README.md

Fohhn-Net Simpl+ Modules and Simpl Sharp Pro library

Developed by Niklas Westman - JaDeVa AB

Tested on CP3 (1.8001.0146) and CP4 (2.7000.00083)

Summary

This repository contains

- `FohhnNet_Device_x.x.x` - Module to control any Fohhn device that supports Fohhn-Net
- SimplSharp (C#) solution containing projects for both SIMPL+ modules and Simpl Sharp Pro
  - `Fohhn.FohhnNet`
    - This project contains the library for use in S#Pro
  - `Fohhn.FohhnNet.Demo`
    - This is a S#Pro demo program. Load this program to your processor and use the Xpanel `Demo_Xpanel.vtp` to test the library.

    - `FohhnNet_Device_CSharp`
        - This project contains wrapper classes that are used in the Simpl+ Modules
- SIMPL Windows Demo program
- VTPro-E project to be used with both Simpl Sharp Pro and SIMPL Windows demo programs
- Documentation for the SIMPL module, as well as this README as a PDF

This README is mainly focused on how to use the classes in `Fohhn.FohhnNet`

Notes

It is recommended to have only one Fohhn-device per connection. It will work with slaved devices, but you might notice slower response times.

Quickstart

```
// Instantiate a configuration object and supply device id, and number of inputs and outputs
// on the device
var config = new FohhnNetDeviceConfig(1, 4, 4);

// Instantiate a device either over UDP or RS485
// UDP
var device = new FohhnNetDevice(config, "192.168.1.50", null);

// RS485 - pass your comports send-method, and when data comes from the comport, call the
// handle serial response method
var device = new FohhnNetDevice(config, yourComPort.Send, null);
HandleSerialResponse(dataFromYourComPort);

// Subscribe to events
device.Events += DeviceOnEvents;
foreach (var output in device.Outputs)
{
    output.Events += OutputOnEvents;

    foreach (var inputRoute in output.InputRoutes)
        inputRoute.Events += RouteOnEvents;
}

// Start communications
device.StartCommunication();

// Make sure to dispose the device on program stop
device.Dispose();


// Event handler for device events
private void DeviceOnEvents(object sender, FohhnNetDeviceEventArgs args)
{
    switch (args.EventType)
    {
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.Responding:
            bool responding = args.BoolValue;
            break;
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.FirmwareVersion:
            string firmwareVersion = args.StringValue;
            break;
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.DeviceAlias:
            string deviceAlias = args.StringValue;
            break;
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.Power:
            bool powerIsOn = args.BoolValue;
```

```
            break;
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.StatusBits:
            bool[] statusBits = args.BoolArrayValue;
            break;
        case FohhnNetDeviceEventArgs.FohhnNetDeviceEventTypes.Temperature:
            double temperature = args.DoubleValue;
            break;
    }
}

// Event handler for output events
private void OutputOnEvents(object sender, FohhnNetDeviceOutputEventArgs args)
{
    switch (args.EventType)
    {
        case FohhnNetDeviceOutputEventArgs.FohhnNetDeviceOutputEventTypes.Volume:
            double volumeInDecibels = args.DoubleValue;
            break;
        case FohhnNetDeviceOutputEventArgs.FohhnNetDeviceOutputEventTypes.Muted:
            bool muted = args.BoolValue;
            break;
    }
}

// Event handler for route events
private void RouteOnEvents(object sender, FohhnNetDeviceRoutePointEventArgs args)
{
    switch (args.EventType)
    {
        case FohhnNetDeviceRoutePointEventArgs.FohhnNetDeviceRoutePointEventTypes.Gain:
            double gainInDecibels = args.DoubleValue;
            break;
        case FohhnNetDeviceRoutePointEventArgs.FohhnNetDeviceRoutePointEventTypes.Muted:
            bool muted = args.BoolValue;
            break;
    }
}
```

FohhnNetDevice

This class enables control for any Fohhn device that supports Fohhn-Net. It is possible to use either UDP or RS485

Constructors

All constructors need a `FohhnNetDeviceConfig` object passed in. You have to create this first, and supply device Id, number of inputs and number of outputs of the device you want to control.

In all constructors it is possible to provide your own method for logging. Set to null to use default. The default logger writes to errorlog if this is a server-based processor. Otherwise prints to console, and errorlog if loglevel is an error

Constructor	Description
`FohhnNetDevice(config, logger)`	For UDP connection.
`FohhnNetDevice(config, host, logger)`	For UDP connection.
`FohhnNetDevice(config, host, port, logger)`	For UDP connection.
`FohhnNetDevice(config, sendDelegate logger)`	For RS485 connection.

Properties

Property	Description
`DeviceAlias`	Returns the alias of the device. If no alias is set, this will be an empty string
`DeviceId`	This is the Id of the device
`FirmwareVersion`	Returns the firmware version of the device. Example "3.2.2"
`Host`	Get or set the Ip/FQDN/Hostname of the device to connect to over UDP
`LogLevel`	Get or set the current log level. Change this to output the desired level of logging.
`Outputs[]`	Contains all outputs on the device This is where you control things such as volume and mute
`PollRateMs`	Get or set at which rate the device should be polled (in milliseconds). It's recommended to not go lower than 5000. Default is 30000
`Port`	Get or set the remote Port to use for UDP
`PowerIsOn`	Returns false if the device is in standby
`Responding`	Returns true if the device is responding properly to commands.
`StatusBits[]`	This contains some status about things. This is different for every Fohhn device type. It contains 4 bits. See Appendix A for information
`Temperature`	Returns the temperature in Celsius of the device

Methods

Method	Description
`Dispose()`	Disposes the underlying connection and any unmanaged resources. Must be done on program stop
`HandleSerialResponse(string)`	Call this method when you get serial data from the device
`PollRouting()`	This will poll all outputs for all routing information. This is done automatically only on connection. It requires a lot of commands to be sent so you have to manually do this if you need it. There are methods for this on each output and routing point.
`RecallPreset(int)`	Recalls a preset in the device (1-100)
`SendCustomCommand(byte[], Action<byte[]>)`	Method to send custom commands. Used if you want additional functionality that this library does not provide. The action is called when/if we get a response from the device
`SetPower(bool)`	Sets the standby state of the device
`StartCommunication()`	Starts communicating with the device. Use this if RS485 is used, or if you have supplied Host address in constructor or directly on the Host property.
`StartCommunication(string)`	Starts communicating with the device using UDP on remote port 2101
`StartCommunication(string, int)`	Starts communicating with the device using UDP
`StopCommunication()`	Stops communicating with the device

Events

Event	Description
`Events`	This will be trigged when a property changes

FohhnNetDeviceEventArgs

Contains information on what property has changed on FohhnNetDevice

Properties

Property	Description
`BoolArrayValue[]`	Contains the new value for EventTypes: `StatusBits`
`BoolValue`	Contains the new value for EventTypes: `Responding` , `Power`
`DoubleValue`	Contains the new value for EventTypes: `Temperature`
`EventType`	Contains which property has changed
`StringValue`	Contains the new value for EventTypes: `FirmwareVersion` , `DeviceAlias`

FohhnNetDeviceOutput

This class represents an output on the device

Properties

Property	Description
`InputRoutes[]`	Contains every input that you can route to this output. This is where you perform routing operations
`IsMuted`	Returns true if the output is muted
`Number`	The 1-based number of this output
`VolumeDb`	Returns the current volume of the output in dB (-80.0 to 12.0)

Methods

Method	Description
`PollRouting()`	This will poll this output for all routing information. This is done automatically only on connection. It requires a lot of commands to be sent so you have to manually do this if you need it. There are methods for this on each routing point, or on the main device as well.
`SetMute(bool)`	Sets the mute state for this output
`SetVolumeAndMute(double, bool)`	Sets volume and mute for this output. Volume is in decibels(-80.0 to 12.0)

Events

Event	Description
`Events`	This will be trigged when a property changes

FohhnNetDeviceOutputEventArgs

Contains information on what property has changed on FohhnNetDeviceOutput

Properties

Property	Description
`BoolValue`	Contains the new value for EventTypes: `Muted`
`DoubleValue`	Contains the new value for EventTypes: `Volume`
`EventType`	Contains which property has changed

FohhnNetDeviceRoutePoint

This class represents an output routing point on the device

Properties

Property	Description
`GainDb`	Returns the current gain of the routing point in dB (-80.0 to 12.0)
`Input`	The input for this routing point
`IsMuted`	Returns true if the routing point is muted
`Output`	The output for this routing point

Methods

Method	Description
`PollRoute()`	This will poll this routing point for all gain and mute. This is done automatically only on connection. It requires a lot of commands to be sent so you have to manually do this if you need it. There are methods for this on each output, or on the main device as well.
`SetGainAndMute(double, bool)`	Sets gain and mute for this routing point. Gain is in decibels(-80.0 to 12.0)

Events

Event	Description
`Events`	This will be trigged when a property changes

FohhnNetDeviceRoutePointEventArgs

Contains information on what property has changed on FohhnNetDeviceRoutePoint

Properties

Property	Description
`BoolValue`	Contains the new value for EventTypes: `Muted`
`DoubleValue`	Contains the new value for EventTypes: `Gain`
`EventType`	Contains which property has changed

Appendix A

Explanation of what each status bit means, depending on the device

Fohhn Devices	Bit[0]	Bit[1]	Bit[2]	Bit[3]
DLI-130/230/330/430	Fault	Audio (AES)	Pilotton	
FV-100/200	Fault	Audio (AES)		
LFI-120/220/350/450	Fault	Piltoton		
FMI-100/110/400	Fault	Piltoton		
DI-2.2000/4000	Protect 1	Protect 2		
DI-4.1000/2000	Protect 1	Protect 2	Protect 3	Protect 4
DFM-100/110/400	Fault	Audio (AES)	Pilotton	
MA-4.100/600	Protect 1	Protect 2	Protect 3	Protect 4

Release notes

1.0.0

- Initial version