# ArbinCTI Manual 1.2.0 Continues02...

## *Annexure*

### *Annexure A : List<StartResumeEx>*

List of structures required for advanced start and advanced resume.

```
public struct StartResumeEx
    {
        public uint channelIndex;
        public int TestID;
        public string TestNames;
        public string Schedules;
        public uint nSelectSteps;
        public uint Cycle;
        public double TestTime;
        public double StepTime;
        public double CCapacity;
        public double DCapacity;
        public double CEnergy;
        public double DEnergy;
        public double TC_Time1;
        public double TC_Time2;
        public double TC_Time3;
        public double TC_Time4;
        public double TC_CCapacity1;
        public double TC_CCapacity2;
        public double TC_DCapacity1;
        public double TC_DCapacity2;
        public double TC_CEnergy1;
        public double TC_CEnergy2;
        public double TC_DEnergy1;
        public double TC_DEnergy2;
        public float MV_Counter1;
        public float MV_Counter2;
        public float MV_Counter3;
        public float MV_Counter4;
        public float MV_Counter5;
        public float MV_Counter6;
        public float MV_Counter7;
        public float MV_Counter8;
        public double ChargeCapacityTime;
        public double DischargeCapacityTime;
        public float MVUD1;
        public float MVUD2;
        public float MVUD3;
        public float MVUD4;
        public float MVUD5;
        public float MVUD6;
        public float MVUD7;
        public float MVUD8;
        public float MVUD9;
        public float MVUD10;
        public float MVUD11;
```

```
        public float MVUD12;
        public float MVUD13;
        public float MVUD14;
        public float MVUD15;
        public float MVUD16;

        public void Reset ()
        {
            nSelectSteps = 0;

            Cycle = 0;
            TestTime = 0;
            StepTime = 0;
            CCapacity = 0;
            DCapacity = 0;
            CEnergy = 0;
            DEnergy = 0;
            TC_Time1 = 0;
            TC_Time2 = 0;
            TC_Time3 = 0;
            TC_Time4 = 0;
            TC_CCapacity1 = 0;
            TC_CCapacity2 = 0;
            TC_DCapacity1 = 0;
            TC_DCapacity2 = 0;
            TC_CEnergy1 = 0;
            TC_CEnergy2 = 0;
            TC_DEnergy1 = 0;
            TC_DEnergy2 = 0;
            MV_Counter1 = 0;
            MV_Counter2 = 0;
            MV_Counter3 = 0;
            MV_Counter4 = 0;
            MV_Counter5 = 0;
            MV_Counter6 = 0;
            MV_Counter7 = 0;
            MV_Counter8 = 0;
            ChargeCapacityTime = 0;
           DischargeCapacityTime=0;
            MVUD1 = 0;
            MVUD2 = 0;
            MVUD3 = 0;
            MVUD4 = 0;
            MVUD5 = 0;
            MVUD6 = 0;
            MVUD7 = 0;
            MVUD8 = 0;
            MVUD9 = 0;
            MVUD10 = 0;
            MVUD11 = 0;
            MVUD12 = 0;
            MVUD13 = 0;
            MVUD14 = 0;
            MVUD15 = 0;
            MVUD16 = 0;
        }
```

## *Annexure B : List<MetaVariableInfo>*

List of Structure required for advance Update Metavariable.

```
public class MetaVariableInfo
    {
        public ushort m_ChannelIndexInGlobal = 0;

        public ushort m_MV_MetaCode = 52;

        public float fMV_Value = 0f;
    }
```

## *Annexure C : List<MetaVariableInfoEx>*

List of Structure required for advance Update Metavariable

```
public class MetaVariableInfoEx : MetaVariableInfo
    {
        public TE_DATA_TYPE DataType = TE_DATA_TYPE.MP_DATA_TYPE_MetaValue;

        public byte Error = 0;

        public MetaVariableInfoEx()
      {
        }
}
```

## Annexure D : List<CMetavariableDataCodeApply>

List of Structure required for apply UDP communication.

```
public class CMetavariableDataCodeApply
    {
        public ushort m_GlobalIndex = ushort.MaxValue;

        public TE_DATA_TYPE m_MV_ValueType = TE_DATA_TYPE.MP_DATA_TYPE_MetaValue;

        public ushort m_MV_MetaCode = ushort.MaxValue;

       public EReadWriteMode ReadWriteMode = EReadWriteMode.Read;

        public CMetavariableDataCodeApply()
        {
         }
   }
```

## *Annexure E : EParameterDataType*

Supports updated variable enumerations.

```
public enum EParameterDataType : ushort
    {
        NormCapacity = 0,
        IMax,
        VMax,
```

```
        VMin,
        Mass,
        SCapacity,
        NIR,
        NVoltage,
        NCapacitance,
        IsAutoCalculate
    }
```

## *Annexure F : TE_DATA_TYPE*

Supports data type enumerations.

```
public enum TE_DATA_TYPE
{
            /// <summary>
            /// Meta Value
            /// </summary>
            MP_DATA_TYPE_MetaValue = 1,
            /// <summary>
            /// Auxiliary temperature
            /// </summary>
            MP_DATA_TYPE_AuxTemperature = 10,
            /// <summary>
            /// Auxiliary temperature dt
            /// </summary>
            MP_DATA_TYPE_AuxdTdt,
            /// <summary>
            /// Auxiliary voltage
            /// </summary>
            MP_DATA_TYPE_AuxVoltage,
            /// <summary>
            /// Auxiliary voltage dt
            /// </summary>
            MP_DATA_TYPE_AuxdVdt,
            /// <summary>
            /// Auxiliary pressure
            /// </summary>
            MP_DATA_TYPE_AuxPressure,
            /// <summary>
            /// Auxiliary pressure dt
            /// </summary>
            MP_DATA_TYPE_AuxdPdt,
            /// <summary>
            /// Auxiliary pH
            /// </summary>
            MP_DATA_TYPE_AuxPh,
            /// <summary>
            /// Auxiliary pH dt
            /// </summary>
            MP_DATA_TYPE_AuxdPhVdt,
            /// <summary>
            /// Auxiliary flow rate
            /// </summary>
            MP_DATA_TYPE_AuxFlowRate,
            /// <summary>
            /// Auxiliary flow rate dt
```

```
            /// </summary>
            MP_DATA_TYPE_AuxdFRdt,
            /// <summary>
            /// Auxiliary density
            /// </summary>
            MP_DATA_TYPE_AuxConcentration,
            /// <summary>
            /// Auxiliary density dt
            /// </summary>
            MP_DATA_TYPE_AuxdConcentrationdt,
            /// <summary>
            /// Auxiliary DI
            /// </summary>
            MP_DATA_TYPE_AuxDI,
            /// <summary>
            /// Auxiliary DO
            /// </summary>
            MP_DATA_TYPE_AuxDO,
            /// <summary>
            /// CANBMS
            /// </summary>
            MP_DATA_TYPE_CANBMS,
            /// <summary>
            /// Auxiliary external charge
            /// </summary>
            MP_DATA_TYPE_AuxExternalCharge,
            /// <summary>
            /// Auxiliary humidity
            /// </summary>
            MP_DATA_TYPE_AuxHumidity,
            /// <summary>
            /// Auxiliary humidity dt
            /// </summary>
            MP_DATA_TYPE_AuxdHumdt,
            /// <summary>
            /// SMB
            /// </summary>
            MP_DATA_TYPE_SMBBMS,
            /// <summary>
            /// Advanced Formula
            /// </summary>
            MP_DATA_TYPE_ADVFORMULA,
            /// <summary>
            /// Auxiliary AO
            /// </summary>
            MP_DATA_TYPE_AuxAO,
            /// <summary>
            /// SPTT_EQ
            /// </summary>
            MP_DATA_TYPE_SPTT_EQ,
            /// <summary>
            /// SPTT_CELL
            /// </summary>
            MP_DATA_TYPE_SPTT_CELL
};
```

## Annexure G: Meta Codes for MV_UDs

Meta codes for Various MV_UDs User Defined Meta Variables

```
public enum Enum_MvUd
    {
        MetaCode_MV_UD1 = 52,
        MetaCode_MV_UD2 = 53,
        MetaCode_MV_UD3 = 54,
        MetaCode_MV_UD4 = 55,
        MetaCode_MV_UD5 = 105,
        MetaCode_MV_UD6 = 106,
        MetaCode_MV_UD7 = 107,
        MetaCode_MV_UD8 = 108,
        MetaCode_MV_UD9 = 109,
        MetaCode_MV_UD10 = 110,
        MetaCode_MV_UD11 = 111,
        MetaCode_MV_UD12 = 112,
        MetaCode_MV_UD13 = 113,
        MetaCode_MV_UD14 = 114,
        MetaCode_MV_UD15 = 115,
        MetaCode_MV_UD16 = 116
    }
```

## Annexure H : Sample Code

This file contains test methods for testing various functions of ArbinClient class.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using ArbinCTI.Core;
using System;
using System.Collections.Generic;
using ArbinCTI.Core.Control;
using ArbinCTI.Core.Inteface;
using System.Diagnostics;
using System.Threading;
using System.IO;

namespace ArbinCTI.Core
{
    /// <summary>
    /// <para>The <see cref="ArbinCTI.Core"/> Provide connection service.</para>
    /// <para><see cref="ArbinClient"/> is used to connect to CTI.</para>
    /// <see cref="ArbinCommand"/> contains the basic information of the CTI return packet, which is the
main base class for all command classes.
    /// </summary>
    [TestClass()]
    public class ArbinClientTests
    {
        private ArbinClient client;
        private MyArbinControl ctrl;
        MyTestData testData;
        private bool? connected;

        static string[] lines = File.ReadAllLines("../../../TestConfig.txt");
        private readonly string MyIP = lines[0];
        private readonly string UserName = lines[1];
        private readonly string Password = lines[2];
```

```
        private readonly string Schedule = lines[3];
        private readonly string SerialNumber = lines[4];
        private float NominalCapacity = float.Parse(lines[5]);
        private readonly int WrongChnlIndex = int.Parse(lines[7]);
        readonly static DateTime A = DateTime.Now;
        readonly static string B = A.ToString("yy/MM/dd HH/mm/ss");
        readonly static string TestName = lines[6] + B;
private readonly List<ushort> Chnl = new List<ushort>() { ushort.Parse(lines[8]) };

        public class MyTestData
        {
            public ArbinCommandLoginFeed LoginFeed { get; set; }
            public ArbinCommandAssignScheduleFeed ScheduleFeed { get; set; }
            public ArbinCommandLogicConnectFeed LogicConnectFeed { get; set; }
            public ArbinCommandStartChannelFeed StartChannelFeed { get; set; }
            public ArbinCommandBrowseDirectoryFeed BrowseDirectoryFeed { get; set; }
            public ArbinCommandStopChannelFeed StopChannelFeed { get; set; }
            public ArbinCommandGetSerialNumberFeed GetSerialNumberFeed { get; set; }
            public ArbinCommandResumChanneleFeed resumChanneleFeed { get; set; }
            public ArbinCommandNewOrDeleteFeed NewOrDeleteFeed { get; set; }
            public ArbinCommandJumpChannelFeed JumpChannelFeed { get; set; }
            public ArbinCommandGetStartDataFeed GetStartDataFeed { get; set; }
            public ArbinCommandGetChannelDataFeed GetChannelDataFeed { get; set; }
            public ArbinCommandSetMetaVariableFeed SetMetaVariableFeed { get; set; }
            public ArbinCommandUpLoadFileFeed UpLoadFileFeed { get; set; }
        }

        public class MyArbinControl : ArbinControl
        {
            public MyTestData TestData { get; }
    public delegate void UploadFileFeedbackEvent(ArbinCommandUpLoadFileFeed feedback);
            public event UploadFileFeedbackEvent UploadFileFeedback;

            public MyArbinControl(MyTestData testData)
            {
                TestData = testData;
            }
            public override void OnAssignScheduleFeedBack(ArbinCommandAssignScheduleFeed cmd)
            {
                TestData.ScheduleFeed = cmd;
            }
            public override void OnBrowseDirectoryBack(ArbinCommandBrowseDirectoryFeed cmd)
            {
                TestData.BrowseDirectoryFeed = cmd;
            }
            public override void OnGetChannelsDataFeedBack(ArbinCommandGetChannelDataFeed cmd)
            {
                TestData.GetChannelDataFeed = cmd;
            }
            public override void OnGetSerialNumberFeedBack(ArbinCommandGetSerialNumberFeed cmd)
            {
                TestData.GetSerialNumberFeed = cmd;
            }
            public override void OnGetStartDataBack(ArbinCommandGetStartDataFeed cmd)
            {
                TestData.GetStartDataFeed = cmd;
            }
            public override void OnJumpChannelFeedBack(ArbinCommandJumpChannelFeed cmd)
```

```
            {
                TestData.JumpChannelFeed = cmd;
            }
            public override void OnLogicConnectFeedBack(ArbinCommandLogicConnectFeed cmd)
            {
                TestData.LogicConnectFeed = cmd;
            }
            public override void OnNewOrDeleteBack(ArbinCommandNewOrDeleteFeed cmd)
            {
                TestData.NewOrDeleteFeed = cmd;
            }

            public override void OnResumeChannelFeedBack(ArbinCommandResumChanneleFeed cmd)
            {
                TestData.resumChanneleFeed = cmd;
            }

            public override void OnUpLoadFileBack(ArbinCommandUpLoadFileFeed cmd)
            {
                TestData.UpLoadFileFeed = cmd;
            }

public override void OnResumeChannelFeedBack(ArbinCommandResumChanneleFeed cmd)
            {
                TestData.resumChanneleFeed = cmd;
            }
            public override void OnSendMsgToCTIBack(ArbinCommandSendMsgToCTIFeed cmd)
            {
                TestData.SendMsgToCTIFeed = cmd;
            }
            public override void OnSetMetaVariableFeedBack(ArbinCommandSetMetaVariableFeed cmd)
            {
                TestData.SetMetaVariableFeed = cmd;
            }
            public override void OnStartAutomaticCalibrationBack(ArbinCommandStartAutomatic
CalibrationFeed cmd)
            {

            }
            public override void OnStartChannelFeedBack(ArbinCommandStartChannelFeed cmd)
            {
                TestData.StartChannelFeed = cmd;
            }
            public override void OnStopChannelFeedBack(ArbinCommandStopChannelFeed cmd)
            {
                TestData.StopChannelFeed = cmd;
            }
            public override void OnUpdateMetaVariableAdvancedFeedBack(ArbinCommandUpdate
MetaVariableAdvancedFeed cmd)
            {

            }
            public override void OnUpLoadFileBack(ArbinCommandUpLoadFileFeed cmd)
            {

            }
            public override void OnUserLoginFeedBack(ArbinCommandLoginFeed cmd)
            {
```

```
                TestData.LoginFeed = cmd;
            }
            public override void OnContinueChannelFeedBack(ArbinCommandContinueChannelFeed cmd)
            {
                throw new NotImplementedException();
            }
            public override void OnDeleteFileBack(ArbinCommandDeleteFileFeed cmd)
            {
                throw new NotImplementedException();
            }
        }

        /// <summary>
        /// This is a test project for CTI.
        /// Initialization of tests.
        /// </summary>
        [TestInitialize]
        public void InitializeTests()
        {
            testData = new MyTestData();
            ctrl = new MyArbinControl(testData);
            ctrl.Start();
            client = new ArbinClient();
            client.OnConnectionChanged += (IArbinSocket Socket,
ArbinSocketEventArgSet.SocketConnectionEventArgs e) =>
            {
                connected = e.Connected;
            };
            ctrl.ListenSocketRecv(client);
            client.ConnectAsync(MyIP, 9031, 0, out int err);
            while (true)
            {
                if (connected.HasValue)
                {
                    ctrl.PostLogicConnect(client, true);
                    ctrl.PostUserLogin(client, UserName, Password);
break;
                }
            }
        }

        [TestMethod()]
        public void UserLoginSuccessTest()
        {
            while (true)
            {
                if (testData.LoginFeed != null)
                {
                    Console.WriteLine(testData.LoginFeed.Result);
                    Assert.IsTrue(testData.LoginFeed.Result == ArbinCommandLoginFeed.LOGIN_RESULT.
CTI_LOGIN_SUCCESS);
                    break;
                }
            }
        }

        [TestMethod()]
        public void UserLoginFailedTest()
```

```
        {
            testData = new MyTestData();
            ctrl = new MyArbinControl(testData);
            ctrl.Start();
            connected = null;
            client = new ArbinClient();
            client.OnConnectionChanged += (IArbinSocket Socket,
ArbinSocketEventArgSet.SocketConnectionEventArgs e) =>
            {
                connected = e.Connected;
            };
            ctrl.ListenSocketRecv(client);
            client.ConnectAsync(MyIP, 9031, 0, out int err);
            Assert.IsTrue(err == 0);
            while (true)
            {
                if (connected.HasValue)
                {
                    ctrl.PostLogicConnect(client, true);
                    ctrl.PostUserLogin(client, "admin", "0001");
                    break;
                }
            }

            while (true)
            {
                if (testData.LoginFeed != null)
                {
                    Console.WriteLine(testData.LoginFeed.Result);
                    Assert.IsTrue(testData.LoginFeed.Result == ArbinCommandLoginFeed.LOGIN_RESULT.
CTI_LOGIN_FAILED);
                    break;
                }
            }
        }

        [TestMethod()]
        public void AssignScheduleSuccessTest()
        {
            ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0);
            while (true)
            {
                if (testData.ScheduleFeed != null)
                {
                    Console.WriteLine(testData.ScheduleFeed.Result);
                    Assert.IsTrue(testData.ScheduleFeed.Result == ArbinCommandAssignScheduleFeed.
ASSIGN_TOKEN.CTI_ASSIGN_SUCCESS);
                    break;
                }
            }
        }
[TestMethod()]
        public void ConnectAsyncTestConnectionSuccesful()
        {
            client = new ArbinClient();
            int result;
            int err;
```

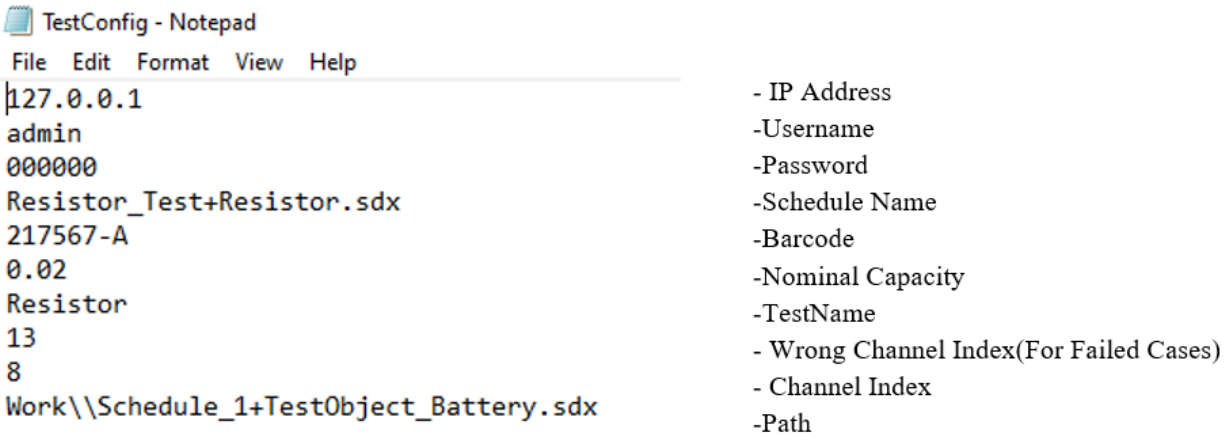
```
            result = client.ConnectAsync("127.0.0.1", 9031, 10, out err); //set IPAddress which you can connect
            Console.WriteLine("Connection Successful!");
            Assert.AreEqual(0, result);
        }

        [TestMethod()]
        public void ConnectAsyncTestConnectionTimeout()
        {
            client = new ArbinClient();
            int result;
            int err;
            result = client.ConnectAsync("192.168.2.114", 9031, 10, out err); //set IPAddress which you cannot connect
            Console.WriteLine("Connection Timeout!");
            Assert.AreEqual(-4, result);
        }

        [TestMethod]
        public void LogicConnectSuccessTest()
        {
            while (true)
            {
                if (testData.LogicConnectFeed != null)
                {
                    Console.WriteLine("Logic Connect: " + testData.LogicConnectFeed.dwSetKickOut);
                    Assert.IsTrue(testData.LogicConnectFeed.dwConnectResult == 0);
                    break;
                }
            }
        }

        [TestMethod()]
        public void IsConnected_Successfull()
        {
            bool Result;
            Result = client.IsConnected();
            Console.WriteLine("Connected to server!");
            Assert.IsTrue(Result);
        }

        [TestMethod()]
        public void IsConnected_Unsuccessfull()
        {
            client = new ArbinClient();
            bool Result;
            Result = client.IsConnected();
            Console.WriteLine("Not Connected to Server!");
            Assert.IsFalse(Result);
        }

        [TestMethod()]
        public void StartChannelSuccessTest()
        {
            ctrl.PostStopChannel(client, Chnl[0], true);
            ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0, false, Chnl[0]);
            ctrl.PostStartChannel(client, TestName, Chnl);
```

```
            while (true)
            {
                if (testData.StartChannelFeed != null)
                {
                    ctrl.PostStopChannel(client, Chnl[0], true);
                    Assert.IsTrue(testData.StartChannelFeed.Result == ArbinCommandStartChannel
Feed.START_TOKEN.CTI_START_SUCCESS);
                    break;
                } } }
[TestMethod]
public void StopChannelSuccessTest()
        {
            ctrl.PostAssignSchedule(client, Schedule, SerialNumber, Nominal Capacity, 0, 0, 0, 0, false,
Chnl[0]);
            ctrl.PostStartChannel(client, TestName, Chnl);
            ctrl.PostStopChannel(client, Chnl[0], false);
            while (true)
            {
                if (testData.StopChannelFeed != null)
                {
                    Console.WriteLine(testData.StopChannelFeed.Result);
                    Assert.IsTrue(testData.StopChannelFeed.Result == ArbinCommand
StopChannelFeed.STOP_TOKEN.SUCCESS);
                    break;
                }
            }
        }
[TestMethod()]
public void ResumeChannelResumeSuccessTest()
        {
            ctrl.PostAssignSchedule(client, Schedule, SerialNumber, Nominal Capacity, 0, 0, 0, 0, false,
Chnl[0]);
            ctrl.PostStartChannel(client, TestName, Chnl);
            Thread.Sleep(30);
            ctrl.PostStopChannel(client, Chnl[0], false);
            ctrl.PostResumeChannel(client, false, Chnl[0]);
            while (true)
            {
                if (testData.resumChanneleFeed != null)
                {
                    Assert.IsTrue(testData.resumChanneleFeed.Result == Arbin
CommandResumChanneleFeed.RESUME_TOKEN.RESUME_SUCCESS);
                    ctrl.PostStopChannel(client, Chnl[0], false);
                    break;
                }
            }
        }
[TestMethod]
        public void SetMetavariableTest()
        {
            ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0, false,
Chnl[0]);
            ctrl.PostStartChannel(client, TestName, Chnl);
            Thread.Sleep(30000);
            ctrl.PostGetChannelsData(client,
(uint)ArbinCommandGetChannelDataFeed.GET_CHANNELS_INFO_NEED_TYPE.THIRD_PARTY_GET_CHANNELS_INFO_NEED_TYPE_A
UX, (short)Chnl[0]);
            float a;
```

```
            while (true)
            {
                if (testData.GetChannelDataFeed != null)
                {
                    a = testData.GetChannelDataFeed.m_Channels[0].Current;
                    break;
                }
            }

            float b = a * 2;

            ctrl.PostSetMetaVariable(client, Chnl[0], 1, 52, 1, b);
            while (true)
            {
                if (testData.SetMetaVariableFeed != null)
                {
                    Assert.IsTrue(testData.SetMetaVariableFeed.Result == ArbinCommandSetMetaVariable
Feed.SET_MV_RESULT.CTI_SET_MV_SUCCESS);
                    break;
                }
            }
        }
[TestMethod]
        public void UpLoadFile_Success()
        {
            var A = DateTime.Now;
            string B = A.ToString("dd/MM mm/ss");
            string fileName = "UploadFile"+B+".sdx";
            var stream = new FileStream(fileName, FileMode.OpenOrCreate);
            for (int i = 0; i < 10; i++)
                stream.WriteByte((byte)'1');
            stream.Position = 0;
            long fileSize = stream.Length;

            AutoResetEvent done = new AutoResetEvent(false);
            ctrl.UploadFileFeedback += (ArbinCommandUpLoadFileFeed feedback) =>
            {
                done.Set();
            };

            long packByteSize = stream.Length;
            byte[] buffer = new byte[packByteSize];
            int packCount = (int)((fileSize % packByteSize) != 0 ? (fileSize / packByteSize) + 1 :
(fileSize / packByteSize));
            int lastPack = packCount - 1;
            for (int packIndex = 0; packIndex < packCount; packIndex++)
            {
                int readCount = stream.Read(buffer, 0, buffer.Length);
                byte[] dest = buffer;
                if (packIndex == lastPack)
                {
                    dest = new byte[readCount];
                    Buffer.BlockCopy(buffer, 0, dest, 0, readCount);
                }
                ctrl.PostUpLoadFile(client, fileName, dest, 0, (uint)packCount, (uint)packIndex);
                while(true)
                {
                    if (testData.UpLoadFileFeed != null)
```

```
                        break;
                }
                Console.WriteLine(testData.UpLoadFileFeed.Result);

                Assert.IsTrue(testData.UpLoadFileFeed.Result == ArbinCommandUpLoadFileFeed.
UPLOAD_RESULT.CTI_UPLOAD_SUCCESS);
            }
        }

```

***Test Config File:***




## Annexure I : ArbinCommand

ArbinCommand class

```
/// <summary>
/// It is the root of the CTI(Console TCP Interface) command type hierarchy.
/// </summary>
public class ArbinCommand
{
    /// <summary>
    /// Command code.
    /// </summary>
    public ArbinCommandCode dwCmd;

    /// <summary>
    /// This parameter is reserved and must be <b>Zero</b>.
    /// </summary>
    public uint dwCmd_Extend;

    /// <summary>
    /// Know that it is the command that the CTI(Console TCP Interface) client gets.
    /// </summary>
    public IArbinSocket socket = null;

    /// <summary>
    /// Channel number starts from zero.
```

```
    /// -1: All Channel
    /// </summary>
    public int Channel = -1;

    /// <summary>
    /// Channel index list
    /// </summary>
    public List<ushort> ChannelIndexList;
}
```

## Annexure J : TimeSensitiveSetMVArgs

TimeSensitiveSetMVArgs class

```
    public class TimeSensitiveSetMVArgs
    {
        public class TimeSensitiveSetMVChannel : IComparer<TimeSensitiveSetMVChannel>
        {
            /// <summary>
            /// IV Channel Global Index, starts from zero.
            /// </summary>
            public int GlobalIndex = 0;

            /// <summary>
            /// If true, it will be logged into the database.
            /// </summary>
            public bool IsDoLog = true;

            public int TimeSensitiveSetMVCount;

            public TimeSensitiveSetMVChannel();

            public TimeSensitiveSetMVChannel(int nGlobalIndex, List<TimeSensitiveSetMV> args, bool bDoLog = true);

            public TimeSensitiveSetMV GetSensitiveSetMV(int nIndex);

            public int Compare(TimeSensitiveSetMVChannel x, TimeSensitiveSetMVChannel y);
        }

        /// <summary>
        /// If the interval between the next call command and the previous call command is greater than the Timeout, the IV channel is automatically stopped this time.
        /// </summary>
        public float Timeout = 1.0f;

        public List<TimeSensitiveSetMVChannel> Channels;
    }

    public class TimeSensitiveSetMV
    {
        /// <summary>
        /// MV UD Enumeration
        /// </summary>
        public enum EMVUD
        {
            MVUD1 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD1,
```

```
            MVUD2 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD2,
            MVUD3 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD3,
            MVUD4 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD4,
            MVUD5 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD5,
            MVUD6 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD6,
            MVUD7 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD7,
            MVUD8 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD8,
            MVUD9 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD9,
            MVUD10 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD10,
            MVUD11 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD11,
            MVUD12 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD12,
            MVUD13 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD13,
            MVUD14 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD14,
            MVUD15 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD15,
            MVUD16 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD16,
        }

        public EMVUD MVUD = EMVUD.MVUD1;
        public float Value = 0.0f;

        public TimeSensitiveSetMV();

        public TimeSensitiveSetMV(EMVUD eMVUD, float fValue);
    }
```