

System Initialize

- [Summary](#)
 - [Code](#)
 - [API](#)
-

```
namespace ...
{
    public class ...
    {
        public void Init()
        {
            // Master Device
            InitMasterDevice();

            // Config Scan
            CheckChannel();

            // 가
            AxisServoOn();

            //
            AxisHomeReturn();
        }

        private void InitMasterDevice()
        {
            //
            DeviceLoad();

            // 가
            CompareSlaveCount();
            // SW Version(FW, WDM, SDK)
            GetVersionCompResult();
            // Input / Output
            CheckReveseConnection();
            // Network alStatus OP
            SetAlStateToOP();
        }

        private void DeviceLoad() {}
        private void CompareSlaveCount() {}
        private void GetVersionCompResult() {}
    }
}
```

```

        private void CheckReveseConnection() {}
        private void SetAlStateTo0P() {}
        private void CheckChannel() {}
        private void AxisServo0n() {}
        private void AxisHomeReturn() {}
    }
}

```

x , , 가 가

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using System.Threading;
using ec = ComiLib.EtherCAT.SafeNativeMethods;

namespace EtherCAT_Examples_CSharp
{
    /// <summary>
    ///
    /// </summary>
    public class CookBook_Initialize
    {
        // SystemInit()
        // SystemInit(), Init() Task
    }

    #if true // .net 4.5 (async / await)
        public async void SystemInit()
        {
            IsStop = false;
            bool isSuccess = await Init();
            AddLog(string.Format("System Initialize {0}", isSuccess ?
"success" : "false"));
        }

        public async Task<bool> Init()
        {
            // Master Device
            if (!await Task.Run(() => InitMasterDevice()))
                return false;

            // Config Scan . ( )
            if (!await Task.Run(() => CheckChannel()))
                return false;
        }
    }
}

```

```

        var taskList = new List<Task<bool>>();

        //
        axisList.ToList().ForEach(axis => taskList.Add(Task.Run(() =>
AxisServoOn(axis)))));
        var resultList = (await Task.WhenAll(taskList)).ToList();
        if (resultList.Any(x => !x))
            return false;

        //
        taskList.Clear();
        axisList.ToList().ForEach(axis => taskList.Add(Task.Run(() =>
AxisHomeReturn(axis)))));
        resultList = (await Task.WhenAll(taskList)).ToList();
        return resultList.All(x => x);
    }

#else // .net 4.0
    public void SystemInit()
    {
        Task<bool>.Factory.StartNew(() => Init()).ContinueWith(x =>
            AddLog(string.Format("System Initialize {0}", x.Result ?
"success" : "false")));
    }

    public bool Init()
    {
        // Master Device
        if(!Task.Factory.StartNew(() => InitMasterDevice()).Result)
            return false;

        // Config Scan
        if (!Task.Factory.StartNew(() => CheckChannel()).Result)
            return false;
        var taskList = new List<Task<bool>>();

        //
        axisList.ToList().ForEach(axis =>
taskList.Add(Task.Factory.StartNew(() => AxisServoOn(axis)))));
        Task.Factory.ContinueWhenAll(taskList.ToArray(), r => { });
        //Task.WaitAll(taskList.ToArray());
        if (taskList.Exists(x => !x.Result))
            return false;

        //
        taskList.Clear();
        axisList.ToList().ForEach(axis =>
taskList.Add(Task.Factory.StartNew(() => AxisHomeReturn(axis)))));
        Task.Factory.ContinueWhenAll(taskList.ToArray(), r => { });
        return (!taskList.Exists(x => !x.Result));
    }
}

```

```
#endif
    public bool IsStop { get; set; }

    int netID = 0;
    uint slaveCount = 0;
    int errorCode = 0;
    byte[] axisList = new byte[32];
    List<string> errorList = new List<string>();
    CancellationTokensSource cts;
#region AddLog

    private void AddLog(int errorCode)
    {
        if (errorCode == 0)
            return;

        Debug.WriteLine(ec.ecUtl_GetErrorString(errorCode));
    }

    private void AddLog(string errorString)
    {
        Debug.WriteLine(errorString);
    }
#endregion

    void Stop()
    {
        IsStop = true;
    }

    /// <summary>
    /// Master Device
    /// </summary>
    /// <returns></returns>
    private bool InitMasterDevice()
    {
        //
        if (!DeviceLoad())
            return false;

        //
        if (!CompareSlaveCount())
            return false;

        // SW Version(FW, WDM, SDK)
        if (!GetVersionCompResult())
        {
            AddLog("Version compare fail");
            return false;
        }
    }
}
```

```

    }
    AddLog("Version compare compt");

//      Input / Output
if (!CheckReveseConnection())
{
    AddLog("                .");
    return false;
}

// Network alStatus OP
if (!SetAlStateToOP())
    return false;

    AddLog("MasterDevice Init Compt");
    return true;
}

private bool DeviceLoad()
{
    try
    {
        // Device
        if (!ec.ecGn_LoadDevice(ref errorCode))
        {
            AddLog(errorCode);
            switch (errorCode)
            {
                case 5:
                    AddLog("Mater Device 12V
.);
                    break;

                case 8:
                    AddLog("Mater Device가      . Windows
FastBoot(      )가      ");
                    break;
            }

            return false;
        }

        return true;
    }
    catch (BadImageFormatException)
    {
        AddLog("ecGn_LoadDevice Failed : DLL      (x86/x64) OS
.);
        return false;
    }
    catch (DllNotFoundException)

```

```

        {
            AddLog("ecGn_LoadDevice Failed : DLL
                .");
            return false;
        }
        catch (Exception ex)
        {
            AddLog(string.Format("ecGn_LoadDevice Failed : Exception -
{0}", ex.ToString()));
            return false;
        }
    }

private bool CompareSlaveCount()
{
    // Config slave
    // Configuration

    // Config
    https://winoar.com/dokuwiki/platform:ethercat:1_setup:10_config:20_configura
tion
    uint cfgCount = ec.ecNet_GetCfgSlaveCount(netID, ref errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

    // slave
    uint slaveCount = ec.ecNet_ScanSlaves(netID, ref errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

    //
    if (cfgCount != slaveCount)
    {
        AddLog("
가
.");
        AddLog(string.Format("ScanSlave : {0}. CfgCount : {1}",
slaveCount, cfgCount));
        AddLog("
가
.");
        AddLog("
가 ScanSlave
Configuration
.");
        return false;
    }
    return true;
}

private bool SetAlStateTo0P()

```

```

    {
        // alStatus :
        https://winoar.com/dokuwiki/platform:ethercat:2_info:10_alstatus
        ec.ecNet_SetAlState(netID, ec.EEcAlState.OP, ref errorCode);
        if (errorCode != 0)
        {
            AddLog(errorCode);
            return false;
        }
        AddLog("Set AlState to OP");

        // Network alStatus가 , Slave alStatus Network
        alStatus
        // slave alStatus가 OP가 , slave
        //
        https://winoar.com/dokuwiki/platform:ethercat:1_setup:10_config:ts:30_safeop
        _failed

        // alStatus가 OP가
        // alStatus가 OP가 slave
        ec.EEcAlState alState = ec.EEcAlState.INITIAL;
        Stopwatch sw = new Stopwatch();
        sw.Start();

        bool isSuccess = false;
        while (sw.ElapsedMilliseconds < 10000 && !isSuccess)
        {
            if (IsStop)
            {
                AddLog("Stop");
                return false;
            }

            isSuccess = true;
            for (int i = 0; i < slaveCount; i++)
            {
                alState = ec.ecSlv_GetAlState_A(netID, i, ref
                errorCode);

                if (alState != ec.EEcAlState.OP || errorCode != 0)
                {
                    isSuccess = false;
                    break;
                }
            }
            Thread.Sleep(200);
        }

        if (!isSuccess)
        {
            for (int i = 0; i < slaveCount; i++)
            {

```

```

        alState = ec.ecSlv_GetAlState_A(netID, i, ref
errorCode);
        if (alState != ec.EEcAlState.OP || errorCode != 0)
            AddLog(string.Format("      : {0}    AlState  OP
.", i));
    }

    return false;
}

return true;
}

/// <summary>
/// Config      Scan      (      )
/// </summary>
/// <returns></returns>
public bool CheckChannel()
{
    AddLog("CheckChannel");
    // Scan  Axis  IO
//
int axisCount = ec.ecmGn_GetAxisList(netID, axisList, 32, ref
errorCode);
Array.Resize(ref axisList, axisCount);

if (errorCode != 0)
{
    AddLog(errorCode);
    return false;
}

if (axisCount == 0)
{
//
    AddLog("      .");
}

// config  DI Channel
int totalDiCount = ec.ecdiGetNumChannels(netID, ref errorCode);
if (errorCode != 0)
{
    AddLog(errorCode);
    return false;
}

// config  DO Channel
int totalDoCount = ec.ecdoGetNumChannels(netID, ref errorCode);
if (errorCode != 0)
{
    AddLog(errorCode);
}

```

```

        return false;
    }

#if false //
        int definedAxisCount = 8; //
        if (definedAxisCount != axisCount)
        {
            // axisCount 가

            // config
            AddLog(" Config 가 .");
            return false;
        }

        int definedDiCount = 32; // di channel
        int definedDoCount = 32; // do channel

        if (definedDiCount != totalDiCount)
        {
            AddLog(" DI Slave가 Config 가 .");
            return false;
        }

        if (definedDoCount != totalDoCount)
        {
            AddLog(" DO Slave가 Config 가 .");
            return false;
        }
#endif

    return true;
}

/// <summary>
/// FW - WDM - DLL
/// </summary>
private bool GetVersionCompResult()
{
    // SW Version(FW, WDM, SDK)

    //
    ec.TEcFileVerInfo_SDK sdkInfo = new ec.TEcFileVerInfo_SDK();
    ec.TEcFileVerInfo_WDM driverInfo = new ec.TEcFileVerInfo_WDM();
    ec.TEcFileVerInfo_FW fwInfo = new ec.TEcFileVerInfo_FW();

    bool isSuccess = ec.ecNet_GetVerInfo(netID, ref sdkInfo, ref
driverInfo, ref fwInfo, ref errorCode);

    if (!isSuccess)
    {
        //FW - SDK
        switch (sdkInfo.nFwCompResult)

```

```

        {
            case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_LOWER:
AddLog("Library version is higher than the Firmware"); return false;
            case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_HIGHER:
AddLog("Library version is lower than the Firmware"); return false;
            case (int)ec.EEcVerCompatResult.ecVER_MATCH: AddLog("FW-
SDK : OK"); break;
            default: AddLog("Firmware Version is invalid"); return
false;
        }

//FW-WDM
switch (driverInfo.nFwCompResult)
{
    case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_LOWER:
AddLog("Driver version is higher than the Firmware"); return false;
    case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_HIGHER:
AddLog("Driver version is lower than the Firmware"); return false;
    case (int)ec.EEcVerCompatResult.ecVER_MATCH: AddLog("FW-
WDM : OK"); break;
    default: AddLog("Firmware Version is invalid"); return
false;
}

//SDK-WDM
switch (sdkInfo.nWdmCompResult)
{
    case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_LOWER:
AddLog("Driver version is lower than the Library"); return false;
    case (int)ec.EEcVerCompatResult.ecVER_MISMATCH_HIGHER:
AddLog("Library version is lower than the Driver"); return false;
    case (int)ec.EEcVerCompatResult.ecVER_MATCH:
AddLog("SDK-WDM : OK"); break;
    default: AddLog("Driver Version is invalid"); return
false;
}
}

return isSuccess;
}

/// <summary>
///     Inport / Outport가     가
/// </summary>
private bool CheckReveseConnection()
{
    if (!CanCheckReverseConnection())
        return false;

//
    int scanSlaveCount = 0;

```

```

        int reverseConnectionCount =
ec.ecNet_CheckReverseConnections(netID, ref scanSlaveCount, ref errorCode);

//          가
//if (definedSlaveCount != scanSlaveCount)
//{
// //          Config
// //  AddLog(string.Format("Disconnected Slave Count = {0}",
definedSlaveCount - scanSlaveCount));
//}

// reverseConnectionCount  0
if (reverseConnectionCount == 0)
{
    AddLog(string.Format("ReverseConnection is nothing."));
    return true;
}
else
{
    AddLog(string.Format("ReverseConnectionCount = {0}",
reverseConnectionCount));

    bool isReverseConnected = false;
//
//    for (ushort i = 0; i < scanSlaveCount; i++)
//    {
//        isReverseConnected =
ec.ecSlv_IsReverseConnected_A(netID, i, ref errorCode);

//        if (isReverseConnected)
//            AddLog(string.Format("Check SlaveIndex {0} :
ReverseConnected", i));
//    }
//    return false;
//
//}

/// <summary>
/// DLL : 1.5.3.2 ( FW : 1.92 / WDM : 1.5.0.6)          가
/// </summary>
private bool CanCheckReverseConnection()
{
    ec.TEcFileVerInfo_SDK sdkInfo = new ec.TEcFileVerInfo_SDK();
    ec.TEcFileVerInfo_WDM driverInfo = new ec.TEcFileVerInfo_WDM();
    ec.TEcFileVerInfo_FW fwInfo = new ec.TEcFileVerInfo_FW();

    //FW / Driver / Library
    bool isSuccess = ec.ecNet_GetVerInfo(netID, ref sdkInfo, ref
driverInfo, ref fwInfo, ref errorCode);
    string sdkVer = string.Format("{0}{1}{2}{3}",

```

```

sdkInfo.CurVer.MajorVer, sdkInfo.CurVer.MinorVer, sdkInfo.CurVer.BuildNo,
sdkInfo.CurVer.RevNo);
    int curVer = int.Parse(sdkVer);

    // Library      1.5.3.2
    if (curVer < 1532)
    {
        AddLog("CheckReverseConnection : Not Supported version");
        return false;
    }

    return true;
}

/// <summary>
/// Alarm Clear & Servo0n
/// </summary>
/// <param name="axisID"></param>
/// <returns></returns>
private bool AxisServo0n(int axisID)
{
    // state
    int motState = ec.ecmSxSt_GetMotState(netID, axisID, ref
errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

    if (motState != 0)
    {
        // State가 Stop
        // Stop
        ec.ecmSxMot_Stop(netID, axisID, 1, 1, ref errorCode);
    }

    Stopwatch sw = new Stopwatch();
    if (motState == -1010) //
    {
        ec.ecmSxCtl_ResetAlm(netID, axisID, ref errorCode); //

        if (errorCode != 0)
        {
            AddLog(errorCode);
            return false;
        }

        sw.Start();
        while (sw.ElapsedMilliseconds < 1000 &&

```

```

ec.ecmSxSt_GetMotState(netID, axisID, ref errorCode) == -1010)
    Thread.Sleep(100);
motState = ec.ecmSxSt_GetMotState(netID, axisID, ref
errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

    // 1
    // 1
    if (motState == -1010)
    {
        AddLog(string.Format("Axis {0} :
.", axisID));
        return false;
    }
}

// 가 Operation Enable
var isOn = ec.ecmSxCtl_GetSvon(netID, axisID, ref errorCode);
if (errorCode != 0)
{
    AddLog(errorCode);
    return false;
}

if (!isOn)
{
    // Operation Enable 가 , Enable ( )
    ec.ecmSxCtl_SetSvon(netID, axisID, 1, ref errorCode);

    sw.Restart();
    while (sw.ElapsedMilliseconds < 2000 &&
!ec.ecmSxCtl_GetSvon(netID, axisID, ref errorCode))
        Thread.Sleep(100);

    // 2 Enable , Enable
    // 2
    isOn = ec.ecmSxCtl_GetSvon(netID, axisID, ref errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

    if (!isOn)
    {

```

```

        AddLog(string.Format("Axis {0} : Servo0n fail.",
axisID));
        return false;
    }
}

AddLog(string.Format("Axis {0} : Servo0n Success.", axisID));
return true;
}

/// <summary>
/// Home Return
/// </summary>
/// <param name="axisID"></param>
/// <returns></returns>
private bool AxisHomeReturn(int axisID)
{
//      Guide
//
https://winoar.com/dokuwiki/platform:ethercat:70\_users\_guide:10\_homing:start

    int homeMode = 114;    //
    ec.ecmHomeCfg_SetMode(netID, axisID, homeMode, ref errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }

//      가      가
//      Offset
    double homeOffset = 0;
#if true
    ec.ecmHomeCfg_SetOffsetEx(netID, axisID, homeOffset, false, 1,
ref errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }
#else
//      , 가
    ec.ecmHomeCfg_SetOffset(netID, axisID, homeOffset, ref
errorCode);
    if (errorCode != 0)
    {
        AddLog(errorCode);
        return false;
    }
#endif
}

```

```

//
//
        int speedMode = 2; // 가 . 0:Constant
1:Trapzoidal 2:S-Curve
        double workSpeed = 100000;
        double accel = workSpeed * 10;
        double decel = workSpeed * 10;
        double specVel = workSpeed / 10; // 1
        , 가 .

        ec.ecmHomeCfg_SetSpeedPatt(netID, axisID, speedMode, workSpeed,
accel, decel, specVel, ref errorCode);
        if (errorCode != 0)
        {
            AddLog(errorCode);
            return false;
        }
        int dir = 0; // . 0:(-) 1:(+)
        ec.ecmHomeMot_MoveStart(netID, axisID, dir, ref errorCode);
        if (errorCode != 0)
        {
            AddLog(errorCode);
            return false;
        }

        // TimeOut
        Stopwatch sw = new Stopwatch();
        sw.Start();

        bool isBusy = true;
        while (sw.ElapsedMilliseconds < 10000 && isBusy)
        {
            if (IsStop)
            {
                AddLog("Stop");
                return false;
            }

            isBusy = ec.ecmHomeSt_IsBusy(netID, axisID, ref errorCode);
            Thread.Sleep(100);
        }

        // isBusy가 true , timeout while
        if (isBusy)
        {
            AddLog(string.Format("Axis {0} Homing Timeout", axisID));
            return false;
        }

        // isBusy가 false ,

```

```

        // , isBusy false
        ec.TEcmHomeSt_Flags homeFlag = new ec.TEcmHomeSt_Flags();
        homeFlag.word = ec.ecmHomeSt_GetFlags(netID, axisID, ref
errorCode);
        bool isSuccess = ((homeFlag.word >> 2) & 1) == 1;
        if (!isSuccess)
        {
            // 가 , 가 가 ,
motState 가
            // MotState == 0 , Stop
            int motState = ec.ecmSxSt_GetMotState(netID, axisID, ref
errorCode);
            AddLog(errorCode);
        }

        AddLog(string.Format("Axis {0} : HomeReturn {1}", axisID,
isSuccess ? "success" : "fail"));
        return isSuccess;
    }
}
}

```

- [ecGn_LoadDevice](#)
- [ecNet_GetCfgSlaveCount](#)
- [ecNet_ScanSlaves](#)
- [ecNet_SetAIState](#)
- ecSlv_GetAIState_A
- ecmGn_GetAxisList
- ecdiGetNumChannels
- ecdoGetNumChannels
- [ecNet_GetVerInfo](#)
- ecNet_CheckReverseConnections
- ecSlv_IsReverseConnected_A
- ecmSxSt_GetMotState
- ecmSxMot_Stop
- ecmSxCtl_ResetAlm
- ecmSxCtl_GetSvon
- ecmSxCtl_SetSvon
- ecmHomeCfg_SetMode
- ecmHomeCfg_SetOffsetEx
- ecmHomeCfg_SetOffset
- ecmHomeCfg_SetSpeedPatt
- ecmHomeMot_MoveStart

- ecmHomeSt_IsBusy
- ecmHomeSt_GetFlags

From:
<http://comizoa.com/info/> - -

Permanent link:
http://comizoa.com/info/doku.php?id=platform:ethercat:70_users_guide:00_cookbook:20_systeminit:start&rev=1620968454

Last update: **2024/07/08 18:22**