<< egTree IO API >>

1 Introduction of egTreeIO API

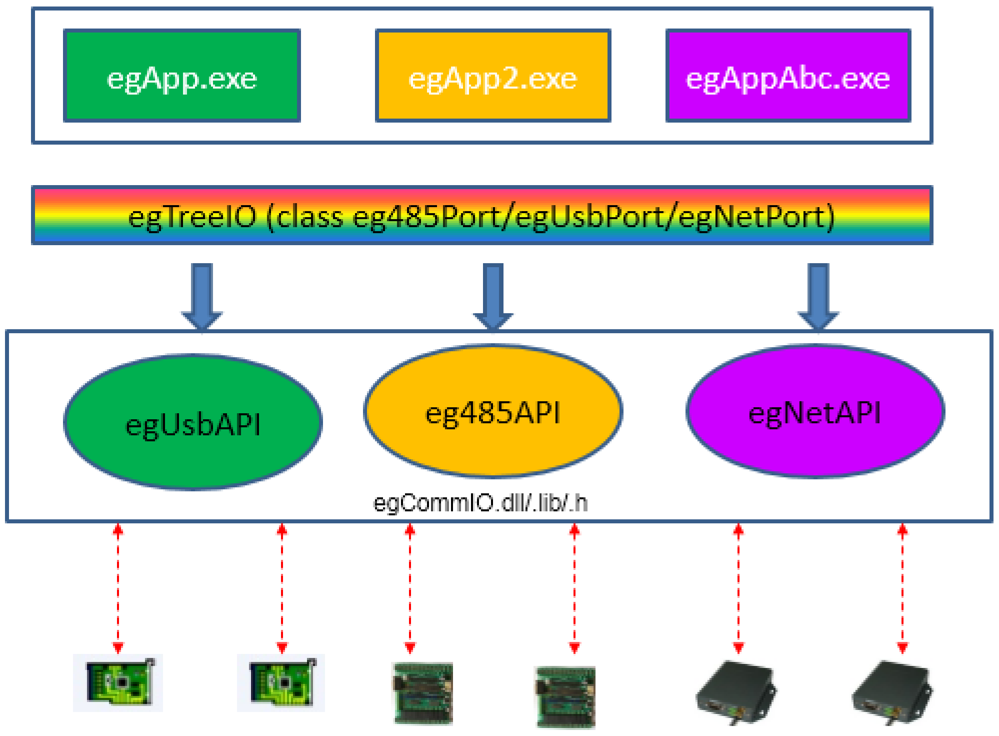
egTree devices facilitate a rich set of communication interfaces. Such interfaces include Ethernet, USB, and RS485, via which devices and hosts can talk each other. In a fully built network a host PC can connect to multiple egTree devices through multiple communication ports. User application running on host PC can acquire sensing data and output control signal to its operating environment. Raynix Ltd provides a full spectrum of device API to support host PC system in application development and as well in running environment.

egTree IO API is a software library package for host PC to use and access egTree series devices. Inside the package are the run-time dll files for user applications and also a set of IO access interfaces for application development in Windows visual studio.

2 API Architecture

Built upon ubiquitous host communication ports (USB, RS232, Net) egTreeIO API facilitate a simple universal API to support bidirectional data transfer between host PC and egTree series devices.

We construct 3 levels of software concept to carry out data transfer from application software to communication ports. API is the middle layer to translate from simple user request to more complicated lower layer protocols to accomplish application level request. The diagram below describes API architecture.



The API functions mainly perform the register content read and write from device IO space. There are also functions to perform IO pins read and write directly from host PC to achieve lowest hardware control in user application environment. egTree series products include 3 categories, they are

egTree-PIO32AD -- USB-based data acquisition and actuation control board, to access IO pins and ADC results directly.

egPLC -- USB, RS485-based program logic control and remote terminal unit for building autonomous control system and network system with host PC. It has built in very strong board-level customization. egPLC comprises egPLC4888 and egPLC5888.

egDAA -- TCP/IP-based remote data acquisition and actuation control for building autonomous control system and network with cloud computing. It is extra powerful with 2-level functional customization.

The following table summarizes the product variants and their API classes. All APIs are associated with communication ports.

Product and Communication Class	USB Class	485 Class	Net Class
egTreePIO32AD	yes		
egPLC	yes	yes	
egDAA		yes	yes
Common API Prototypes	egConnect egInput egOutput egSampleAll	eg485Open eg485Input eg485Output eg485SampleAll	egNetInput egNetOutput egNetSampleAll
C/C++ library files	egCommIO.dll egCommIO.lib egCommIO.h	egCommIO.dll egCommIO.lib egCommIO.h	egCommIO.dll egCommIO.lib egCommIO.h

DotNet C#/VB file and Class	egTreeIO.dll
	class eg485Port
	class egUSBPort
	class egNetPort
		egTreeIO.dll
		class eg485Port
		class egUSBPort
		class egNetPort
			egTreeIO.dll
			class eg485Port
			class egUSBPort
			class egNetPort

3 C/C++ Interface

There are 3 categories of C/C++ interface. USB class, RS485 class, and TCP/IP (Net) class

USB class

```
data structure for SampleAll
typedef struct _jf_IEEadxy {
  unsigned short ok;
  unsigned short an[8]; //eight-channel analog data
  unsigned short xx;
  unsigned short yy;
} IEE_ADXY;
```

int egConnect(void)

Connect to egTree control boards, then it gets a device handles for internal management. If successful it returns the number of all egTree control boards currently with USB connected in the system otherwise it returns 0.

```
   input:  none
   output: create an internal table to manage device handles, 16 boards for max capacity
   remark: application software must call egConnect() routine before attempting to
communicate with egTree device board.
```

int egIdentifier(int idx, char *message)

Have specific egTree control board return its identifier, which is device-type specific. For the same type of egTree boards their identifiers are the same.

```
   input:  int  idx -- 0-based index to logically tag the boards fitted in the system
           char *message -- user provided buffer to contain returned text message.
   output: identifier of this type of egTree board
   remark: users use this to gain the board ID and manufacture information
```

int egRead(int idx, long *buf) ONLY for egTree-PIO32AD

Read 32-bit data from a specific egTree control board. The buffer size provided by user must at least be 32-bit. It returns 1 on valid data in buffer, returns 0 on undefined data in buffer.

```
   input:  int  idx -- 0-based index to logically tag the boards fitted in the system
   output: long *buf -- user provided buffer to receive 32-bit data, which reflect PIO
pin voltage
```

int egReadADC(int idx, int chn, long *dat) ONLY for egTree-PIO32AD

Read 10-bit ADC data from a specific egTree device board and a given channel. The 10-bit result is then saved in *dat. It returns 1 on valid data in buffer, returns 0 on undefined data in buffer.

```
   input:  int  idx -- 0-based index to logically tag the boards fitted in the system
           int  chn -- 0, 1, 2, 3 four available channels
   output: long *dat -- user provided buffer to receive 32-bit data, only lower 10 bits
are defined
```

int egSampleAll(int idx, IEE_ADXY *dat)

Read 8 channel 10-bit ADC data and digital input X<7..0>, digital output Y<7..0> from a specific egTree device board. Prior to any data, the first field is ok flag. It returns 1 on valid data in buffer, returns 0 on undefined data in buffer.

```
   input:  int  idx -- 0-based index to logically tag the boards fitted in the system
   output: IEE_ADXY *dat -- user provided buffer to receive 11 unsigned short
```

int egWrite(int idx, long mask, long dat) ONLY for egTree-PIO32AD

Write 32-bit data to egTree device board. It returns 1 on data successfully written, returns 0 on writing failure.

```
   input:  int idx  -- 0-based index to logically tag the boards fitted in the system
```

```
           long mask-- 32-bit mask, for each bit 1 allow for change, 0 unaffected
           long dat -- 32-bit data, bit31 is mapped to PIO pin31 on the board, bit0 to
PIO pin0
   output: none
```

int egConfigurePio32(int idx, long cfg) ONLY for egTree-PIO32AD

Write 32-bit data from users buffer to egTree device board to configure IO pins. Write 0 to configure output, 1 to configure input. This action is validated by a mask bit. It returns 1 on data successfully written, returns 0 on writing failure.

```
   input:  int idx -- 0-based index to logically tag the boards fitted in the system
           long cfg  -- 32-bit to indicate which pins are configured as 1 input or 0
output
   output: none
```

int egGetCFGPio32(int idx, long *cfg) ONLY for egTree-PIO32AD

Retrieve 32-bit configuration data from egTree device board PIO32 port. Current configuration is stored in cfg. It returns 1 on CFG successfully read, returns 0 on reading failure.

```
   input:  int idx -- 0-based index to logically tag the boards fitted in the system
   output: long *cfg  -- 32-bit to indicate which pins were configured as 1 input or 0
output
```

int egSoftReset(int idx)

Signal egTree control board to perform a soft reset, including reset all IO peripherals. It always returns 0 due to timeout on HID response. PC will be notified by a new HID plug-in.

```
   input:  int idx -- 0-based index to logically tag the boards fitted in the system
   output: none
```

int egInput(int idx, unsigned short addreg, long *dat)

Read register data from egTree deice over USB to PC buffer. It returns 1 for success and 0 for failure.

```
   input:  int idx -- one of the USB port slot index
           unsigned short addreg -- the starting address of register of egTree device in
IO space
   Output: long *dat -- data buffer to hold the IO space register content returned
```

int egOutput(int idx, unsigned short addreg, long dat)

Write register data from PC buffer to egTree deice over USB. It returns 1 for success and 0 for failure.

```
   input:  int idx -- one of the USB port slot index
           unsigned short addreg -- the starting address of register of egTree device in
IO space
           long dat -- data to write to IO register from PC to egTree device
```

void egClose(void)

```
Close all egTree USB connections when USB communication is no longer required for a
specific application session.
   input:  none
   output: close all File handles managed in an internal table
   remark: after this routine is called it releases system resource to windows
```

NET Class

```
/* data structure for SampleAll */
typedef struct _jf_IEEadxy {
  unsigned short ok;
  unsigned short an[8]; //eight-channel analog data
  unsigned short xx;
  unsigned short yy;
} IEE_ADXY;
```

int Net_egIdentifier(char *ipaddr, int idx, char *message)

```
Have specific egTree device return its identifier, which is device-type specific. For
same type of device boards their identifiers are the same. It returns 1 on success, 0 on
failure.
  input:  char *ipaddr -- the IP address of target egTree device, eg. egDAA
          int  idx -- 0-based index to logically tag the boards fitted in the device
system
          char *message -- user provided buffer to contain returned text message, if
egIdentifier(ipaddr, idx, 0) is called, then this message is not returned.
  output: identifier of this type of egTree board
  remark: users use this to gain the board ID and manufacture information
```

int Net_egSoftReset(char *ipaddr, int idx)

```
Signal egTree device to perform a soft reset, including reset all IO peripherals. It
always returns 0 due to timeout on HID response. PC will be notified by a new HID plug-in.
  input:  ipaddr -- the IP address of target device
          int idx -- 0-based index to logically label the boards fitted in the device
system
  output: none
```

int Net_egSampleAll(char *ipaddr, int idx, IEE_ADXY *egt)

```
Read 8 channel 10-bit ADC data and digital input X<7..0>, digital output Y<7..0> from a
specific egTree device. Prior to any data, the first field is ok flag. It returns 1 on
valid data in buffer, returns 0 on undefined data in buffer.
  input:  char *ipaddr -- the ip address of target device
          int idx -- 0-based index to logically label the boards fitted in the device
system
  output: IEE_ADXY *egt -- user provided buffer to receive 11 unsigned short
```

int Net_egInput(char *ipaddr, int idx, unsigned short addreg, long *dat)

```
Read 32-bit register content from egTree device. It returns 1 on success, 0 on failure
  input:  char *ipaddr -- the ip address of target device
          int idx -- 0-based index to logically label the boards fitted in the system
          unsigned short addreg -- register address defined by egTree device
  output: long *dat -- hold the content of the deice register
```

int Net_egOutput(char *ipaddr, int idx, unsigned short addreg, long dat)

```
Write 32-bit data to egTree device register. It returns 1 on success, 0 on failure.
  input:  char *ipaddr -- the ip address of target device
          int idx -- 0-based index to logically label the boards fitted in the system
          unsigned short addreg -- register address defined by egTree device
          long dat -- the 32-bit data (although as few as 1 bit to be used) to be sent
and written to deice register
```

RS485 class

```
data structure for SampleAll
typedef struct _jf_IEEadxy {
  unsigned short ok;
  unsigned short an[8]; //eight-channel analog data
  unsigned short xx;
  unsigned short yy;
} IEE_ADXY;
```

void eg485Close(void)

```
Close egTree 485 connection when 485 communication is no longer required for a specific
application session, only one RS485 network is supported.
   input:  none
   output: close RS485 File handles managed in an internal table
   remark: after this routine is called it releases system resource to windows
```

int eg485Open(char *devname, unsigned long baud)

```
Open egTree 485 connection before accessing the egTree device. It returns 0 on success,
otherwise -1 on failure.
   input:  char *devname -- deice name of the RS485 port, eg COM1, COM2,... if the port
is USB-RS485 adapter, then "VID&PID" of the USB connection is used, eg "1efa&011b"
           unsigned long baud -- baud rate of serial communication of the RS485 network
data format use 8-bit for data, 0-bit for parity, 2-bit for stop
```

int eg485Output(unsigned char aa, unsigned short addr, unsigned short regpoints, unsigned char *dat)

```
Send out and write a string of data to egTree device over RS485 from a buffer. It returns
the number of bytes written to the egTree device successfully.
   input:  unsigned char aa -- one of the RS485 slave address on the network through 485
bus
           unsigned short addr -- the starting address of register of egTree device
```

```
            unsigned short regpoints -- this is the 16-bit register number to be accessed,
            in our case, for each single 32-bit register, this field should be 2 for each
            unsigned char *dat -- this is data buffer to write to egTree device register,
the byte order should comply with modbus convention, which is most significant byte comes
first, the byte number in buffer is denoted by (regpoints * 2)
```

int eg485Input(unsigned char aa, unsigned short addr, unsigned short regpoints, unsigned char *dat)

```
Read register data from egTree deice over RS485 to PC buffer. It returns the number of
bytes read from the egTree device successfully.
   input:  unsigned char aa -- one of the RS485 slave address on the network through 485
bus
           unsigned short addr -- the starting address of register of egTree device
           unsigned short regpoints -- this is the 16-bit register number to be accessed,
           in our case, for each single 32-bit register, this field should be 2 for each
   output: unsigned char *dat -- this is data buffer to hold the data come from egTree
device register, the space should be big enough to hold the arrival data (regpoints * 2),
the byte order should comply with modbus convention, which is most significant byte comes
first, the byte number in buffer should be equivalent to (regpoints * 2)
   remark: sample all function can be achieved by: eg485Input(aa, 0xe000, 0x0a, &dat[0]);
```

int eg485SampleAll(unsigned char aa, void *dat)

```
Read all 8-ADC channel and packed X & Y from egTree device. It returns the number of
bytes read from the egTree device successfully, 20 in this case.
   input:  unsigned char aa -- one of the RS485 slave address on the network through 485
bus
           void *dat -- this is data buffer to hold the data come from egTree device
register, here particularly is data structure of IEE_ADXY
```

4 .Net (VB/C#/C++) Interface

.Net (VB/C#/C++) interface is encapsulated in object classes. The 3 category IO communication interfaces have 3 corresponding classes defined under the same naming space: “egTreeIO”. The USB IO class has the class name: egUSBPort. The RS485 IO class has the class name: eg485Port. The Net IO class has the name: egNetPort. Apart from the above 3 classes, the .Net Interface also defines a data structure for all of them, which is used for efficient data acquisition.

```
    [StructLayout(LayoutKind.Sequential, Pack = 2)]
    public struct IEE_ADXY
    {
        public UInt16 ok;  //the number of valid fields followed
        public UInt16 an0, an1, an2, an3, an4, an5, an6, an7;
        public UInt16 xx;
        public UInt16 yy;
    }
```

USB class (egUSBPort)

```
public egUSBPort();
```

This constructor includes the function of opening all currently connected egTree devices

```
public int Boards
```
This read only property returns the egTree device number currently connected to the host PC.

```
public int Select
```
Read/write property. If there is more than one egTree device connected to the host, then a default index can be selected by this property

```
public void SoftReset();
```
Do soft reset for all the egTree devices

```
public StringBuilder Identity();
public StringBuilder Identity(int idx);
```
Get the device description of the egTree device. The shorter form takes the default index.

```
public IEE_ADXY SampleAll();
public IEE_ADXY SampleAll(int idx);
```
Sample all the ADC and X7..0, Y7..0 from the egPLC devices. The shorter form takes the default index.

```
public int Input(UInt16 addr, ref UInt32 dat);
public int Input(int idx, UInt16 addr, ref UInt32 dat);
```
Read a register content from IO space. The shorter form takes the default index.

```
public int Output(UInt16 addr, UInt32 dat);
public int Output(int idx, UInt16 addr, UInt32 dat);
```
Write a register of IO space. The shorter form takes the default index.

NET Class (egNetPort)

```
public egNetPort(String ipaddr);
```
Contructor of the class, IP address will be registered internally as default for the following short form methods.

```
public void SoftReset(int idx);
public void SoftReset(String ipaddr, int idx);
```
Do soft reset of all the IO boards in an egDAA device. The shorter form takes the default IP address.

```
public String Identity(int idx);
public String Identity(String ipaddr, int idx);
```
Get the device description of the egDAA device with IO module index. The shorter form takes the default IP address. The egDAA device has at least one IO module fitted.

```
public IEE_ADXY SampleAll(int idx);
public IEE_ADXY SampleAll(String ipaddr, int idx);
```
Sample all the ADC and X7..0, and Y7..0 from the egDAA device with IO module index. The shorter form takes the default IP address. The egDAA device has at least one IO module fitted.

```
public int Input(int idx, UInt16 addreg, ref UInt32 dat);
public int Input(String ipaddr, int idx, UInt16 addreg, ref UInt32 dat);
```
Read a register content from IO space of egDAA with IO module index. The shorter form takes the default IP address. The egDAA device has at least one IO module fitted.

```
public int Output(int idx, UInt16 addreg, UInt32 dat);
public int Output(String ipaddr, int idx, UInt16 addreg, UInt32 dat);
```

Write a register of IO space of egDAA with IO module index. The shorter form takes the default IP address. The egDAA device has at least one IO module fitted.

RS485 class (eg485Port)

```
public eg485Port();
public eg485Port(String devname, UInt32 baud);
```

constructor o the class. The constructor can take the parameters of device name and baud rate. The device name is com port name, eg. Com1, com2,…etc. For USB-CDC converters, the device name can be in form of "vid_pid". For example "1e84_0112". For the non-parameter constructor, a subsequent Open method is called to realize the connection.

```
public bool IsOpened();
```

Query the object whether or not already opened and live.

```
public int Open(String devname, UInt32 baud);
```

Open method with parameters device name and baud rate.

```
public void Close();
```

Close connection method

```
public int Output(Byte aa, UInt16 addr, UInt32 dat);
```

Write a 32-bit data to remote device IO space register, with the target slave address aa.

```
public int Input(Byte aa, UInt16 addr, ref UInt32 dat);
```

Read a 32-bit data from remote device IO space register, with the target slave address aa.

```
public IEE_ADXY SampleAll(byte aa);
```

Sample all the ADC and X7..0, and Y7..0 from remote slave device with target address aa.

The methods in .Net class is organized in the same way as it is organized in C/C++ API. They are all encapsulated in a single egTreeIO.dll module. It is worth pointing out that some egTree devices which have more communication ports can be connected and accessed through each of their IO communication port.

5 Build User Application

Here will show the way to utilize the API for C/C++ and for .Net with 2 examples.

C/C++ Example

```
#include <stdio.h>
#include "egCommIO.h"

void main(void)
{
  char msg[64];
  int  idx, j, num;

  num = egConnect();
  printf("%d egTree boards:\n", num);
  for (idx=0; idx<num; idx++) {
    for (j=0; j<64; j++) msg[j] = 0;
    j = egIdentifier(idx, msg);
    printf("%d: %s\n", idx, msg);
  }
}
```

The above example open all the egTree devices connected to host PC through USB ports, and print and display all the device description.

To compile this program egCommIO.h is required to be included in the source file and the function prototypes can be referred from this .H header file.

To link this program egCommIO.lib is required to be provided in command line along with the .OBJ files to generate the application exe file.

To run the program egCommIO.dll is required to be provided and accessible in run-time environment.

.Net Example

The following is a windows form application where it periodical samples egPLC ADC channels through “timerSample_Tick” routine and display 4 channels (CHN0, CHN2, CHN3, CHN4) of them on the form panel in real-time. The displayed channels are from sensors: SF6, O2, temperature, and relative humidity.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using egTreeIO;

namespace user_ivySensor
{
    public partial class Form1 : Form
    {
        private egUSBPort usbIO = null;

        public Form1()
        {
            InitializeComponent();
        }

        void stop()
        {
            usbIO = null;
            buttonStart.Enabled = true;
            buttonStop.Enabled = false;
            timerSample.Enabled = false;
        }

        void start()
        {
            usbIO = new egUSBPort();
            buttonStart.Enabled = false;
            buttonStop.Enabled = true;
            timerSample.Enabled = true;
        }

        private void buttonStart_Click(object sender, EventArgs e)
        {
            start();
        }
        private void buttonStop_Click(object sender, EventArgs e)
        {
            stop();
        }


        private void timerSample_Tick(object sender, EventArgs e)
        {
            IEE_ADXY adxy = usbIO.SampleAll();
            if (adxy.ok != 0)
            {
                ucVertBarAD0.Adc = adxy.an0 * 100 / 1024;
                ucVertBarAD1.Adc = adxy.an1 * 100 / 1024;
                ucVertBarAD2.Adc = adxy.an2 * 100 / 1024;
                ucVertBarAD3.Adc = adxy.an3 * 100 / 1024;
                ucVertBarAD4.Adc = adxy.an4 * 100 / 1024;
                ucVertBarAD5.Adc = adxy.an5 * 100 / 1024;
                ucVertBarAD6.Adc = adxy.an6 * 100 / 1024;
                ucVertBarAD7.Adc = adxy.an7 * 100 / 1024;


                //display sensor value
                int val = adxy.an0 - 204;
                if (val < 0) val = 0;
                val = (int)(val * 2.45);
                labelSF6.Text = val.ToString();

                val = adxy.an2;
                labelO2.Text = (val / 41.0).ToString("F2");

                val = adxy.an3;
                labelTMP.Text = (val / 10.0).ToString("F2");

                val = adxy.an4;
                labelHMD.Text = ((val - 530) * 0.106 + 40).ToString("F2");
            }
        }


    }
}
```

The egTreeIO.dll exports all the 3 IO communication classes, which is provided to visual studio for reference. “using egTreeIO” is required to designate naming space.