tbody>
</table>
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)
  ```
◆ begin()

virtual void Server::begin ( )

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

**Parameters**

- [in] **none**
- [out]

Implemented in **WiFiServer**.
<table>
<thead>
<tr>
<th>Static Protected Member Functions</th>
<th>Protected Attributes</th>
<th>Static Protected Attributes</th>
</tr>
</thead>
</table>

**W60X_Arduino**

**SList< T > Class Template Reference**
## Static Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>static void add(T *self)</td>
</tr>
<tr>
<td>remove</td>
<td>static void remove(T *self)</td>
</tr>
</tbody>
</table>
Protected Attributes

| T * _next |
### Static Protected Attributes

| static T * _s_first |

Generated by [doxygen](https://www.doxygen.org/) 1.8.14
<table>
<thead>
<tr>
<th>Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIClass Class Reference</td>
</tr>
</tbody>
</table>
### 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 *__bufout, void *__bufin, 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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setBitOrder (BitOrder)</td>
<td>Deprecated function. Configure the bit order: MSB first or LSB first. More...</td>
</tr>
<tr>
<td>void setDataMode (uint8_t _mode)</td>
<td>Deprecated function. Configure the data mode (clock polarity and clock phase) More...</td>
</tr>
<tr>
<td>void setFrequency (uint32_t freq)</td>
<td>Configure the spi frequency. More...</td>
</tr>
</tbody>
</table>

Generated by doxygen 1.8.14
<table>
<thead>
<tr>
<th>W60X_Arduino</th>
</tr>
</thead>
</table>

**SPISettings Class Reference**

<table>
<thead>
<tr>
<th>Public Member Functions</th>
<th>Friends</th>
</tr>
</thead>
</table>

Public Member Functions

**SPISettings** (uint32_t clock, BitOrder bitOrder, uint8_t dataMode)
class SPIClass
Inherits **Print**.

Inherited by **Client, HardwareSerial, TwoWire, and UDP**.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual int</td>
<td><strong>available ()=0</strong> <strong>available()</strong> gets the number of bytes available in the stream. This is only for bytes that have already arrived. More...</td>
</tr>
<tr>
<td>virtual int</td>
<td><strong>read ()=0</strong> <strong>read()</strong> reads characters from an incoming stream to the buffer. More...</td>
</tr>
<tr>
<td>virtual int</td>
<td><strong>peek ()=0</strong> <strong>peek()</strong> reads a byte from the file without advancing to the next one. That is, successive calls to <strong>peek()</strong> will return the same value, as will the next call to <strong>read()</strong>. More...</td>
</tr>
<tr>
<td>void</td>
<td><strong>setTimeout (unsigned long timeout)</strong> <strong>setTimeout()</strong> sets the maximum milliseconds to wait for stream data, it defaults to 1000 milliseconds. This function is part of the <strong>Stream</strong> class, and is called by any class that inherits from it (Wire, Serial, etc). See the <strong>Stream</strong> class main page for more information. More...</td>
</tr>
<tr>
<td>bool</td>
<td>*<em>find (const char <em>target)</em></em> <strong>find()</strong> 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...</td>
</tr>
<tr>
<td>bool</td>
<td>*<em>find (uint8_t <em>target)</em></em> <strong>find()</strong> 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...</td>
</tr>
<tr>
<td>bool</td>
<td>*<em>find (const char <em>target, size_t length)</em></em> <strong>find()</strong> 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...</td>
</tr>
<tr>
<td>Type</td>
<td>Function</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>bool</td>
<td>find (const uint8_t *target, size_t length)</td>
</tr>
<tr>
<td>bool</td>
<td>find (char target)</td>
</tr>
<tr>
<td>bool</td>
<td>findUntil (const char *target, const char *terminator)</td>
</tr>
<tr>
<td>bool</td>
<td>findUntil (const uint8_t *target, const char *terminator)</td>
</tr>
<tr>
<td>bool</td>
<td>findUntil (const char *target, size_t targetLen, const char *terminate, size_t termLen)</td>
</tr>
<tr>
<td>bool</td>
<td>findUntil (const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen)</td>
</tr>
<tr>
<td>long</td>
<td>parseInt ()</td>
</tr>
<tr>
<td>float</td>
<td>parseFloat ()</td>
</tr>
</tbody>
</table>

More...
digits (or the minus sign) are skipped. \texttt{parseFloat()} is terminated by the first character that is not a floating point number. More...

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

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

\begin{Verbatim}
\textbf{size_t} \texttt{readBytesUntil (char terminator, char *buffer, size_t length)}
\end{Verbatim}
\texttt{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 \texttt{setTimeout()}). More...

\begin{Verbatim}
\textbf{size_t} \texttt{readBytesUntil (char terminator, uint8_t *buffer, size_t length)}
\end{Verbatim}
\texttt{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 \texttt{setTimeout()}). More...

\begin{Verbatim}
\textbf{String} \texttt{readString ()}
\end{Verbatim}
\texttt{readString()} reads characters from a stream into a string. The function terminates if it times out (see \texttt{setTimeout()}). More...

\begin{Verbatim}
\textbf{String} \texttt{readStringUntil (char terminator)}
\end{Verbatim}
\texttt{readStringUntil()} reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see \texttt{setTimeout()}). More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `int getWriteError()` | This function is used to get write error number. 
| `void clearWriteError()` | This function is used to clear write error number. 
| `virtual size_t write(uint8_t)=0` | This pure virtual function is used to define the operation that writes binary data. 
| `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. 
| `size_t write(const char *str)` | This function is used to write buffer to the interface defined by the object. 
| `size_t write(const char *buffer, size_t size)` | This function is used to write buffer to the interface defined by the object. 
| `size_t print(const String&)` | This function is used to print buffer to the interface defined by the object. 
| `size_t print(const char [])` | This function is used to print buffer to the interface defined by the object. 
| `size_t print(char)` | This function is used to print buffer to the interface defined by the object. 
<p>| <code>size_t print(unsigned char, int=DEC)</code> | This function is used to print target to the interface |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t print</td>
<td>(int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t print</td>
<td>(unsigned int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t print</td>
<td>(long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t print</td>
<td>(unsigned long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t print</td>
<td>(double, int=BIN)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t print</td>
<td>(const Printable &amp;)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t println</td>
<td>(void)</td>
<td>This function is used to print target to the interface with carriage ret (\n) and new line (\n).</td>
<td></td>
</tr>
<tr>
<td>size_t println</td>
<td>(const String &amp;s)</td>
<td>This function is used to print target to the interface with carriage ret (\n) and new line (\n).</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>println</strong> (const char [])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (char)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (unsigned char, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (int, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (unsigned int, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (long, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (unsigned long, int=DEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th><strong>println</strong> (double, int=BIN)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
</tr>
<tr>
<td>size_t println (const Printable &amp;)</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| This function is used to print target to the interface defined by the object with carriage ret ("") and new line (‘
’). More... |
### Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timedRead()</code></td>
<td>int</td>
</tr>
<tr>
<td><code>timedPeek()</code></td>
<td>int</td>
</tr>
<tr>
<td><code>peekNextDigit()</code></td>
<td>int</td>
</tr>
<tr>
<td><code>parseInt(char skipChar)</code></td>
<td>long</td>
</tr>
<tr>
<td><code>parseFloat(char skipChar)</code></td>
<td>float</td>
</tr>
</tbody>
</table>

- **Protected Member Functions inherited from Print**
  - `setWriteError(int err=1)`
## Protected Attributes

- `unsigned long _timeout`
- `unsigned long _startMillis`
Inherits Stream.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>begin</code></td>
<td><code>void begin (int sda, int scl)</code></td>
</tr>
<tr>
<td>pins</td>
<td><code>void pins (int sda, int scl) __attribute__((deprecated))</code></td>
</tr>
<tr>
<td><code>begin</code></td>
<td><code>void begin ()</code></td>
</tr>
<tr>
<td><code>begin</code></td>
<td><code>void begin (uint8_t)</code></td>
</tr>
<tr>
<td><code>begin</code></td>
<td><code>void begin (int)</code></td>
</tr>
<tr>
<td><code>setClock</code></td>
<td><code>void setClock (uint32_t)</code></td>
</tr>
<tr>
<td><code>setClockStretchLimit</code></td>
<td><code>void setClockStretchLimit (uint32_t)</code></td>
</tr>
<tr>
<td><code>beginTransmission</code></td>
<td><code>void beginTransmission (uint8_t)</code></td>
</tr>
<tr>
<td><code>beginTransmission</code></td>
<td><code>void beginTransmission (int)</code></td>
</tr>
<tr>
<td><code>endTransmission</code></td>
<td><code>uint8_t endTransmission (void)</code></td>
</tr>
<tr>
<td><code>endTransmission</code></td>
<td><code>uint8_t endTransmission (uint8_t)</code></td>
</tr>
<tr>
<td><code>requestFrom</code></td>
<td><code>size_t requestFrom (uint8_t address, size_t size, bool sendStop)</code></td>
</tr>
<tr>
<td><code>status</code></td>
<td><code>uint8_t status ()</code></td>
</tr>
<tr>
<td><code>requestFrom</code></td>
<td><code>uint8_t requestFrom (uint8_t, uint8_t)</code></td>
</tr>
<tr>
<td><code>requestFrom</code></td>
<td><code>uint8_t requestFrom (uint8_t, uint8_t, uint8_t)</code></td>
</tr>
<tr>
<td><code>requestFrom</code></td>
<td><code>uint8_t requestFrom (int, int)</code></td>
</tr>
<tr>
<td><code>requestFrom</code></td>
<td><code>uint8_t requestFrom (int, int, int)</code></td>
</tr>
</tbody>
</table>
### 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
### find (const uint8_t *target, size_t length)

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

### find (char target)

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

### 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 (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...

### 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...

### 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...

### 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...

### parseFloat ()

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...
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td><strong>print</strong> (unsigned int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>print</strong> (long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>print</strong> (unsigned long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>print</strong> (double, int=BIN)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>print</strong> (const Printable &amp;)</td>
<td>This function is used to print target to the interface defined by the object.</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>println</strong> (void)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\r').</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>println</strong> (const String &amp;s)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\r').</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>println</strong> (const char [])</td>
<td>This function is used to print target to the interface defined by the object with carriage ret ('\n') and new line ('\r').</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td><strong>println</strong> (char)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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** 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)`

- **Protected Attributes inherited from** Stream
  - `unsigned long _timeout`
  - `unsigned long _startMillis`
Member Function Documentation
**write()** [1/4]

```cpp
size_t Print::write
```

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).
**write() [3/4]**

```cpp
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`

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](https://www.doxygen.org) 1.8.14
Inherits **Stream**.

Inherited by **WiFiUDP**.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual uint8_t</code></td>
<td><code>begin (uint16_t)=0</code></td>
<td>This function is used to initializes the UDP library and network settings, Starts UDP socket, listening at local port. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual void</code></td>
<td><code>stop ()=0</code></td>
<td>This function is used to disconnect from the server. Release any resource being used during the UDP session. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual int</code></td>
<td><code>beginPacket (IPAddress ip, uint16_t port)=0</code></td>
<td>This function is used to starts a connection to write UDP data to the remote connection. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual int</code></td>
<td><code>beginPacket (const char *host, uint16_t port)=0</code></td>
<td>This function is used to starts a connection to write UDP data to the remote connection. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual int</code></td>
<td><code>endPacket ()=0</code></td>
<td>This function is used to called after writing UDP data to the remote connection. It finishes off the packet and send it. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual size_t</code></td>
<td><code>write (uint8_t)=0</code></td>
<td>This function is used to writes UDP data to the remote connection. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual size_t</code></td>
<td><code>write (const uint8_t *buffer, size_t size)=0</code></td>
<td>This function is used to writes UDP data to the remote connection. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>virtual int</code></td>
<td><code>parsePacket ()=0</code></td>
<td>It starts processing the next available incoming packet, checks for the presence of a UDP packet, <a href="#">More...</a></td>
</tr>
</tbody>
</table>
and reports the size. More...

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>available ()</strong> = 0</td>
<td>Get the number of bytes (characters) available for reading from the buffer. This is data that's already arrived. More...</td>
</tr>
<tr>
<td><strong>read ()</strong> = 0</td>
<td>Reads <strong>UDP</strong> data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...</td>
</tr>
<tr>
<td>*<em>read (unsigned char <em>buffer, size_t len)</em></em> = 0</td>
<td>Reads <strong>UDP</strong> data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...</td>
</tr>
<tr>
<td>*<em>read (char <em>buffer, size_t len)</em></em> = 0</td>
<td>Reads <strong>UDP</strong> data from the specified buffer. If no arguments are given, it will return the next character in the buffer. More...</td>
</tr>
<tr>
<td><strong>peek ()</strong> = 0</td>
<td>Read a byte from the file without advancing to the next one. That is, successive calls to <strong>peek()</strong> will return the same value, as will the next call to <strong>read()</strong>. More...</td>
</tr>
<tr>
<td><strong>flush ()</strong> = 0</td>
<td>Discard any bytes that have been written to the client but not yet read. More...</td>
</tr>
<tr>
<td><strong>IPAddress remoteIP ()</strong> = 0</td>
<td>This function is used to gets the IP address of the remote connection. More...</td>
</tr>
<tr>
<td><strong>uint16_t remotePort ()</strong> = 0</td>
<td></td>
</tr>
</tbody>
</table>
This function is used to gets the port of the remote **UDP** connection. More...

<table>
<thead>
<tr>
<th>Public Member Functions inherited from <strong>Stream</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
|     | `find()` reads data from the stream until the target
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string of given length is found</td>
<td>The function returns true if target string is found, false if timed out. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool findUntil(const char *target, const char *terminator)</td>
<td>findUntil() reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool findUntil(const uint8_t *target, const char *terminator)</td>
<td>findUntil() reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool findUntil(const char *target, size_t targetLen, const char *terminate, size_t termLen)</td>
<td>findUntil() reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool findUntil(const uint8_t *target, size_t targetLen, const char *terminate, size_t termLen)</td>
<td>findUntil() reads data from the stream until the target string of given length or terminator string is found. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long parseInt()</td>
<td>parseInt() returns the first valid (long) integer number from the serial buffer. Characters that are not integers (or the minus sign) are skipped. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float parseFloat()</td>
<td>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...</td>
</tr>
</tbody>
</table>
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...
setTimeout(). More...

Public Member Functions inherited from Print

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>getError</code></td>
<td>This function is used to get write error number.</td>
</tr>
<tr>
<td>void <code>clearWriteError</code></td>
<td>This function is used to clear write error number.</td>
</tr>
<tr>
<td>size_t <code>write</code> (const char *str)</td>
<td>This function is used to write buffer to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>write</code> (const char *buffer, size_t size)</td>
<td>This function is used to write buffer to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>print</code> (const String &amp;string)</td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>print</code> (const char [])</td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>print</code> (char)</td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>print</code> (unsigned char, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object.</td>
</tr>
<tr>
<td>size_t <code>print</code> (int, int=DEC)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>size_t print</code> (unsigned int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t print</code> (long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t print</code> (unsigned long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t print</code> (double, int=BIN)</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t print</code> (const Printable &amp;)</td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t println</code> (void)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (''). More...</td>
</tr>
<tr>
<td><code>size_t println</code> (const String &amp;s)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (''). More...</td>
</tr>
<tr>
<td><code>size_t println</code> (const char [])</td>
<td>This function is used to print target to the interface</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>size_t println (char)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t println (unsigned char, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t println (int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t println (unsigned int, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t println (long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t println (unsigned long, int=DEC)</td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (' '). More...</td>
</tr>
<tr>
<td>size_t</td>
<td>println (double, int=BIN)</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (')</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size_t</th>
<th>println (const Printable &amp;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line (')</td>
<td>More...</td>
</tr>
</tbody>
</table>
# 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)`

- **Protected Attributes inherited from** `Stream`
  - `unsigned long _timeout`
  - `unsigned long _startMillis`
Member Function Documentation
available()

virtual int UDP::available ( )

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

Parameters

[\textbf{in}] None
[\textbf{out}] None

Return values

0 \hspace{1em} \text{parsePacket hasn't been called yet}
other \hspace{1em} the number of bytes available in the current packet

Note

This function can only be successfully called after \texttt{UDP.parsePacket()}. available() inherits from the \texttt{Stream} utility class.

Implements \texttt{Stream}.

Implemented in \texttt{WiFiUDP}.
begin()

virtual uint8_t UDP::begin ( uint16_t )

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

Implemented in WiFiUDP.
◆ beginPacket() [1/2]

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

This function is used to start 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**

Implemented in WiFiUDP.
virtual int UDP::beginPacket ( const char * host,  
    uint16_t port  
)

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

**Parameters**

- **[in] host**  the address of the remote host. It accepts a character string or an IPAddress
- **[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**

Implemented in WiFiUDP.
◆ endPacket()

virtual int UDP::endPacket ( )

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**

Implemented in WiFiUDP.
◆ flush()

virtual void UDP::flush ( )

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.

Implemented in WiFiUDP.
parsePacket()

virtual int UDP::parsePacket ( )

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

- 0 no packets are available
- other the size of the packet in bytes

Note

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

Implemented in WiFiUDP.
virtual int UDP::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()`.

**Parameters**

- [in] None
- [out] None

**Return values**

- `-1` none is available
- `other` the next byte or character

**Note**

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

Implements `Stream`.

Implemented in `WiFiUDP`. 
virtual int UDP::read()  

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**

-1 no buffer is available

**other** the characters in the buffer (char)

**Note**

Implements **Stream**.

Implemented in **WiFiUDP**.
**read()** [2/3]

```cpp
virtual int UDP::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.

**Parameters**

- **buffer** buffer to hold incoming packets (unsigned char*)
- **len** maximum size of the buffer (int)

**Return values**

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

**Note**

Implemented in **WiFiUDP**.
virtual int UDP::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.

**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**

Implemented in **WiFiUDP**.
remoteIP()

virtual IPAddress UDP::remoteIP ()

This function is used to get 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().

Implemented in WiFiUDP.
remotePort()

virtual uint16_t UDP::remotePort ()

This function is used to get 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().

Implemented in WiFiUDP.
virtual void UDP::stop()  

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  
Implemented in WiFiUDP.
**write() [1/2]**

```cpp
template int size_t UDP::write ( uint8_t )
```

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**.

Implemented in **WiFiUDP**.
virtual size_t UDP::write ( const uint8_t * buffer, size_t size )

This function is used to write 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`.

Implemented in `WiFiUDP`.

Generated by [doxygen](https://www.doxygen.org/) 1.8.14
## UdpContext Class Reference

<table>
<thead>
<tr>
<th>Public Types</th>
<th>Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>W60X_Arduino</td>
<td>UdpContext</td>
</tr>
</tbody>
</table>
Public Types

typedef std::function< void(void)>  rxhandler_t
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void ref ()</td>
<td></td>
</tr>
<tr>
<td>void unref ()</td>
<td></td>
</tr>
<tr>
<td>bool connect (ip_addr_t addr, uint16_t port)</td>
<td>Connect to a remote address and port</td>
</tr>
<tr>
<td>bool listen (ip_addr_t addr, uint16_t port)</td>
<td>Listen for incoming messages on a specified port</td>
</tr>
<tr>
<td>void disconnect ()</td>
<td>Disconnect from the remote address and port</td>
</tr>
<tr>
<td>void setMulticastInterface (const ip_addr_t &amp;addr)</td>
<td>Set the multicast interface to a specific address</td>
</tr>
<tr>
<td>void setMulticastTTL (int ttl)</td>
<td>Set the multicast TTL to a specific value</td>
</tr>
<tr>
<td>void onRx (rxhandler_t handler)</td>
<td>Specify the callback function for incoming messages</td>
</tr>
<tr>
<td>size_t getSize () const</td>
<td>Get the size of the remote address packet</td>
</tr>
<tr>
<td>size_t tell () const</td>
<td>Tell the position of the remote address packet</td>
</tr>
<tr>
<td>void seek (const size_t pos)</td>
<td>Seek to a specific position in the remote address packet</td>
</tr>
<tr>
<td>bool isValidOffset (const size_t pos) const</td>
<td>Validate if a specific offset is within bounds</td>
</tr>
<tr>
<td>uint32_t getRemoteAddress ()</td>
<td>Get the remote address of the packet</td>
</tr>
<tr>
<td>uint16_t getRemotePort ()</td>
<td>Get the remote port of the packet</td>
</tr>
<tr>
<td>uint32_t getDestAddress ()</td>
<td>Get the destination address of the packet</td>
</tr>
<tr>
<td>uint16_t getLocalPort ()</td>
<td>Get the local port of the packet</td>
</tr>
<tr>
<td>bool next ()</td>
<td>Move to the next packet in the stream</td>
</tr>
</tbody>
</table>
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)
## W600InnerFlashClass

Class Reference
Public Member Functions

**int begin ()**
This function is used to initialize 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 initialize inner flash driver.

**Parameters**

- [in] None
- [out] None

**Returns**

None

**Note**
### flashEraseSector()

```c
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()

```cpp
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 0x00000 to 0xFFFFF
- **in/out** data : data to be read from flash
- **in** size : data length to be written to flash
- **out** None

**Returns**

None

**Note**
◆ flashWrite()

```cpp
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.
Inherited by WiFiClass.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool softAP (const char *ssid, const char *passphrase=NULL, int channel=1, int ssid_hidden=0, int max_connection=4)</code></td>
<td>This function is used to start wifi module as AP mode. More...</td>
</tr>
<tr>
<td><code>bool softAPConfig (uint32_t local_ip, uint32_t gateway, uint32_t subnet)</code></td>
<td>This function is used to config soft AP parameters. More...</td>
</tr>
<tr>
<td><code>bool softAPConfig (const char *local_ip, const char *gateway, const char *subnet)</code></td>
<td>This function is used to config soft AP parameters. More...</td>
</tr>
<tr>
<td><code>bool softAPdisconnect (bool wifioff=false)</code></td>
<td>This function is used to disconnect the ap function. More...</td>
</tr>
<tr>
<td><code>bool softAPdestroy ()</code></td>
<td>This function is used to destroy the AP function. More...</td>
</tr>
<tr>
<td><code>uint8_t softAPgetStationNum ()</code></td>
<td>This function is used to get the number of station. More...</td>
</tr>
<tr>
<td><code>char * softAPIP ()</code></td>
<td>This function is used to get AP’s IP. More...</td>
</tr>
<tr>
<td><code>uint8_t * softAPmacAddress (uint8_t *mac)</code></td>
<td>This function is used to get AP’s MAC address. More...</td>
</tr>
<tr>
<td><code>char * softAPmacAddress (void)</code></td>
<td>This function is used to get the AP’s MAC address. More...</td>
</tr>
<tr>
<td><code>char * softAPSSID () const</code></td>
<td>This function is used to get AP’s SSID. More...</td>
</tr>
</tbody>
</table>
char * softAPPSK () const

This function is used to get AP’s PSK. More...
**softAP()**

```c
bool WiFiAPClass::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.

**Parameters**

- **[in] ssid** Specify the SSID.
- **[in] passphrase** Specify the passphrase.
- **[in] channel** Specify the channel.
- **[in] ssid_hidden** Specify the ssid_hidden flag.
- **[in] max_connection** Specify the max_connection.

**Returns**

true - operate successfully, false - operate failure.

**Note**
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.

---

**Note**
This function is used to configure 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.

## Note
◆ softAPdestroy()

```cpp
def WiFiAPClass::softAPdestroy()
```

This function is used to destroy the AP function.

**Parameters**

[in] None

**Returns**

true - operate successfully, false - operate failure.

**Note**
**softAPdisconnect()**

```cpp
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.

**Note**
◆ softAPgetStationNum()

```c
uint8_t WiFiAPClass::softAPgetStationNum()
```

This function is used to get the number of station.

**Parameters**

[in] None

**Returns**

The number of station.

| Note |
◆ softAPIP()

```c
char * WiFiAPClass::softAPIP()
```

This function is used to get AP's IP.

**Parameters**
- `[in] None`

**Returns**
- The local AP's IPv4 address.

**Note**
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.

| Note |
◆ softAPPSK()

char * WiFiAPClass::softAPPSK () const

This function is used to get AP's PSK.

Parameters

[in] None

Returns

The AP's PSK is returned.

Note
**softAPSSID()**

```cpp
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](https://www.doxygen.org/) 1.8.14
**W60X_Arduino**

### WiFiClass Class Reference

Inherits **WiFiGenericClass**, **WiFiSTAClass**, **WiFiScanClass**, **WiFiAPClass**, and **WiFiOneshotClass**.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char * SSID ()</code></td>
<td>This function is used to get the SSID used by the module. More...</td>
</tr>
<tr>
<td><code>uint8_t * BSSID ()</code></td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td><code>char * BSSIDstr ()</code></td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td><code>int32_t RSSI ()</code></td>
<td>This function is used to get the RSSI. More...</td>
</tr>
<tr>
<td><code>char * SSID (uint8_t networkItem)</code></td>
<td>This function is used to get ssid. More...</td>
</tr>
<tr>
<td><code>uint8_t * BSSID (uint8_t networkItem)</code></td>
<td>This function is used to get the bssid. More...</td>
</tr>
<tr>
<td><code>char * BSSIDstr (uint8_t networkItem)</code></td>
<td>This function is used to get the bssid. More...</td>
</tr>
<tr>
<td><code>int32_t RSSI (uint8_t networkItem)</code></td>
<td>This function is used to get RSSI. More...</td>
</tr>
<tr>
<td><code>int32_t channel (uint8_t networkItem)</code></td>
<td>This function is used to get the bssid. More...</td>
</tr>
</tbody>
</table>

Public Member Functions inherited from `WiFiGenericClass`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WiFiMode_t getMode ()</code></td>
<td>This function is used to Get the WiFi's work mode.</td>
</tr>
</tbody>
</table>
### Public Member Functions inherited from WiFiSTAClass

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int hostByName (const char *aHostname, IPAddress &amp;aResult)</code></td>
<td>This function is used to resolve hostname. More...</td>
</tr>
<tr>
<td><code>int hostByName (const char *aHostname, IPAddress &amp;aResult, uint32_t timeout_ms)</code></td>
<td>This function is used to resolve hostname. More...</td>
</tr>
<tr>
<td><code>int begin (const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)</code></td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td><code>int begin (char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)</code></td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td><code>int begin ()</code></td>
<td>This function is used to start the wifi module as station mode. More...</td>
</tr>
<tr>
<td><code>bool reconnect ()</code></td>
<td>This function is used to reconnect the AP. More...</td>
</tr>
<tr>
<td><code>bool disconnect (bool wifioff=false)</code></td>
<td>This function is used to disconnect the wifi. More...</td>
</tr>
<tr>
<td><code>bool isConnected ()</code></td>
<td>This function is used to get the wifi mode connect status. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>bool setAutoConnect (bool autoConnect)</code></td>
<td>This function is used to set auto connect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool getAutoConnect ()</code></td>
<td>This function is used to get auto connect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool setAutoReconnect (bool autoReconnect)</code></td>
<td>This function is used to set auto reconnect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>bool getAutoReconnect ()</code></td>
<td>This function is used to get auto reconnect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint8_t waitForConnectResult ()</code></td>
<td>This function is used to suspend until the WiFi is connected. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IPAddress localIP ()</code></td>
<td>This function is used to get the local ip address. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>char * macAddress ()</code></td>
<td>This function is used to get local MAC address used by the module. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>char * macAddressStr ()</code></td>
<td>This function is used to get local MAC address used by the module. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IPAddress subnetMask ()</code></td>
<td>This function is used to get subnet mask. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IPAddress gatewayIP ()</code></td>
<td>This function is used to get gateway IP address. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>IPAddress dnsIP (uint8_t dns_no=0)</code></td>
<td>This function is used to get DNS IP address. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
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...

<table>
<thead>
<tr>
<th>Public Member Functions inherited from WiFiAPClass</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bool</strong> softAP (const char *ssid, const char *passphrase=NULL, int channel=1, int ssid_hidden=0, int max_connection=4)</td>
</tr>
<tr>
<td>This function is used to start wifi module as AP mode. More...</td>
</tr>
</tbody>
</table>

| **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 wifiOpt=false) |
| This function is used to disconnect the ap function. More... |

| **bool** softAPdestroy () |
| This function is used to destroy 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... |

<p>| **char ** softAPmacAddress (void) |
| This function is used to get the AP's MAC address. More... |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char * softAPSSID () const</code></td>
<td>This function is used to get AP's SSID. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>char * softAPPSK () const</code></td>
<td>This function is used to get AP's PSK. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong>Public Member Functions inherited from WiFiOneshotClass</strong></td>
<td></td>
</tr>
<tr>
<td><code>int oneshotStart ()</code></td>
<td>This function is used to start oneshot configuration network. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>int oneshotStop ()</code></td>
<td>This function is used to stop oneshot configuration network. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>int oneshotGetState ()</code></td>
<td>This function is used to get oneshot state. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>int oneshotSetMode (ONESHOT_MODE mode)</code></td>
<td>This function is used to get oneshot state. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>int oneshotGetMode ()</code></td>
<td>This function is used to get oneshot mode. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
### Additional Inherited Members

- **Static Protected Member Functions inherited from WiFiScanClass**
  
  ```
  static void _scanDone ()
  ```

  ```
  static void * _getScanInfoByIndex (int i)
  ```
Member Function Documentation
```c
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.

**Note**
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.

Note
This function is used to get the bssid.

**Parameters**

- **[in]** `networkItem` Specify the index of scan result.

**Returns**

The string of the bssid.

**Note**
**channel()**

```c
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.

**Note**
This function is used to get the RSSI.

**Parameters**

[in] None

**Returns**

the value of rssi in this connect.

**Note**
### RSSI() [2/2]

```c
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

**Note**
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.

Note
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

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
</table>

Generated by doxygen 1.8.14
## WiFiClient Class Reference

Inherits **Client**, and **SList<WiFiClient >**.
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WiFiClient ()</strong></td>
<td>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().</td>
<td>More...</td>
</tr>
<tr>
<td><strong>WiFiClient (const WiFiClient &amp;)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WiFiClient &amp; operator= (const WiFiClient &amp;)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>virtual <strong>~WiFiClient ()</strong></td>
<td>This function is deconstructor, it's used to release WiFiClient class.</td>
<td>More...</td>
</tr>
<tr>
<td>uint8_t <strong>status ()</strong></td>
<td>return tcp status of WiFiClient.</td>
<td>More...</td>
</tr>
<tr>
<td>virtual int <strong>connect (IPAddress ip, uint16_t port)</strong></td>
<td>This function is used to connect to the IP address and port specified in the constructor.</td>
<td>More...</td>
</tr>
<tr>
<td>virtual int *<em>connect (const char <em>host, uint16_t port)</em></em></td>
<td>This function is used to connect to the IP address and port specified in the constructor.</td>
<td>More...</td>
</tr>
<tr>
<td>virtual int <strong>connect (const String host, uint16_t port)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>virtual size_t <strong>write (uint8_t)</strong></td>
<td>This function is used to write data to the server the client is connected to.</td>
<td>More...</td>
</tr>
<tr>
<td>virtual size_t *<em>write (const uint8_t <em>buf, size_t size)</em></em></td>
<td>This function is used to write data to the server the client is connected to.</td>
<td>More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>write_P</td>
<td><code>virtual size_t write_P(PGM_P buf, size_t size)</code></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td><code>size_t write(Stream &amp;stream)</code></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td><code>size_t write(Stream &amp;stream, size_t unitSize)</code> <strong>attribute</strong>((deprecated))</td>
<td></td>
</tr>
<tr>
<td>available</td>
<td><code>virtual int available()</code> 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...</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td><code>virtual int read()</code> Read the next byte received from the server the client is connected to (after the last call to read()). More...</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td><code>virtual int read(uint8_t *buf, size_t size)</code> Read the next byte received from the server the client is connected to (after the last call to read()). More...</td>
<td></td>
</tr>
<tr>
<td>peek</td>
<td><code>virtual int peek()</code> 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...</td>
<td></td>
</tr>
<tr>
<td>peekBytes</td>
<td><code>virtual size_t peekBytes(uint8_t *buffer, size_t length)</code></td>
<td></td>
</tr>
<tr>
<td>peekBytes</td>
<td><code>size_t peekBytes(char *buffer, size_t length)</code></td>
<td></td>
</tr>
<tr>
<td>flush</td>
<td><code>virtual void flush()</code> Discard any bytes that have been written to the client but not yet read. More...</td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td><code>virtual void stop()</code> This function is used to disconnect from the server. More...</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>virtual uint8_t <strong>connected</strong> ()</td>
<td>Whether or not the client is connected. More...</td>
<td></td>
</tr>
<tr>
<td>virtual <strong>operator bool</strong> ()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPAddress <strong>remoteIP</strong> ()</td>
<td>This function is used to get the IP address of the remote connection. More...</td>
<td></td>
</tr>
<tr>
<td>uint16_t <strong>remotePort</strong> ()</td>
<td>This function is used to get the port of the remote connection. More...</td>
<td></td>
</tr>
<tr>
<td>IPAddress <strong>localIP</strong> ()</td>
<td>This function is used to get the IP address of the local tcp connection. More...</td>
<td></td>
</tr>
<tr>
<td>uint16_t <strong>localPort</strong> ()</td>
<td>This function is used to get the port of the local tcp connection. More...</td>
<td></td>
</tr>
<tr>
<td><strong>bool</strong> <strong>getNoDelay</strong> ()</td>
<td>This function is used to get whether no delay of the tcp connection. More...</td>
<td></td>
</tr>
<tr>
<td><strong>void</strong> <strong>setNoDelay</strong> (bool nodelay)</td>
<td>This function is used to set no delay for the tcp connection. More...</td>
<td></td>
</tr>
<tr>
<td>size_t <strong>availableForWrite</strong> ()</td>
<td>This function is used to get the length that can be written. More...</td>
<td></td>
</tr>
<tr>
<td><strong>void</strong> <strong>keepAlive</strong> (uint16_t idle_sec=TCP_DEFAULT_KEEPALIVE_IDLE_SEC,</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Methods for Keep-Alive

#### `bool isKeepAliveEnabled () const`
This function is used to get whether enable keep alive. More...

#### `uint16_t getKeepAliveIdle () 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 Methods

#### `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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void timeout()</td>
<td>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...</td>
</tr>
<tr>
<td>bool find()</td>
<td>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...</td>
</tr>
<tr>
<td>bool find()</td>
<td>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...</td>
</tr>
<tr>
<td>bool find()</td>
<td>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...</td>
</tr>
<tr>
<td>bool find()</td>
<td>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...</td>
</tr>
<tr>
<td>bool find()</td>
<td>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...</td>
</tr>
<tr>
<td>bool findUntil()</td>
<td>reads data from the stream until the target string of given length or terminator string is found.</td>
</tr>
</tbody>
</table>
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...

<table>
<thead>
<tr>
<th>Size_t</th>
<th><code>readBytesUntil</code> (char terminator, char *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>readBytesUntil()</code> 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 <code>setTimeout()</code>). More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size_t</th>
<th><code>readBytesUntil</code> (char terminator, uint8_t *buffer, size_t length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>readBytesUntil()</code> 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 <code>setTimeout()</code>). More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th><code>readString</code> ()</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>readString()</code> reads characters from a stream into a string. The function terminates if it times out (see <code>setTimeout()</code>). More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th><code>readStringUntil</code> (char terminator)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>readStringUntil()</code> reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see <code>setTimeout()</code>). More...</td>
<td></td>
</tr>
</tbody>
</table>

Public Member Functions inherited from Print

<table>
<thead>
<tr>
<th>int</th>
<th><code>getWriteError</code> ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to get write error number. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th><code>clearWriteError</code> ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to clear write error number. More...</td>
<td></td>
</tr>
</tbody>
</table>

| 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)
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_t print (double, int=BIN)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t print (const Printable &amp;)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (void)</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (const String &amp;s)</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (const char [])</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (char)</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (unsigned char, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object with carriage ret (&quot;&quot;) and new line ('). More...</td>
<td></td>
</tr>
<tr>
<td><code>size_t println (int, int=DEC)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setLocalPortStart(uint16_t port)</code></td>
<td>This function is used to set local port number. More...</td>
</tr>
<tr>
<td><code>stopAll()</code></td>
<td>This function is used to stop all WiFiClient session. More...</td>
</tr>
<tr>
<td><code>stopAllExcept(WiFiClient *c)</code></td>
<td>This function is used to stop all WiFiClient session without exC. More...</td>
</tr>
</tbody>
</table>
## Protected Member Functions

### WiFiClient

- **Function**: `WiFiClient(ClientContext *client)`

### int8_t

- **Function**: `_connected (void *tpcb, int8_t err)`

### void

- **Function**: `_err (int8_t err)`

### Protected Member Functions inherited from Stream

- **Function**: `int timedRead ()`

- **Function**: `int timed Peek ()`

- **Function**: `int peekNextDigit ()`

- **Function**: `long parseInt (char skipChar)`

- **Function**: `float parseFloat (char skipChar)`

### Protected Member Functions inherited from Print

- **Function**: `void setWriteError (int err=1)`
## Static Protected Member Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static int8_t _s_connected (void *arg, void *tpcb, int8_t err)</code></td>
<td>Function to check the connection status.</td>
</tr>
<tr>
<td><code>static void _s_err (void *arg, int8_t err)</code></td>
<td>Function to handle errors.</td>
</tr>
</tbody>
</table>

### Static Protected Member Functions inherited from `SList<
WiFiClient>`

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void _add (WiFiClient *self)</code></td>
<td>Function to add a WiFiClient to the list.</td>
</tr>
<tr>
<td><code>static void _remove (WiFiClient *self)</code></td>
<td>Function to remove a WiFiClient from the list.</td>
</tr>
</tbody>
</table>
## Protected Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClientContext *</code></td>
<td><code>_client</code></td>
</tr>
</tbody>
</table>

- Protected Attributes inherited from `Stream`
  - `unsigned long` _timeout_  
  - `unsigned long` _startMillis_  

- Protected Attributes inherited from `SList< WiFiClient >`
  - `WiFiClient *` _next_
### Static Protected Attributes

<table>
<thead>
<tr>
<th>Static uint16_t</th>
<th>_localPort = 0</th>
</tr>
</thead>
</table>

- Static Protected Attributes inherited from `SLList< 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).

Note
**write()** [2/4]

```cpp
size_t Print::write
```

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**
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**
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**
Inherited by Wi-FiClass.
**Public Member Functions**

<table>
<thead>
<tr>
<th>WiFiMode_t</th>
<th>getMode()</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to Get the WiFi's work mode.</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>hostByName (const char *aHostname, IPAddress &amp;aResult)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to resolve hostname.</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>hostByName (const char *aHostname, IPAddress &amp;aResult, uint32_t timeout_ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function is used to resolve hostname.</td>
<td>More...</td>
</tr>
</tbody>
</table>
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

**Note**
hostByName() [1/2]

```cpp
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 resolving the hostname.

**Returns**

1 - success, 0 - failure.

**Note**
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 resolve the hostname.

Returns
1 - success, 0 - failure.

Note

Generated by doxygen 1.8.14
Inherited by WiFiClass.
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int oneshotStart ()</td>
<td>This function is used to start oneshot configuration network. More...</td>
</tr>
<tr>
<td>int oneshotStop ()</td>
<td>This function is used to stop oneshot configuration network. More...</td>
</tr>
<tr>
<td>int oneshotGetState ()</td>
<td>This function is used to get oneshot state. More...</td>
</tr>
<tr>
<td>int oneshotSetMode (ONESHOT_MODE mode)</td>
<td>This function is used to get oneshot state. More...</td>
</tr>
<tr>
<td>int oneshotGetMode ()</td>
<td>This function is used to get oneshot mode. More...</td>
</tr>
</tbody>
</table>
Member Function Documentation
◆ oneshotGetMode()

```cpp
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

**Note**
◆ oneshotGetState()

```c
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

| Note |
◆ oneshotSetMode()

```cpp
int WiFiOneshotClass::oneshotSetMode ( ONESHOT_MODE mode )
```

This function is used to get oneshot state.

**Parameters**

- **[in]** `mode`
  - 0: UDP
  - 1: AP+SOCKET
  - 2: AP+WEBSEVER

- **[out]** `None`

**Returns**

None

**Note**
◆ oneshotStart()

```c
int WiFiOneshotClass::oneshotStart ()
```

This function is used to start oneshot configuration network.

**Parameters**

- `[in]` None
- `[out]` None

**Returns**

None

| Note |
◆ 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
Inherited by **WiFiClass**.
## Public Member Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8_t</td>
<td><code>scanNetworks</code> (bool async=false, bool show_hidden=false, uint8_t <code>channel</code>=0, uint8_t *ssid=NULL)</td>
<td>This function is used to begin scan the WiFi network.</td>
<td></td>
</tr>
<tr>
<td>int8_t</td>
<td><code>scanComplete</code> ()</td>
<td>This function is used to get the scan status.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><code>scanDelete</code> ()</td>
<td>This function is used to free the buffer of last scan result.</td>
<td>More...</td>
</tr>
<tr>
<td>bool</td>
<td><code>getNetworkInfo</code> (uint8_t networkItem, char *&amp;ssid, uint8_t &amp;encryptionType, int32_t &amp;RSSI, uint8_t *&amp;BSSID, int32_t &amp;channel, bool &amp;isHidden)</td>
<td>This function is used to get the network info via last scan.</td>
<td>More...</td>
</tr>
<tr>
<td>char *</td>
<td><code>SSID</code> (uint8_t networkItem)</td>
<td>This function is used to get ssid.</td>
<td>More...</td>
</tr>
<tr>
<td>uint32_t</td>
<td><code>encryptionType</code> (uint8_t networkItem)</td>
<td>This function is used to get encryption type.</td>
<td>More...</td>
</tr>
<tr>
<td>int32_t</td>
<td><code>RSSI</code> (uint8_t networkItem)</td>
<td>This function is used to get RSSI.</td>
<td>More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td><code>BSSID</code> (uint8_t networkItem)</td>
<td>This function is used to get the bssid.</td>
<td>More...</td>
</tr>
<tr>
<td>char *</td>
<td><code>BSSIDstr</code> (uint8_t networkItem)</td>
<td>This function is used to get the bssid.</td>
<td>More...</td>
</tr>
<tr>
<td>int32_t</td>
<td><code>channel</code> (uint8_t networkItem)</td>
<td>This function is used to get the bssid.</td>
<td>More...</td>
</tr>
</tbody>
</table>
bool isHidden (uint8_t networkItem)
This function is used to get the hidden flag. More...
### Static Protected Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void _scanDone ()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static void * _getScanInfoByIndex (int i)</td>
<td></td>
<td>(int i)</td>
</tr>
</tbody>
</table>
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()

```c
char * WiFiScanClass::BSSIDstr ( uint8_t networkItem )
```

This function is used to get the bssid.

**Parameters**
- `[in]` `networkItem` Specify the index of scan result.

**Returns**
- The string of the bssid.

**Note**
channel()

```c
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.

**Note**
◆ encryptionType()

```c
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

| Note |
◆ getNetworkInfo()

```cpp
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** The 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

**Note**
isHidden()

```cpp
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

**Note**
**RSSI()**

```c
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

| Note |
**scanComplete()**

```c
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.

**Note**
**scanDelete()**

void WiFiScanClass::scanDelete ( )

This function is used to free the buffer of last scan result.

**Parameters**

[in] None

**Returns**

None

**Note**
**scanNetworks()**

```c
int8_t WiFiScanClass::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.

**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.

**Note**
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
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...

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td><strong>hasClient</strong> ()</td>
</tr>
<tr>
<td>void</td>
<td><strong>begin</strong> ()</td>
</tr>
<tr>
<td>void</td>
<td><strong>begin</strong> (uint16_t port)</td>
</tr>
<tr>
<td>void</td>
<td><strong>setNoDelay</strong> (bool nodelay)</td>
</tr>
<tr>
<td>bool</td>
<td><strong>getNoDelay</strong> ()</td>
</tr>
<tr>
<td>virtual size_t</td>
<td><strong>write</strong> (uint8_t)</td>
</tr>
<tr>
<td>virtual size_t</td>
<td><strong>write</strong> (const uint8_t *buf, size_t size)</td>
</tr>
<tr>
<td>uint8_t</td>
<td><strong>status</strong> ()</td>
</tr>
<tr>
<td>void</td>
<td><strong>close</strong> ()</td>
</tr>
<tr>
<td></td>
<td>This function is used to start the WiFiServer.  More...</td>
</tr>
<tr>
<td></td>
<td>This function is used to set no-delay flag.  More...</td>
</tr>
<tr>
<td></td>
<td>This function is used to get no-delay flag.  More...</td>
</tr>
<tr>
<td></td>
<td>This function is used to send the message (one byte) to peer.  More...</td>
</tr>
<tr>
<td></td>
<td>This function is used to send the message to peer.  More...</td>
</tr>
<tr>
<td></td>
<td>This function is used to close the connection.  More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void stop ()</code></td>
<td>This function is used to close the connection.</td>
</tr>
<tr>
<td><code>long _accept (tcp_pcb *newpcb, long err)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Public Member Functions inherited from Print</strong></td>
<td></td>
</tr>
<tr>
<td><code>int getWriteError ()</code></td>
<td>This function is used to get write error number.</td>
</tr>
<tr>
<td><code>void clearWriteError ()</code></td>
<td>This function is used to clear write error number.</td>
</tr>
<tr>
<td><code>size_t write (const char *str)</code></td>
<td>This function is used to write buffer to the interface defined by the object.</td>
</tr>
<tr>
<td><code>size_t write (const char *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object.</td>
</tr>
<tr>
<td><code>size_t print (const String &amp;)</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td><code>size_t print (const char [])</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td><code>size_t print (char)</code></td>
<td>This function is used to print buffer to the interface defined by the object.</td>
</tr>
<tr>
<td><code>size_t print (unsigned char, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object.</td>
</tr>
<tr>
<td>Function</td>
<td>Signature</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td><code>print</code></td>
<td><code>size_t print(int, int=DEC)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t print(unsigned int, int=DEC)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t print(long, int=DEC)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t print(unsigned long, int=DEC)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t print(double, int=BIN)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t print(const Printable &amp;)</code></td>
</tr>
<tr>
<td><code>println</code></td>
<td><code>size_t println(void)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t println(const String &amp;s)</code></td>
</tr>
<tr>
<td></td>
<td><code>size_t println(const char []])</code></td>
</tr>
</tbody>
</table>


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 ('\n'). More...
Static Public Member Functions

static err_t _s_accept (void *arg, tcp pcb *newpcb, err_t err)
<table>
<thead>
<tr>
<th>Protected Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t _port</td>
<td></td>
</tr>
<tr>
<td>IPAddress _addr</td>
<td></td>
</tr>
<tr>
<td>tcp pcb * _pcb</td>
<td></td>
</tr>
<tr>
<td>bool _noDelay = false</td>
<td></td>
</tr>
</tbody>
</table>
Additional Inherited Members

- Protected Member Functions inherited from Print
  
  ```cpp
  void setWriteError (int err=1)
  ```
Inherited by WiFiClass.
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><code>int begin(const char *ssid, const char *passphrase=NULL, unsigned int channel=0, const unsigned char bssid[6]=NULL, bool connect=true)</code></strong></td>
<td>This function is used to start the wifi module as station mode. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>int begin(char *ssid, char *passphrase=NULL, int channel=0, unsigned char bssid[6]=NULL, bool connect=true)</code></strong></td>
<td>This function is used to start the wifi module as station mode. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>int begin()</code></strong></td>
<td>This function is used to start the wifi module as station mode. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool reconnect()</code></strong></td>
<td>This function is used to reconnect the AP. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool disconnect(bool wifioff=false)</code></strong></td>
<td>This function is used to disconnect the wifi. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool isConnected()</code></strong></td>
<td>This function is used to get the wifi mode connect status. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool setAutoConnect(bool autoConnect)</code></strong></td>
<td>This function is used to set auto connect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool getAutoConnect()</code></strong></td>
<td>This function is used to get auto connect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td><strong><code>bool setAutoReconnect(bool autoReconnect)</code></strong></td>
<td>This function is used to set auto reconnect flag. <a href="#">More...</a></td>
</tr>
<tr>
<td>Type</td>
<td>Function Name</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>bool</td>
<td><code>getAutoReconnect()</code></td>
</tr>
<tr>
<td>uint8_t</td>
<td><code>waitForConnectResult()</code></td>
</tr>
<tr>
<td>IPAddress</td>
<td><code>localIP()</code></td>
</tr>
<tr>
<td>char *</td>
<td><code>macAddress()</code></td>
</tr>
<tr>
<td>char *</td>
<td><code>macAddressStr()</code></td>
</tr>
<tr>
<td>IPAddress</td>
<td><code>subnetMask()</code></td>
</tr>
<tr>
<td>IPAddress</td>
<td><code>gatewayIP()</code></td>
</tr>
<tr>
<td>IPAddress</td>
<td><code>dnsIP(uint8_t dns_no=0)</code></td>
</tr>
<tr>
<td>char *</td>
<td><code>hostname()</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>hostname(char *aHostname)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>hostname(const char *aHostname)</code></td>
</tr>
<tr>
<td>wl_status_t</td>
<td><code>status()</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>char * statusStr ()</td>
<td>This function is used to get the status during Station mode. More...</td>
</tr>
<tr>
<td>char * SSID () const</td>
<td>This function is used to get the SSID used by the module. More...</td>
</tr>
<tr>
<td>char * psk () const</td>
<td>This function is used to get the psk used by the module. More...</td>
</tr>
<tr>
<td>uint8_t * BSSID ()</td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td>char * BSSIDstr ()</td>
<td>This function is used to get the BSSID which is connected by the module. More...</td>
</tr>
<tr>
<td>int32_t RSSI ()</td>
<td>This function is used to get the RSSI. More...</td>
</tr>
</tbody>
</table>
Inherits `UDP`, and `SList< WiFiUDP >`. 
# Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WiFiUDP ()</strong></td>
<td>This function is constructor, it's used to creates a named instance of the WiFiUDP class that can send and receive UDP messages. More...</td>
</tr>
<tr>
<td><strong>WiFiUDP (const WiFiUDP &amp;other)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>WiFiUDP &amp; operator= (const WiFiUDP &amp;rhs)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>~WiFiUDP ()</strong></td>
<td>This function is deconstructor, it's used to release WiFiUDP class. More...</td>
</tr>
<tr>
<td><strong>operator bool () const</strong></td>
<td></td>
</tr>
<tr>
<td><strong>virtual uint8_t begin (uint16_t port)</strong></td>
<td>This function is used to initializes the WiFiUDP library and network settings, Starts UDP socket, listening at local port. More...</td>
</tr>
<tr>
<td><strong>virtual void stop ()</strong></td>
<td>This function is used to disconnect from the server. Release any resource being used during the UDP session. More...</td>
</tr>
<tr>
<td><strong>uint8_t beginMulticast (IPAddress interfaceAddr, IPAddress multicast, uint16_t port)</strong></td>
<td>This function is used to join a multicast group and listen on the given port. More...</td>
</tr>
<tr>
<td><strong>virtual int beginPacket (IPAddress ip, uint16_t port)</strong></td>
<td>This function is used to starts a connection to write UDP data to the remote connection. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Signature</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td><code>beginPacket</code></td>
<td><code>(const char *host, uint16_t port)</code></td>
</tr>
<tr>
<td><code>beginPacketMulticast</code></td>
<td><code>(IPAddress multicastAddress, uint16_t port, IPAddress interfaceAddress, int ttl=1)</code></td>
</tr>
<tr>
<td><code>endPacket</code></td>
<td><code>()</code></td>
</tr>
<tr>
<td><code>write</code></td>
<td><code>(uint8_t)</code></td>
</tr>
<tr>
<td><code>write</code></td>
<td><code>(const uint8_t *buffer, size_t size)</code></td>
</tr>
<tr>
<td><code>parsePacket</code></td>
<td><code>()</code></td>
</tr>
<tr>
<td><code>available</code></td>
<td><code>()</code></td>
</tr>
<tr>
<td><code>read</code></td>
<td><code>()</code></td>
</tr>
</tbody>
</table>
### 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 get the IP address of the remote connection. More...

### virtual uint16_t `remotePort`()

This function is used to get 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 get 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...

<table>
<thead>
<tr>
<th>Function Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual size_t write (const uint8_t *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t write (const char *str)</code></td>
<td>This function is used to write buffer to the interface defined by the object. More...</td>
</tr>
<tr>
<td><code>size_t write (const char *buffer, size_t size)</code></td>
<td>This function is used to write buffer to the interface defined by the object. More...</td>
</tr>
</tbody>
</table>

Public Member Functions inherited from Stream

<table>
<thead>
<tr>
<th>Function Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setTimeout (unsigned long timeout)</code></td>
<td><code>setTimeout()</code> 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...</td>
</tr>
<tr>
<td><code>bool find (const char *target)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool find (uint8_t *target)</code></td>
<td><code>find()</code> 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...</td>
</tr>
<tr>
<td><code>bool find (const char *target, size_t length)</code></td>
<td></td>
</tr>
</tbody>
</table>
**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...

```cpp
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...

```cpp
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...

```cpp
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...

```cpp
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...

```cpp
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...

```cpp
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.
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 $\text{setTimeout()}$). More...

String $\text{readString()}$
$\text{readString()}$ reads characters from a stream into a string. The function terminates if it times out (see $\text{setTimeout()}$). More...

String $\text{readStringUntil (char terminator)}$
$\text{readStringUntil()}$ reads characters from a stream into a string. The function terminates if the terminator character is detected or it times out (see $\text{setTimeout()}$). More...

Public Member Functions inherited from Print

int $\text{getWriteError()}$
This function is used to get write error number. More...

void $\text{clearWriteError()}$
This function is used to clear write error number. More...

size_t $\text{write (const char *str)}$
This function is used to write buffer to the interface defined by the object. More...

size_t $\text{write (const char *buffer, size_t size)}$
This function is used to write buffer to the interface defined by the object. More...

size_t $\text{print (const String &)}$
This function is used to print buffer to the interface defined by the object. More...

size_t $\text{print (const char [])}$
<table>
<thead>
<tr>
<th>function</th>
<th>signature</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print</td>
<td><code>size_t print (char)</code></td>
<td>This function is used to print buffer to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (unsigned char, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (unsigned int, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (unsigned long, int=DEC)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (double, int=BIN)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>print</td>
<td><code>size_t print (const Printable &amp;)</code></td>
<td>This function is used to print target to the interface defined by the object. More...</td>
</tr>
<tr>
<td>println</td>
<td><code>size_t println (void)</code></td>
<td></td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>stopAll ()</strong></td>
<td>This function is used to stop all <strong>WiFiUDP</strong> session. <a href="#">More...</a></td>
</tr>
<tr>
<td>static void *<em>stopAllExcept (WiFiUDP <em>exC)</em></em></td>
<td>This function is used to stop all <strong>WiFiUDP</strong> session without <strong>exC</strong>. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
## 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<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]

```cpp
size_t Print::write
```

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]**

```c
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

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

<table>
<thead>
<tr>
<th>b</th>
<th>c</th>
<th>d</th>
<th>h</th>
<th>i</th>
<th>p</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferDataSource</td>
<td>DhcpClass</td>
<td>Printable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BufferedStreamDataSource</td>
<td>DNSClient</td>
<td>ProgmemStream</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>DNSServer</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientContext</td>
<td>HardwareSerial</td>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudClass</td>
<td>IPAddress</td>
<td>SLList</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataSource</td>
<td>IP</td>
<td>SPIClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Print</td>
<td>SPISettings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TwoWire</td>
<td>Stream</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generated by [doxygen] 1.8.14
## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<table>
<thead>
<tr>
<th>ClientContext</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudClass</td>
<td></td>
</tr>
<tr>
<td>DataSource</td>
<td></td>
</tr>
<tr>
<td>BufferDataSource</td>
<td></td>
</tr>
<tr>
<td>BufferedStreamDataSource&lt; TStream &gt;</td>
<td></td>
</tr>
<tr>
<td>DhcpClass</td>
<td></td>
</tr>
<tr>
<td>DNSClient</td>
<td></td>
</tr>
<tr>
<td>DNSServer</td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>WiFiServer</td>
<td></td>
</tr>
<tr>
<td>Stream</td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td></td>
</tr>
<tr>
<td>WiFiClient</td>
<td></td>
</tr>
<tr>
<td>HardwareSerial</td>
<td></td>
</tr>
<tr>
<td>TwoWire</td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
</tr>
<tr>
<td>WiFiUDP</td>
<td></td>
</tr>
<tr>
<td>Printable</td>
<td></td>
</tr>
<tr>
<td>IPAddress</td>
<td></td>
</tr>
<tr>
<td>ProgmemStream</td>
<td></td>
</tr>
<tr>
<td>SLIST&lt; T &gt;</td>
<td></td>
</tr>
<tr>
<td>SLIST&lt; WiFiClient &gt;</td>
<td></td>
</tr>
</tbody>
</table>
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
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() : WiFiScanClass
  - clearWriteError() : Print
  - close() : WiFiServer
  - CloudClass() : CloudClass
  - CloudInit() : CloudClass
  - connect() : Client, WiFiClient
  - connected() : Client, WiFiClient

Generated by [doxygen](http://www.stackoverflow.com) 1.8.14
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() : WiFiClient
- disconnect() : WiFiSTAClass
- dnsIP() : WiFiSTAClass
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

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:

- **h** -

  - HardwareSerial() : [HardwareSerial](#)
  - hostByName() : [WiFiGenericClass](#)

Generated by [doxygen](https://code.google.com/p/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:

- i -

- IPAddress() : IPAddress
- isConnected() : WiFiSTAClass
- isHidden() : 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:

- l -

- `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()` : WiFiSTAClass
  - `macAddressStr()` : 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:

- o -

- 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() : Print
- println() : 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:

- `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
- `softAPdisconnect()` : WiFiAPClass
- `softAPgetStationNum()` : WiFiAPClass
- `softAPIP()` : WiFiAPClass
- `softAPmacAddress()` : WiFiAPClass
- `softAPSSK()` : 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()` : [WiFiClient](#)
  - `WiFiServer()` : [WiFiServer](#)
  - `WiFiUDP()` : [WiFiUDP](#)
  - `write()` : [Client](#), [HardwareSerial](#), [Print](#), [TwoWire](#), [UDP](#), [WiFiClient](#), [WiFiServer](#), [WiFiUDP](#)

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:

- ~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() : WiFiScanClass
- clearWriteError() : Print
- close() : WiFiServer
- CloudClass() : CloudClass
- CloudInit() : 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
- fromString() : IPAddress
- g -

- getAutoConnect() : WiFiSTAClass
- getAutoReconnect() : WiFiSTAClass
- getHostByName() : DNSClient
- getKeepAliveCount() : WiFiClient
- getKeepAliveIdle() : WiFiClient
- getKeepAliveInterval() : WiFiClient
- getMode() : WiFiGenericClass
- getNetworkInfo() : WiFiScanClass
- getNoDelay() : WiFiClient, WiFiServer
- getwayIP() : WiFiSTAClass
- getWriteError() : Print
- h -

- HardwareSerial() : HardwareSerial
- hostByName() : WiFiGenericClass

Generated by doxygen 1.8.14
- i -

- IPAddress()
- isConnected()
- isHidden()
- isKeepAliveEnabled()
W60X_Arduino

- k -

- keepAlive() : WiFiClient
localIP() : WiFiClient , WiFiSTAClass
localPort() : WiFiClient , WiFiUDP
- m -

- macAddress() : WiFiSTAClass
- macAddressStr() : WiFiSTAClass
- o -

- oneshotGetMode() : WiFiOneshotClass
- oneshotGetState() : WiFiOneshotClass
- oneshotSetMode() : WiFiOneshotClass
- oneshotStart() : WiFiOneshotClass
- oneshotStop() : WiFiOneshotClass
- operator uint32_t() : IPAddress
- operator=() : IPAddress
- operator==() : IPAddress
- operator[]() : IPAddress
- p -

- parseFloat(): Stream
- parseInt(): Stream
- parsePacket(): UDP, WiFiUDP
- peek(): Client, HardwareSerial, Stream, TwoWire, UDP, WiFiClient, WiFiUDP
- print(): Print
- println(): Print
- printTo(): IPAddress, Printable
- psk(): WiFiSTAClass
- r -

- read() : \texttt{Client}, \texttt{HardwareSerial}, \texttt{Stream}, \texttt{TwoWire}, \texttt{UDP}, \texttt{WiFiClient}, \texttt{WiFiUDP}
- readBytes() : \texttt{Stream}
- readBytesUntil() : \texttt{Stream}
- readString() : \texttt{Stream}
- readStringUntil() : \texttt{Stream}
- reconnect() : \texttt{WiFiSTAClass}
- remoteIP() : \texttt{UDP}, \texttt{WiFiClient}, \texttt{WiFiUDP}
- remotePort() : \texttt{UDP}, \texttt{WiFiClient}, \texttt{WiFiUDP}
- RSSI() : \texttt{WiFiScanClass}, \texttt{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
- 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
W60X_Arduino

- t -

- toString() : IPAddress
- transfer() : SPIClass
- transfer16() : SPIClass
- transferRead() : SPIClass
- transferWrite() : SPIClass
waitForConnectResult() : WiFiSTAClass
WiFiClient() : WiFiClient
WiFiServer() : WiFiServer
WiFiUDP() : WiFiUDP
write() : Client, HardwareSerial, Print, TwoWire, UDP, WiFiClient, WiFiServer, WiFiUDP
- ~ -

- ~WiFiClient() : WiFiClient
- ~WiFiUDP() : WiFiUDP