WDR Socket Communication
Sample Program
(Windows C#)

Contents

1. Overview

The following is an overview of sample programs to control the WDR via socket communication.

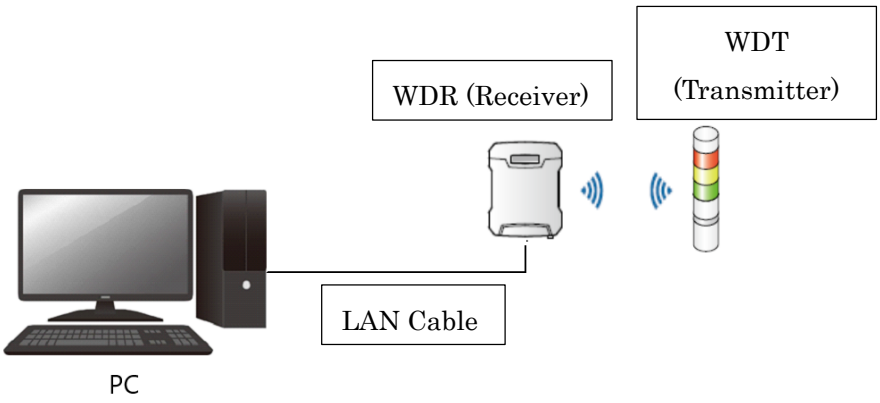
This programs are intended to be controlled with Microsoft Visual C#.

Since the programs are just samples, it is necessary to design your own unique system separately.

1.1. System Overview

The system configuration diagram of this program is as follows.

In this program, one WDR device is controlled by socket communication.



2. Development Environment

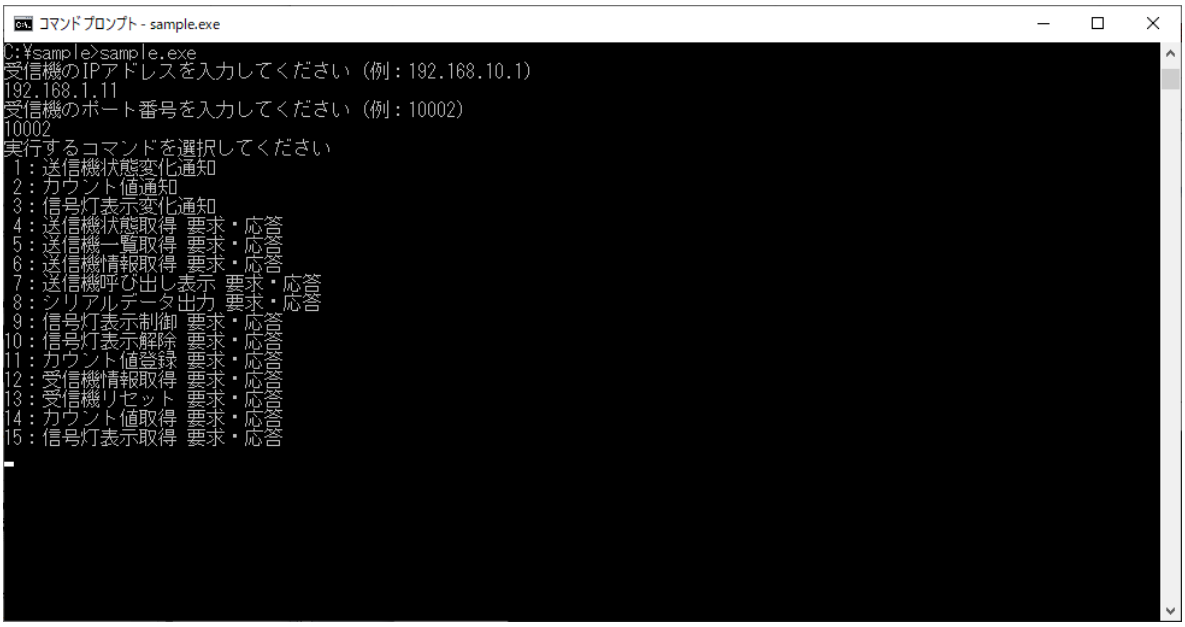
The development environment of the sample program is shown below.

Development Environment		Remarks
Development OS	Windows 10 64bit	
Development Language	C#	.Net Framework 4.5 or later
App type	CUI application	
Development Tools	Visual Studio 2019 Professional	

3. Application Overview

3.1. Command Operation

On the command prompt, navigate to the location of the sample.exe file created at build time and execute it. After entering the IP address and port number of the receiver, you will be prompted to select a command to execute. After selecting a command, follow the instructions to enter parameters.



3.1.1. Command List

Command Name	Contents
Transmitter Status Change Notification	Receives commands that notifies you when the transmitter status changes.
Count Value Notification	Receives a command notifying when the transmitter count value is updated.
Signal Tower Display Change Notification	Receives notification command when the transmitter's control of the display (LED Unit) is released (cancelled).
Transmitter Status Acquisition Request / Response	Acquires transmitter status information (change information).
Transmitter List Acquisition Request / Response	Gets a list of transmitters managed by the receiver.
Transmitter Information Acquisition Request / Response	Acquires information for a specified transmitter.
Transmitter Call Display Request / Response	The specified transmitter lights up when called.
Serial Data Output Request / Response	Outputs serial data from the RS232C interface of the specified transmitter.
Signal Tower Display Control	Controls the Signal Tower display of the specified transmitter.

Request / Response	
Signal Tower Display cancellation Request / Response	Cancels the Signal Tower display control of the specified transmitter.
Count Value Registration Request / Response	Registers the count value of the specified transmitter.
Receiver Information Acquisition Request / Response	Gets receiver information.
Receiver Reset Request / Response	Resets receiver.
Count Value Acquisition Request / Response	Gets the count value for the specified transmitter.
Signal Tower Display Acquisition Request / Response	Acquires the Signal Tower display status of the specified transmitter.

3.1.2. Transmitter Status Change Notification Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	1

3.1.3. Count Value Notification Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	2

3.1.4. Signal Tower Display Change Notification Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	3

3.1.5. Transmitter Status Acquisition Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	4
2	IEEE Transmitter Address	IEEE address of transmitter whose status is to be acquired

3.1.6. Transmitter List Acquisition Request / Response Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	5

3.1.7. Transmitter Information Acquisition Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	6
2	IEEE Transmitter Address	IEEE address of transmitter whose information is to be acquired

3.1.8. Transmitter Call Display Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	7
2	IEEE Transmitter Address	IEEE address of transmitter to be called and have it light up

3.1.9. Serial Data Output Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	8
2	IEEE Transmitter Address	IEEE address of transmitter outputting serial data
3	Serial Number	Number to determine whether or not infomration was resent
4	Output Information	Data to be output

3.1.10. Signal Tower Display Control Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	9
2	IEEE Transmitter Address	IEEE address of the transmitter that controls the display of the Signal Tower
3	Control Time	Time to control the display of the Signal Tower
4	Red Unit Lighting Pattern	LED Unit control method 00: Control by control line 10: Off 11: Continuous on 12: Blinking 13: Triple flash
5	Amber Unit Lighting Pattern	
6	Green Unit Lighting Pattern	
7	Blue Unit Lighting Pattern	
8	White Unit Lighting Pattern	
9	Alarm Pattern	Buzzer Unit control method 00: Control by control line 10: Silent 11: Continuous sound 12: Intermittent sound

3.1.11. Signal Tower Display Cancellation Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	10
2	IEEE Transmitter Address	IEEE address of the transmitter that cancels the display control of the Signal Tower

3.1.12. Count Value Registration Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	11
2	IEEE Transmitter Address	IEEE address of the transmitter that sets the count value
3	Count Registration Value	Count value to register

3.1.13. Receiver Information Acquisition Request / Response Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	12

3.1.14. Receiver Reset Request / Response Command

Specify the following command ID and execute.

No.	Input Value	Value
1	Command ID	13

3.1.15. Count Value Acquisition Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	14
2	IEEE Transmitter Address	IEEE address of transmitter from which to get count value

3.1.16. Signal Tower Display Acquisition Request / Response Command

Specify the following command ID and parameters and execute.

No.	Input Value	Value
1	Command ID	15
2	IEEE Transmitter Address	IEEE address of transmitter to acquire Signal Tower control status

3.2. Function Description

3.2.1. Function List

Function Name	Explanation
SocketOpen	Connect to WDR
SocketClose	Close the socket
RecvDatagujde	Divide received data into commands
SendCommand	Send a request command and receive a response command
RecvCommand	Receive notification command
WDR_StatusChangeNoticeCommand	Receive transmitter status change notification
WDR_CountNoticeCommand	Receive count value notification
WDR_SignalLightChangeNoticeCommand	Receive Signal Tower display change notification
WDR_TransmitterStatusRequest	Send transmitter status acquisition request and receive transmitter status acquisition response
WDR_TransmitterListRequest	Send transmitter list acquisition request and receive transmitter list acquisition response
WDR_TransmitterDataRequest	Send transmitter information acquisition request and receive transmitter information acquisition response
WDR_TransmitterCallRequest	Send transmitter call display request and receive transmitter call display response
WDR_SerialOutputRequest	Send a serial data output request and receive a serial data output response
WDR_SignalLightControlRequest	Send a Signal Tower display control request and receive a Signal Tower display control response
WDR_SignalLightLiftRequest	Send a Signal Tower display cancellation request and receive a Signal Tower display cancellation response
WDR_SignalLightCountSetRequest	Send a count value registration request and receive a count value registration response
WDR_ReceiveDataRequest	Send receiver information acquisition request and receive receiver information acquisition response
WDR_ReceiverResetRequest	Send a receiver reset request and receive a receiver reset response
WDR_SignalLightCountGetRequest	Send a count value acquisition request and receive a count value acquisition response
WDR_SignalLightDataGetRequest	Send a Signal Tower display acquisition request and receive a Signal Tower display acquisition response

3.2.2. Connect to WDR

Function Name	public static int SocketOpen(string ip, int port)	
Parameters	string ip	WDR IP address
	int port	WDR port number
Return Value	int	Success: 0, Failure: Other than 0
Explanation	Connect to the WDR with the specified IP address and port number by socket communication	

How to Use	// Define socket class variables
	private static Socket sock = null;
	
	// Main function
	static void Main()
	{
	  // Connect to WDR
	  ret = SocketOpen("192.168.10.1", 10002);
	  if (ret == -1)
	  {
	  return;
	  }
	}	

Remarks	See "4.1 Connect to WDR" for an overview of the program.4.1Close Socketan overview of the program.4.2Send a Request Command and Receive a Response Commandan overview of the program .4.4Receive Notification Commands" for an overview of the program .4.5Receive " for an overview of the program.4.7Send a Receiver	

3.2.3. Send a Count Value Acquisition Request and Receive a Count Value Acquisition Response

Function Name	public static int WDR_SignalLightCountGetRequest(ulong IEEEAddress, out WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA Data)	
Parameters	ulong IEEEAddress	IEEE address of the target transmitter
	out WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA Data	Received data
Return Value	int	Success: 0, Failure: Other than 0
Explanation	Sends a count value acquisition request command, receives a count value acquisition response command, and returns data.	

How to Use	// Main function
	static void Main()
	{
	  // Connect to WDR
	  ret = SocketOpen("192.168.10.1", 10002);
	  if (ret == -1) {
	  return;
	  }
	
	  // Send and receive count value acquisition commands
	  WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA Data;
	  Data = new WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA();	

	WDR_SignalLightCountGetRequest(IEEEAddress, out Data);
	
	// Close socket
	SocketClose();
	}

Remarks	See "4.19Send a Count Value " for an overview of the program.

3.2.4. Send Signal Tower Display Acquisition Request and Receive Signal Tower Display Acquisition Response

Function Name	public static int WDR_SignalLightDataGetRequest(ulong IEEEAddress, out WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA Data)	
Parameters	ulong IEEEAddress	IEEE address of the target transmitter
	out WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA Data	Received data
Return Value	int	Success: 0, Failure: Other than 0
Explanation	Sends a Signal Tower display acquisition request command, receives a Signal Tower display acquisition response command, and returns data.	

How to Use	// Main function
	static void Main()
	{
	// Connect to WDR
	ret = SocketOpen("192.168.10.1", 10002);
	if (ret == -1) {
	return;
	}
	
	// Send and receive Signal Tower display acquisition commands
	WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA Data;
	Data = new WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA();
	
	WDR_SignalLightDataGetRequest(IEEEAddress, out Data);
	
	// Close socket
	SocketClose();
	}	

Remarks	See "4.20Send a Signal Tower Display Acquisition Request and Receive a Signal Tower Display Acquisition " for an overview of the program.	

3.3. Constant Description

3.3.1. Receive Timeout Code

Constant Name	Value	Explanation
WSAETIMEDOUT	10060	Receive timeout judgment code

3.3.2. Product Category

Constant Name	Value	Explanation
WDR_PRODUCT_ID	0x5842	WDR product category

3.3.3. WDR Identifier

Constant Name	Value	Explanation
WDR_COMMAND	0x01	WDR command identifier code

3.3.4. Expansion

Constant Name	Value	Explanation
WDR_EXPANSION	0x00	WDR command extension code

3.3.5. Command Type

Constant Name	Value	Explanation
WDR_COMMAND_KIND_NOTICE	0x10	Notification command
WDR_COMMAND_KIND_REQUEST	0x20	Request command
WDR_COMMAND_KIND_RESPONSE	0x30	Response command

3.3.6. Command Mode

Constant Name	Value	Explanation
WDR_COMMAND_MODE_STATUS_CHANGE_NOTICE	0x2001	Transmitter status change notification
WDR_COMMAND_MODE_COUNT_NOTICE	0x2007	Count value notification
WDR_COMMAND_MODE_SIGNAL_LIGHT_CHANGE_NOTICE	0x2008	Signal Tower display change notification
WDR_COMMAND_MODE_TRANSMITTER_STATUS_REQUEST	0x2002	Transmitter status acquisition
WDR_COMMAND_MODE_TRANSMITTER_LIST_REQUEST	0x2003	Get transmitter list
WDR_COMMAND_MODE_TRANSMITTER_DATA_REQUEST	0x2004	Transmitter information acquisition
WDR_COMMAND_MODE_TRANSMITTER_CALL_REQUEST	0x4010	Transmitter call display
WDR_COMMAND_MODE_SERIAL_OUTPUT_REQUEST	0x9011	Serial data output
WDR_COMMAND_MODE_SIGNAL_LIGHT_CONTROL_REQUEST	0xE001	Signal Tower display

		control request
WDR_COMMAND_MODE_SIGNAL_LIGHT_CONTROL_RESPONSE	0xE000	Signal Tower display control response
WDR_COMMAND_MODE_SIGNAL_LIGHT_LIFT_REQUEST	0xE0FF	Signal Tower display cancellation
WDR_COMMAND_MODE_SIGNAL_LIGHT_COUNT_SET_REQUEST	0x6001	Count value registration
WDR_COMMAND_MODE_RECEIVER_DATA_REQUEST	0x2005	Receiver information acquisition
WDR_COMMAND_MODE_RECEIVER_RESET_REQUEST	0x2006	Receiver reset
WDR_COMMAND_MODE_SIGNAL_LIGHT_COUNT_GET_REQUEST	0x2009	Get count value
WDR_COMMAND_MODE_SIGNAL_LIGHT_DATA_GET_REQUEST	0x200A	Get Signal Tower display

3.3.7. Receive Timeout Time by Command Mode

Constant Name	Value	Explanation
WDR_STATUS_CHANGE_NOTICE_TIMEOUT	60*1000	For transmitter status change notification
WDR_COUNT_NOTICE_TIMEOUT	60*1000	Count value notification
WDR_SIGNAL_LIGHT_CHANGE_NOTICE_TIMEOUT	60*1000	Signal Tower display change notification
WDR_TRANSMITTER_STATUS_REQUEST_TIMEOUT	2*100	Transmitter status acquisition
WDR_TRANSMITTER_LIST_REQUEST_TIMEOUT	2*1000	Get transmitter list
WDR_TRANSMITTER_DATA_REQUEST_TIMEOUT	2*1000	Transmitter information acquisition
WDR_TRANSMITTER_CALL_REQUEST_TIMEOUT	15*1000	Transmitter call display
WDR_SERIAL_OUTPUT_REQUEST_TIMEOUT	15*1000	Serial data output
WDR_SIGNAL_LIGHT_CONTROL_TIMEOUT	15*1000	For Signal Tower display control
WDR_SIGNAL_LIGHT_LIFT_TIMEOUT	15*1000	Signal Tower display cancellation
WDR_SIGNAL_LIGHT_COUNT_SET_TIMEOUT	2*1000	Count value registration
WDR_RECEIVER_DATA_REQUEST_TIMEOUT	2*1000	Receiver information acquisition
WDR_RECEIVER_RESET_REQUEST_TIMEOUT	2*1000	Receiver reset
WDR_SIGNAL_LIGHT_COUNT_GET_TIMEOUT	2*1000	Get count value
WDR_SIGNAL_LIGHT_DATA_GET_TIMEOUT	2*1000	Get Signal Tower display

3.3.8. PNS Command Response Data

Constant Name	Value	Explanation
PNS_ACK	0x06	Normal response
PNS_NAK	0x15	Abnormal response

3.4. Structure Description

3.4.1. Version Structure

Name	WDR_VERSION_DATA

Definition	public class WDR_VERSION_DATA
	{
	// Major version
	public byte major = 0;
	// Minor version
	public byte minor = 0;
	};

Explanation	Version number structure in the response command

3.4.2. WDT Version Information Structure

Name	WDR_WDT_VERSION_DATA

Definition	public class WDR_WDT_VERSION_DATA
	{
	// Major version
	public byte major = 0;
	// Minor version
	public byte minor = 0;
	// Dummy data
	public byte dummy = 0;

Explanation	WDT version information structure in the response command

3.4.3. WDT Information Structure

Name	WDR_INFO_DATA

Definition	public class WDR_INFO_DATA
	{
	// Version information
	public WDR_WDT_VERSION_DATA version = null;
	// Status information
	public byte status = 0;
	};

Explanation	WDT information structure in the response command

3.4.4. Base Unit Information Structure

Name	WDR_BASEUNIT_DATA

Definition	public class WDR_BASEUNIT_DATA
	{
	// Unit format
	public byte format = 0;
	// Version information
	public WDR_WDT_VERSION_DATA version = null;
	// DIP switch information

	public byte dipSwitch = 0; };
Explanation	Base unit information structure in the response command

3.4.5. WDT Status Information Structure

Name	WDR_WDT_STATUS_DATA

Definition	public class WDR_WDT_STATUS_DATA
	{
	// WDT IEEE address
	public ulong IEEEAddress = 0;
	// WDT registration status
	public byte registration = 0;
	// WDT connection status
	public byte connect = 0;
	};

Explanation	WDT state information structure in the response command

3.4.6. RS232C Data Information Structure

Name	WDR_RS232C_DATA

Definition	public class WDR_RS232C_DATA
	{
	// Input information size
	public byte size = 0;
	// Serial number
	public byte serialNumber = 0;
	// Input information
	public byte[] data = new byte[60];
	};

Explanation	RS232C data information structure in the response command

3.4.7. Transmitter Status Change Notification Structure

Name	WDR_STATUS_CHANGE_NOTICE_RECV_DATA

Definition	public class WDR_STATUS_CHANGE_NOTICE_RECV_DATA
	{
	// IEEE address
	public ulong IEEEAddress = 0;
	// Serial number
	public uint serialNumber = 0;
	// Time information
	public ulong time = 0;
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information

	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Signal Tower information (red)
	public byte redUnit = 0;
	// Signal Tower information (amber)
	public byte yellowUnit = 0;
	// Signal Tower information (green)
	public byte greenUnit = 0;
	// Signal Tower information (blue)
	public byte blueUnit = 0;
	// Signal Tower information (white)
	public byte whiteUnit = 0;
	// Alarm information
	public byte buzzerUnit = 0;
	// WDT monitoring information
	public byte surveillance = 0;
	// External input information
	public byte externalInput = 0;
	// RS232C data
	public WDR_RS232C_DATA RS232CData = null;
	};

Explanation	Received data structure of transmitter status change notification command

3.4.8. Count Value Notification Structure

Name	WDR_COUNT_NOTICE_RECV_DATA

Definition	public class WDR_COUNT_NOTICE_RECV_DATA
	{
	// IEEE address
	public ulong IEEEAddress = 0;
	// Time information
	public ulong time = 0;
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information
	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Count value
	public uint countValue = 0;
	};

Explanation	Received data structure of count value notification command

3.4.9. Signal Tower Display Change Notification Structure

Name	WDR_SIGNAL_LIGHT_CHANGE_NOTICE_RECV_DATA

Definition	public class WDR_SIGNAL_LIGHT_CHANGE_NOTICE_RECV_DATA
	{
	// IEEE address
	public ulong IEEEAddress = 0;
	// Time information
	public ulong time = 0;
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information
	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Red unit
	public byte redUnit = 0;
	// Amber unit
	public byte yellowUnit = 0;
	// Green unit
	public byte greenUnit = 0;
	// Blue unit
	public byte blueUnit = 0;
	// White unit
	public byte whiteUnit = 0;
	// Alarm unit
	public byte buzzerUnit = 0;
	};

Explanation	Received data structure of Signal Tower display change notification command

3.4.10. Transmitter Status Acquisition Structure

Name	WDR_TRANSMITTER_STATUS_REQUEST_RES_DATA

Definition	public class WDR_TRANSMITTER_STATUS_REQUEST_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// Time information
	public ulong time = 0;
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information
	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Signal Tower information (red)

	public byte redUnit = 0;
	// Signal Tower information (amber)
	public byte yellowUnit = 0;
	// Signal Tower information (green)
	public byte greenUnit = 0;
	// Signal Tower information (blue)
	public byte blueUnit = 0;
	// Signal Tower information (white)
	public byte whiteUnit = 0;
	// Alarm information
	public byte buzzerUnit = 0;
	// WDT monitoring information
	public byte surveillance = 0;
	// External input information
	public byte externalInput = 0;
	// RS232C data
	public WDR_RS232C_DATA RS232CData = null;
	};

Explanation	Received data structure of transmitter status acquisition command

3.4.11. Transmitter List Acquisition Structure

Name	WDR_TRANSMITTER_LIST_REQUEST_RES_DATA

Definition	public class WDR_TRANSMITTER_LIST_REQUEST_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// Number of acquisitions
	public byte unitCount = 0;
	// WDT status information
	public WDR_WDT_STATUS_DATA[] wdtStatus = new WDR_WDT_STATUS_DATA[70];
	};

Explanation	Received data structure of transmitter list acquisition command

3.4.12. Transmitter Information Acquisition Structure

Name	WDR_TRANSMITTER_DATA_REQUEST_RES_DATA

Definition	public class WDR_TRANSMITTER_DATA_REQUEST_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// username
	public byte[] userName = new byte[121];
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information

	public WDR_BASEUNIT_DATA baseUnitData = null;
	// ExtendedPanID
	public ulong extendedPanID = 0;
	// Frequency channel
	public uint frequencyChannel = 0;
	// Signal Tower input judgment
	public byte signalLightInputJudge = 0;
	// Power settings
	public byte powerSetting = 0;
	// Counter setting
	public byte counterSetting = 0;
	// Send mode
	public ushort sendMode = 0;
	};

Explanation	Received data structure of transmitter information acquisition command

3.4.13. Transmitter Information Acquisition Extended Structure

Name	WDR_TRANSMITTER_DATA_REQUEST_RES_ADD_DATA

Definition	public class WDR_TRANSMITTER_DATA_REQUEST_RES_ADD_DATA
	{
	// Input information transmission method
	public byte inputDataTranform = 0;
	// Signal Tower format
	public byte signalLightFormat = 0;
	// Periodic transmission
	public byte regularSend = 0;
	// Simultaneous input judgment sensitivity setting
	public byte concInputSensitiveSetting = 0;
	// Received data file format
	public byte recvDataFileFormat = 0;
	// Communication setting baud rate
	public byte baudrate = 0;
	// Communication setting data length
	public byte dataLength = 0;
	// Communication setting parity
	public byte parity = 0;
	// Communication setting stop bit
	public byte stopBit = 0;
	};

Explanation	Transmitter information acquisition command data structure received when WDT_6LR_Z2_PRO is added.

3.4.14. Transmitter Call Display Structure

Name	WDR_TRANSMITTER_CALL_REQUEST_RES_DATA

Definition	public class WDR_TRANSMITTER_CALL_REQUEST_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	};

Explanation	Received data structure of transmitter call display command

3.4.15. Serial Data Output Request Structure

Name	WDR_SERIAL_OUTPUT_REQ_DATA

Definition	public class WDR_SERIAL_OUTPUT_REQ_DATA
	{
	// IEEEE address
	public ulong IEEEAddress = 0;
	// Serial number
	public byte serialNumber = 0;
	// Output information
	public List<byte> outputData = new List<byte>();
	};

Explanation	Transmission data structure of serial data output command

3.4.16. Serial Data Output Response Structure

Name	WDR_SERIAL_OUTPUT_RES_DATA

Definition	public class WDR_SERIAL_OUTPUT_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	};

Explanation	Received data structure of serial data output command

3.4.17. Signal Tower Display Control Request Structure

Name	WDR_SIGNAL_LIGHT_CONTROL_REQ_DATA

Definition	public class WDR_SIGNAL_LIGHT_CONTROL_REQ_DATA
	{
	// IEEEE address
	public ulong IEEEAddress = 0;
	// Control time
	public byte controlTime = 0;
	// Red unit
	public byte redUnit = 0;
	// Amber unit
	public byte yellowUnit = 0;
	// Green unit
	public byte greenUnit = 0;
	// Blue unit

	public byte blueUnit = 0;
	// White unit
	public byte whiteUnit = 0;
	// Alarm unit
	public byte buzzerUnit = 0;
	};

Explanation	Transmission data structure of Signal Tower display control command

3.4.18. Signal Tower Display Control Response Structure

Name	WDR_SIGNAL_LIGHT_CONTROL_RES_DATA

Definition	public class WDR_SIGNAL_LIGHT_CONTROL_RES_DATA
	{
	// Response status
	public byte recvState = 0;
	// Control state
	public byte controlState = 0;
	// Red unit
	public byte redUnit = 0;
	// Amber unit
	public byte yellowUnit = 0;
	// Green unit
	public byte greenUnit = 0;
	// Blue unit
	public byte blueUnit = 0;
	// White unit
	public byte whiteUnit = 0;
	// Alarm unit
	public byte buzzerUnit = 0;
	};

Explanation	Received data structure of Signal Tower display control command

3.4.19. Signal Tower Display Cancellation Structure

Name	WDR_SIGNAL_LIGHT_LIFT_RES_DATA

Definition	public class WDR_SIGNAL_LIGHT_LIFT_RES_DATA
	{
	// Response status
	public byte recvState = 0;
	// Control state
	public byte controlState = 0;
	// Red unit
	public byte redUnit = 0;
	// Amber unit
	public byte yellowUnit = 0;
	// Green unit
	public byte greenUnit = 0;
	// Blue unit
	public byte blueUnit = 0;
	// White unit

	public byte whiteUnit = 0;
	// Alarm unit
	public byte buzzerUnit = 0;
	};

Explanation	Received data structure of Signal Tower display cancellation command

3.4.20. Count Value Registration Request Structure

Name	WDR_SIGNAL_LIGHT_COUNT_SET_REQ_DATA

Definition	public class WDR_SIGNAL_LIGHT_COUNT_SET_REQ_DATA
	{
	// IEEEE address
	public ulong IEEEAddress = 0;
	// Count registration value
	public uint setCount = 0;
	};

Explanation	Transmission data structure of count value registration command

3.4.21. Count Value Registration Response Structure

Name	WDR_SIGNAL_LIGHT_COUNT_SET_RES_DATA

Definition	public class WDR_SIGNAL_LIGHT_COUNT_SET_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	};

Explanation	Received data structure of count value registration command

3.4.22. Receiver Information Acquisition Structure

Name	WDR_RECEIVER_DATA_REQUEST_RES_DATA

Definition	public class WDR_RECEIVER_DATA_REQUEST_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// ExtendedPanID
	public ulong extendedPanID = 0;
	// Frequency channel
	public uint frequencyChannel = 0;
	// Firmware version
	public WDR_VERSION_DATA version = null;
	// Network status
	public byte networkStatus = 0;
	// How to boot the network
	public byte networkBoot = 0;
	// Running ExtendedPanID
	public ulong actionExtendedPanID = 0;

	// Operating frequency channel
	public byte actionFrequencyChannel = 0;
	};

Explanation	Received data structure of receiver information command

3.4.23. Receiver Reset Structure

Name	WDR_RECEIVER_RESET_RES_DATA

Definition	public class WDR_RECEIVER_RESET_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	};

Explanation	Received data structure of receiver reset command

3.4.24. Count Value Acquisition Structure

Name	WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA

Definition	public class WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// Time information
	public ulong time = 0;
	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information
	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Count value
	public uint count = 0;
	};

Explanation	Received data structure of count value acquisition command

3.4.25. Signal Tower Display Acquisition Structure

Name	WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA

Definition	public class WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA
	{
	// Response status
	public byte controlState = 0;
	// Time information
	public ulong time = 0;

	// Version information
	public WDR_VERSION_DATA version = null;
	// Action mode
	public byte actionMode = 0;
	// WDT information
	public WDR_INFO_DATA wdtData = null;
	// Base unit information
	public WDR_BASEUNIT_DATA baseUnitData = null;
	// Red unit
	public byte redUnit = 0;
	// Amber unit
	public byte yellowUnit = 0;
	// Green unit
	public byte greenUnit = 0;
	// Blue unit
	public byte blueUnit = 0;
	// White unit
	public byte whiteUnit = 0;
	// Alarm unit
	public byte buzzerUnit = 0;
	};

Explanation	Received data structure of Signal Tower display acquisition command

4. Program Overview

Only the main points of the Program operation are described.

4.1. Connect to WDR

Program	Explanation
Program.cs `private static Socket sock = null;`	→ Define socket member variables

Program.cs SocketOpen()
`public static int SocketOpen(string ip, int port)`
`{`
`  try`
`  {`
`  // Set IP address and port`
`  IPAddress ipAddress = IPAddress.Parse(ip);`
`  IPEndPoint remoteEP = new IPEndPoint(ipAddress`
`  // Creating a socket`
`  sock = new Socket(ipAddress.AddressFamily, Soc`
`  if (sock == null)`
`  {`
`  Console.WriteLine("failed to create socket`
`  return -1;`
`  }`
`  // Connect to WDR`
`  sock.Connect(remoteEP);`
`  }`
`  catch (Exception ex)`
`  {`
`  Console.WriteLine("socket open error");`
`  if (sock != null)`
`  {`
`  sock.Close();`
`  }`
`  return -1;`
`  }`
`  return 0;`
`}`
	→ Specify device IP address and port number
	IP address entered from screen
	Port number entered from screen
	→ Create a socket instance
	→ Connect to the device with the socket's Connect function

4.2. Close Socket

Program	Explanation

Program.cs SocketClose()
`public static void SocketClose()`
`{`
`  if (sock != null)`
`  {`
`  //Closing the socket。`
`  sock.Shutdown(SocketShutdown.Both);`
`  sock.Close();`
`  }`
`}`
	→ Shut down the socket and then call close

4.3. Divide Received Data into Commands

Program	Explanation

Program.cs RecvDatagujde()
`public static bool RecvDatagujde(byte[] recvdata, int recvS`
`{`
`  bool ret = false;`
`  ushort resSize = 0;`
`  ushort resMode = 0;`
 	
`  startPos = 0;`
`  getSize = 0;`
 	
`  // If the reception size is 0 or less, it has not been`
`  if (recvSize <= 0)`
`  {`
`  return ret;`
`  }`
 	
`  do`
`  {`
`  // Get the size of one response`
`  resSize = (ushort)IPAddress.NetworkToHostOrder(BitC`
 	
`  // Get command`
`  resMode = (ushort)IPAddress.NetworkToHostOrder(BitC`
 	
`  // Command judgment`
`  if (resMode == mode)`
`  {`
`  // If it is the target command, set the size fo`
`  getSize = resSize + 6;`
`  ret = true;`
`  break;`
`  }`
 	
`  // Add the size of one response to startPos`
`  startPos += (resSize + 6);`
 	
`  } while (recvSize > startPos);  // When startPos become`
 	
`  return ret;`
`}`
	→ Get the data size from the received data
	→ Get command mode from received data
	→ If the command mode is the specified mode, set the data size and return “command exists”.
	
	→ Calculate the start position of data for each command
	→ Repeat until the start position is larger than the size.

4.4. Send a Request Command and Receive a Response Command

Program	Explanation

Program.cs SendCommand()
 	
`public static int SendCommand(ushort mode, byte[] sendD`
`{`
`int ret;`
`recvData = null;`
 	
`try`
`{`
`if (sock == null)`
`{`
`Console.WriteLine("socket is not");`
`return -1;`
`}`
 	
`// Set timeout`
`sock.SendTimeout = 1000;`
`sock.ReceiveTimeout = recvTimeout;`
 	
`// send`
`ret = sock.Send(sendData);`
`if (ret < 0)`
`{`
`Console.WriteLine("failed to send");`
`return -1;`
`}`
 	
`// Receive response data`
`byte[] bytes = new byte[1024];`
`int recvSize = 0;`
`int pos;`
`while (true)`
`{`
`// Data reception`
`recvSize = sock.Receive(bytes);`
 	
`if (recvSize < 0)`
`{`
`Console.WriteLine("failed to recv");`
`return -1;`
`}`
 	
`// Get the data of the target command from the re`
`if (true == RecvDatagujde(bytes, recvSize, mode,`
`{`
`break;`
`}`
`}`
`recvData = new byte[recvSize];`
`Array.Copy(bytes, pos, recvData, 0, recvSize);`
	→ Send the created send data with the Send function
	
	→ After sending, get response from the device with the Recive function
	
	→ Determine whether the received data contains the data of the specified command, and acquire the start position and size of the data.

```
// Judgment of command error response
ushort commandSize = (ushort)IPAddress.NetworkToHostOrder(BitC
bool errorFlag = false;
string errorMessage = "";
if (commandSize == 0x000C)
{
    errorFlag = true;
    switch (recvData[17])
    {
        case 0x80:
            errorMessage = "Command error";
            break;

        case 0x81:
            errorMessage = "Mode error";
            break;

        case 0x82:
            errorMessage = "Data error";
            break;

        case 0x83:
            errorMessage = "Connection unit error";
            break;

        case 0x84:
            errorMessage = "Wireless module response error";
            break;

        case 0x86:
            errorMessage = "Data acquisition error";
            break;

        case 0xC0:
            errorMessage = "Initialization abnormality";


        case 0xFF:
            errorMessage = "Exception anomaly";
            break;

        default:
            errorFlag = false;
            break;
    }
}

if (errorFlag)
{
    Console.WriteLine(errorMessage);
    return -1;
}
```

→ Determine the size because there is a possibility of command error response

→ If the response status is one of the command error responses, a message is displayed and the error ends.

→ If the response status is not a command error response, there is no problem and the process ends normally.

```
    }
    catch (SocketException e)
    {
        if (e.ErrorCode == WSAETIMEDOUT)
        {
            Console.WriteLine("TimeOut");
            return -1;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Determines the communication timeout and ends with an error.

4.5. Receive Notification Commands

Program	Explanation

Program.cs RecvCommand()
`public static int RecvCommand(ushort mode, out byte[] recvD`
`{`
`  recvData = null;`
 	
`  try`
`  {`
`  if (sock == null)`
`  {`
`  Console.WriteLine("socket is not");`
`  return -1;`
`  }`
 	
`  // Set timeout`
`  sock.ReceiveTimeout = recvTimeout;`
 	
`  // Receive notification data`
`  byte[] bytes = new byte[1024];`
`  int recvSize = 0;`
`  int pos;`
`  while (true)`
`  {`
`  // Data reception`
`  recvSize = sock.Receive(bytes);`
`  if (recvSize < 0)`
`  {`
`  Console.WriteLine("failed to recv");`
`  return -1;`
`  }`
 	
`  // Get the data of the target command from the`
`  if (true == RecvDatagujde(bytes, recvSize, mode`
`  {`
`  break;`
`  }`
`  }`
`  recvData = new byte[recvSize];`
`  Array.Copy(bytes, pos, recvData, 0, recvSize);`
	→ Get notification from device with Recive function
	
	→ Determine whether the received data contains the data of the specified command, and acquire the start position and size of the data.

```
        }
        catch (SocketException e)
        {
            if (e.ErrorCode == WSAETIMEDOUT)
            {
                Console.WriteLine("TimeOut");
                return -1;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return -1;
        }

        return 0;
    }
```

→ Determines the communication timeout and ends with an error.

4.6. Receive Transmitter Status Change Notification

Program	Explanation

Program.cs WDR_StatusChangeNoticeCommand()
 	
`public static int WDR_StatusChangeNoticeCommand(out WDR_STATUS_CHANGE_NOTICE_RE`
`{`
`int ret;`
`Data = new WDR_STATUS_CHANGE_NOTICE_RECV_DATA();`
 	
`// Command reception`
`byte[] recvData;`
`ret = RecvCommand(WDR_COMMAND_MODE_STATUS_CHANGE_NOTICE, out recvData, WDR_`
 	
`// Check for response data`
`if (recvData == null)`
`{`
`// Exception error (including timeout) occurred`
`return -1;`
`}`
 	
`// Check the response data`
`if (recvData[0] == PNS_NAK)`
`{`
`// Receive an abnormal response`
`Console.WriteLine("negative acknowledge");`
`return -1;`
`}`
 	
`// IEEE address`
`Data.IEEEAddress = (ulong)IPAddress.NetworkToHostOrder(BitConverter.ToInt64`
 	
`// serial number`
`Data.serialNumber = (uint)IPAddress.NetworkToHostOrder(BitConverter.ToInt32`
 	
`// Time information`
`Data.time = (ulong)IPAddress.NetworkToHostOrder(BitConverter.ToInt64(recvDa`
 	
`// version information`
`Data.version = new WDR_VERSION_DATA`
`{`
`major = recvData[30],  // Major version`
`minor = recvData[31],  // Minor version`
`};`
 	
`// action mode`
`Data.actionMode = recvData[32];`
 	
`// WDT information`
`Data.wdtData = new WDR_INFO_DATA`
`{`
`version = new WDR_WDT_VERSION_DATA  // version information`
`{`
`major = recvData[33],  // Major version`
`minor = recvData[34],  // Minor version`
`dummy = recvData[35]  // Fixed value`
`},`
`status = recvData[36],  // Status information`
`};`
	→ 4.5Receive Notification Commands
	
	→ Check if there is an abnormal response after reception
	
	→ Store the received data in the structure

```
    // Base unit information
    Data.baseUnitData = new WDR_BASEUNIT_DATA
    {
        format = recvData[37],                  // Unit type
        version = new WDR_WDT_VERSION_DATA      // version information
        {
            major = recvData[38],               // Major version
            minor = recvData[39],               // Minor version
            dummy = recvData[40]                // Fixed value
        },
        dipSwitch = recvData[41]                // DIP switch informati
    };

    // Signal light information (red)
    Data.redUnit = recvData[47];

    // Signal light information (yellow)
    Data.yellowUnit = recvData[48];

    // Signal light information (green)
    Data.greenUnit = recvData[49];

    // Signal light information (blue)
    Data.blueUnit = recvData[50];

    // Signal light information (white)
    Data.whiteUnit = recvData[51];

    // Buzzer information
    Data.buzzerUnit = recvData[52];

    // WDT monitoring information
    Data.surveillance = recvData[53];

    // External input information
    Data.externalInput = recvData[54];


    // RS232C data
    Data.RS232CData = new WDR_RS232C_DATA();
    Data.RS232CData.size = recvData[55];
    Data.RS232CData.serialNumber = recvData[56];
    Data.RS232CData.data = new byte[60];
    Array.Copy(recvData, 57, Data.RS232CData.data, 0, Data.RS232CData.dat

    return 0;
}
```

4.7. Receive Count Value Notification

Program	Explanation

Program.cs WDR_CountNoticeCommand()
`public static int WDR_CountNoticeCommand(out WDR_COUNT_NOTICE_RECV_DAT`
`{`
`  int ret;`
`  Data = new WDR_COUNT_NOTICE_RECV_DATA();`
 	
`  // Command reception`
`  byte[] recvData;`
`  ret = RecvCommand(WDR_COMMAND_MODE_COUNT_NOTICE, out recvData, WDF`
 	
`  // Check for response data`
`  if (recvData == null)`
`  {`
`  // Exception error (including timeout) occurred`
`  return -1;`
`  }`
 	
`  // Check for response data`
`  if (recvData == null)`
`  {`
`  // Exception error (including timeout) occurred`
`  return -1;`
`  }`
 	
`  // Check the response data`
`  if (recvData[0] == PNS_NAK)`
`  {`
`  // Receive an abnormal response`
`  Console.WriteLine("negative acknowledge");`
`  return -1;`
`  }`
 	
`  // IEEE address`
`  Data.IEEEAddress = (ulong)IPAddress.NetworkToHostOrder(BitConverte`
 	
`  // Time information`
`  Data.time = (ulong)IPAddress.NetworkToHostOrder(BitConverter.ToInt`
 	
`  // version information`
`  Data.version = new WDR_VERSION_DATA`
`  {`
`  major = recvData[30],  // Major version`
`  minor = recvData[31],  // Minor version`
`  };`
 	
`  // action mode`
`  Data.actionMode = recvData[32];`
 	
`  // WDT information`
`  Data.wdtData = new WDR_INFO_DATA`
`  {`
`  version = new WDR_WDT_VERSION_DATA  // version information`
`  {`
`  major = recvData[33],  // Major version`
`  minor = recvData[34],  // Minor version`
`  dummy = recvData[35]  // Fixed value`
`  },`
`  status = recvData[36],  // Status information`
`  };`
	→ 4.5Receive Notification Commands
	
	→ Check if there is an abnormal response after reception
	
	→ Store the received data in the structure

```
    // Base unit information
    Data.baseUnitData = new WDR_BASEUNIT_DATA
    {
        format = recvData[37],                  // Unit type
        version = new WDR_WDT_VERSION_DATA      // version informat
        {
            major = recvData[38],               // Major version
            minor = recvData[39],               // Minor version
            dummy = recvData[40]                // Fixed value
        },
        dipSwitch = recvData[41]                // DIP switch infor
    };

    // Count value
    Data.countValue = (uint)IPAddress.NetworkToHostOrder(BitConvert

    return 0;
}
```

4.8. Receive Signal Tower Display Change Notification

Program	Explanation

Program.cs WDR_SignalLightChangeNoticeCommand()
 	
`public static int WDR_SignalLightChangeNoticeCommand(out WDR_SIGNAL_LIGH`
`{`
`  int ret;`
`  Data = new WDR_SIGNAL_LIGHT_CHANGE_NOTICE_RECV_DATA();`
 	
`  // Command reception`
`  byte[] recvData;`
`  ret = RecvCommand(WDR_COMMAND_MODE_SIGNAL_LIGHT_CHANGE_NOTICE, out r`
 	
`  // Check for response data`
`  if (recvData == null)`
`  {`
`  // Exception error (including timeout) occurred`
`  return -1;`
`  }`
 	
`  // Check the response data`
`  if (recvData[0] == PNS_NAK)`
`  {`
`  // Receive an abnormal response`
`  Console.WriteLine("negative acknowledge");`
`  return -1;`
`  }`
 	
`  // IEEE address`
`  Data.IEEEAddress = (ulong)IPAddress.NetworkToHostOrder(BitConverter.`
 	
`  // Time information`
`  Data.time = (ulong)IPAddress.NetworkToHostOrder(BitConverter.ToInt64`
 	
`  // version information`
`  Data.version = new WDR_VERSION_DATA`
`  {`
`  major = recvData[30],  // Major version`
`  minor = recvData[31],  // Minor version`
`  };`
 	
`  // action mode`
`  Data.actionMode = recvData[32];`
 	
`  // WDT information`
`  Data.wdtData = new WDR_INFO_DATA`
`  {`
`  version = new WDR_WDT_VERSION_DATA  // version information`
`  {`
`  major = recvData[33],  // Major version`
`  minor = recvData[34],  // Minor version`
`  dummy = recvData[35]  // Fixed value`
`  },`
`  status = recvData[36],  // Status information`
`  };`
	→ 4.5Receive Notification Commands
	
	→ Check if there is an abnormal response after reception
	
	→ Store the received data in the Structure

```
    // Base unit information
    Data.baseUnitData = new WDR_BASEUNIT_DATA
    {
        format = recvData[37],                  // Unit type
        version = new WDR_WDT_VERSION_DATA      // version info
        {
            major = recvData[38],               // Major versio
            minor = recvData[39],               // Minor versio
            dummy = recvData[40]                // Fixed value
        },
        dipSwitch = recvData[41]                // DIP switch i
    };

    // Red unit
    Data.redUnit = recvData[47];

    // Yellow unit
    Data.yellowUnit = recvData[48];

    // Green unit
    Data.greenUnit = recvData[49];

    // Blue unit
    Data.blueUnit = recvData[50];

    // White unit
    Data.whiteUnit = recvData[51];

    // Buzzer unit
    Data.buzzerUnit = recvData[52];

    return 0;
}
```

4.9. Send a Transmitter Status Acquisition Request and Receive a Transmitter Status Acquisition Response

Program.cs WDR_TransmitterStatusRequest()

```
public static int WDR_TransmitterStatusRequest(ulong IEEEAddress,
{
    int ret;
    Data = new WDR_TRANSMITTER_STATUS_REQUEST_RES_DATA();

    try
    {
        byte[] sendData = { };

        // Product category
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_PRODU

        // identifier
        sendData = sendData.Concat(new byte[] { WDR_COMMAND }).To

        // Expansion
        sendData = sendData.Concat(new byte[] { WDR_EXPANSION }).T

        // size
        sendData = sendData.Concat(BitConverter.GetBytes((ushort)(

        // Command type
        sendData = sendData.Concat(new byte[] { WDR_COMMAND_KIND_F

        // IEEE address
        sendData = sendData.Concat(BitConverter.GetBytes((ulong)IE

        // Command mode
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_COMMA


        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_TRANSMITTER_STATUS_REQUES
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }
```

→ Create transmission data

→ 4.4Send a Request Command and Receive a Response Command

→ Check if there is an abnormal response after reception

```
        // Response status
        Data.controlState = recvData[17];

        // Time information
        Data.time = (ulong)IPAddress.NetworkToHostOrder(BitConverter.ToInt64(rec

        // version information
        Data.version = new WDR_VERSION_DATA
        {
            major = recvData[30],      // Major version
            minor = recvData[31],      // Minor version
        };

        // action mode
        Data.actionMode = recvData[32];

        // WDT information
        Data.wdtData = new WDR_INFO_DATA
        {
            version = new WDR_WDT_VERSION_DATA  // version information
            {
                major = recvData[33],   // Major version
                minor = recvData[34],   // Minor version
                dummy = recvData[35]    // Fixed value
            },
            status = recvData[36],      // Status information
        };

        // Base unit information
        Data.baseUnitData = new WDR_BASEUNIT_DATA
        {
            format = recvData[37],                  // Unit type
            version = new WDR_WDT_VERSION_DATA      // version informat
            {
                major = recvData[38],               // Major version
                minor = recvData[39],               // Minor version
                dummy = recvData[40]                // Fixed value
            },
            dipSwitch = recvData[41]                // DIP switch infor
        };

        // Signal light information (red)
        Data.redUnit = recvData[47];

        // Signal light information (yellow)
        Data.yellowUnit = recvData[48];

        // Signal light information (green)
        Data.greenUnit = recvData[49];

        // Signal light information (blue)
        Data.blueUnit = recvData[50];

        // Signal light information (white)
        Data.whiteUnit = recvData[51];
```

→ Store the received data in the structure

```
        // Buzzer information
        Data.buzzerUnit = recvData[52];

        // WDT monitoring information
        Data.surveillance = recvData[53];

        // External input information
        Data.externalInput = recvData[54];

        // RS232C data
        Data.RS232CData = new WDR_RS232C_DATA();
        Data.RS232CData.size = recvData[55];
        Data.RS232CData.serialNumber = recvData[56];
        Data.RS232CData.data = new byte[60];
        Array.Copy(recvData, 57, Data.RS232CData.data, 0, Data.RS2


    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

4.10. Send a Transmitter List Acquisition Request and Receive a Transmitter List Acquisition Response

Program.cs WDR_TransmitterListRequest()

```
public static int WDR_TransmitterListRequest(out WDR_TRANSMITTER_LIST
{
    int ret;
    Data = new WDR_TRANSMITTER_LIST_REQUEST_RES_DATA();

    try
    {
        byte[] sendData = { };

        // Product category
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_PRODUCT_

        // identifier
        sendData = sendData.Concat(new byte[] { WDR_COMMAND }).ToArra

        // Expansion
        sendData = sendData.Concat(new byte[] { WDR_EXPANSION }).ToAr

        // size
        sendData = sendData.Concat(BitConverter.GetBytes((ushort)0x0C

        // Command type
        sendData = sendData.Concat(new byte[] { WDR_COMMAND_KIND_REQU

        // IEEE address
        byte[] IEEEEdata = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
        sendData = sendData.Concat(IEEEEdata).ToArray();

        // Command mode
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_COMMAND_


        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_TRANSMITTER_LIST_RE
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }
```

→ Create transmission data

→ 4.4Send a Request Command and Receive a Response Command

```
        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

        // Number of acquisitions
        Data.unitCount = recvData[18];

        // WDT status information
        for (int count = 0; count < Data.unitCount; count++)
        {
            // Object creation
            WDR_WDT_STATUS_DATA setData = new WDR_WDT_STATUS_DATA()

            // IEEE address
            setData.IEEEAddress = (ulong)IPAddress.NetworkToHostOrd

            // Registration status
            setData.registration = recvData[27 + (count * 10)];

            // Connection Status
            setData.connect = recvData[28 + (count * 10)];


            // Set in an array
            Data.wdtStatus[count] = setData;
        }

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.11.Send a Transmitter Information Acquisition Request and Receive a Transmitter Information Acquisition Response

Program.cs WDR_TransmitterDataRequest()

```
public static int WDR_TransmitterDataRequest(ulong IEEEAddress, out WDR_TRAI
{
    int ret;
    Data = new WDR_TRANSMITTER_DATA_REQUEST_RES_DATA();
    addData = null;

    try
    {
        byte[] sendData = [ ];

        // Product category
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_PRODUCT_ID).Rev

        // identifier
        sendData = sendData.Concat(new byte[] { WDR_COMMAND }).ToArray();

        // Expansion
        sendData = sendData.Concat(new byte[] { WDR_EXPANSION }).ToArray();

        // size
        sendData = sendData.Concat(BitConverter.GetBytes((ushort)0x000B).Rev

        // Command type
        sendData = sendData.Concat(new byte[] { WDR_COMMAND_KIND_REQUEST })

        // IEEE address
        sendData = sendData.Concat(BitConverter.GetBytes((ulong)IEEEAddress

        // Command mode
        sendData = sendData.Concat(BitConverter.GetBytes(WDR_COMMAND_MODE_TI

        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_TRANSMITTER_DATA_REQUEST,
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }
```

→ Create transmission data

→ 4.4Send a Request Command and Receive a Response Command

→ Check if there is an abnormal response after reception

```
        // Response status
        Data.controlState = recvData[17];

        // User name
        Data.userName = new byte[121];
        Array.Copy(recvData, 18, Data.userName, 0, Data.userName.Length);

        // version information
        Data.version = new WDR_VERSION_DATA
        {
            major = recvData[139],      // Major version
            minor = recvData[140],      // Minor version
        };

        // action mode
        Data.actionMode = recvData[141];

        // WDT information
        Data.wdtData = new WDR_INFO_DATA
        {
            version = new WDR_WDT_VERSION_DATA  // version information
            {
                major = recvData[142],   // Major version
                minor = recvData[143],   // Minor version
                dummy = recvData[144]    // Fixed value
            },
            status = recvData[145],      // Status information
        };

        // Base unit information
        Data.baseUnitData = new WDR_BASEUNIT_DATA
        {
            format = recvData[146],                 // Unit type
            version = new WDR_WDT_VERSION_DATA      // version informati
            {
                major = recvData[147],              // Major version
                minor = recvData[148],              // Minor version
                dummy = recvData[149]               // Fixed value
            },
            dipSwitch = recvData[150]               // DIP switch inform
        };

        // ExtendedPanID
        Data.extendedPanID = (ulong)IPAddress.NetworkToHostOrder(BitConv

        // Frequency channel
        Data.frequencyChannel = (uint)IPAddress.NetworkToHostOrder(BitCo

        // Signal light input judgment
        Data.signalLightInputJudge = recvData[168];

        // Power settings
        Data.powerSetting = recvData[169];

        // Counter setting
        Data.counterSetting = recvData[170];

        // Send mode
        Data.sendMode = (ushort)IPAddress.NetworkToHostOrder(BitConverte
```

→ Store the received data in the structure

```
        // In the case of WDT-6LR-Z2-PRO, the expansion structure is als
        if (Data.actionMode == 0xFF)
        {
            addData = new WDR_TRANSMITTER_DATA_REQUEST_RES_ADD_DATA();

            // Input information transmission method
            addData.inputDataTranform = recvData[173];

            // Signal light format
            addData.signalLightFormat = recvData[174];

            // Periodic transmission
            addData.regularSend = recvData[175];

            // Simultaneous input judgment sensitivity setting
            addData.concInputSensitiveSetting = recvData[176];

            // Received data file format
            addData.recvDataFileFormat = recvData[177];

            // Communication setting baud rate
            addData.baudrate = recvData[178];

            // Communication setting data length
            addData.dataLength = recvData[179];

            // Communication setting parity
            addData.parity = recvData[180];

            // Communication setting stop bit
            addData.stopBit = recvData[181];
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

4.12. Send a Transmitter Call Display Request and Receive a Transmitter Call

Display Response

Program	Explanation

Program.cs WDR_TransmitterCallRequest ()
 	
`public static int WDR_TransmitterCallRequest(ulong IEEEAddress, out WDR`
`{`
`  int ret;`
`  Data = new WDR_TRANSMITTER_CALL_REQUEST_RES_DATA();`
 	
`  try`
`  {`
`  byte[] sendData = { };`
 	
`  // Product category`
`  sendData = sendData.Concat(BitConverter.GetBytes(WDR_PRODUCT_ID`
 	
`  // identifier`
`  sendData = sendData.Concat(new byte[] { WDR_COMMAND }).ToArray(`
 	
`  // Expansion`
`  sendData = sendData.Concat(new byte[] { WDR_EXPANSION }).ToArra`
 	
`  // size`
`  sendData = sendData.Concat(BitConverter.GetBytes((ushort)0x000B`
 	
`  // Command type`
`  sendData = sendData.Concat(new byte[] { WDR_COMMAND_KIND_REQUES`
 	
`  // IEEE address`
`  sendData = sendData.Concat(BitConverter.GetBytes((ulong)IEEEAdd`
 	
`  // Command mode`
`  sendData = sendData.Concat(BitConverter.GetBytes(WDR_COMMAND_MO`
	→ Create transmission data

```
        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_TRANSMITTER_CALL_RE
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ 4.4Send a Request Command and Receive a Response Command

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.13. Send a Serial Data Output Request and Receive a Serial Data Output

Response

Program	Explanation

Program.cs WDR_SerialOutputRequest()
`public static int WDR_SerialOutputRequest(WDR_SERIAL_OUTPUT_RE`
`{`
`int ret;`
`Data = new WDR_SERIAL_OUTPUT_RES_DATA();`
`try`
`{`
`byte[] sendData = { };`
`// Product category`
`sendData = sendData.Concat(BitConverter.GetBytes(WDR_PI`
`// identifier`
`sendData = sendData.Concat(new byte[] { WDR_COMMAND })`
`// Expansion`
`sendData = sendData.Concat(new byte[] { WDR_EXPANSION`
`// size`
`ushort sendSize = (ushort)(14 + outputData.outputData.`
`sendData = sendData.Concat(BitConverter.GetBytes(sendS`
`// Command type`
`sendData = sendData.Concat(new byte[] { WDR_COMMAND_KI`
`// IEEE address`
`sendData = sendData.Concat(BitConverter.GetBytes((ulon,`
`// Command mode`
`sendData = sendData.Concat(BitConverter.GetBytes(WDR_C`
`// Dummy data`
`sendData = sendData.Concat(BitConverter.GetBytes((usho`
`// serial number`
`sendData = sendData.Concat(new byte[] { outputData.ser`
`// Output information`
`sendData = sendData.Concat(outputData.outputData.ToArray()).ToArray`
`// Send request command`
`byte[] recvData;`
`ret = SendCommand(WDR_COMMAND_MODE_SERIAL_OUTPUT_REQUEST, sendData,`
`if (ret != 0)`
`{`
`Console.WriteLine("failed to send data");`
`return -1;`
`}`
	→ Create transmission data
	
	→ 4.4Send a Request Command and Receive a Response Command

```
        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.14. Send a Signal Tower Display Control Request and Receive a Signal Tower Display Control Response

Program	Explanation

Program.cs WDR_SignalLightControlRequest()
`public static int WDR_SignalLightControlRequest(WDR_SIGNAL_LIGHT`
`{`
`int ret;`
`Data = new WDR_SIGNAL_LIGHT_CONTROL_RES_DATA();`
`try`
`{`
`byte[] sendData = { };`
`// Product category`
`sendData = sendData.Concat(BitConverter.GetBytes(WDR_PRC`
`// identifier`
`sendData = sendData.Concat(new byte[] { WDR_COMMAND }).T`
`// Expansion`
`sendData = sendData.Concat(new byte[] { WDR_EXPANSION })`
`// size`
`sendData = sendData.Concat(BitConverter.GetBytes((ushort`
`// Command type`
`sendData = sendData.Concat(new byte[] { WDR_COMMAND_KIND`
`// IEEE address`
`sendData = sendData.Concat(BitConverter.GetBytes((ulong)`
`// Command mode`
`sendData = sendData.Concat(BitConverter.GetBytes(WDR_COM`
	→ Create transmission data

```
        // Data size, data area
        byte[] data = {
            controlData.controlTime,    // Control time
            controlData.redUnit,        // Red unit lighting pa
            controlData.yellowUnit,     // Yellow unit lighting
            controlData.greenUnit,      // Green unit lighting
            controlData.blueUnit,       // Blue unit lighting p
            controlData.whiteUnit,      // White unit lighting
            controlData.buzzerUnit      // Buzzer unit pattern
        };
        sendData = sendData.Concat(data).ToArray();

        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_SIGNAL_LIGHT_CONTROL,
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }


        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.recvState = recvData[17];

        // Control state
        Data.controlState = recvData[18];

        // Red unit
        Data.redUnit = recvData[19];

        // Yellow unit
        Data.yellowUnit = recvData[20];

        // Green unit
        Data.greenUnit = recvData[21];
```

→ 4.4Send a Request Command and Receive a Response Command

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

```
            // Blue unit
            Data.blueUnit = recvData[22];

            // White unit
            Data.whiteUnit = recvData[23];

            // Buzzer unit
            Data.buzzerUnit = recvData[24];

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return -1;
        }

        return 0;
    }
```

4.15. Send a Signal Tower Display Cancellation Request and Receive a Signal

Tower Display Cancellation Response

Program	Explanation

Program.cs WDR_SignalLightLiftRequest()
`public static int WDR_SignalLightLiftRequest(ulong IEEEAd`
`{`
`int ret;`
`Data = new WDR_SIGNAL_LIGHT_LIFT_RES_DATA();`
`try`
`{`
`byte[] sendData = { };`
`// Product category`
`sendData = sendData.Concat(BitConverter.GetBytes(`
`// identifier`
`sendData = sendData.Concat(new byte[] { WDR_COMMA`
`// Expansion`
`sendData = sendData.Concat(new byte[] { WDR_EXPAN`
`// size`
`sendData = sendData.Concat(BitConverter.GetBytes(`
`// Command type`
`sendData = sendData.Concat(new byte[] { WDR_COMMA`
`// IEEE address`
`sendData = sendData.Concat(BitConverter.GetBytes(`
`// Command mode`
`sendData = sendData.Concat(BitConverter.GetBytes(`
`// Send request command`
`byte[] recvData;`
`ret = SendCommand(WDR_COMMAND_MODE_SIGNAL_LIGHT_C`
`if (ret != 0)`
`{`
`Console.WriteLine("failed to send data");`
`return -1;`
`}`
	→ Create transmission data
	
	→ 4.4Send a Request Command and Receive a Response Command

```
        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.recvState = recvData[17];

        // Control state
        Data.controlState = recvData[18];

        // Red unit
        Data.redUnit = recvData[19];

        // Yellow unit
        Data.yellowUnit = recvData[20];

        // Green unit
        Data.greenUnit = recvData[21];

        // Blue unit
        Data.blueUnit = recvData[22];

        // White unit
        Data.whiteUnit = recvData[23];

        // Buzzer unit
        Data.buzzerUnit = recvData[24];

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.16. Send a Count Value Registration Request and Receive a Count Value

Registration Response

Program	Explanation

Program.cs WDR_SignalLightCountSetRequest()
public static int WDR_SignalLightCountSetRequest(WDR_SIGNA
{
int ret;
Data = new WDR_SIGNAL_LIGHT_COUNT_SET_RES_DATA();
 	
try
{
byte[] sendData = { };
 	
// Product category
sendData = sendData.Concat(BitConverter.GetBytes(W
 	
// identifier
sendData = sendData.Concat(new byte[] { WDR_COMMAN
 	
// Expansion
sendData = sendData.Concat(new byte[] { WDR_EXPANS
 	
// size
sendData = sendData.Concat(BitConverter.GetBytes((
 	
// Command type
sendData = sendData.Concat(new byte[] { WDR_COMMAN
 	
// IEEE address
sendData = sendData.Concat(BitConverter.GetBytes((
 	
// Command mode
sendData = sendData.Concat(BitConverter.GetBytes(W
 	
// Count registration value
sendData = sendData.Concat(BitConverter.GetBytes(s
	→ Create transmission data

```
        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_SIGNAL_LIGHT_COUNT_SET_RE
        if (ret != 0)
        {
            Console.WriteLine("failed to send data");
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) occurred
            return -1;
        }

        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ 4.4Send a Request Command and Receive a Response Command

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.17. Send a Receiver Information Acquisition Request and Receive a Receiver Information Acquisition Response

Program	Explanation

Program.cs WDR_ReceiveDataRequest()
 	
`public static int WDR_ReceiveDataRequest(out WDR_RE(`
`{`
`  int ret;`
`  Data = new WDR_RECEIVER_DATA_REQUEST_RES_DATA()`
 	
`  try`
`  {`
`  byte[] sendData = { };`
 	
`  // Product category`
`  sendData = sendData.Concat(BitConverter.GetE`
 	
`  // identifier`
`  sendData = sendData.Concat(new byte[] { WDR_`
 	
`  // Expansion`
`  sendData = sendData.Concat(new byte[] { WDR_`
 	
`  // size`
`  sendData = sendData.Concat(BitConverter.GetE`
 	
`  // Command type`
`  sendData = sendData.Concat(new byte[] { WDR_`
 	
`  // IEEE address`
`  byte[] IEEEEdata = { 0x00, 0x00, 0x00, 0x00`
`  sendData = sendData.Concat(IEEEEdata).ToArra`
 	
`  // Command mode`
`  sendData = sendData.Concat(BitConverter.GetE`
	→ Create transmission data

```
        // Send request command
        byte[] recvData;
        ret = SendCommand(WDR_COMMAND_MODE_RECEIVER,
        if (ret != 0)
        {
            Console.WriteLine("failed to send data"
            return -1;
        }

        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeout) o
            return -1;
        }
```

→ 4.4Send a Request Command and Receive a Response Command

```
        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowlec
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

        // ExtendedPanID
        Data.extendedPanID = (ulong)IPAddress.Net

        // Frequency channel
        Data.frequencyChannel = (uint)IPAddress.N

        // Firmware version
        Data.version = new WDR_VERSION_DATA
        {
            major = recvData[30],       // Major v
            minor = recvData[31],       // Minor v
        };

        // Network status
        Data.networkStatus = recvData[32];

        // How to boot the network
        Data.networkBoot = recvData[33];

        // Running ExtendedPanID
        Data.actionExtendedPanID = (ulong)IPAddre

        // Operating frequency channel
        Data.actionFrequencyChannel = recvData[42

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.18. Send a Receiver Reset Request and Receive a Receiver Reset Response

Program	Explanation

Program.cs WDR_ReceiverResetRequest()
`public static int WDR_ReceiverResetRequest(out WDR_RE`
`{`
`  int ret;`
`  Data = new WDR_RECEIVER_RESET_RES_DATA();`
 	
`  try`
`  {`
`  byte[] sendData = { };`
 	
`  // Product category`
`  sendData = sendData.Concat(BitConverter.GetBy`
 	
`  // identifier`
`  sendData = sendData.Concat(new byte[] { WDR_C`
 	
`  // Expansion`
`  sendData = sendData.Concat(new byte[] { WDR_E`
 	
`  // size`
`  sendData = sendData.Concat(BitConverter.GetBy`
 	
`  // Command type`
`  sendData = sendData.Concat(new byte[] { WDR_C`
 	
`  // IEEE address`
`  byte[] IEEEEdata = { 0x00, 0x00, 0x00, 0x00,`
`  sendData = sendData.Concat(IEEEEdata).ToArray`
 	
`  // Command mode`
`  sendData = sendData.Concat(BitConverter.GetBy`
 	
`  // Send request command`
`  byte[] recvData;`
`  ret = SendCommand(WDR_COMMAND_MODE_RECEIVER_F`
`  if (ret != 0)`
`  {`
`  Console.WriteLine("failed to send data");`
`  return -1;`
`  }`
 	
`  // Check for response data`
`  if (recvData == null)`
`  {`
`  // Exception error (including timeout) oc`
`  return -1;`
`  }`
	→ Create transmission data
	
	→ 4.4Send a Request Command and Receive a Response Command

```
        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowledge");
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

4.19. Send a Count Value Acquisition Request and Receive a Count Value

Acquisition Response

Program	Explanation

Program.cs WDR_SignalLightCountGetRequest()
 	
`public static int WDR_SignalLightCountGetRequest(ulong Il`
`{`
`  int ret;`
`  Data = new WDR_SIGNAL_LIGHT_COUNT_GET_RES_DATA();`
 	
`  try`
`  {`
`  byte[] sendData = { };`
 	
`  // Product category`
`  sendData = sendData.Concat(BitConverter.GetBytes`
 	
`  // identifier`
`  sendData = sendData.Concat(new byte[] { WDR_COMM.`
 	
`  // Expansion`
`  sendData = sendData.Concat(new byte[] { WDR_EXPAI`
 	
`  // size`
`  sendData = sendData.Concat(BitConverter.GetBytes`
 	
`  // Command type`
`  sendData = sendData.Concat(new byte[] { WDR_COMM.`
 	
`  // IEEE address`
`  sendData = sendData.Concat(BitConverter.GetBytes`
 	
`  // Command mode`
`  sendData = sendData.Concat(BitConverter.GetBytes`
 	
`  // Send request command`
`  byte[] recvData;`
`  ret = SendCommand(WDR_COMMAND_MODE_SIGNAL_LIGHT_I`
`  if (ret != 0)`
`  {`
`  Console.WriteLine("failed to send data");`
`  return -1;`
`  }`
	→ Create transmission data
	
	→ 4.4Send a Request Command and Receive a Response Command

```
        // Check for response data
        if (recvData == null)
        {
            // Exception error (including timeo
            return -1;
        }
```

```
        // Check the response data
        if (recvData[0] == PNS_NAK)
        {
            // Receive an abnormal response
            Console.WriteLine("negative acknowled
            return -1;
        }

        // Response status
        Data.controlState = recvData[17];

        // Time information
        Data.time = (ulong)IPAddress.NetworkToHos

        // version information
        Data.version = new WDR_VERSION_DATA
        {
            major = recvData[30],       // Major v
            minor = recvData[31],       // Minor v
        };

        // action mode
        Data.actionMode = recvData[32];

        // WDT information
        Data.wdtData = new WDR_INFO_DATA
        {
            version = new WDR_WDT_VERSION_DATA  /
            {
                major = recvData[33],   // Major
                minor = recvData[34],   // Minor
                dummy = recvData[35]    // Fixed
            },
            status = recvData[36],      // Status
        };
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

```
        // Base unit information
        Data.baseUnitData = new WDR_BASEUNIT_DATA
        {
            format = recvData[37],
            version = new WDR_WDT_VERSION_DATA
            {
                major = recvData[38],
                minor = recvData[39],
                dummy = recvData[40]
            },
            dipSwitch = recvData[41]
        };

        // Count value
        Data.count = (uint)IPAddress.NetworkToHost

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return -1;
    }

    return 0;
}
```

4.20. Send a Signal Tower Display Acquisition Request and Receive a Signal Tower Display Acquisition Response

Program	Explanation

Program.cs WDR_SignalLightDataGetRequest()
`public static int WDR_SignalLightDataGetRequest(ulong`
`{`
`int ret;`
`Data = new WDR_SIGNAL_LIGHT_DATA_GET_RES_DATA();`
`try`
`{`
`byte[] sendData = { };`
`// Product category`
`sendData = sendData.Concat(BitConverter.GetByt`
`// identifier`
`sendData = sendData.Concat(new byte[] { WDR_CO`
`// Expansion`
`sendData = sendData.Concat(new byte[] { WDR_EX`
`// size`
`sendData = sendData.Concat(BitConverter.GetByt`
`// Command type`
`sendData = sendData.Concat(new byte[] { WDR_CO`
`// IEEE address`
`sendData = sendData.Concat(BitConverter.GetByt`
`// Command mode`
`sendData = sendData.Concat(BitConverter.GetByt`
`// Send request command`
`byte[] recvData;`
`ret = SendCommand(WDR_COMMAND_MODE_SIGNAL_LIGH`
`if (ret != 0)`
`{`
`Console.WriteLine("failed to send data");`
`return -1;`
`}`
	→ Create transmission data
	
	→ 4.4Send a Request Command and Receive a Response Command

```
    // Check for response data
    if (recvData == null)
    {
        // Exception error (including timeout) occu
        return -1;
    }



    // Check the response data
    if (recvData[0] == PNS_NAK)
    {
        // Receive an abnormal response
        Console.WriteLine("negative acknowledge");
        return -1;
    }

    // Response status
    Data.controlState = recvData[17];

    // Time information
    Data.time = (ulong)IPAddress.NetworkToHostOrder

    // version information
    Data.version = new WDR_VERSION_DATA
    {
        major = recvData[30],       // Major version
        minor = recvData[31],       // Minor version
    };

    // action mode
    Data.actionMode = recvData[32];
```

→ Check if there is an abnormal response after reception

→ Store the received data in the structure

```
        // WDT information
        Data.wdtData = new WDR_INFO_DATA
        {
            version = new WDR_WDT_VERSION_DATA  /,
            {
                major = recvData[33],    // Major \
                minor = recvData[34],    // Minor \
                dummy = recvData[35]     // Fixed \
            },
            status = recvData[36],       // Status
        };

        // Base unit information
        Data.baseUnitData = new WDR_BASEUNIT_DATA
        {
            format = recvData[37],
            version = new WDR_WDT_VERSION_DATA
            {
                major = recvData[38],
                minor = recvData[39],
                dummy = recvData[40]
            },
            dipSwitch = recvData[41]
        };
```

```
            // Red unit
            Data.redUnit = recvData[47];

            // Yellow unit
            Data.yellowUnit = recvData[48];

            // Green unit
            Data.greenUnit = recvData[49];

            // Blue unit
            Data.blueUnit = recvData[50];

            // White unit
            Data.whiteUnit = recvData[51];

            // Buzzer unit
            Data.buzzerUnit = recvData[52];

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return -1;
        }

        return 0;
    }
```