

Annexure

Annexure A : List<StartResumeEx>

▼ List of structures required for advanced start and advanced resume.

```
1 public struct StartResumeEx
2 {
3     public uint channelIndex;
4     public int TestID;
5     public string TestNames;
6     public string Schedules;
7     public uint nSelectSteps;
8     public uint Cycle;
9     public double TestTime;
10    public double StepTime;
11    public double CCapacity;
12    public double DCapacity;
13    public double CEnergy;
14    public double DEnergy;
15    public double TC_Time1;
16    public double TC_Time2;
17    public double TC_Time3;
18    public double TC_Time4;
19    public double TC_CCapacity1;
20    public double TC_CCapacity2;
21    public double TC_DCapacity1;
22    public double TC_DCapacity2;
23    public double TC_CEnergy1;
24    public double TC_CEnergy2;
25    public double TC_DEnergy1;
26    public double TC_DEnergy2;
27    public float MV_Counter1;
28    public float MV_Counter2;
29    public float MV_Counter3;
30    public float MV_Counter4;
31    public float MV_Counter5;
32    public float MV_Counter6;
33    public float MV_Counter7;
34    public float MV_Counter8;
35    public double ChargeCapacityTime;
36    public double DischargeCapacityTime;
37    public float MVUD1;
38    public float MVUD2;
39    public float MVUD3;
40    public float MVUD4;
41    public float MVUD5;
42    public float MVUD6;
43    public float MVUD7;
44    public float MVUD8;
45    public float MVUD9;
46    public float MVUD10;
47    public float MVUD11;
```

```
48     public float MVUD12;
49     public float MVUD13;
50     public float MVUD14;
51     public float MVUD15;
52     public float MVUD16;
53
54     public void Reset ()
55     {
56         nSelectSteps = 0;
57
58         Cycle = 0;
59         TestTime = 0;
60         StepTime = 0;
61         CCapacity = 0;
62         DCapacity = 0;
63         CEnergy = 0;
64         DEnergy = 0;
65         TC_Time1 = 0;
66         TC_Time2 = 0;
67         TC_Time3 = 0;
68         TC_Time4 = 0;
69         TC_CCapacity1 = 0;
70         TC_CCapacity2 = 0;
71         TC_DCapacity1 = 0;
72         TC_DCapacity2 = 0;
73         TC_CEnergy1 = 0;
74         TC_CEnergy2 = 0;
75         TC_DEnergy1 = 0;
76         TC_DEnergy2 = 0;
77         MV_Counter1 = 0;
78         MV_Counter2 = 0;
79         MV_Counter3 = 0;
80         MV_Counter4 = 0;
81         MV_Counter5 = 0;
82         MV_Counter6 = 0;
83         MV_Counter7 = 0;
84         MV_Counter8 = 0;
85         ChargeCapacityTime = 0;
86         DischargeCapacityTime=0;
87         MVUD1 = 0;
88         MVUD2 = 0;
89         MVUD3 = 0;
90         MVUD4 = 0;
91         MVUD5 = 0;
92         MVUD6 = 0;
93         MVUD7 = 0;
94         MVUD8 = 0;
95         MVUD9 = 0;
96         MVUD10 = 0;
97         MVUD11 = 0;
98         MVUD12 = 0;
99         MVUD13 = 0;
100        MVUD14 = 0;
101        MVUD15 = 0;
102        MVUD16 = 0;
103    }
```

Annexure B : List<MetaVariableInfo>

- ▼ List of Structure required for advance Update Metavariable.

```
1 public class MetaVariableInfo
2 {
3     public ushort m_ChannelIndexInGlobal = 0;
4
5     public ushort m_MV_MetaCode = 52;
6
7     public float fMV_Value = 0f;
8 }
```

Annexure C : List<MetaVariableInfoEx>

- ▼ List of Structure required for advance Update Metavariable

```
1 public class MetaVariableInfoEx : MetaVariableInfo
2 {
3     public TE_DATA_TYPE DataType = TE_DATA_TYPE.MP_DATA_TYPE_MetaValue;
4
5     public byte Error = 0;
6
7     public MetaVariableInfoEx()
8     {
9     }
10 }
```

Annexure D : List<CMetavariableDataCodeApply>

- ▼ List of Structure required for apply UDP communication.

```
1 public class CMetavariableDataCodeApply
2 {
3     public ushort m_GlobalIndex = ushort.MaxValue;
4
5     public TE_DATA_TYPE m_MV_ValueType = TE_DATA_TYPE.MP_DATA_TYPE_MetaValue;
6
7     public ushort m_MV_MetaCode = ushort.MaxValue;
8
9     public EReadWriteMode ReadWriteMode = EReadWriteMode.Read;
10
11     public CMetavariableDataCodeApply()
12     {
13     }
14 }
```

Annexure E : EParameterDataType

- ▼ Supports updated variable enumerations.

```
1 public enum EParameterDataType : ushort
2 {
3     NormCapacity = 0,
4     IMax,
5     VMax,
```

```

6      VMin,
7      Mass,
8      SCapacity,
9      NIR,
10     NVoltage,
11     NCapacitance,
12     IsAutoCalculate
13 }

```

Annexure F : TE_DATA_TYPE

▼ Supports data type enumerations.

```

1  public enum TE_DATA_TYPE
2  {
3      /// <summary>
4      /// Meta Value
5      /// </summary>
6      MP_DATA_TYPE_MetaValue = 1,
7      /// <summary>
8      /// Auxiliary temperature
9      /// </summary>
10     MP_DATA_TYPE_AuxTemperature = 10,
11     /// <summary>
12     /// Auxiliary temperature dt
13     /// </summary>
14     MP_DATA_TYPE_AuxdTdt,
15     /// <summary>
16     /// Auxiliary voltage
17     /// </summary>
18     MP_DATA_TYPE_AuxVoltage,
19     /// <summary>
20     /// Auxiliary voltage dt
21     /// </summary>
22     MP_DATA_TYPE_AuxVdt,
23     /// <summary>
24     /// Auxiliary pressure
25     /// </summary>
26     MP_DATA_TYPE_AuxPressure,
27     /// <summary>
28     /// Auxiliary pressure dt
29     /// </summary>
30     MP_DATA_TYPE_AuxPdtdt,
31     /// <summary>
32     /// Auxiliary pH
33     /// </summary>
34     MP_DATA_TYPE_AuxPh,
35     /// <summary>
36     /// Auxiliary pH dt
37     /// </summary>
38     MP_DATA_TYPE_AuxPhVdt,
39     /// <summary>
40     /// Auxiliary flow rate
41     /// </summary>
42     MP_DATA_TYPE_AuxFlowRate,
43     /// <summary>
44     /// Auxiliary flow rate dt

```

```

45      /// </summary>
46      MP_DATA_TYPE_AuxdFRdt,
47      /// <summary>
48      /// Auxiliary density
49      /// </summary>
50      MP_DATA_TYPE_AuxConcentration,
51      /// <summary>
52      /// Auxiliary density dt
53      /// </summary>
54      MP_DATA_TYPE_AuxdConcentrationdt,
55      /// <summary>
56      /// Auxiliary DI
57      /// </summary>
58      MP_DATA_TYPE_AuxDI,
59      /// <summary>
60      /// Auxiliary DO
61      /// </summary>
62      MP_DATA_TYPE_AuxDO,
63      /// <summary>
64      /// CANBMS
65      /// </summary>
66      MP_DATA_TYPE_CANBMS,
67      /// <summary>
68      /// Auxiliary external charge
69      /// </summary>
70      MP_DATA_TYPE_AuxExternalCharge,
71      /// <summary>
72      /// Auxiliary humidity
73      /// </summary>
74      MP_DATA_TYPE_AuxHumidity,
75      /// <summary>
76      /// Auxiliary humidity dt
77      /// </summary>
78      MP_DATA_TYPE_AuxdHumdt,
79      /// <summary>
80      /// SMB
81      /// </summary>
82      MP_DATA_TYPE_SMBBMS,
83      /// <summary>
84      /// Advanced Formula
85      /// </summary>
86      MP_DATA_TYPE_ADVFORMULA,
87      /// <summary>
88      /// Auxiliary A0
89      /// </summary>
90      MP_DATA_TYPE_AuxA0,
91      /// <summary>
92      /// SPTT_EQ
93      /// </summary>
94      MP_DATA_TYPE_SPTT_EQ,
95      /// <summary>
96      /// SPTT_CELL
97      /// </summary>
98      MP_DATA_TYPE_SPTT_CELL
99  };

```

Annexure G: Meta Codes for MV_UDs

▼ Meta codes for Various MV_UDs User Defined Meta Variables

```
1 public enum Enum_MvUd
2 {
3     MetaCode_MV_UD1 = 52,
4     MetaCode_MV_UD2 = 53,
5     MetaCode_MV_UD3 = 54,
6     MetaCode_MV_UD4 = 55,
7     MetaCode_MV_UD5 = 105,
8     MetaCode_MV_UD6 = 106,
9     MetaCode_MV_UD7 = 107,
10    MetaCode_MV_UD8 = 108,
11    MetaCode_MV_UD9 = 109,
12    MetaCode_MV_UD10 = 110,
13    MetaCode_MV_UD11 = 111,
14    MetaCode_MV_UD12 = 112,
15    MetaCode_MV_UD13 = 113,
16    MetaCode_MV_UD14 = 114,
17    MetaCode_MV_UD15 = 115,
18    MetaCode_MV_UD16 = 116
19 }
```

Annexure H : Sample Code

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

```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2 using ArbinCTI.Core;
3 using System;
4 using System.Collections.Generic;
5 using ArbinCTI.Core.Control;
6 using ArbinCTI.Core.Interface;
7 using System.Diagnostics;
8 using System.Threading;
9 using System.IO;
10
11 namespace ArbinCTI.Core
12 {
13     /// <summary>
14     /// <para>The <see cref="ArbinCTI.Core"/> Provide connection service.</para>
15     /// <para><see cref="ArbinClient"/> is used to connect to CTI.</para>
16     /// <see cref="ArbinCommand"/> contains the basic information of the CTI return packet, which is the
17     main base class for all command classes.
18     /// </summary>
19     [TestClass]
20     public class ArbinClientTests
21     {
22         private ArbinClient client;
23         private MyArbinControl ctrl;
24         MyTestData testData;
25         private bool? connected;
26
27         static string[] lines = File.ReadAllLines("../TestConfig.txt");
28         private readonly string MyIP = lines[0];
29         private readonly string UserName = lines[1];
30         private readonly string Password = lines[2];
```

```

30     private readonly string Schedule = lines[3];
31     private readonly string SerialNumber = lines[4];
32     private float NominalCapacity = float.Parse(lines[5]);
33     private readonly int WrongChnlIndex = int.Parse(lines[7]);
34     readonly static DateTime A = DateTime.Now;
35     readonly static string B = A.ToString("yy/MM/dd HH/mm/ss");
36     readonly static string TestName = lines[6] + B;
37     private readonly List<ushort> Chnl = new List<ushort>() { ushort.Parse(lines[8]) };
38
39     public class MyTestData
40     {
41         public ArbinCommandLoginFeed LoginFeed { get; set; }
42         public ArbinCommandAssignScheduleFeed ScheduleFeed { get; set; }
43         public ArbinCommandLogicConnectFeed LogicConnectFeed { get; set; }
44         public ArbinCommandStartChannelFeed StartChannelFeed { get; set; }
45         public ArbinCommandBrowseDirectoryFeed BrowseDirectoryFeed { get; set; }
46         public ArbinCommandStopChannelFeed StopChannelFeed { get; set; }
47         public ArbinCommandGetSerialNumberFeed GetSerialNumberFeed { get; set; }
48         public ArbinCommandResumChanneleFeed resumChanneleFeed { get; set; }
49         public ArbinCommandNewOrDeleteFeed NewOrDeleteFeed { get; set; }
50         public ArbinCommandJumpChannelFeed JumpChannelFeed { get; set; }
51         public ArbinCommandGetStartDataFeed GetStartDataFeed { get; set; }
52         public ArbinCommandGetChannelDataFeed GetChannelDataFeed { get; set; }
53         public ArbinCommandSetMetaVariableFeed SetMetaVariableFeed { get; set; }
54         public ArbinCommandUpLoadFileFeed UpLoadFileFeed { get; set; }
55     }
56
57     public class MyArbinControl : ArbinControl
58     {
59         public MyTestData TestData { get; }
60         public delegate void UploadFileFeedbackEvent(ArbinCommandUpLoadFileFeed feedback);
61         public event UploadFileFeedbackEvent UploadFileFeedback;
62
63         public MyArbinControl(MyTestData testData)
64         {
65             TestData = testData;
66         }
67         public override void OnAssignScheduleFeedBack(ArbinCommandAssignScheduleFeed cmd)
68         {
69             TestData.ScheduleFeed = cmd;
70         }
71         public override void OnBrowseDirectoryBack(ArbinCommandBrowseDirectoryFeed cmd)
72         {
73             TestData.BrowseDirectoryFeed = cmd;
74         }
75         public override void OnGetChannelsDataFeedBack(ArbinCommandGetChannelDataFeed cmd)
76         {
77             TestData.GetChannelDataFeed = cmd;
78         }
79         public override void OnGetSerialNumberFeedBack(ArbinCommandGetSerialNumberFeed cmd)
80         {
81             TestData.GetSerialNumberFeed = cmd;
82         }
83         public override void OnGetStartDataBack(ArbinCommandGetStartDataFeed cmd)
84         {
85             TestData.GetStartDataFeed = cmd;
86         }
87         public override void OnJumpChannelFeedBack(ArbinCommandJumpChannelFeed cmd)

```

```

88         {
89             TestData.JumpChannelFeed = cmd;
90         }
91     public override void OnLogicConnectFeedBack(ArbinCommandLogicConnectFeed cmd)
92     {
93         TestData.LogicConnectFeed = cmd;
94     }
95     public override void OnNewOrDeleteBack(ArbinCommandNewOrDeleteFeed cmd)
96     {
97         TestData.NewOrDeleteFeed = cmd;
98     }
99
100     public override void OnResumeChannelFeedBack(ArbinCommandResumChanneleFeed cmd)
101     {
102         TestData.resumChanneleFeed = cmd;
103     }
104
105     public override void OnUpLoadFileBack(ArbinCommandUpLoadFileFeed cmd)
106     {
107         TestData.UpLoadFileFeed = cmd;
108     }
109
110     public override void OnResumeChannelFeedBack(ArbinCommandResumChanneleFeed cmd)
111     {
112         TestData.resumChanneleFeed = cmd;
113     }
114     public override void OnSendMsgToCTIBack(ArbinCommandSendMsgToCTIFeed cmd)
115     {
116         TestData.SendMsgToCTIFeed = cmd;
117     }
118     public override void OnSetMetaVariableFeedBack(ArbinCommandSetMetaVariableFeed cmd)
119     {
120         TestData.SetMetaVariableFeed = cmd;
121     }
122     public override void OnStartAutomaticCalibrationBack(ArbinCommandStartAutomatic
CalibrationFeed cmd)
123     {
124
125     }
126     public override void OnStartChannelFeedBack(ArbinCommandStartChannelFeed cmd)
127     {
128         TestData.StartChannelFeed = cmd;
129     }
130     public override void OnStopChannelFeedBack(ArbinCommandStopChannelFeed cmd)
131     {
132         TestData.StopChannelFeed = cmd;
133     }
134     public override void OnUpdateMetaVariableAdvancedFeedBack(ArbinCommandUpdate
MetaVariableAdvancedFeed cmd)
135     {
136
137     }
138     public override void OnUpLoadFileBack(ArbinCommandUpLoadFileFeed cmd)
139     {
140
141     }
142     public override void OnUserLoginFeedBack(ArbinCommandLoginFeed cmd)
143     {

```



```

144         TestData.LoginFeed = cmd;
145     }
146     public override void OnContinueChannelFeedBack(ArbinCommandContinueChannelFeed cmd)
147     {
148         throw new NotImplementedException();
149     }
150     public override void OnDeleteFileBack(ArbinCommandDeleteFileFeed cmd)
151     {
152         throw new NotImplementedException();
153     }
154 }
155
156 /// <summary>
157 /// This is a test project for CTI.
158 /// Initialization of tests.
159 /// </summary>
160 [TestInitialize]
161 public void InitializeTests()
162 {
163     testData = new MyTestData();
164     ctrl = new MyArbinControl(testData);
165     ctrl.Start();
166     client = new ArbinClient();
167     client.OnConnectionChanged += (IArbinSocket Socket,
ArbinSocketEventArgs e) =>
168     {
169         connected = e.Connected;
170     };
171     ctrl.ListenSocketRecv(client);
172     client.ConnectAsync(MyIP, 9031, 0, out int err);
173     while (true)
174     {
175         if (connected.HasValue)
176         {
177             ctrl.PostLogicConnect(client, true);
178             ctrl.PostUserLogin(client, UserName, Password);
179             break;
180         }
181     }
182 }
183
184 [TestMethod()]
185 public void UserLoginSuccessTest()
186 {
187     while (true)
188     {
189         if (testData.LoginFeed != null)
190         {
191             Console.WriteLine(testData.LoginFeed.Result);
192             Assert.IsTrue(testData.LoginFeed.Result == ArbinCommandLoginFeed.LOGIN_RESULT.
CTI_LOGIN_SUCCESS);
193             break;
194         }
195     }
196 }
197
198 [TestMethod()]
199 public void UserLoginFailedTest()

```

```

200     {
201         testData = new MyTestData();
202         ctrl = new MyArbinControl(testData);
203         ctrl.Start();
204         connected = null;
205         client = new ArbinClient();
206         client.OnConnectionChanged += (IArbinSocket Socket,
ArbinSocketEventArgs e) =>
207         {
208             connected = e.Connected;
209         };
210         ctrl.ListenSocketRecv(client);
211         client.ConnectAsync(MyIP, 9031, 0, out int err);
212         Assert.IsTrue(err == 0);
213         while (true)
214         {
215             if (connected.HasValue)
216             {
217                 ctrl.PostLogicConnect(client, true);
218                 ctrl.PostUserLogin(client, "admin", "0001");
219                 break;
220             }
221         }
222
223         while (true)
224         {
225             if (testData.LoginFeed != null)
226             {
227                 Console.WriteLine(testData.LoginFeed.Result);
228                 Assert.IsTrue(testData.LoginFeed.Result == ArbinCommandLoginFeed.LOGIN_RESULT.
CTI_LOGIN_FAILED);
229                 break;
230             }
231         }
232     }
233
234     [TestMethod()]
235     public void AssignScheduleSuccessTest()
236     {
237         ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0);
238         while (true)
239         {
240             if (testData.ScheduleFeed != null)
241             {
242                 Console.WriteLine(testData.ScheduleFeed.Result);
243                 Assert.IsTrue(testData.ScheduleFeed.Result == ArbinCommandAssignScheduleFeed.
ASSIGN_TOKEN.CTI_ASSIGN_SUCCESS);
244                 break;
245             }
246         }
247     }
248     [TestMethod()]
249     public void ConnectAsyncTestConnectionSuccessful()
250     {
251         client = new ArbinClient();
252         int result;
253         int err;

```

```

254         result = client.ConnectAsync("127.0.0.1", 9031, 10, out err); //set IPAddress which you can
connect
255         Console.WriteLine("Connection Successful!");
256         Assert.AreEqual(0, result);
257     }
258
259     [TestMethod()]
260     public void ConnectAsyncTestConnectionTimeout()
261     {
262         client = new ArbinClient();
263         int result;
264         int err;
265         result = client.ConnectAsync("192.168.2.114", 9031, 10, out err); //set IPAddress which you
cannot connect
266         Console.WriteLine("Connection Timeout!");
267         Assert.AreEqual(-4, result);
268     }
269
270     [TestMethod]
271     public void LogicConnectSuccessTest()
272     {
273         while (true)
274         {
275             if (testData.LogicConnectFeed != null)
276             {
277                 Console.WriteLine("Logic Connect: " + testData.LogicConnectFeed.dwSetKickOut);
278                 Assert.IsTrue(testData.LogicConnectFeed.dwConnectResult == 0);
279                 break;
280             }
281         }
282     }
283
284     [TestMethod()]
285     public void IsConnected_Successfull()
286     {
287         bool Result;
288         Result = client.IsConnected();
289         Console.WriteLine("Connected to server!");
290         Assert.IsTrue(Result);
291     }
292
293     [TestMethod()]
294     public void IsConnected_Unsuccessfull()
295     {
296         client = new ArbinClient();
297         bool Result;
298         Result = client.IsConnected();
299         Console.WriteLine("Not Connected to Server!");
300         Assert.IsFalse(Result);
301     }
302
303     [TestMethod()]
304     public void StartChannelSuccessTest()
305     {
306         ctrl.PostStopChannel(client, Chnl[0], true);
307         ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0, false,
Chnl[0]);
308         ctrl.PostStartChannel(client, TestName, Chnl);

```

```

309         while (true)
310         {
311             if (testData.StartChannelFeed != null)
312             {
313                 ctrl.PostStopChannel(client, Chnl[0], true);
314                 Assert.IsTrue(testData.StartChannelFeed.Result == ArbinCommandStartChannel
Feed.START_TOKEN.CTI_START_SUCCESS);
315                 break;
316             } } }
317 [TestMethod]
318 public void StopChannelSuccessTest()
319     {
320         ctrl.PostAssignSchedule(client, Schedule, SerialNumber, Nominal Capacity, 0, 0, 0, 0, false,
Chnl[0]);
321         ctrl.PostStartChannel(client, TestName, Chnl);
322         ctrl.PostStopChannel(client, Chnl[0], false);
323         while (true)
324         {
325             if (testData.StopChannelFeed != null)
326             {
327                 Console.WriteLine(testData.StopChannelFeed.Result);
328                 Assert.IsTrue(testData.StopChannelFeed.Result == ArbinCommand
StopChannelFeed.STOP_TOKEN.SUCCESS);
329                 break;
330             }
331         }
332     }
333 [TestMethod()]
334 public void ResumeChannelResumeSuccessTest()
335     {
336         ctrl.PostAssignSchedule(client, Schedule, SerialNumber, Nominal Capacity, 0, 0, 0, 0, false,
Chnl[0]);
337         ctrl.PostStartChannel(client, TestName, Chnl);
338         Thread.Sleep(30);
339         ctrl.PostStopChannel(client, Chnl[0], false);
340         ctrl.PostResumeChannel(client, false, Chnl[0]);
341         while (true)
342         {
343             if (testData.resumeChanneleFeed != null)
344             {
345                 Assert.IsTrue(testData.resumeChanneleFeed.Result == Arbin
CommandResumChanneleFeed.RESUME_TOKEN.RESUME_SUCCESS);
346                 ctrl.PostStopChannel(client, Chnl[0], false);
347                 break;
348             }
349         }
350     }
351 [TestMethod]
352 public void SetMetavariabaleTest()
353     {
354         ctrl.PostAssignSchedule(client, Schedule, SerialNumber, NominalCapacity, 0, 0, 0, 0, false,
Chnl[0]);
355         ctrl.PostStartChannel(client, TestName, Chnl);
356         Thread.Sleep(30000);
357         ctrl.PostGetChannelsData(client,
(uint)ArbinCommandGetChannelDataFeed.GET_CHANNELS_INFO_NEED_TYPE.THIRD_PARTY_GET_CHANNELS_INFO_NEED_TYPE_A
UX, (short)Chnl[0]);
358         float a;

```

```

359         while (true)
360         {
361             if (testData.GetChannelDataFeed != null)
362             {
363                 a = testData.GetChannelDataFeed.m_Channels[0].Current;
364                 break;
365             }
366         }
367
368         float b = a * 2;
369
370         ctrl.PostSetMetaVariable(client, Chnl[0], 1, 52, 1, b);
371         while (true)
372         {
373             if (testData.SetMetaVariableFeed != null)
374             {
375                 Assert.IsTrue(testData.SetMetaVariableFeed.Result == ArbinCommandSetMetaVariable
Feed.SET_MV_RESULT.CTI_SET_MV_SUCCESS);
376                 break;
377             }
378         }
379     }
380     [TestMethod]
381     public void UploadFile_Success()
382     {
383         var A = DateTime.Now;
384         string B = A.ToString("dd/MM mm/ss");
385         string fileName = "UploadFile"+B+".sdx";
386         var stream = new FileStream(fileName, FileMode.OpenOrCreate);
387         for (int i = 0; i < 10; i++)
388             stream.WriteByte((byte)'1');
389         stream.Position = 0;
390         long fileSize = stream.Length;
391
392         AutoResetEvent done = new AutoResetEvent(false);
393         ctrl.UploadFileFeedback += (ArbinCommandUploadFileFeed feedback) =>
394         {
395             done.Set();
396         };
397
398         long packByteSize = stream.Length;
399         byte[] buffer = new byte[packByteSize];
400         int packCount = (int)((fileSize % packByteSize) != 0 ? (fileSize / packByteSize) + 1 :
(fileSize / packByteSize));
401         int lastPack = packCount - 1;
402         for (int packIndex = 0; packIndex < packCount; packIndex++)
403         {
404             int readCount = stream.Read(buffer, 0, buffer.Length);
405             byte[] dest = buffer;
406             if (packIndex == lastPack)
407             {
408                 dest = new byte[readCount];
409                 Buffer.BlockCopy(buffer, 0, dest, 0, readCount);
410             }
411             ctrl.PostUpLoadFile(client, fileName, dest, 0, (uint)packCount, (uint)packIndex);
412             while(true)
413             {
414                 if (testData.UpLoadFileFeed != null)

```

```

415             break;
416         }
417         Console.WriteLine(testData.UploadFileFeed.Result);
418
419         Assert.IsTrue(testData.UploadFileFeed.Result == ArbinCommandUploadFileFeed.
UPLOAD_RESULT.CTI_UPLOAD_SUCCESS);
420     }
421 }
422

```

Test Config File:



TestConfig - Notepad

File Edit Format View Help

127.0.0.1	- IP Address
admin	-Username
000000	-Password
Resistor_Test+Resistor.sdx	-Schedule Name
217567-A	-Barcode
0.02	-Nominal Capacity
Resistor	-TestName
13	- Wrong Channel Index(For Failed Cases)
8	- Channel Index
Work\\Schedule_1+TestObject_Battery.sdx	-Path

Annexure I : ArbinCommand

▼ ArbinCommand class

```

1  /// <summary>
2  /// It is the root of the CTI(Console TCP Interface) command type hierarchy.
3  /// </summary>
4  public class ArbinCommand
5  {
6      /// <summary>
7      /// Command code.
8      /// </summary>
9      public ArbinCommandCode dwCmd;
10
11     /// <summary>
12     /// This parameter is reserved and must be <b>Zero</b>.
13     /// </summary>
14     public uint dwCmd_Extend;
15
16     /// <summary>
17     /// Know that it is the command that the CTI(Console TCP Interface) client gets.
18     /// </summary>
19     public IArbinSocket socket = null;
20
21     /// <summary>
22     /// Channel number starts from zero.

```

```

23     /// -1: All Channel
24     /// </summary>
25     public int Channel = -1;
26
27     /// <summary>
28     /// Channel index list
29     /// </summary>
30     public List<ushort> ChannelIndexList;
31 }

```

Annexure J : TimeSensitiveSetMVArgs

▼ TimeSensitiveSetMVArgs class

```

1     public class TimeSensitiveSetMVArgs
2     {
3         public class TimeSensitiveSetMVChannel : IComparer<TimeSensitiveSetMVChannel>
4         {
5             /// <summary>
6             /// IV Channel Global Index, starts from zero.
7             /// </summary>
8             public int GlobalIndex = 0;
9
10            /// <summary>
11            /// If true, it will be logged into the database.
12            /// </summary>
13            public bool IsDoLog = true;
14
15            public int TimeSensitiveSetMVCount;
16
17            public TimeSensitiveSetMVChannel();
18
19            public TimeSensitiveSetMVChannel(int nGlobalIndex, List<TimeSensitiveSetMV> args, bool bDoLog =
20            true);
21
22            public TimeSensitiveSetMV GetSensitiveSetMV(int nIndex);
23
24            public int Compare(TimeSensitiveSetMVChannel x, TimeSensitiveSetMVChannel y);
25        }
26
27        /// <summary>
28        /// If the interval between the next call command and the previous call command is greater than the
29        Timeout, the IV channel is automatically stopped this time.
30        /// </summary>
31        public float Timeout = 1.0f;
32
33        public List<TimeSensitiveSetMVChannel> Channels;
34    }
35
36    public class TimeSensitiveSetMV
37    {
38        /// <summary>
39        /// MV UD Enumeration
40        /// </summary>
41        public enum EMVUD
42        {
43            MVUD1 = ETE_META VARIABLE_CODE.MetaCode_MV_UD1,

```

```
42         MVUD2 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD2,
43         MVUD3 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD3,
44         MVUD4 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD4,
45         MVUD5 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD5,
46         MVUD6 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD6,
47         MVUD7 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD7,
48         MVUD8 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD8,
49         MVUD9 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD9,
50         MVUD10 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD10,
51         MVUD11 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD11,
52         MVUD12 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD12,
53         MVUD13 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD13,
54         MVUD14 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD14,
55         MVUD15 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD15,
56         MVUD16 = ETE_METAVARIABLE_CODE.MetaCode_MV_UD16,
57     }
58
59     public EMVUD MVUD = EMVUD.MVUD1;
60     public float Value = 0.0f;
61
62     public TimeSensitiveSetMV();
63
64     public TimeSensitiveSetMV(EMVUD eMVUD, float fValue);
65 }
```