

# Qlight RF Library API Manual

---

v1.3.2

2019-08-16

## 1. Summary

### ■ Summary

This document is C# DLL library API reference for Qlight Wireless Solution Control/ Management.

### ■ Hardware requirement

- Processor: 1GHz or higher
- RAM : 2GByte or more
- Disk space (minimum): 4.5GB or more

### ■ Software requirement

- Development tool: Recommend Visual Studio 2017 or higher
- .NET Framework 4.6.1





### ■ USB Device Driver

- Install USB dongle & Gateway USB Device Driver : **CP210x\_Windows\_Drivers**
- USB TO SERIAL DEVICE DRIVER must be installed for USB connection. It can be downloaded via website.

### ■ DLL and API application example provided

- DLL : **QLightRFLibrary.dll**
- SDK program : QLightRFLibraryExample\_Vx.x.x\_Portable ->**QLightRFLibraryExample.exe**
- SDK SOURCE : **QLightRFLibraryExample\_Vx.x.x\_source**

## ■ Wireless Products name and description

관리 프로그램	GATEWAY	USB DONGLE	라우터(무선신호정보기기)
			

- **Application** : Application for the PC supports a comprehensive control by setting up, monitoring, controlling and analyzing the wireless signal information device.  
Provide API for developers based on C# along with management tool.  
Take advantage of mobile apps to extend your manager's coverage.
- **GATEWAY** : Devices that control and monitor the router wirelessly by connecting to PC application via LAN or USB
- **USB DONGLE** : Device that controls and monitors the router wirelessly through serial communication by connecting to PC's USB port.
- **Router(Wireless Signaling Device)** : **Wireless signaling device** with LED color display and sounder

## 2. Manual for Basic operational procedure

- Basic operational procedure means executing sequences of API to control Router.
- It is a basic API execution sequence when controlling the Router by using USB DONGLE or GATEWAY. It assumes that all devices' status are already confirmed.
- Please refer to "4. QLightAPI" for more about API.

### 2.1 API for DLL configuration

- DLL configuration is to set default environment for operating the wireless product. Software engineer can make a decision in terms of configuration based on the environment.
- API that creates a list for Router information on DLL and scanning Routers' status information in a regular basis as well as using repeat function.

#### 1) Register Router Scanning

- Routers with the same PAN ID are automatically searched and registered on DLL per group. Once Routers are registered on DLL, you can periodically update Router's status by setting scanning cycle.
- It can be manually scanned when Router status information is required in case Routers are not registered.

Please refer to "4. QLightAPI" for details.

```
public static void SearchRouters (void)
```

## 2) Setting Router scanning cycle

Scan cycle to check the status information of registered Router. 2 seconds is recommended since frequent scans increase wireless traffic. If set 0, scanning is disabled.

```
public static void SetScanInterval(int seconds)
```

## 3) How to set Router scanning

When taking care of multiple groups via GATEWAY, set whether to scan GATEWAY by group or to scan every single Router by PC Application. If you use multiple groups, we recommend scanning by group.

```
public static bool SetScanMethod(ScanMethod method)
```

## 4) Set REPEAT time

API that sets the number of times to repeat for hopping between Routers. Recommended value is 3 to 5 because too many REPEATs increase communication traffic. This setting is the number of REPEAT when sending packets from USB DONGLE or GATEWAY to the Router.

```
public static void SetTTL (ushort ttl)
```

## 5) Set transmission time

When Router or USB DONGLE/ GATEWAY sends data, it decides default transmission times. Even if data transmission command is issued, the data will be transmitted in accordance with the automatic set number of times in case the wireless environment is not good or data transmission rate decreases. Recommended value is 2 to 3. Too many REPEATs may increase communication traffic.

```
public static bool SetSendNumber(ushort num)
```

## 2.2 Router control sequence by using USB DONGLE

### 1) Connect USB DONGLE

- It adopts a USB to Serial converter so necessary to check COM Port before connection.

```
static bool ConnectUSB Dongle(string portName)
```

### 2) Check USB DONGLE information

- Check configuration on USB DONGLE(GROUP ID, PAN ID, CHANNEL, TX POWER..)

```
static Task<USB DongleInfo> GetInfoUSB Dongle (string portName)
```

### 3) Control LED module on ROUTER

- Control ROUTER's LED Color per NODE ID.

```
static bool ControlRoutersLED(ushort[] nodeIds, LEDState red,  
LEDState orange, LEDState green, LEDState blue, LEDState white)
```

### 4) Control ROUTER Sounder

- Select ROUTER's sound pattern and play per NODE ID.

```
static bool ControlRoutersSound(ushort[] nodeIds, ushort soundNumber)
```

### 5) Termination of USB DONGLE connection

- Terminate connection of USB DONGLE.

```
static void DisconnectUSB Dongle (string portName)
```

## 2.3 Control sequence of Router via USB connection of GATEWAY

### 1) USB connection of GATEWAY

- It adopts a USB to Serial converter so necessary to check COM Port before connection.

```
static bool ConnectGatewayUSB (string portName)
```

### 2) Check USB connection of GATEWAY

- Check configuration on GATEWAY.(Ethernet Networks information, PORT, GROUP ID, PAN ID, CHANNEL, TX POWER...)

```
static Task<GatewayInfo> GetInfoGatewayUSB (string portName)
```

### 3) Control LED module on ROUTER

- Please refer to "2.2.3) Control LED module on ROUTER"

### 4) Control ROUTER Sounder

- Please refer to "2.2.4) Control ROUTER Sounder"

### 5) Termination of USB GATEWAY connection

- Terminate connection of Gateway from USB.

```
static void DisconnectGatewayUSB (string portName)
```

## 2.4 LAN connection of GATEWAY

### 1) LAN connection of GATEWAY

- How to LAN connection of Gateway

```
static bool ConnectGatewayLAN (string ipAddress, int portNumber)
```

### 2) Check GATEWAY LAN information

- Check configuration on GATEWAY.(Ethernet Networks information, PORT, GROUP ID, PAN ID, CHANNEL, TX POWER..)

```
static Task<GatewayInfo> GetInfoGatewayLAN (string ipAddress)
```

### 3) Control LED module on ROUTER

- Please refer to "2.2.3) Control LED module on ROUTER"

#### 4) Control ROUTER Sounder

- Please refer to "2.2.4) Control ROUTER Sounder"

#### 5) Termination of GATEWAY LAN connection

- Terminate connection of Gateway from LAN.

```
static void DisconnectGatewayLAN (string ipAddress)
```

### 3. CLASS

#### 3.1 ENUM

```
public enum LEDState { UNKNOWN, OFF, ON, BLINK }
```

```
public enum SoundType { UNKNOWN, SPEAKER, BUZZER }
```

```
public enum SoundSource { UNKNOWN, NONE, WS, WP, WM, WA, WB }
```

```
public enum ScanMethod{ GROUP = 0, ALL = 1 }
```

#### 3.2 STRUCTURE

##### 1) RouterStatus

Structure shows Router's status.

Variable name	Type	Comment
<b>Red</b>	LEDState	Red LED status
<b>Orange</b>	LEDState	Orange LED status
<b>Green</b>	LEDState	Green LED status
<b>Blue</b>	LEDState	Blue LED status
<b>White</b>	LEDState	White LED status
<b>SoundNumber</b>	ushort	Sound playback status (0 means not play)

##### 2) RouterInfo

Structure shows Router's information

Variable name	Type	Comment
---------------	------	---------

<b>NodeId</b>	ushort	Router node ID
<b>GroupId</b>	ushort	Router group ID
<b>Name</b>	string	Router name(Max. 20 bytes)
<b>PanId</b>	ushort	Wireless network PAN ID, to be used as customer ID
<b>Channel</b>	ushort	Wireless network channel (11 to 26)
<b>TxPower</b>	int	Wireless network strength (-32 to 10, The higher the stronger)
<b>SoundType</b>	SoundType	Sound type (Please refer to Enum)
<b>SoundSource</b>	SoundSource	Sound source (Please refer to Enum)
<b>LampLayerCount</b>	ushort	Number of LED module (0 to 5)

### 3) USB DongleInfo

Structure shows USB DONGLE's information

Variable name	Type	Comment
<b>PortName</b>	string	Port name to be connected (ex. "COM4")
<b>GroupId</b>	ushort	Group ID
<b>Name</b>	string	Name of USB DONGLE(Max. 20 bytes)
<b>PanId</b>	ushort	Wireless network PAN ID, to be used as customer ID
<b>Channel</b>	ushort	Wireless network channel
<b>TxPower</b>	ushort	Wireless network strength (Read only)

### 4) GatewayInfo

Structure shows Gateway's information

Variable name	Type	Comment
<b>GroupId</b>	ushort	Group ID
<b>Name</b>	string	Name of gateway (Max. 20 bytes)
<b>PanId</b>	ushort	Wireless network PAN ID, to be used as customer ID
<b>Channel</b>	ushort	Wireless network channel (11 to 26)
<b>TxPower</b>	int	Wireless network strength (-32 to 10, The higher the stronger)
<b>IpAddress</b>	string	IP address of gateway itself
<b>SubnetMask</b>	string	Subnet mask of network adaptor
<b>DefaultGatewayAddress</b>	string	Gateway address of network adaptor
<b>MacAddress</b>	string	MAC address of network adaptor
<b>TCPPortNumber</b>	int	TCP port No. of gateway itself(Default : 32177)
<b>IsMaster</b>	bool	Check if it is Master gateway

<b>MasterIp</b>	string	IP address of Master gateway
<b>MasterGatewayPortNumber</b>	int	TCP port No. of Master gateway(Default : 32177)
<b>CompanyId</b>	string	CompanyId for Mobile push

## 5) ControllerInfo

Common structure shows USB DONGLE or Gateway information

Variable name	Type	Comment
<b>Address string</b>	ushort	Port or IP address
<b>GroupId ushort</b>	string	Group ID
<b>Name</b>	String	Name of USB DONGLE or gateway
<b>PanId</b>	ushort	Wireless network PAN ID
<b>Channel</b>	ushort	Wireless network channel (11 to 26)
<b>TxPower</b>	ushort	Wireless network strength (-32 to 10, The higher the stronger)

## 6) MobileInfo (Soon to be updated)

Structure shows mobile connection and push alarm information

Variable name	Type	Comment
<b>MobileId</b>	string	Mobile connection ID
<b>MobilePassword</b>	string	Mobile connection network
<b>MobileAlarmEnable</b>	bool	Mobile push alarm (All push alarm off if false)
<b>MobileAlarmRed</b>	bool	Push alarm set when Red LED status is changed (Only if MobileAlarmEnable is true)
<b>MobileAlarmOrange</b>	bool	Push alarm set when Orange LED status is changed (Only if MobileAlarmEnable is true)
<b>MobileAlarmGreen</b>	bool	Push alarm set when Greeb LED status is changed (Only if MobileAlarmEnable is true)
<b>MobileAlarmBlue</b>	bool	Push alarm set when Blue LED status is changed (Only if MobileAlarmEnable is true)
<b>MobileAlarmWhite</b>	bool	Push alarm set when White LED status is changed (Only if MobileAlarmEnable is true)
<b>MobileAlarmSound</b>	bool	Push alarm set when Sounder status is changed (Only if MobileAlarmEnable is true)

## 4. QLightAPI

### 4.1 Functions for USB Dongle

#### 1) static bool **ConnectUSB Dongle**(string portName)

Function for USB dongle connection.

##### Parameters

- portName : Port name to connect (ex. "COM3")

##### Return

- bool : Return True if successfully connected

##### Example

```
if(QLightAPI.ConnectUSB Dongle(comboBox1.Text))
{
    PrintMessage("[USB Dongle] Connected to " + comboBox1.Text);
}
else
{
    PrintMessage("[USB Dongle] Fail to connect to " + comboBox1.Text);
}
```

#### 2) static void **DisconnectUSB Dongle** (string portName)

Disconnect USB Dongle.

##### Parameters

- portName : Port name to disconnect (ex. "COM3")

##### Return

- N/A

##### Example

```
if (QLightAPI.GetConnectStatusUSB Dongle(comboBox1.Text))
{
    QLightAPI.DisconnectUSB Dongle(comboBox1.Text);
}
```

#### 3) static bool **GetConnectStatusUSB Dongle** (string portName)

Check status of USB Dongle connection.

### Parameters

- portName : Port name to check status of connection. (ex. "COM3")

### Return

- bool : Return True if connected.

### Example

*(Refer to DisconnectUSB Dongle example)*

## 4) static Task<USB Dongle Info> **GetInfoUSB Dongle** (string portName)

Get USB Dongle configuration information(USB Dongle must be connected)

### Parameters

- portName : Serial port name that is connected to USB Dongle (ex. "COM3")

### Return

- Task<USB Dongle Info> : Return USB Dongle Info asynchronously by using await command(Please refer to structure manual). Functions that are using await must use async command.

### Example

```
private async void Button4_Click(object sender, EventArgs e)
{
    USB Dongle Info info = await QLightAPI.GetInfoUSB Dongle(comboBox1.Text);

    if (info.PortName == "")
    {
        PrintMessage("[USB Dongle] Get Info - null");
    }
    else
    {
        string msg = String.Format("[USB Dongle] Get Info - portName:{0}, dongleId:{1},
name:{2}, groupId:{3}, channel:{4}, panId:{5}, txPower:{6}", info.PortName, info.DongleId,
info.Name, info.GroupId, info.Channel, info.PanId, info.TxPower);
        PrintMessage(msg);
    }
}
```

## 4.2 Functions for Gateway (USB Connection)

### 1) static bool **ConnectGatewayUSB** (string portName)

Connect to Gateway via USB.

#### Parameters

- portName : Serial port name that is connected to Gateway (ex. "COM3")

#### Return

- bool : Return True if connection is successful.

#### Example

```
if (QLightAPI.ConnectGatewayUSB(comboBox4.Text))
{
    PrintMessage("[GW USB] Connected to " + comboBox4.Text);
}
else
{
    PrintMessage("[GW USB] Fail to connect to " + comboBox1.Text);
}
```

### 2) static void **DisconnectGatewayUSB** (string portName)

Disconnect Gateway connected to USB.

#### Parameters

- portName : Port name to disconnect (ex. "COM3")

#### Return

- N/A

#### Example

```
if (QLightAPI.GetConnectStatusGatewayUSB(comboBox4.Text))
{
    QLightAPI.DisconnectGatewayUSB(comboBox4.Text);
}
```

### 3) static bool **GetConnectStatusGatewayUSB** (string portName)

Check Gateway's USB connection status.

## Parameters

- portName : Port name to check connection status (ex. "COM3")

## Return

- bool : Return True if Gateway is connected to USB.

## Example

*(Please refer to DisconnectGatewayUSB example)*

### 4) static Task<GatewayInfo> **GetInfoGatewayUSB** (string portName)

Get Gateway information connected to USB(The Gateway must be connected)

## Parameters

- portName : Serial port name connected to Gateway (ex. "COM3")

## Return

- Task<GatewayInfo> : Return GatewayInfo asynchronously by using await command(Please refer to structure manual). Functions that are using await must use async command.

## Example

```
private async void Button13_Click(object sender, EventArgs e)
{
    GatewayInfo info = await QLightAPI.GetInfoGatewayUSB(comboBox4.Text);

    if (info.IpAddress == "")
    {
        PrintMessage("[GW USB] Get Info - null");
    }
    else
    {
        string msg = String.Format("[GW USB] Get Info - name:{0}, groupId:{1},
channel:{2}, panId:{3}, txPower:{4}, " + "\n" + "WtWtipAddr:{5}, subnet:{6}, gw:{7},
mac:{8}, tcpPort:{9}" + "\n" + "WtWtisMaster:{10}, masterIp:{11}, masterPort:{12},
companyId:{13}", info.Name, info.GroupId, info.Channel, info.PanId, info.TxPower,
info.IpAddress, info.SubnetMask, info.DefaultGatewayAddress, info.MacAddress,
info.TCPPortNumber, info.IsMaster, info.MasterIp, info.MasterPortNumber, info.CompanyId);
        PrintMessage(msg);
    }
}
```

## 4.3 Functions for Gateway(LAN connection)

### 1) static bool **ConnectGatewayLAN** (string ipAddress, int portNumber)

Connect to Gateway via LAN.

#### Parameters

- ipAddress : Gateway IP address to be connected (ex. "192.168.0.3")
- portNumber : TCP port number to be connected (default: 32177)

#### Return

- bool : Return true when connection is successful.

#### Example

```
if (QLightAPI.ConnectGatewayLAN(comboBox5.Text, 32177))
{
    PrintMessage("[GW LAN] Connected to " + comboBox5.Text);
}
else
{
    PrintMessage("[GW LAN] Fail to connect to " + comboBox5.Text);
}
```

### 2) static void **DisconnectGatewayLAN** (string ipAddress)

Disconnect Gateway connected to LAN.

#### Parameters

- ipAddress: Gateway IP address to be disconnected (ex. "192.168.0.3")

#### Return

- N/A

#### Example

```
if (QLightAPI.GetConnectStatusGatewayLAN(comboBox5.Text))
{
    QLightAPI.DisconnectGatewayLAN(comboBox5.Text);
}
```

### 3) static bool **GetConnectStatusGatewayLAN** (string ipAddress)

Check LAN connection status of Gateway.

#### Parameters

- ipAddress: IP address of gateway to check connection status (ex. "192.168.0.3")

#### Return

- bool : Return true if the Gateway is connected to LAN.

#### Example

*(Please refer to DisconnectGatewayLAN example)*

### 4) static Task<GatewayInfo> **GetInfoGatewayLAN** (string ipAddress)

Get Gateway information connected to LAN(The Gateway must be connected).

#### Parameters

- ipAddress: IP address connected to Gateway (ex. "192.168.0.3")

#### Return

- Task<GatewayInfo> : Return GatewayInfo asynchronously by using await command(Please refer to structure manual on page #3). Functions that are using await must use async command.

#### Example

```
private async void Button13_Click(object sender, EventArgs e)
{
    GatewayInfo info = await QLightAPI.GetInfoGatewayLAN(comboBox5.Text);

    if (info.IpAddress == "")
    {
        PrintMessage("[GW LAN] Get Info - null");
    }
    else
    {
        string msg = String.Format("[GW LAN] Get Info - name:{0}, groupId:{1},
channel:{2}, panId:{3}, txPower:{4}, " + "\n" + "WtWtipAddr:{5}, subnet:{6}, gw:{7},
mac:{8}, tcpPort:{9}" + "\n" + "WtWtisMaster:{10}, masterIp:{11}, masterPort:{12},
companyId:{13}", info.Name, info.GroupId, info.Channel, info.PanId, info.TxPower,
info.IpAddress, info.SubnetMask, info.DefaultGatewayAddress, info.MacAddress,
info.TCPPortNumber, info.IsMaster, info.MasterIp, info.MasterPortNumber, info.CompanyId);
        PrintMessage(msg);
    }
}
```

## 5) static Task<Dictionary<string, string> FindMacGateways (void)

Return Dictionary as ipAddress value by using Mac Address over all Gateways that are connected to LAN.  
Detect nearby gateways via UDP and return only the gateway's IP address.  
If the Mac address of the gateway is wrong or the IP address is not obtained, it will not be detected.

### Parameters

- N/A

### Return

- Task<Dictionary<string,string>> : Return Dictionary as ipAddress Value asynchronously by using Mac Address along with await command. Function that are using await command must use async command.

### Example

```
private async void Button17_Click(object sender, EventArgs e)
{
    var gwList = await QLightAPI.FindMacGateways();
    if (gwList == null || gwList.Count == 0)
    {
        PrintMessage("[GW LAN] There is no gateway...");
        return;
    }
    string[] ipList = gwList.Values.ToArray();
    Array.Sort(ipList);
    comboBox5.Items.Clear();
    comboBox5.Items.AddRange(ipList);
    PrintMessage(String.Format("[GW LAN] Found {0} gateway(s)...", ipList.Length));
}
```

## 4.4 Functions for USB Dongle or Gateway connection information

### 1) static List<ControllerInfo> GetConnectedControllerList()

Return all USB Dongle and Gateway information as a list.

### Parameters

- N/A

### Return

- List<ControllerInfo> : Return all connected USB Dongle and Gateway information as a ControllerInfo type list regardless of connection sequence. Return Null if there's no USB Dongle or Gateway.

## Example

```
List<ControllerInfo> list = QLightAPI.GetConnectedControllerList();
string printMsg = "- Connected Controller List - WrWn";
if (list == null) PrintMessage("(No list)");
else
{
    foreach (ControllerInfo info in list)
    {
        printMsg += String.Format("{0}. Addr:{1}, Name:{2}, GroupId:{3}Wn",
list.IndexOf(info), info.Address, info.Name, info.GroupId);
    }
    PrintMessage(printMsg);
}
```

## 4.5 Functions for Router control

1) static bool **ControlRoutersLED**(ushort[] nodeIds, LEDState red, LEDState orange, LEDState green, LEDState blue, LEDState white)

Control all router's LED status written on NodeIds array.

### Parameters

- nodeIds : All Routers' NodeID array to be controlled. Control pre-registered routers only.
- red : Status of red LED to be controlled (ex. LEDState.ON, LEDState.OFF, LEDState.BLINK)
- orange : Status of orange LED to be controlled (ex. LEDState.ON, LEDState.OFF, LEDState.BLINK)
- green : Status of green LED to be controlled (ex. LEDState.ON, LEDState.OFF, LEDState.BLINK)
- blue : Status of blue LED to be controlled (ex. LEDState.ON, LEDState.OFF, LEDState.BLINK)
- white : Status of white LED to be controlled (ex. LEDState.ON, LEDState.OFF, LEDState.BLINK)

### Return

- bool : If there's no controller(USB Dongle or Gateway) with the same group ID as the router to control, it returns false. It returns true if receives at least one control message.

## Example

```
ushort[] nodeId = new ushort[] { ushort.Parse(comboBox3.Text.Split('-')[0]) };
bool result = QLightAPI.ControlRoutersLED(nodeId, controlLedState[0],
controlLedState[1], controlLedState[2], controlLedState[3], controlLedState[4]);
if(result)
{
    PrintMessage(String.Format("Sent the control LED msg to {0}.",
```

```

comboBox3.Text));
    }
    else
    {
        PrintMessage(String.Format("Fail to Send the control LED msg to {0}.",
comboBox3.Text));
    }

```

## 2) static bool **ControlRoutersSound**(ushort[] nodeIds, ushort soundNumber)

Control replay status of all Routers listed in NodeIds array.

### Parameters

- nodeIds : NodeId array of all Routers to be controlled. Control pre-registered Routers only.
- soundNumber : Values are 0 to 5. Sound playback is stopped if it is 0.

### Return

- bool : If there's no controller in the same group as the Router to control, it returns false. It returns true at least one control message is received.

### Example

```

bool result = QLightAPI.ControlRoutersSound(nodeIds, (ushort)num);
if (result)
{
    PrintMessage(String.Format("Sent the control SOUND msg to {0}.", comboBox3.Text));
}
else
{
    PrintMessage(String.Format("Fail to Send the control SOUND msg to {0}.",
comboBox3.Text));
}

```

## 3) static RouterStatus **GetRouterStatus** (ushort nodeId)

Get the latest status information that is renewed inside of the library. The Router status information is renewed and returned by push data which is transferred by regular scanning or Router contact point change inside of library.

### Parameters

- nodeId : Router's node ID. Return empty if it is not pre-registered Router.

## Return

- RouterStatus : Please refer to RouterStatus structure. Return NULL if it is not pre-registered Router because it does not have monitored data.

- **Example**

```
RouterStatus status = QLightAPI.GetStatusRouter(nodeId);

SetLEDButton("Red", status.Red); // Update information of LED status button
SetLEDButton("Orange", status.Orange);
SetLEDButton("Green", status.Green);
SetLEDButton("Blue", status.Blue);
SetLEDButton("White", status.White);
SetSoundButton(status.SoundNumber);
```

## 4) static RouterInfo **GetInfoRouter** (ushort nodeId)

Get the latest Router's information that is renewed inside of library.

### Parameters

- nodeId : Router's Node ID.

## Return

- RouterInfo : Please refer to RouterInfo structure. Return NULL if it is not pre-registered Router because it does not have monitored data.

- **Example**

```
RouterInfo info = QLightAPI.GetInfoRouter(nodeId);

lbNodeId.Text = info.NodeId.ToString();
lbName.Text = (info.Name == "" || info.Name==null) ? "(none)":info.Name;
lbGroupId.Text = info.GroupId.ToString();
...
```

## 5) static void **SetRouterStatusEventHandler**(RouterStatusEventHandler handler)

Register a delegate function to be executed when the current Router status is different from the past. In order to see auto pushed data when Router's contact point changes, user defined function must be registered by this function.

### Parameters

- handler : User RouterStatusEventHandler type delegate function. Call when the Router's status value changes as below reference.

```
public delegate void RouterStatusEventHandler(ushort nodeId, RouterStatus status);
```

- nodeId : Router's nodeId that status is changed
- status : Updated latest Router's status

### Return

- N/A

### Example

```
private void Form1_Load(object sender, EventArgs e)
{
    QLighAPI.SetRouterStatusEventHandler(new RouterStatusEventHandler(handler));
}
private void handler(ushort nodeId, RouterStatus status)
{
    RefreshRouterList();
    PrintMessage(String.Format("[new Router Status.] nodeId:{0} - Red:{1}, Orange:{2},
Green:{3}, Blue:{4}, White:{5}, Sound:{6}", nodeId, status.Red, status.Orange,
status.Green, status.Blue, status.White, status.SoundNumber));
}
```

## 4.6 Function for Router registration

Routers that are not registered inside the library do not periodically scan status values inside the library. The registered router list is all deleted when the program is closed. If you want to keep the registered list even after restarting the program, save it as a file or a database in the user application and synchronize using the following functions.

### 1) static Dictionary<ushort,ushort> **GetAllRouterList(void)**

Get a list of all routers registered in the library.

#### Parameters

- N/A

#### Return

- Dictionary<ushort, ushort> : Returns a list of all routers registered inside the library(Dictionary

with node ID as key and group ID as value)

### Example

```
var list = QLightAPI.GetAllRouterList();
comboBox2.Items.Clear();
List<string> strs = new List<string>();
ushort[] nodeIds = list.Keys.ToArray();
for (int i=0; i< nodeIds.Length; i++)
{
    ushort nodeId = nodeIds[i];
    if (list.TryGetValue(nodeId, out ushort groupId))
    {
        strs.Add(String.Format("{0}-{1}", nodeId, groupId));
    }
}
comboBox2.Items.AddRange(strs.ToArray());
```

## 2) static bool **AddRouterListManually**(ushort nodeIds, ushort groupId)

Add Routers to be controlled manually from library list.

### Parameters

- nodeIds : Router's node ID to be registered manually
- groupId : Router's group ID to be registered manually. It must be the same as Group ID of USB DONGLE or Gateway connected to PC to get or control the router status information.

### Return

- bool : Returns true if registration is successful.

### Example

```
if( ushort.TryParse(textBox1.Text, out ushort nodeId) && ushort.TryParse(textBox2.Text, out ushort groupId) )
{
    if(QLightAPI.AddRouterListManually(nodeId, groupId))
    {
        PrintMessage(String.Format("Add Router Manually (nodeId:{0}, groupId:{1})", nodeId, groupId));
        RefreshRouterList();
    }
}
```

## 3) public static bool **DeleteRouterList**(ushort nodeId)

Delete the router registered in the library. The deleted router no longer performs periodic scans.

### Parameters

- nodeId : Node ID of the router to be deleted.

### Return

- bool : Returns true if the delete is successful.

### Example

```
if(ushort.TryParse(textBox1.Text, out ushort nodeId))
{
    if(QLightAPI.DeleteRouterList(nodeId))
    {
        PrintMessage(String.Format("Deleted {0} in the Router list.", nodeId));
        RefreshRouterList();
    }
}
```

## 4) public static void **SearchRouters** (void)

Search for nearby routers and register automatically. At this time, the router periodically transmits (beacons) a wireless packet containing its information and wirelessly sends a command to the router to stop the beacon when it receives it from inside the library. (It may take a long time depending on the wireless network traffic, the number of routers, the router installation location, etc.) Periodic scan of registered routers pauses while seeking. If you know node ID and group ID, you can register manually through AddRouterListManually function.)

### Parameters

- N/A.

### Return

- N/A

### Example

```
QLightAPI.SearchRouters();
PrintMessage("Request to search Routers ...");
```

## 5) public static ushort **GetGroupIdFromRouterList** (ushort nodeId)

Seeking router's group ID by using the list of routers registered in the library.

### Parameters

- nodeId : Node ID of Router that wants to know group ID

### Return

- ushort : Returns group ID. If there is no router in the list of registered routers, it returns 0.

## 4.7 Environment setting functions

You can change the environment variables necessary for library operation through the following functions. These environment variables are maintained even after restarting the program and are implemented in the file DB format inside the library.

### 1) public static void **SetScanInterval**(int seconds)

Set scanning cycle for registered routers. Periodical scan to check the status information of registered routers is used as a function to recover packet loss caused by radio collisions. Increasing scanning speed of routers can increase wireless traffic, so recommended default value is 2 seconds or more. (If set to 0, the scan function is not used.)

### 2) public static uint **GetScanInterval**(void)

Get the router scanning cycle from the library.

### 3) public static void **SetTTL** (ushort ttl)

TTL is a variable that sets the number of packet retransmissions between wireless routers when the management program sends a packet to the router. It is better to increase the value when the wireless environment is poor and the number of routers is too high. As the traffic increases, it may cause the failure of wireless communication. Therefore, an appropriate value should be set according to the environment. In general, Recommended value is 3 to 5.

### 4) public static ushort **GetTTL**(void)

Get TTL value that is configured one from the library.

### 5) public static bool **SetScanMethod**(ScanMethod method)

Depending on the method parameter value, this function defines and executes whether to scan each group at the same time or scan the whole router individually. When using multiple groups using GATEWAY, set whether to scan subroutines by GATEWAY group or routers individually. If you use multiple GATEWAYS, we recommend scanning by group.

### Parameters

- method: Refer to ScanMethod Enum

### Return

- bool : Returns true if the execution is successful.

### Example

```
if(comboBox6.Text.Equals("ALL"))
{
    if (QLightAPI.SetScanMethod(ScanMethod.ALL))
    {
        PrintMessage("Scan Method is changed to " + comboBox6.Text + ".");
    }
}
else if(comboBox6.Text.Equals("GROUP"))
{
    if (QLightAPI.SetScanMethod(ScanMethod.GROUP))
    {
        PrintMessage("Scan Method is changed to " + comboBox6.Text + ".");
    }
}
```

## 6) public static ScanMethod **GetScanMethod()**

This function is to check whether the type of router's status information is scanned in groups or in the entire list.

### Parameters

- N/A

### Return

- ScanMethod: Please refer to ScanMethod Enum

### Example

```
ScanMethod m = QLightAPI.GetScanMethod();
if (m == ScanMethod.ALL) comboBox6.SelectedItem = "ALL";
else if (m == ScanMethod.GROUP) comboBox6.SelectedItem = "GROUP";
```

## 7) public static bool **SetSendNumber**(ushort num)

This function is to set retransmission time of the same packet in order to increase a possibility of wireless signal reception when transmit control commands. If the wireless environment is bad or the possibility of data transmission decreases, it is recommended to increase the number of retransmissions, but the wireless traffic will be increased because the data is automatically retransmitted as many as the retransmission time setting. Normally 2 to 3 times is recommended. Since the traffic increases, it is necessary to extend the router scan frequency or adjust the number of repeats appropriately.

### Parameters

- num : Packet retransmission time

### Return

- bool : Returns true if it is successfully deleted..

### Example

```
if (ushort.TryParse(textBox5.Text, out ushort num) && num >= 1 && num <= 10)
{
    if (QLightAPI.SetSendNumber(num))
    {
        PrintMessage("Send Number is changed : " + num + ".");
    }
}
```

## 8) public static ushort **GetSendNumber**()

This function is to check retransmission time in accordance with preset value in order to increase possibility of wireless signal reception.

### Parameters

- N/A

### Return

- ushort : Packet retransmission time

### Example

```
textBox5.Text = QLightAPI.GetSendNumber().ToString();
```