Table of Contents

System Initialize


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

            // Config된 정보와 Scan된 정보를 비교합니다. (설정값과 실제값 비교)
            CheckChannel();

    // 각 축의 알람을 클리어하고, 모터 구동 가능 상태로 변경합니다.
            AxisServoOn();

    // 각 축의 원점 복귀를 수행하여 원점을 설정합니다.
            AxisHomeReturn();
        }

        private void InitMasterDevice()
        {
    //마스터 디바이스를 로드합니다.
            DeviceLoad();

    // 설정 된 슬레이브 개수와 연결 된 슬레이브 개수가 동일한지 확인합니다.
            CompareSlaveCount();
            // SW Version(FW, WDM, SDK)이 서로 호환되는 버전인지 확인합니다.
            GetVersionCompResult();
    // 슬레이브의 Input / Output이 반대로 연결된 모듈이 있는지 확인합니다.
            CheckReveseConnection();
            // Network의 alStatus를 OP로 설정합니다.
            SetAlStateToOP();
        }

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

```
        private void GetVersionCompResult() {}
        private void CheckReveseConnection() {}
        private void SetAlStateToOP() {}
        private void CheckChannel() {}
        private void AxisServoOn() {}
        private void AxisHomeReturn() {}
    }
}
```

x 디바이스 로드, 알람 클리어, 홈복귀 등을 수행하여 시스템이 이송 가능한 상태가 되도록 초기화합니다.

```
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>
  /// 시스템 초기화
    /// </summary>
    public class CookBook_Initialize
    {
        // SystemInit() 함수를 수행하여 시스템을 초기화 합니다.
        // SystemInit(), Init() 함수는 Task 등을 사용한 비동기처리 예시이며, 중요한 것
은 함수의 실행 순서입니다.

#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 Device를 초기화합니다.
            if (!await Task.Run(() => InitMasterDevice()))
                return false;

            // Config된 정보와 Scan된 정보를 비교합니다. (설정값과 실제값 비교)
            if (!await Task.Run(() => CheckChannel()))
```

```
                return false;

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

    // 각 축의 알람을 클리어하고, 모터 구동 가능 상태로 변경합니다.
            axisList.ToList().ForEach(axis => taskList.Add(Task.Run(() =>
AxisServoOn(axis))));
            var resultList = (await Task.WhenAll(taskList)).ToList();
            if (resultList.Any(x => !x))
                return false;

    // 각 축의 원점 복귀를 수행하여 원점을 설정합니다.
            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 Device를 초기화합니다.
            if(!Task.Factory.StartNew(() => InitMasterDevice()).Result)
                return false;

            // Config된 정보와 Scan된 정보를 비교합니다. (설정값과 실제값 비교)
            if (!Task.Factory.StartNew(() => CheckChannel()).Result)
                return false;
            var taskList = new List<Task<bool>>();

    // 각 축의 알람을 클리어하고, 모터 구동 가능 상태로 변경합니다.
            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;

    // 각 축의 원점 복귀를 수행하여 원점을 설정합니다.
            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>();
        CancellationTokenSource 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 Device를 초기화합니다.
        /// </summary>
        /// <returns></returns>
        private bool InitMasterDevice()
        {
     //마스터 디바이스를 로드합니다.
            if (!DeviceLoad())
                return false;

     // 설정 된 슬레이브 개수와 연결 된 슬레이브 개수가 동일한지 확인합니다.
            if(!CompareSlaveCount())
                return false;

            // SW Version(FW, WDM, SDK)이 서로 호환되는 버전인지 확인합니다.
            if (!GetVersionCompResult())
```

```
            {
                AddLog("Version compare fail");
                return false;
            }
            AddLog("Version compare compt");

    // 슬레이브의 Input / Output이 반대로 연결된 모듈이 있는지 확인합니다.
            if (!CheckReveseConnection())
            {
                AddLog("역삽입된 모듈이 있습니다.");
                return false;
            }

            // Network의 alStatus를 OP로 설정합니다.
            if (!SetAlStateToOP())
                return false;

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

        private bool DeviceLoad()
        {
            try
            {
                // Device를 초기화합니다.
                if (!ec.ecGn_LoadDevice(ref errorCode))
                {
                    AddLog(errorCode);
                    switch (errorCode)
                    {
                        case 5:
                            AddLog("Mater Device에 12V 전원이 입력되었는지 확인 바
랍니다.");
                            break;

                        case 8:
                            AddLog("Mater Device가 부팅되지 않았습니다. Windows의
FastBoot(빠른시작켜기)가 활성화 되어 있는 경우 비활성화 하세요");
                            break;
                    }

                    return false;
                }

                return true;
            }
            catch (BadImageFormatException)
            {
                AddLog("ecGn_LoadDevice Failed : DLL 버전(x86/x64)이 OS와 맞지
```

```
않습니다.");
                return false;
            }
            catch (DllNotFoundException)
            {
                AddLog("ecGn_LoadDevice Failed : DLL을 찾을 수 없습니다.");
                return false;
            }
            catch (Exception ex)
            {
                AddLog(string.Format("ecGn_LoadDevice Failed : Exception -
{0}", ex.ToString()));
                return false;
            }
        }

        private bool CompareSlaveCount()
        {
            // Config된 slave 수를 확인합니다.
            // Configuration 단계에서 설정된 슬레이브의 수로 현재 연결된 슬레이브 수와는
관련이 없습니다.
            // Config 상세 정보
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;
            }

    // 현재 네트워크에 연결되어 있는 slave 수를 확인합니다.
            uint slaveCount = ec.ecNet_ScanSlaves(netID, ref errorCode);
            if (errorCode != 0)
            {
                AddLog(errorCode);
                return false;
            }

    // 설정된 모듈 수만큼 스캔되었는지 확인합니다.
            if (cfgCount != slaveCount)
            {
                AddLog("현재 스캔된 슬레이브의 모듈 수와 설정된 슬레이브의 모듈 수가 다릅니
다.");
                AddLog(string.Format("ScanSlave : {0}. CfgCount : {1}",
slaveCount, cfgCount));
                AddLog("전원이 들어가지 않았거나 네트워크와 연결되지 않은 슬레이브 모듈이
있는지 확인하시기 바랍니다.");
                AddLog("마스터를 포함한 슬레이브 모듈의 개수가 ScanSlave와 같다면
Configuration 을 재실행 하시기 바랍니다.");
```

```
                return false;
            }
            return true;
        }

        private bool SetAlStateToOP()
        {
            // 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");

            // Network의 alStatus가 변경되는 경우, 모든 Slave의 alStatus도 Network의
alStatus로 변경됩니다.
            // slave의 alStatus가 OP가 되지 않는 경우, 해당 slave를 점검하시기 바랍니다.
            //
https://winoar.com/dokuwiki/platform:ethercat:1_setup:10_config:ts:30_safeop
_failed

    // 모든 모듈의 alStatus가 OP가 되었는지 확인합니다.
            // alStatus가 OP가 아닌 slave는 정상적으로 제어되지 않습니다.
            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("슬레이브:{0}의 AlState를 OP로 설정
하는데 실패하였습니다.", i));
                }

                return false;
            }

            return true;
        }

        /// <summary>
        /// Config된 정보와 Scan된 정보 비교 (설정값과 실제값 비교)
        /// </summary>
        /// <returns></returns>
        public bool CheckChannel()
        {
            AddLog("CheckChannel");
            // Scan 된 Axis나 IO 채널 수를 비교하려면 아래 코드를 수행합니다.
    // 연결된 축리스트를 확인합니다.
            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)
            {
      // 서보를 사용하는 경우에만 에러처리 합니다.
                AddLog("연결된 축이 없습니다.");
            }

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

```
            // config 된 DO Channel 수를 확인합니다.
            int totalDoCount = ec.ecdoGetNumChannels(netID, ref errorCode);
            if (errorCode != 0)
            {
                AddLog(errorCode);
                return false;
            }

#if false // 미리 정의 된 축 수나 채널수가 있는 경우, 비교하여 에러처리 할 수 있다.
            int definedAxisCount = 8; // 실제 설치되어 있는 서보 드라이버의 수
            if (definedAxisCount != axisCount)
            {
                // axisCount 가 다른 경우 물리적으로 연결되지 않은 축이 있는지 확인합니다.
      // 모두 연결되어 있다면 config를 다시 실행합니다.
                AddLog("연결되지 않은 축이 있거나 Config 정보가 다릅니다.");
                return false;
            }

            int definedDiCount = 32; // 실제 설치되어 있는 di channel 수
            int definedDoCount = 32; // 실제 설치되어 있는 do channel 수

            if (definedDiCount != totalDiCount)
            {
                AddLog("연결되지 않은 DI Slave가 있거나 Config 정보가 다릅니다.");
                return false;
            }

            if (definedDoCount != totalDoCount)
            {
                AddLog("연결되지 않은 DO Slave가 있거나 Config 정보가 다릅니다.");
                return false;
            }
#endif
            return true;
        }


        /// <summary>
        /// FW - WDM - DLL 간 버전 호환성을 확인
        /// </summary>
        private bool GetVersionCompResult()
        {
            // SW Version(FW, WDM, SDK)이 서로 호환되는 버전인지 확인합니다.
    // 호환되지 않는 버전이 설치되어 있는 경우 오동작 할 수 있습니다.
            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 호환성 결과
                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 호환성 결과
                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>
   /// 슬레이브의 Inport / Outport가 반대가 삽입된 모듈을 검색합니다.
        /// </summary>
        private bool CheckReveseConnection()
        {
            if (!CanCheckReverseConnection())
```

```
                return false;

    // 역삽입된 슬레이브의 수를 확인합니다.
            int scanSlaveCount = 0;
            int reverseConnectionCount =
ec.ecNet_CheckReverseConnections(netID, ref scanSlaveCount, ref errorCode);

    // 정의된 슬레이브 수가 있다면, 스캔된 슬레이브의 수와 비교합니다.
            //if (definedSlaveCount != scanSlaveCount)
            //{
    //  // 두 변수값의 차이는 Config 되었지만 현재 연결되어 있지 않은 슬레이브의 수입니다.
            //    AddLog(string.Format("Disconnected Slave Count = {0}",
definedSlaveCount - scanSlaveCount));
            //}

            // reverseConnectionCount 값이 0인 경우, 역삽입된 모듈이 없는 경우이므로 정
상입니다.
            if (reverseConnectionCount == 0)
            {
                AddLog(string.Format("ReverseConnection is nothing."));
                return true;
            }
            else
            {
                AddLog(string.Format("ReverseConnectionCount = {0}",
reverseConnectionCount));

                bool isReverseConnected = false;
      // 역삽입된 모듈이 있는 경우, 슬레이브별로 역삽입 여부를 확인합니다.
                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>
   /// 역삽입 검출 기능을 사용할 수 있는지 확인
        /// DLL : 1.5.3.2 ( FW : 1.92 / WDM : 1.5.0.6)  이상의 버전에서 사용 가능
        /// </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 / Library의 버전을 확인합니다.
            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);

            // Library의 버전이 1.5.3.2 이하인 경우 해당 기능을 사용할 수 없습니다.
            if (curVer < 1532)
            {
                AddLog("CheckReverseConnection : Not Supported version");
                return false;
            }

            return true;
        }

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

            if (motState != 0)
            {
      // 마지막 명령에 의해 모터의 State가 Stop 이 아닌 경우, 일부 명령이 에러처리될 수 있으므
로
                // Stop 명령을 먼저 수행합니다.
                ec.ecmSxMot_Stop(netID, axisID, 1, 1, ref errorCode);
            }

            Stopwatch sw = new Stopwatch();
            if (motState == -1010)  // 서보에 알람이 발생한 경우
            {
                ec.ecmSxCtl_ResetAlm(netID, axisID, ref errorCode); // 알람
클리어 명령 실행
                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;
                }

      // 1초후에도 알람이 남아있으면, 클리어되지 않는 알람으로 간주하고 실패로 처리합니다.
      // 1초는 예시이며, 드라이버에 따라 알람 처리 시간은 다를 수 있습니다.
                if (motState == -1010)
                {
                    AddLog(string.Format("Axis {0} : 클리어되지 않는 알람이 있습니
다.", axisID));
                    return false;
                }
            }


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

            if (!isOn)
            {
                // Operation Enable 상태가 아닌 경우, Enable 명령(서보온)을 실행합니
다.
                ec.ecmSxCtl_SetSvon(netID, axisID, 1, ref errorCode);

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

      // 2초후까지 Enable이 아닌 경우, Enable을 할 수 없는 상황으로 간주하고 실패로 처리합니
다.
      // 2초는 예시이며, 드라이버에 따라 처리 시간은 다를 수 있습니다.
                isOn = ec.ecmSxCtl_GetSvon(netID, axisID, ref errorCode);
                if (errorCode != 0)
                {
```

```
                    AddLog(errorCode);
                    return false;
                }

                if (!isOn)
                {
                    AddLog(string.Format("Axis {0} : ServoOn fail.",
axisID));
                    return false;
                }
            }

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

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

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

    // 홈복귀가 완료된 후 추가 이송 거리를 설정합니다.
            // Offset은 옵션이므로 사용하지 않을 경우 함수를 호출하지 않습니다.
            double homeOffset = 0;
#if true
            ec.ecmHomeCfg_SetOffsetEx(netID, axisID, homeOffset, false, 1,
ref errorCode);
            if (errorCode != 0)
            {
                AddLog(errorCode);
                return false;
            }
#else
    // 홈복귀 완료 후, 추가 이송 없이 위치값만 변경하는 경우 다음 함수를 사용합니다.
            ec.ecmHomeCfg_SetOffset(netID, axisID, homeOffset, ref
errorCode);
            if (errorCode != 0)
```

```
            {
                AddLog(errorCode);
                return false;
            }
#endif

    // 홈복귀 속도를 설정합니다.
    // 홈복귀 속도는 단축 이송 속도와 다르므로, 별개로 설정해야 합니다.

            int speedMode = 2;  // 가감속 방식을 설정합니다. 0:Constant
1:Trapzoidal 2:S-Curve
            double workSpeed = 100000;
            double accel = workSpeed * 10;
            double decel = workSpeed * 10;
            double specVel = workSpeed / 10; // 1차 센서감지 후 빠지는 속도 또는 재
진입 속도이며, 해당 값이 낮을 수록 센서 위치에서 가깝게 정지합니다.

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

            // TimeOut을 설정합니다.
            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 인 경우, timeout 조건에 의해 while 종료
```

```
            if (isBusy)
            {
                AddLog(string.Format("Axis {0} Homing Timeout", axisID));
                return false;
            }

            // isBusy가 false 이면 일단 홈복귀는 종료된 상태이지만, 성공한 경우와 실패한 경우를 구분해야 합니다.
     // 예를 들어, 드라이버에 알람이 발생하여 정지한 경우에도 isBusy는 false로 리턴됩니다.

            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)
            {
       // 홈복귀가 실패한 경우, 대부분은 현재 축의 상태가 홈복귀 불가능 상태이며, 이는 motState로 확인 가능합니다.
                // MotState == 0 인 경우, Stop 명령에 의해 정지했다는 의미입니다.
                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_SetAlState
- ecSlv_GetAlState_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